"Байт Пайтона" (українська версія)

Реліз 2025

author: Swaroop Chitlur Translator: Daria JENS

трав. 07, 2026

Зміст

Це українське видання книги «A Byte of Python». Щиро вітаємо!

- URL-адреса цього проєкту: https://spielend-programmieren.at/byte_of_python_ukraine/
- Автор книги: Swaroop CH https://www.swaroopch.com/
  - Оригінал цієї книги (english): https://python.swaroopch.com/
- Перекладач: Daria JENS https://https://github.com/Daria-Jens
- Ліцензія цієї книги: creative-commons attribution share-alike 4.0 International (cc-by-sa 4.0)
  - en: https://creativecommons.org/licenses/by-sa/4.0/deed.en
  - ukr: https://creativecommons.org/licenses/by-sa/4.0/deed.uk
  - 

Розділ 1

Про цей переклад та про мене – коротка інформація для читачів

Дарія Йенс (Daria JENS)

Мене звати Дарія Йенс (Daria JENS), я родом із Одеси, Україна.Маю вищу економічну освіту та 13

років професійного досвіду в Україні. У 2019 році я переїхала до Відня, Австрія, де розпочала вивчення мови програмування Python, аби долучитися до освітнього проєкту мого чоловіка з викладання Python

Під час вивчення Python я використовувала оригінальну книгу "A Byte of Python" від Swaroop англійською мовою. Я знайшла кілька сайтів, які переклали окремі частини цієї книги українською, але повного перекладу не було, а також не було посилань на оригінальних авторів (Swaroop C H) чи на ліцензію Creative Commons Share-Alike.

Це мій перший переклад підручника з програмування, я додавала контент, коли це було необхідно для мого розуміння теми.

Якщо цей переклад буде корисний для вашого навчання, буду дуже рада отримати лист на адресу `jensdarya@gmail.com`. Якщо ви хочете допомогти покращити переклад, будь ласка, скористайтеся кнопкою у верхній частині кожної сторінки, щоб написати issue або надіслати pull request.

Розділ 2

Завантажити

- *Завантажити PDF версію*
- *Завантажити версію EPUB*

Розділ 3

Зміст

3.1 Посвята

Kalyan Varma і багатьом іншим старшим співробітникам PESIT , які познайомили нас із GNU/Linux і світом відкритого програмного забеспечення .

Пам'яті Atul Chitnis, друга та провідника, за яким будемо дуже сумувати.

Піонерам, які створили Інтернет. Ця книга була вперше написана в 2003 році. Вона досі залишається популярною завдяки характеру обміну знаннями в Інтернеті, як передбачали піонери.

3.2 Передмова

Python, ймовірно, одна з небагатьох мов програмування, яка одночасно є простою та потужною. Це добре як для початківців, так і для експертів, і, що важливіше,на Python цікаво програмувати. Ця книга має на меті допомогти вам вивчити цю чудову мову та показати, як виконувати завдання швидко й безболісно — фактично "Антиотрута проти всіх проблем у програмуванні".

3.2.1 Для кого ця книга

Ця книга служить провідником або посібником з мови програмування Python. Насамперед вона орієнтована на новачків. Проте вона також корисна для досвідчених програмістів.

Автор задумав цю книгу так, щоб вивчити Python по ній зміг будь-хто, хто вміє хоча б зберігати текстові файли, втім, і досвід програмування не перешкода.

Якщо у вас усе-таки є попередній досвід програмування, вас зацікавлять відмінності між Python і вашою улюбленою мовою програмування – я відзначив багато таких відмінностей. Невелике застереження, незабаром Python стане вашою улюбленою мовою програмування!

3.2.2 Офіційний веб-сайт

Офіційний веб-сайт книги https://python.swaroopch.com , де ви можете прочитати всю книгу онлайн, завантажити останні версії книги, купити друковану копію, а також надіслати мені відгук.

3.2.3 Є над чим замислитись

> Існує два способи побудови дизайну програмного забезпечення: один спосіб полягає в тому, щоб зробити його настільки простим, щоб вочевидь не було недоліків; інший — зробити його настільки складним, щоб не було очевидних недоліків. – C. A. R. Hoare
>
> Успіх у житті залежить не стільки від таланту та можливостей, скільки від зосередженості та наполегливості. – C. W. Wendte

3.3 Про мову програмування Python

Python — одна з тих рідкісних мов, які можуть бути водночас *простими* і *потужними*. Ви будете приємно здивовані, побачивши, як легко зосередитися на вирішенні поставленого завдання, а не на синтаксисі та структурі мови, на якій ви програмуєте.

Офіційно мову Python представляють так:

> Python — це потужна мова програмування, яка проста у вивченні. Вона має ефективні високорівневі структури даних та простий, але ефективний підхід до об'єктно-орієнтованого програмування. Елегантний синтаксис і динамічна типізація мови Python разом із інтерпретованою природою роблять її ідеальною мовою для створення сценаріїв і швидкої розробки додатків у багатьох сферах на більшості платформ.

Більш детально я розповім про більшість із цих особливостей у наступному розділі.

3.3.1 Історія, що стоїть за назвою

Гвідо ван Россум, автор мови програмування Python, назвав свою розробку на честь телешоу на BBC під назвою "Літаючий Цирк Монті Пайтона"(англ."Monty Python's Flying Circus"). А зовсім не тому,що він любить змій, які вбивають тварин для їжі шляхом звивання їхніх довгих тіл навколо них і розчавлювання їх.

3.3.2 Особливості Python

Проста мова програмування

Python — проста та мінімалістична мова. Читання якісного коду на мові Python виглядає майже як читання англійською, хоча англійська дуже сувора! Така «псевдокодова» природа мови Python є однією з її найбільших переваг. Це дозволяє зосередитися на вирішенні завдання, а не на самій мові.

Легкість у вивченні

Як ви побачите,з мови Python дуже легко почати програмувати. Python має надзвичайно простий синтаксис, як уже згадувалося.

Вільне і відкрите програмне забезпечення

Python є прикладом *ВВПЗ* (Вільне та Відкрите Програмне Забезпечення), (англ."*FLOSS*" (Free/Libre and Open Source Software)).Простіше кажучи, ви можете вільно поширювати копії цього програмного забезпечення, читати його вихідний код, вносити в нього зміни та використовувати його частини в нових безкоштовних програмах. ВВПЗ базується на концепції спільноти, яка ділиться знаннями. Це одна

з причин, чому мова Python така гарна - вона була створена і постійно вдосконалюється спільнотою, яка прагне зробити мову ще кращою.

Мова високого рівня

Коли ви пишете програми на мові Python, вам ніколи не доведеться турбуватися про деталі низького рівня, такі, як наприклад, керування пам'ятю, яку використовує ваша програма, тощо.

Python — портативна мова програмування

Через свою відкриту природу, мова Python була портована (тобто змінена, щоб змусити її працювати) на багатьох платформах. Усі ваші програми на мові Python можуть працювати на будь-якій із цих платформ, не вимагаючи жодних змін, якщо ви будете достатньо обережні, щоб уникнути будь-яких системно-залежних функцій. Ви можете використовувати мову Python на GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE і PocketPC!

Ви навіть можете використовувати таку платформу, як Kivy, щоб створювати ігри для свого комп'ютера *і* для iPhone, iPad і Android.

Інтерпретована мова

Це вимагає трохи пояснень.

Програма, написана компільованою мовою програмування, як наприклад, C або C+ +, перетворюється з вихідного коду(тобто C або C++) на мову, зрозумілу компьютеру (бінарний код, тобто нулі та одиниці), використовуючи компілятор з різними опціями. Коли ви запускаєте програму, компонувальник/завантажувач (англ."linker/loader software") копіює програму з жорсткого диска в оперативну пам'ять і починає її виконання.

Мова Python, напроти, не потребує компіляції у бінарний код. Ви просто *запускаєте* програму безпосередньо з вихідного коду(англ."source code"). Автоматично мова Python перетворює вихідний код у деяку проміжну форму, звану байт-кодами, а потім перекладає байт-коди на рідну мову вашого комп'ютера (бінарний код), і запускає. Все це, насправді, значно полегшує використання мови Python, оскільки вам не потрібно турбуватися про компіляцію програми, переконуватися, що відповідні бібліотеки пов'язані та завантажені тощо.Це також робить ваші програми на Python набагато більш портативними, оскільки ви можете просто скопіювати свою програму,написану мовою Python на інший комп'ютер, і вона просто запрацює!

Об'єктно-орієнтоване програмування

Мова Python підтримує процедурно-орієнтоване програмування, а також об'єктно-орієнтоване програмування (ООП). У *процедурно-орієнтованих* мовах програма побудована навколо процедур або функцій, які є не що інше, як багаторазові частини програм. У *об'єктно-орієнтованих* мовах програма побудована навколо об'єктів, які поєднують дані та функціонал. Python має дуже простий, але потужний спосіб створення ООП, особливо в порівнянні з такими великими мовами, як C++ або Java.

Розширювана мова

Якщо вам потрібно,щоб деякий критичний фрагмент коду працював дуже швидко, або ви хочете, щоб якийсь фрагмент алгоритму не був відкритим для інших, ви можете програмувати цю частину вашої програми на мові C або C++, а потім викликати її з ваших програм на мові Python.

Можливість вбудовування

Ви можете вбудовувати Python у свої програми C/C++, щоб надати користувачам "*scripting* capabilities" (можливості написати маленький фрагмент коду на мові Python всередені вашої великої програми(наприклад, C/C++,Java,тощо)).

Обширні бібліотеки

Стандартна бібліотека мови Python дійсно величезна. Вона може допомогти з вирішенням самих різноманітних завдань, пов'язаних з використанням регулярних виразів, генеруваням документації, перевіркою блоків коду, розпаралелювання процесів, базами даних, веб-браузерами, CGI, FTP, електронною поштою, XML, XML-RPC, HTML, файлами WAV, криптографією, GUI (графічними інтерфейсами користувача), та іншими системно-залежними речами. Пам'ятайте, що все це завжди доступно, де б не було встановлено Python. Це називається філософією Python *Все в комплекті* ("*Batteries Included*").

Крім стандартної бібліотеки, існують інші високоякісні бібліотеки, які ви можете знайти в каталозі пакетів Python.

Резюме

Мова Python справді захоплююча та потужна. Вона має правильне поєднання продуктивність і можливості, які роблять написання програм на мові Python одночасно цікавим і легким.

3.3.3 Різниця між версією Python 3 та 2

Ви можете проігнорувати цей розділ, якщо вас не цікавить різниця між версіями «Python 2» і «Python 3». Але майте на увазі, яку версію ви використовуєте. Ця книга написана для версії Python 3.

Пам'ятайте, що як тільки ви правильно зрозумієте та навчитеся використовувати одну версію, ви зможете легко дізнатися про відмінності та використовувати іншу. Найважче – навчитися програмуванню та зрозуміти основи самої мови Python.Це наша мета в цій книзі, і коли ви досягнете цієї мети, ви зможете легко використовувати Python 2 або Python 3 залежно від вашої ситуації.

Докладніше про відмінності між Python 2 і Python 3 дивиться:

- Майбутнє Python 2 (англ."The future of Python 2")
- Перенесення коду Python 2 на Python 3 (англ."Porting Python 2 Code to Python 3")
- Написання коду, який працює на Python2 і 3(англ."Writing code that runs under both Python 2 and 3")
- Підтримка Python 3: докладний посібник(англ."Supporting Python 3: An in-depth guide")

3.3.4 Що кажуть програмісти

Можливо, вам буде цікаво почитати, що такі великі хакери, як Ерік С. Реймонд (ESR), мають сказати про Python:

- *Ерік Стівен Реймонд* є автором книги «Собор і базар», а також людиною, яка ввела термін *відкритого програмного забезпечення*. Він каже, що Python став його улюбленою мовою програмування. Ця стаття стала справжнім натхненням моїх перших робіт з Python.
- *Брюс Екель* є автором знаменитих книг «Мислення на Java» та «Мислення на C++». Він каже, що жодна мова не зробила його більш продуктивним, ніж мова Python. Він вважає, що Python, мабуть, єдина мова, яка спрямована на полегшення роботи програміста. Прочитайте повне інтерв'ю, щоб дізнатися більше.
- *Пітер Норвіг* — відомий автор Lisp і директор із якості пошуку в Google (дякую Гвідо ван Россуму за те, що звернув увагу на це). Він каже, що написання на Python — це все одно, що написання в

псевдокоді. Він каже, що Python завжди був невід'ємною частиною Google. Ви можете перевірити це твердження, переглянувши сторінку Вакансії в Google, на якій знання Python є обов'язковими для інженерів програмного забезпечення.

3.4 Інсталяція

Коли ми згадуємо "Python 3" у цій книзі, ми маємо на увазі будь-яку версію Python, що дорівнює або перевищує версію 3.0.

Перейдіть на офіційний веб-сайт Python і натисніть посилання "Завантажити" (англ."Download"), щоб знайти список усіх версій Python для вашої операційної системи: https://www.python.org/downloads

3.4.1 Інсталяція на Windows

Відвідайте сайт https://www.python.org/downloads/windows/ і завантажте останню версію Python для Windows. На момент написання цієї статті це був Python 3.12. Встановлення відбувається так само, як і будь-яке інше програмне забезпечення для Windows.

Зауважте, що якщо у вас версія старіша, ніж версія Windows Vista, вам слід завантажити лише Python 3.4, оскільки для пізніших версій потрібні новіші версії Windows.

УВАГА: переконайтеся, що ви позначили опцію «Додати Python до PATH».

3.4.2 Інсталяція на Mac OS X

Відвідайте сайт https://www.python.org/downloads/macos/ і завантажте останню версію Python для MacOS.

Для користувачів Mac OS X використовуйте Homebrew: `brew install python3`.

Щоб перевірити інсталяцію, відкрийте термінал, натиснувши клавіші [Command + Space] (щоб відкрити пошук Spotlight), введіть terminal і натисніть клавішу [enter]. Тепер запустіть python3 і переконайтеся, що немає помилок.

3.4.3 Інсталяція на GNU/Linux

Для користувачів GNU/Linux скористайтеся диспетчером пакунків дистрибутива, щоб установити Python3, наприклад, на Debian і Ubuntu: `sudo apt-get update && sudo apt-get install python3`.

Домашня сторінка Python може мати новішу версію, ніж версія вашого дистрибутива Linux. Якщо ви не можете дочекатися, поки ваш дистрибутив Linux оновиться до найновішої версії Python, ви завжди можете спробувати вручну встановити найновішу версію Python безпосередньо з веб-сайту Python:

Відвідайте сайт https://www.python.org/downloads/source/ і завантажте останню версію Python для інсталяції вручну.

У будь-якому випадку,щоб перевірити чи Python інсталюван вірно, відкрийте термінал. Якщо це не спрацює, зверніться до документації вашого конкретного дистрибутива GNU/Linux. Тепер запустіть `python3` і переконайтеся, що немає помилок.

Ви можете побачити версію Python на екрані, виконавши:

```
python3 -v
```

Результат повинен бути приблизно:

```
python 3.12
```

Попередження

Результати можуть відрізнятися на вашому комп'ютері залежно від версії програмного забезпечення Python, встановленого на вашому комп'ютері.

3.4.4 Резюме

Відтепер ми будемо вважати, що у вашій системі встановлено Python.

Далі ми напишемо нашу першу програму на Python.

3.5 Перші кроки

Давайте подивимося, як створити традиційну програму «Привіт, Світ!» на Python. Це навчить вас писати, зберігати та запускати програми на Python.

Є два способи використання Python для запуску вашої програми - використання інтерактивного командного рядка інтерпретатора та використання файлу з текстом програми. Ми розглянемо обидва ці методи.

3.5.1 Використання командного рядка інтерпретатора

англійська: *Using The Interpreter Prompt*

Запустіть інтерпретатор Python, ввівши команду `python3`та натиснувши клавішу`[Enter]`.

Щойно ви запустите Python, ви побачите >>>, де ви можете почати вводити текст.Це називається *командний рядок інтерпретатора Python*(англ."*Python interpreter prompt*").

У командному рядку інтерпретатора Python введіть:

код python

```
print("Привіт, Світ!")
```

після чого натиснить клавішу `[Enter]`. Ви повинні побачити на екрані слова `Привіт, Світ!`.

Приклад:

```
$ python3
Python 3.6.0 (default, Jan 12 2017, 11:26:36)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Привіт, Світ!")
Привіт, Світ!
```

Це приклад того, що ви повинні бачити під час використання комп'ютера Mac OS X. Інформація о версії програмного забезпечення Python може відрізнятися в наступних рядках:

```
Python 3.6.0 (default, Jan 12 2017, 11:26:36)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
```

залежно від вашого комп'ютера, але частина командного рядка (тобто від >>> і далі) має бути однаковою незалежно від операційної системи:

```
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Привіт, Світ!")
Привіт, Світ!
```

Зауважте, що Python миттєво дає вам результат рядка! Те, що ви щойно ввели, є одиночним *оператором* Python. Ми використовуємо `print`, щоб (як це не дивно) надрукувати будь-яке значення, яке ви йому надаєте. Тут ми надаємо текст «Привіт, Світ!», і він негайно друкується на екрані.

Як вийти з командного рядка інтерпретатора

 англійська: *How to Quit the Interpreter Prompt*

Якщо ви використовуєте оболонку GNU/Linux або OS X, ви можете вийти з *Python interpreter prompt*, натиснувши `[ctrl + d]` або ввівши `exit()`(примітка: не забудьте включити дужки, `()`) ,а потім натиснути клавіші `[Enter]`.

Якщо ви використовуєте *Python interpreter prompt* Windows, натисніть `[ctrl + z]`, а потім клавішу `[Enter]`.

3.5.2 Вибір редактора

 англійська: *Choosing An Editor*

Оскільки ми не можемо набирати програму в *Python interpreter prompt* щоразу, коли нам потрібно щось запустити, нам доведеться зберігати програми у файлах, щоб потім мати можливість запускати їх скільки завгодно разів.

Перш ніж приступити до написання програм на Python у файлах, нам потрібний редактор для роботи із файлами програм. Вибір редактора дуже важливий. Підходити до вибору редактора треба так само, як і до вибору особистого автомобіля. Хороший редактор допоможе вам легко писати програми на Python, роблячи вашу подорож більш комфортною, а також дозволяючи швидше та безпечніше досягти вашої мети.

Одна з основних вимог є *підсвічування синтаксису*, коли різні елементи програми на Python забарвлені, щоб ви могли легко *бачити* вашу програму та хід виконання.

Якщо ви не знаєте, з чого почати, я б порекомендував використовувати програмне забезпечення PyCharm Educational Edition, яке доступне для Windows, Mac OS X і GNU/Linux. Подробиці в наступному розділі.

Якщо ви використовуєте Windows, *не використовуйте Блокнот* - це поганий вибір, оскільки в нього не має функції підсвічування синтаксису,а також він не дозволяє вставляти відступи, що дуже важливо у нашому випадку, як ми побачимо пізніше. Хороші редактори зроблять це автоматично.

Якщо ви досвідчений програміст, ви повинні вже використовувати Vim або Emacs. Зайве говорити, що це два найпотужніші редактори, і ви отримаєте користь від їх використання для написання своїх програм на Python. Я особисто використовую обидва для більшості своїх програм і навіть написав цілу книгу про Vim.

Якщо ви готові витратити час на вивчення Vim або Emacs, я настійно рекомендую вам навчитися використовувати будь-який з них, оскільки це буде дуже корисно для вас у довгостроковій перспективі. Однак, як я вже згадував раніше, початківці можуть почати з PyCharm і зосередити навчання на Python, а не на редакторі на даний момент.

Повторюю, будь ласка, виберіть відповідний редактор - це може зробити написання програм Python веселішим і легшим.

Якщо ви зацікавлені в детальному обговоренні цієї теми, перегляньте Пошук ідеального редактора коду Python.

3.5.3 PyCharm

PyCharm (Community Edition) це безкоштовний редактор, спеціально розроблений для написання програм на Python. Щоб завантажити PyCharm, дотримуйтесь інструкцій на

Зверніть увагу, що PyCharm також пропонує професійну версію редактора з більшими можливостями, однак ця версія не є безкоштовною.

3.5.4 Visual Studio code

Visual Studio Code — безкоштовний редактор коду, який можна використовувати для написання програм на Python.

1.Завантажте та встановіть Visual Studio Code з https://code.visualstudio.com/

2.Коли ви пишете свою першу програму на Python з Visual Studio code, як наприклад, 'print("Привіт,Світ")' і запускаєте її, Visual Studio Code автоматично завантажить і встановить розширення для мови Python. Як тільки цей плагін буде встановлено, ви готові.

Дивіться більше інформації про те, як використовувати код Visual Studio з python тут:

- https://code.visualstudio.com/docs/languages/python

3.5.5 Vim

Vim є відомим безкоштовним текстовим редактором, який можна використовувати для написання програм на Python.

1.Завантажте та встановіть Vim з офіційної домашньої сторінки: http://www.vim.org

2.Перегляньте статті про те, як адаптувати Vim для використання з програмами Python у вашій операційній системі, наприклад:

- https://realpython.com/vim-and-python-a-match-made-in-heaven/
- https://rapphil.github.io/vim-python-ide/

3.5.6 Emacs

Emacs є дуже відомим безкоштовним текстовим редактором, який можна використовувати майже для всього, включно з написанням програм на Python.

1. Завантажте та встановіть Emacs http://www.gnu.org/software/emacs/
2. Перегляньте статті про те, як адаптувати Emacs для написання програм на Python у вашій операційній системі, наприклад:

- https://realpython.com/emacs-the-best-python-editor/
- https://www.emacswiki.org/emacs/PythonProgrammingInEmacs

3.5.7 Інші редактори

 англійська: *Other editors*

Існує багато інших редакторів для написання коду на Python, більшість із яких є безкоштовним програмним забезпеченням. Кожен із них має свої особливі переваги та недоліки. Виберіть на свій смак, що найкраще підходить саме для вас.

Список редакторів Pytho можна знайти тут:

- https://wiki.python.org/moin/PythonEditors

3.5.8 Без використання редактора - робота безпосередньо з вихідними файлами

англійська: *Using no editor - working directly with source files*

Вам не потрібен спеціальний редактор для написання програм на Python, ви можете використовувати будь-який редактор, якщо збережете свою програму на Python як текстовий файл із правильним розширенням файлу ".py".

Наприклад, напишіть цю програму та збережіть її як "привіт.py":

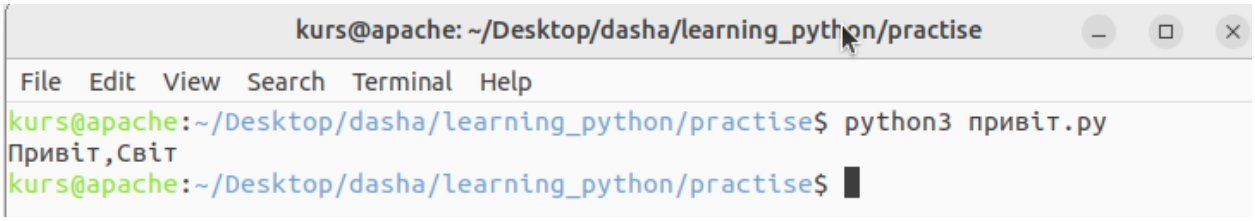
```
print("Привіт,Світ")
```

Більшість редакторів Python пропонують кнопку «Пуск» або «Виконати», на якій можна натиснути, щоб запустити програму. Однак, якщо у вас є відкритий термінал у тій же папці, де збережена і ваша програма `привіт.py`, і вона збережена правильно, ви можете просто ввести в терміналі:

```
python3 привіт.py
```

щоб запустити програму.

Це припускає, що ви трохи знаєте про роботу термінала і що ви зберегли програму привіт.py у тому самому місці, де ви відкривали термінал.



3.6 Основи

англійська: *Basics*

Просто надрукувати `Привіт, Світ!` недостатньо, чи не так? Хочеться більшого – ви хочете ввести щось у програму, обробити та отримати щось на виході. Ми можемо досягти цього у Python за допомогою констант і змінних, і ми також вивчимо деякі інші концепції в цьому розділі.

3.6.1 Коментарі

Коментарі – це будь-який текст,що пишеться після символу `#` і в основному корисний як примітка для читача програми.

Наприклад:

```
print('Привіт, Світ!') # Зауважте, що print - це функція
```

або:

```
# Зауважте, що print - це функція
print('Привіт, Світ!')
```

Використовуйте якомога більше корисних коментарів у своїй програмі, щоб пояснювати:

- припущення;
- важливі рішення;
- важливі деталі;
- проблеми, які ви намагаєтеся вирішити;
- зафіксувати труднощі, які ви намагаєтеся подолати у своїй програмі тощо.

Код програми підкаже вам, ЯК, а коментарі мають розповісти вам, ЧОМУ.

Це корисно для читачів вашої програми, щоб вони могли легко зрозуміти, що робить програма. Пам'ятайте, що такою людиною можете виявитися ви самі через півроку!

3.6.2 Літеральні константи

 англійська: *Literal Constants*

Прикладом літеральної константи є числа, такі як `5`, `1,23`, або рядок, як-от `Це рядок` або `It's a string!`. Вони називаються літеральними, тому що вони *буквальні* - ви використовуєте їхнє значення буквально. Число `2` завжди представляє саме себе і нічого іншого - це *константа*, оскільки її значення не можна змінити. Отже, усі вони називаються літеральними константами.

3.6.3 Числа

 англійська: *Numbers*

Числа в основному бувають двох типів - цілі (англ."integers") і з рухомою комою (англ."floating point numbers" (або «floats»скорочено)).

Прикладом цілого числа може служити "2", яке є цілим числом.

Прикладами чисел з рухомою комою можуть бути `3.23` і `52.3E-4`. Позначення E показує ступені числа 10. У цьому випадку `52,3E-4` означає 52,3 * 10-4.

Примітка від перекладача: *РІЗНИЦЯ між поняттями «Рухома кома» та «Рухома крапка»*. Оскільки в деяких, переважно в Америці та англомовних країнах, при запису чисел ціла частина відділяється від дробової крапкою (Decimal Point), то в термінології цих країн фігурує назва «рухома крапка» (англ. floating point). В Європейських країнах,як і в Україні, ціла частина числа від дробової традиційно відділяється комою (Decimal Comma), то для позначення того ж поняття історично використовується термін «рухома кома» (англ. floating comma), проте в літературі та технічній документації можна зустріти обидва варіанти. У програмуванні Python використовується при позначені числа з рухомою комою *крапка*.

Примітка для досвідчених програмістів

На відміну від інших мов програмування (наприклад, C, Java тощо), число *int* на мові Python не має верхньої межі.

3.6.4 Рядки

 англійська: *Strings*

Рядок — це послідовність символів. Найчастіше рядки - це просто набір слів.

Ви будете використовувати рядки майже в кожній програмі на мові Python, яку пишете, тому варто уважно ознайомитись із цим розділом.

Одинарні лапки

 англійська: *Single Quote*

Ви можете вказати рядки, використовуючи одинарні лапки, наприклад `'Фраза в лапках'`.

Усі пробіли та знаки табуляції, зберігаються як є.

Подвійні лапки

**  англійська: *Double Quotes*

Рядки у подвійних лапках працюють так само, як рядки в одинарних лапках. Приклад: `"Як тебе звати?"`.

Потрійні лапки

 англійська: *Triple Quotes*

Ви можете вказати багаторядкові рядки, використовуючи потрійні лапки - (`"""` або `'''`). Ви можете вільно використовувати одинарні та подвійні лапки в потрійних лапках. Наприклад:

```
'''Це багаторядковий рядок.Це перший рядок.
Це другий рядок.
"Як тебе звати?", - запитав я.
Він сказав "Бонд, Джеймс Бонд".
'''
```

Рядки незмінні

 англійська: *Strings Are Immutable*

Це означає, що коли ви створили рядок, ви не можете його змінити. На перший погляд це може показатися недоліком, насправді це не так. Пізніше,на прикладі різних програм,ми побачимо, чому це не є обмеженням.

Примітка для програмістів C/C++

У Python немає окремого типу даних `char`. У цьому немає справжньої потреби, і я впевнений, ви точно не будете за ним сумувати.

Примітка для програмістів Perl/PHP

Пам'ятайте, що рядки в одинарних лапках і рядки в подвійних лапках однакові – вони нічим не відрізняються.

Метод форматування

 англійська: *The format method*

Іноді нам може знадобитися побудувати рядки на основі будь-яких даних. Ось де корисний метод `format()`.

Завантажуйте наступні рядки як файл `str_format.py`.

Примітка від перекладача:

1. Пунктуація, як апостроф, двокрапка тощо, не можно використовувати у змінних (variable), тільки підкреслення "_". Тому у прикладі замість грамотично вірного написання слова "ім'я" використано -"ім_я";
2. `Swaroop` - Swaroop Chitlur - автор книги ("Bite of Python"– "Укус Пітона").

код python

```
вік = 20
ім_я = 'Swaroop'

print('{0} було {1} років, коли він написав цю книгу'.format(ім_я, вік))
print('Чому {0} грає з цим Python?'.format(ім_я))
```

Висновок:

```
$ python str_format.py
Swaroop було 20 років, коли він написав цю книгу.
Чому Swaroop грає з цим Python?
```

Як це працює

Рядок може використовувати певні позначення, і згодом можна викликати метод `format`, щоб замінити ці позначення відповідними аргументами в метод `format`.

Зверніть увагу на перший випадок, коли ми використовуємо `{0}`, і це відповідає змінній `ім_я`, яка є першим аргументом методу format. Подібним чином друге позначення — `{1}`, що відповідає змінній `вік`,є другим аргументом методу format. Зверніть увагу, що Python починає відлік з 0, що означає, що перша позиція знаходиться в індексі 0, друга позиція в індексі 1 і так далі.

Зверніть увагу, що ми могли б досягти того ж, використовуючи об'єднання рядків:

```
ім_я + ' було ' + str(вік) + ' років '
```

але це набагато гірше та більше схильне до помилок. По-друге, перетворення в рядок буде виконано автоматично за допомогою методу `format` на відміну від явного перетворення в нашому прикладі. По-третє, використовуючи метод `format`, ми можемо змінити повідомлення, не торкаючись використовуваних змінних, і навпаки.

Також зауважте, що числа є необов'язковими, тому ви також можете записати так:

```
вік = 20
ім_я = 'Swaroop'

print('{} було {} років, коли він написав цю книгу'.format(ім_я, вік))
print('Чому {} грає з цим  Python?'.format(ім_я))
```

та отримати такий самий результат, як і раніше.

Ми також можемо назвати параметри (де вік, ім_я є параметрами):

```
вік = 20
ім_я = 'Swaroop'

print('{ім_я} було {вік} років, коли він написав цю книгу'.format(ім_я=ім_я, вік=вік))
print('Чому {ім_я} грає з цим Python?'.format(ім_я=ім_я))
```

та отримати такий самий результат, як і раніше.

Python 3.6 представив короткий спосіб створення іменованих параметрів, який називається «f-strings»:

```
вік = 20
ім_я = 'Swaroop'

print(f'{ім_я} було {вік} років, коли він написав цю книгу')
# зверніть увагу на 'f' перед рядком
print(f'Чому {ім_я} грає з цим Python?')
# зверніть увагу на 'f' перед рядком
```

та отримати такий самий результат, як і раніше.

Те, що Python робить у методі `format`, полягає в тому, що він поміщає значення кожного аргументу(більш детальніше пояснення про аргументи та параметри пізніше) в зазначене місце. Можуть бути більш детальні позначення, такі як (для отримання додаткової інформації дивіться документацію https://docs.python.org/3/library/string.html#formatspec):

код python

```
# десяткове число (.) з точністю в 3 знаки для числа з рухомою комою '0.333'
print('{0:.3f}'.format(1.0/3))
# заповнити підкресленнями (_) з центруванням тексту
# (^) по ширині 11: '___привіт___'
print('{0:_^11}'.format('привіт'))
# на основі ключових слів 'Swaroop написав «Укус Пітона»'
print('{ім_я} написав {книга}'.format(
      ім_я='Swaroop', книга='Укус Пітона'))
```

Висновок:

```
0.333
___привіт___
Swaroop написав Укус Пітона
```

Оскільки ми обговорюємо форматування, зауважте, що `print` завжди закінчується невидимим символом «нового рядка» (`\n`), тому повторні виклики `print` будуть виводитися з нового рядка. Фактично

print() виводить вказане значення, а після цього переводить курсор на наступний рядок. Щоб запобігти друку символу нового рядка, ви можете вказати, що він повинен закінчуватися без пробілів:

код python

```
print('a', end='')
print('b', end='')
```

Висновок:

```
ab
```

Або ви можете завершити пробілом:

код python

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

Висновок:

```
a b c
```

Екранована послідовность,символ ,зворотній слеш

 англійська: *Escape Sequences*

Припустімо, ви хочете мати рядок, який містить одинарну лапку ('), як ви вкажете цей рядок? Наприклад, рядок є"Як Ваше ім'я?". Ви не можете вказати 'Як Ваше ім'я?' тому що Python заплутається щодо того, де починається і де закінчується рядок. Отже, вам доведеться вказати, що ця одинарна лапка не вказує на кінець рядка. Це можна зробити за допомогою так званої *екранована послідовность* (англ.*"escape sequence"*). Ви вказуєте одинарну лапку як \': зверніть увагу на зворотній слеш. Тепер ви можете вказати рядок як 'Як Ваше ім\'я?'.

Іншим способом вказати цей конкретний рядок буде "Як Ваше ім'я?", тобто використовуючи подвійні лапки.Подібним чином, ви повинні використовувати escape sequence для позначення подвійних лапок всередині рядку з подвійними лапками.Приклад від перекладача:

```
# цей варіант не працює, оскільки подвійні лапки знаходяться всередині рядка подвійних лапок:
print("Мова програмування "Python" займає у моєму серці важливе місце.")

# цей варіант спрацює, оскільки подвійні лапки навколо слова Python екрановані:
print("Мова програмування \"Python\" займає у моєму серці важливе місце.")
```

Ви також можете вказати подвійний зворотній слеш за допомогою escape sequence \\. Приклад від перекладача:

код python

```
#  \n - це екранована послідовність для нового рядка
#  \f - form feed - йде на один рядок нижче
```

```
# це речення буде виводитися у три рядки,але з нюансами:
print("мої фотографії збережено на  диску c:\new_pictures\foto.jpg")
```

Висновок:

```
мої фотографії збережено на  диску c:
ew_pictures
           oto.jpg
```

щоб надрукувати речення правильно, треба використати подвійний зворотній слеш:

код python

```
print("мої фотографії збережено на  диску c:\\new_picture\\foto.jpg")
```

Висновок:

```
мої фотографії збережено на  диску c:\new_picture\foto.jpg
```

Що, якби ви хотіли вказати дворядковий рядок? Одним із способів є використання рядка в потрійних лапках, як показано *раніше* або ви можете використати escape sequence для символу нового рядка - `\n`, щоб вказати початок нового рядка. Наприклад:

код python

```
'Це перший рядок\nЦе другий рядок'
```

Висновок:

```
Це перший рядок
Це другий рядок
```

Приклад від перекладача

код python

```
# з потрійними лапками
вірш1 = """я можу бути програмістом
і мова Python потрібна звісно."""
print(вірш1)
# з екранованої послідовностю \n
вірш2 = "я можу бути програмістом\ні мова Python потрібна звісно."
print(вірш2)
```

Висновок:

```
я можу бути програмістом
і мова Python потрібна звісно.
```

Ще одна корисна *escape sequence*, про яку слід знати, це послідовність символів:`\t`(розглядається як один символ табуляції). Існує ще багато escape sequences, але я згадав тут лише найкорисніші.

Варто зауважити, що зворотній слеш в кінці рядка означає, що рядок продовжується в наступному

рядку, але новий рядок не додається. Наприклад:

```
"Це перше речення. \
Це друге речення."
```

еквівалентно

```
"Це перше речення. Це друге речення."
```

3.6.5 Необроблений рядок

 англійська: *Raw String*

Якщо вам потрібно вказати деякі рядки, де не обробляються спеціальні символи, наприклад escape sequences, тоді вам потрібно вказати *raw string*, додавши до рядка префікс `r` або `R`.Необроблений рядок Python розглядає символ зворотної похилої риски (\ ) як буквальний символ.Наприклад:

```
r"Нові рядки позначаються \n"
```

Примітка для користувачів регулярного виразу

 англійська: *Regular Expression User*

Завжди використовуйте raw strings під час роботи з регулярними виразами. Інакше може знадобитися чимало зворотних слешей. Наприклад, абревіатуру з регулярними виразами в Python можна знайти за допомогою `'\\1'` or `r'\1'` (на відміну роботи з Python, пошук абревіатури з регулярними виразами в інших системах виконується символом `\1`)

3.6.6 Змінні

 англійська: *Variable*

Використання лише літеральних констант незабаром може набриднути — нам потрібен якийсь спосіб зберігання будь-якої інформації та маніпулювання нею. Ось тут і з'являються *змінні*. Змінні — це саме те, що випливає з назви — їхнє значення може змінюватися, тобто ви можете зберігати будь-що за допомогою змінної. Змінні — це лише частини пам'яті комп'ютера, де зберігається деяка інформація. На відміну від літеральних констант, вам потрібен певний метод доступу до цих змінних і, отже, ви даєте їм імена.

3.6.7 Іменування ідентифікатора

 англійська: *Identifier Naming*

Змінні є прикладами ідентифікаторів. *Ідентифікатори* — це імена, надані для ідентифікації *чогось*. Існують деякі правила, яких слід дотримуватися для ідентифікаторів імен:

- Першим символом ідентифікатора має бути літера алфавіту (символ ASCII в верхньому(великі літери) або нижньому(малі літери) регістрі, або символ Unicode), а також знак підкреслення(_).
- Решта назви ідентифікатора може складатися з літер (символ ASCII в верхньому або нижньому регістрі, або символ Unicode),знак підкреслення (_) або цифр (0-9).
- Імена ідентифікаторів чутливі до регістру. Наприклад, `myname` і `myName` *не* те саме. Зверніть увагу на `n` у нижньому регістрі у першому випадку та на `N` у верхньому регістрі у другому випадку.

- Прикладами *допустимих* імен ідентифікаторів є `i`, `name_2_3`. Прикладами *неприпустимих* імен ідентифікаторів є `2things`, `this is spaced out`, `my-name` і `>a1b2_c3`.

3.6.8 Типи даних

 англійська: *Data Types*

Змінні можуть містити значення різних типів, які називаються *типами даних.*Основними типами є числа та рядки, які ми вже обговорювали. У наступних розділах ми побачимо, як створювати власні типи за допомогою *класів*.

3.6.9 Об'єкт

 англійська: *Object*

Пам'ятайте, Python називає все, що використовується в програмі, *об'єктом*. Це мається на увазі в загальному значенні. Замість того, щоб говорити "*щось*"", ми говоримо "*об'єкт*".

Примітка для користувачів об'єктно-орієнтованого програмування

Python сильно об'єктно-орієнтований у тому сенсі, що все є об'єктом, включаючи числа, рядки та функції.

Тепер ми побачимо, як використовувати змінні разом із літеральними константами. Збережіть наступний приклад і запустіть програму.

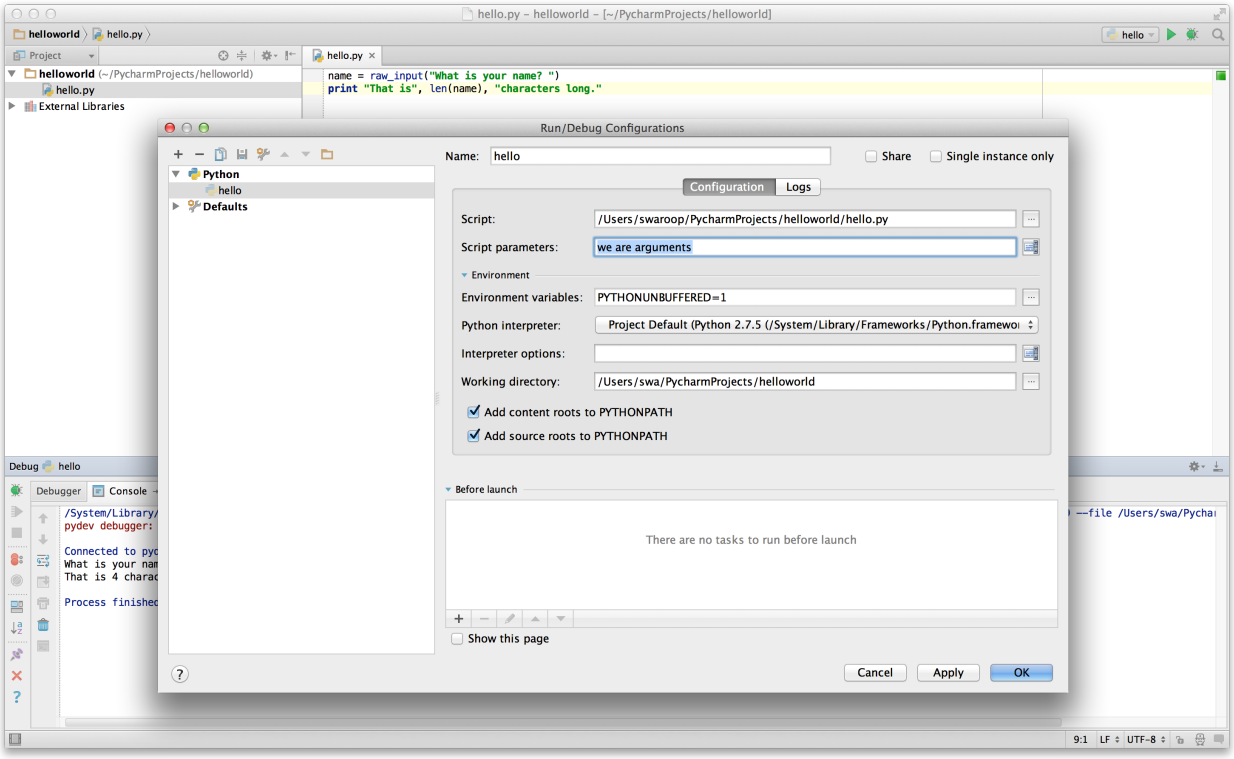
3.6.10 Як писати програми на Python

Відтепер стандартна процедура збереження та запуску програми Python така:

Для PyCharm

1. Відкрийте *PyCharm*.
2. Створіть новий файл з назвою файлу.
3. Введіть програмний код, наведений у прикладі.
4. Клацніть правою кнопкою миші та запустіть поточний файл.

ПРИМІТКА. Щоразу, коли вам потрібно надати *command line arguments ("аргументи командного рядка")*, натисніть кнопку`Run -> Edit Configurations`, введіть аргументи в розділ `Script parameters:` і натисніть кнопку `OK`:



Для інших редакторів

1. Відкрийте вибраний редактор.
2. Введіть програмний код, наведений у прикладі.
3. Збережіть його як файл з назвою файлу.
4. Запустіть інтерпретатор командою `python program.py`, щоб запустити програму.

Приклад: використання змінних і літеральних констант

Введіть і запустіть таку програму:

код python

```
# Ім'я файлу : var.py
i = 5
print(i)
i = i + 1
print(i)

s = '''Це багаторядковий рядок.
Це другий рядок.'''
print(s)
```

Висновок:

```
5
6
Це багаторядковий рядок.
Це другий рядок.
```

Як це працює

Ось як працює ця програма. Спочатку ми привласнюємо значення константи `5` змінній `i` за допомогою оператора присвоювання (=).Цей рядок називається statement - так би мовити "інструкція роботи для комп'ютера". Вираз i=5 має три частини:

1. "i" - назва змінної;
2. "=" - оператор присвоювання;
3. "5" - літеральна константа.

Ці частини не працюють поодинці, але разом вони створюють інструкцію роботи для комп'ютера. Цей рядок називається statement,оскільки в ньому зазначено, що потрібно щось зробити, і в цьому випадку ми з'єднуємо назву змінної `i` зі значенням `5`. Далі ми друкуємо значення `i` за допомогою функції `print`, яка, як не дивно, просто виводить значення змінної на екран.

Потім ми додаємо «1» до значення, збереженого в «i», і зберігаємо його там. Потім ми друкуємо його i, як очікується, отримуємо значення "6".

Подібним чином ми присвоюємо рядковий літерал змінній `s`, а потім друкуємо його.

Примітка для програмістів статичної мови

Змінні використовуються простим наданням їм значень. Ніякого попередньоного оголошення або визначення типу даних не потрібно/застосовується.

3.6.11 Логічні та фізичні рядки

 англійська: *Logical And Physical Line*

Фізична лінія - це те, що ви *бачите*, коли пишете програму. Логічний рядок — це те, що *Python сприймає* як "a single statement" (одну інструкцію). Python неявно припускає, що кожен *фізичний рядок* відповідає *логічному рядку*.

Прикладом логічного рядка є statement `print('Привіт, Світ!')`- якщо воно в одному рядку(як ви бачите це в редакторі), то цей рядок також відповідає фізичному рядку.

Неявно Python заохочує використання одного statement на рядок, що робить код більш читабельним.

Якщо ви хочете вказати більше ніж один логічний рядок в одному фізичному рядку, ви повинні явно вказати це за допомогою крапки з комою (`;`), яка вказує на кінець логічного рядка/statement. Наприклад:

```
i = 5
print(i)
```

те саме, що

```
i = 5;
print(i);
```

і те саме може бути записано у вигляді

```
i = 5; print(i);
```

або

```
i = 5; print(i)
```

Однак я *настійно рекомендую* вам дотримуватися *написання максимум одного логічного рядка в кожному фізичному рядку*. Ідея полягає в тому, що ви можете обійтися без крапки з комою. Фактично, я *ніколи* не використовував і навіть не бачив крапку з комою в програмі Python.

Існує одна ситуація, коли ця концепція дійсно корисна: якщо у вас є довгий рядок коду, ви можете розбити його на кілька фізичних рядків, використовуючи зворотній слеш. Це називається *явним з'єднанням рядків (англ."explicit line joining")*:

```
s = 'Це рядок. \
Цей рядок продовжується.'
print(s)
```

Висновок:

```
Це рядок. Цей рядок продовжується.
```

Аналогічно,

```
i = \
5
```

те саме, що

```
i = 5
```

Іноді існує неявне припущення, що вам не потрібно використовувати зворотній слеш. Це випадок, коли в логічному рядку є відкриваюча кругла, квадратна або фігурна дужка, але немає закриваючої . Це називається *неявним з'єднанням ліній* (англ."implicit line joining") . Ви можете побачити це в дії, коли ми пишемо програми за допомогою *list* у наступних розділах.

3.6.12 Відступи

 англійська: *Indentation*

Пробіли (англ."Whitespace") важливі в Python. Точніше *пробіли на початку рядка важливі*. Це називається *відступами*. Передні відступи (пробіли та табуляції) на початку логічного рядка використовуються для визначення рівня відступу логічного рядка, який, у свою чергу, використовується для групування statement.

Це означає, що statement, які йдуть разом, *повинні* мати однаковий відступ. Кожен такий набір statement називається *блоком*. У наступних розділах ми побачимо приклади важливості блоків.

Пам'ятайте, що неправильний відступ може спричинити помилки. Наприклад:

код python

```
i = 5
# Помилка нижче! Зверніть увагу на один пробіл на початку рядка
 print('Значення є', i)
print('Я повторюю, значення є', i)
```

Коли ви запускаєте це, ви отримуєте таку помилку:

```
  File "whitespace.py", line 3
    print('Значення є', i) # Помилка! Пробіл на початку рядка
    ^
IndentationError: unexpected indent
```

Зверніть увагу, що на початку другого рядка є один пробіл. Помилка, відображена у Python, повідомляє нам, що синтаксис програми невірний, тобто програму було написано неправильно.Для вас це означає те, що *ви не можете довільно починати нові блоки statement* (за винятком головного блоку за замовчуванням, який ви використовували протягом всієї програми,). Випадки, коли ви можете використовувати нові блоки, будуть детально описані в наступних розділах, наприклад *control flow*.

Як зробити відступ

Використовуйте чотири пробіли для відступу. Це офіційна рекомендація щодо мови Python. Хороші редактори автоматично зроблять це за вас. Переконайтеся, що ви використовуєте однакову кількість пробілів для відступів, інакше ваша програма не працюватиме або матиме неочікувану поведінку.

Примітка для програмістів статичної мови програмування

Python завжди використовуватиме відступи для блоків і ніколи не використовуватиме дужки. Запустіть `from __future__ import braces`, щоб дізнатися більше.

3.6.13 Резюме

Тепер, коли ми пройшли через багато дрібних деталей, ми можемо перейти до більш цікавих речей, таких як оператори потоку керування. Переконайтеся, що ви зрозуміли те, що ви прочитали в цьому розділі.

3.7 Основи

 англійська: *Basics*

Просто надрукувати `Привіт, Світ!` недостатньо, чи не так? Хочеться більшого – ви хочете ввести щось у програму, обробити та отримати щось на виході. Ми можемо досягти цього у Python за допомогою констант і змінних, і ми також вивчимо деякі інші концепції в цьому розділі.

3.7.1 Коментарі

Коментарі – це будь-який текст,що пишеться після символу `#` і в основному корисний як примітка для читача програми.

Наприклад:

```
print('Привіт, Світ!') # Зауважте, що print - це функція
```

або:

```
# Зауважте, що print - це функція
print('Привіт, Світ!')
```

Використовуйте якомога більше корисних коментарів у своїй програмі, щоб пояснювати:

- припущення;
- важливі рішення;
- важливі деталі;
- проблеми, які ви намагаєтеся вирішити;
- зафіксувати труднощі, які ви намагаєтеся подолати у своїй програмі тощо.

Код програми підкаже вам, ЯК, а коментарі мають розповісти вам, ЧОМУ.

Це корисно для читачів вашої програми, щоб вони могли легко зрозуміти, що робить програма. Пам'ятайте, що такою людиною можете виявитися ви самі через півроку!

3.7.2 Літеральні константи

 англійська: *Literal Constants*

Прикладом літеральної константи є числа, такі як `5`, `1,23`, або рядок, як-от `Це рядок` або `It's a string!`. Вони називаються літеральними, тому що вони *буквальні* - ви використовуєте їхнє значення буквально. Число `2` завжди представляє саме себе і нічого іншого - це *константа*, оскільки її значення не можна змінити. Отже, усі вони називаються літеральними константами.

3.7.3 Числа

 англійська: *Numbers*

Числа в основному бувають двох типів - цілі (англ."integers") і з рухомою комою (англ."floating point numbers" (або «floats»скорочено)).

Прикладом цілого числа може служити "2", яке є цілим числом.

Прикладами чисел з рухомою комою можуть бути `3.23` і `52.3E-4`. Позначення E показує ступені числа 10. У цьому випадку `52,3E-4` означає 52,3 * 10-4.

Примітка від перекладача: *РІЗНИЦЯ між поняттями «Рухома кома» та «Рухома крапка».* Оскільки в деяких, переважно в Америці та англомовних країнах, при запису чисел ціла частина відділяється від дробової крапкою (Decimal Point), то в термінології цих країн фігурує назва «рухома крапка» (англ. floating point). В Європейських країнах,як і в Україні, ціла частина числа від дробової традиційно відділяється комою (Decimal Comma), то для позначення того ж поняття історично використовується термін «рухома кома» (англ. floating comma), проте в літературі та технічній документації можна зустріти обидва варіанти. У програмуванні Python використовується при позначені числа з рухомою комою *крапка*.

Примітка для досвідчених програмістів

На відміну від інших мов програмування (наприклад, C, Java тощо), число *int* на мові Python не має верхньої межі.

3.7.4 Рядки

 англійська: *Strings*

Рядок — це послідовність символів. Найчастіше рядки - це просто набір слів.

Ви будете використовувати рядки майже в кожній програмі на мові Python, яку пишете, тому варто уважно ознайомитись із цим розділом.

Одинарні лапки

 англійська: *Single Quote*

Ви можете вказати рядки, використовуючи одинарні лапки, наприклад `'Фраза в лапках'`.

Усі пробіли та знаки табуляції, зберігаються як є.

Подвійні лапки

**  англійська: *Double Quotes*

Рядки у подвійних лапках працюють так само, як рядки в одинарних лапках. Приклад: `"Як тебе звати?"`.

Потрійні лапки

 англійська: *Triple Quotes*

Ви можете вказати багаторядкові рядки, використовуючи потрійні лапки - (`"""` або `'''`). Ви можете вільно використовувати одинарні та подвійні лапки в потрійних лапках. Наприклад:

```
'''Це багаторядковий рядок.Це перший рядок.
Це другий рядок.
"Як тебе звати?", - запитав я.
Він сказав "Бонд, Джеймс Бонд".
'''
```

Рядки незмінні

 англійська: *Strings Are Immutable*

Це означає, що коли ви створили рядок, ви не можете його змінити. На перший погляд це може показатися недоліком, насправді це не так. Пізніше,на прикладі різних програм,ми побачимо, чому це не є обмеженням.

Примітка для програмістів C/C++

У Python немає окремого типу даних `char`. У цьому немає справжньої потреби, і я впевнений, ви точно не будете за ним сумувати.

Примітка для програмістів Perl/PHP

Пам'ятайте, що рядки в одинарних лапках і рядки в подвійних лапках однакові – вони нічим не відрізняються.

Метод форматування

 англійська: *The format method*

Іноді нам може знадобитися побудувати рядки на основі будь-яких даних. Ось де корисний метод `format()`.

Завантажуйте наступні рядки як файл `str_format.py`.

Примітка від перекладача:

1. Пунктуація, як апостроф, двокрапка тощо, не можно використовувати у змінних (variable), тільки підкреслення "_". Тому у прикладі замість грамотично вірного написання слова "ім'я" використано -"ім_я";
2. `Swaroop` - Swaroop Chitlur - автор книги ("Bite of Python"– "Укус Пітона").

код python

```
вік = 20
ім_я = 'Swaroop'

print('{0} було {1} років, коли він написав цю книгу'.format(ім_я, вік))
print('Чому {0} грає з цим Python?'.format(ім_я))
```

Висновок:

```
$ python str_format.py
Swaroop було 20 років, коли він написав цю книгу.
Чому Swaroop грає з цим Python?
```

Як це працює

Рядок може використовувати певні позначення, і згодом можна викликати метод `format`, щоб замінити ці позначення відповідними аргументами в метод `format`.

Зверніть увагу на перший випадок, коли ми використовуємо `{0}`, і це відповідає змінній `ім_я`, яка є першим аргументом методу format. Подібним чином друге позначення — `{1}`, що відповідає змінній `вік`,є другим аргументом методу format. Зверніть увагу, що Python починає відлік з 0, що означає, що перша позиція знаходиться в індексі 0, друга позиція в індексі 1 і так далі.

Зверніть увагу, що ми могли б досягти того ж, використовуючи об'єднання рядків:

```
ім_я + ' було ' + str(вік) + ' років '
```

але це набагато гірше та більше схильне до помилок. По-друге, перетворення в рядок буде виконано автоматично за допомогою методу `format` на відміну від явного перетворення в нашому прикладі. По-третє, використовуючи метод `format`, ми можемо змінити повідомлення, не торкаючись використовуваних змінних, і навпаки.

Також зауважте, що числа є необов'язковими, тому ви також можете записати так:

```
вік = 20
ім_я = 'Swaroop'

print('{} було {} років, коли він написав цю книгу'.format(ім_я, вік))
print('Чому {} грає з цим  Python?'.format(ім_я))
```

та отримати такий самий результат, як і раніше.

Ми також можемо назвати параметри (де вік, ім_я є параметрами):

```
вік = 20
ім_я = 'Swaroop'

print('{ім_я} було {вік} років, коли він написав цю книгу'.format(ім_я=ім_я, вік=вік))
print('Чому {ім_я} грає з цим Python?'.format(ім_я=ім_я))
```

та отримати такий самий результат, як і раніше.

Python 3.6 представив короткий спосіб створення іменованих параметрів, який називається «f-strings»:

```
вік = 20
ім_я = 'Swaroop'

print(f'{ім_я} було {вік} років, коли він написав цю книгу')
# зверніть увагу на 'f' перед рядком
print(f'Чому {ім_я} грає з цим Python?')
# зверніть увагу на 'f' перед рядком
```

та отримати такий самий результат, як і раніше.

Те, що Python робить у методі `format`, полягає в тому, що він поміщає значення кожного аргументу(більш детальніше пояснення про аргументи та параметри пізніше) в зазначене місце. Можуть бути більш детальні позначення, такі як (для отримання додаткової інформації дивіться документацію https://docs.python.org/3/library/string.html#formatspec):

код python

```
# десяткове число (.) з точністю в 3 знаки для числа з рухомою комою '0.333'
print('{0:.3f}'.format(1.0/3))
# заповнити підкресленнями (_) з центруванням тексту
# (^) по ширині 11: '___привіт___'
print('{0:_^11}'.format('привіт'))
# на основі ключових слів 'Swaroop написав «Укус Пітона»'
print('{ім_я} написав {книга}'.format(
      ім_я='Swaroop', книга='Укус Пітона'))
```

Висновок:

```
0.333
___привіт___
Swaroop написав Укус Пітона
```

Оскільки ми обговорюємо форматування, зауважте, що `print` завжди закінчується невидимим символом «нового рядка» (`\n`), тому повторні виклики `print` будуть виводитися з нового рядка. Фактично

print() виводить вказане значення, а після цього переводить курсор на наступний рядок. Щоб запобігти друку символу нового рядка, ви можете вказати, що він повинен закінчуватися без пробілів:

код python

```
print('a', end='')
print('b', end='')
```

Висновок:

```
ab
```

Або ви можете завершити `пробілом`:

код python

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

Висновок:

```
a b c
```

Екранована послідовность,символ ,зворотній слеш

 англійська: *Escape Sequences*

Припустімо, ви хочете мати рядок, який містить одинарну лапку (`'`), як ви вкажете цей рядок? Наприклад, рядок є`"Як Ваше ім'я?"`. Ви не можете вказати `'Як Ваше ім'я?'` тому що Python заплутається щодо того, де починається і де закінчується рядок. Отже, вам доведеться вказати, що ця одинарна лапка не вказує на кінець рядка. Це можна зробити за допомогою так званої *екранована послідовность* (англ.*"escape sequence"*). Ви вказуєте одинарну лапку як `\'`: зверніть увагу на зворотній слеш. Тепер ви можете вказати рядок як `'Як Ваше ім\'я?'`.

Іншим способом вказати цей конкретний рядок буде `"Як Ваше ім'я?"`, тобто використовуючи подвійні лапки.Подібним чином, ви повинні використовувати escape sequence для позначення подвійних лапок всередині рядку з подвійними лапками.Приклад від перекладача:

```
# цей варіант не працює, оскільки подвійні лапки знаходяться всередині рядка подвійних лапок:
print("Мова програмування "Python" займає у моєму серці важливе місце.")

# цей варіант спрацює, оскільки подвійні лапки навколо слова Python екрановані:
print("Мова програмування \"Python\" займає у моєму серці важливе місце.")
```

Ви також можете вказати подвійний зворотній слеш за допомогою escape sequence `\\`. Приклад від перекладача:

код python

```
#  \n - це екранована послідовність для нового рядка
#  \f - form feed - йде на один рядок нижче
```

```
# це речення буде виводитися у три рядки,але з нюансами:
print("мої фотографії збережено на  диску c:\new_pictures\foto.jpg")
```

Висновок:

```
мої фотографії збережено на  диску c:
ew_pictures
           oto.jpg
```

щоб надрукувати речення правильно, треба використати подвійний зворотній слеш:

код python

```
print("мої фотографії збережено на  диску c:\\new_picture\\foto.jpg")
```

Висновок:

```
мої фотографії збережено на  диску c:\new_picture\foto.jpg
```

Що, якби ви хотіли вказати дворядковий рядок? Одним із способів є використання рядка в потрійних лапках, як показано *раніше* або ви можете використати escape sequence для символу нового рядка - `\n`, щоб вказати початок нового рядка. Наприклад:

код python

```
'Це перший рядок\nЦе другий рядок'
```

Висновок:

```
Це перший рядок
Це другий рядок
```

Приклад від перекладача

код python

```
# з потрійними лапками
вірш1 = """я можу бути програмістом
і мова Python потрібна звісно."""
print(вірш1)
# з екранованої послідовностю \n
вірш2 = "я можу бути програмістом\ні мова Python потрібна звісно."
print(вірш2)
```

Висновок:

```
я можу бути програмістом
і мова Python потрібна звісно.
```

Ще одна корисна *escape sequence*, про яку слід знати, це послідовність символів:`\t`(розглядається як один символ табуляції). Існує ще багато escape sequences, але я згадав тут лише найкорисніші.

Варто зауважити, що зворотній слеш в кінці рядка означає, що рядок продовжується в наступному

рядку, але новий рядок не додається. Наприклад:

```
"Це перше речення. \
Це друге речення."
```

еквівалентно

```
"Це перше речення. Це друге речення."
```

3.7.5 Необроблений рядок

 англійська: *Raw String*

Якщо вам потрібно вказати деякі рядки, де не обробляються спеціальні символи, наприклад escape sequences, тоді вам потрібно вказати *raw string*, додавши до рядка префікс `r` або `R`.Необроблений рядок Python розглядає символ зворотної похилої риски (\ ) як буквальний символ.Наприклад:

```
r"Нові рядки позначаються \n"
```

Примітка для користувачів регулярного виразу

 англійська: *Regular Expression User*

Завжди використовуйте raw strings під час роботи з регулярними виразами. Інакше може знадобитися чимало зворотних слешей. Наприклад, абревіатуру з регулярними виразами в Python можна знайти за допомогою `'\\1'` or `r'\1'` (на відміну роботи з Python, пошук абревіатури з регулярними виразами в інших системах виконується символом `\1`)

3.7.6 Змінні

 англійська: *Variable*

Використання лише літеральних констант незабаром може набриднути — нам потрібен якийсь спосіб зберігання будь-якої інформації та маніпулювання нею. Ось тут і з'являються *змінні*. Змінні — це саме те, що випливає з назви — їхнє значення може змінюватися, тобто ви можете зберігати будь-що за допомогою змінної. Змінні — це лише частини пам'яті комп'ютера, де зберігається деяка інформація. На відміну від літеральних констант, вам потрібен певний метод доступу до цих змінних і, отже, ви даєте їм імена.

3.7.7 Іменування ідентифікатора

 англійська: *Identifier Naming*

Змінні є прикладами ідентифікаторів. *Ідентифікатори* — це імена, надані для ідентифікації *чогось*. Існують деякі правила, яких слід дотримуватися для ідентифікаторів імен:

- Першим символом ідентифікатора має бути літера алфавіту (символ ASCII в верхньому(великі літери) або нижньому(малі літери) регістрі, або символ Unicode), а також знак підкреслення(_).
- Решта назви ідентифікатора може складатися з літер (символ ASCII в верхньому або нижньому регістрі, або символ Unicode),знак підкреслення (_) або цифр (0-9).
- Імена ідентифікаторів чутливі до регістру. Наприклад, `myname` і `myName` *не* те саме. Зверніть увагу на `n` у нижньому регістрі у першому випадку та на `N` у верхньому регістрі у другому випадку.

- Прикладами *допустимих* імен ідентифікаторів є `i`, `name_2_3`. Прикладами *неприпустимих* імен ідентифікаторів є `2things`, `this is spaced out`, `my-name` і `>a1b2_c3`.

3.7.8 Типи даних

 англійська: *Data Types*

Змінні можуть містити значення різних типів, які називаються *типами даних.*Основними типами є числа та рядки, які ми вже обговорювали. У наступних розділах ми побачимо, як створювати власні типи за допомогою *класів*.

3.7.9 Об'єкт

 англійська: *Object*

Пам'ятайте, Python називає все, що використовується в програмі, *об'єктом*. Це мається на увазі в загальному значенні. Замість того, щоб говорити "*щось*"", ми говоримо "*об'єкт*".

Примітка для користувачів об'єктно-орієнтованого програмування

Python сильно об'єктно-орієнтований у тому сенсі, що все є об'єктом, включаючи числа, рядки та функції.

Тепер ми побачимо, як використовувати змінні разом із літеральними константами. Збережіть наступний приклад і запустіть програму.

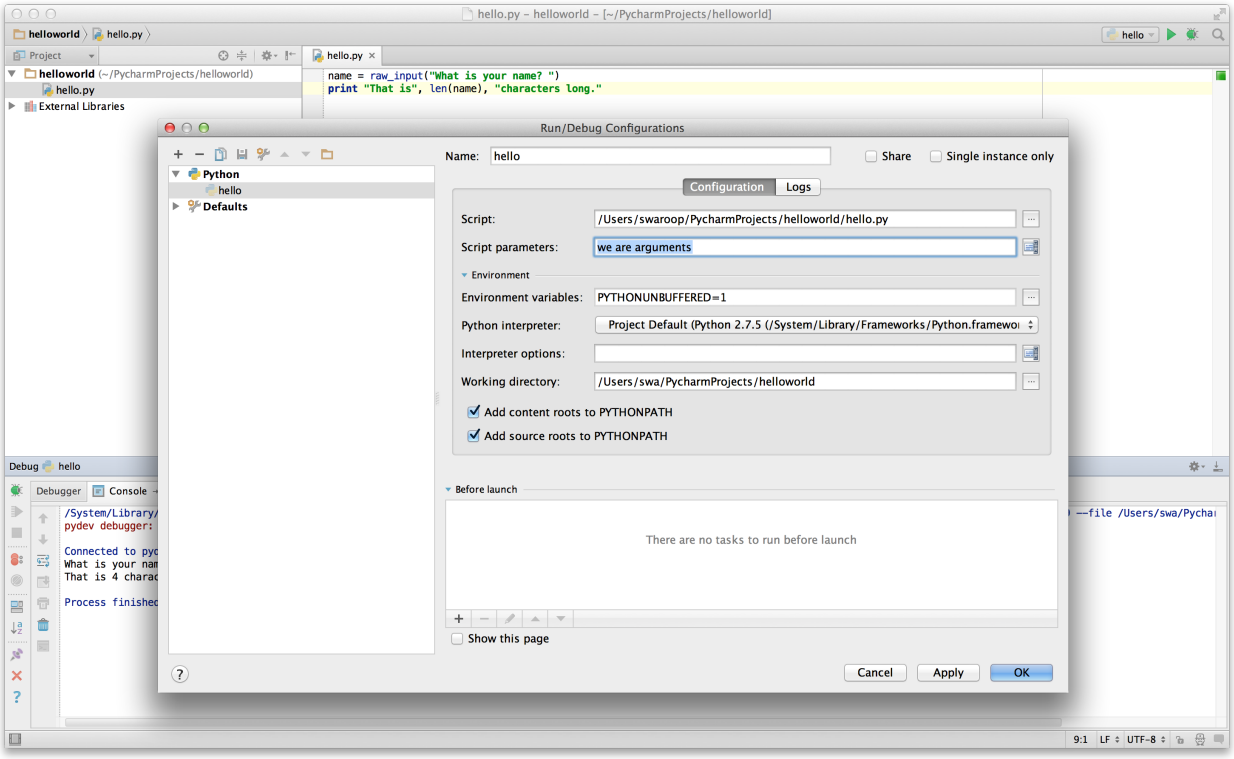
3.7.10 Як писати програми на Python

Відтепер стандартна процедура збереження та запуску програми Python така:

Для PyCharm

1. Відкрийте *PyCharm*.
2. Створіть новий файл з назвою файлу.
3. Введіть програмний код, наведений у прикладі.
4. Клацніть правою кнопкою миші та запустіть поточний файл.

ПРИМІТКА. Щоразу, коли вам потрібно надати *command line arguments ("аргументи командного рядка")*, натисніть кнопку`Run -> Edit Configurations`, введіть аргументи в розділ `Script parameters:` і натисніть кнопку `OK`:



Для інших редакторів

1. Відкрийте вибраний редактор.
2. Введіть програмний код, наведений у прикладі.
3. Збережіть його як файл з назвою файлу.
4. Запустіть інтерпретатор командою `python program.py`, щоб запустити програму.

Приклад: використання змінних і літеральних констант

Введіть і запустіть таку програму:

код python

```
# Ім'я файлу : var.py
i = 5
print(i)
i = i + 1
print(i)

s = '''Це багаторядковий рядок.
Це другий рядок.'''
print(s)
```

Висновок:

```
5
6
Це багаторядковий рядок.
Це другий рядок.
```

Як це працює

Ось як працює ця програма. Спочатку ми привласнюємо значення константи `5` змінній `i` за допомогою оператора присвоювання (=).Цей рядок називається statement - так би мовити "інструкція роботи для комп'ютера". Вираз i=5 має три частини:

1. "i" - назва змінної;
2. "=" - оператор присвоювання;
3. "5" - літеральна константа.

Ці частини не працюють поодинці, але разом вони створюють інструкцію роботи для комп'ютера. Цей рядок називається statement,оскільки в ньому зазначено, що потрібно щось зробити, і в цьому випадку ми з'єднуємо назву змінної `i` зі значенням `5`. Далі ми друкуємо значення `i` за допомогою функції `print`, яка, як не дивно, просто виводить значення змінної на екран.

Потім ми додаємо «1» до значення, збереженого в «i», і зберігаємо його там. Потім ми друкуємо його i, як очікується, отримуємо значення "6".

Подібним чином ми присвоюємо рядковий літерал змінній `s`, а потім друкуємо його.

Примітка для програмістів статичної мови

Змінні використовуються простим наданням їм значень. Ніякого попередньоного оголошення або визначення типу даних не потрібно/застосовується.

3.7.11 Логічні та фізичні рядки

 англійська: *Logical And Physical Line*

Фізична лінія - це те, що ви *бачите*, коли пишете програму. Логічний рядок — це те, що *Python сприймає* як "a single statement" (одну інструкцію). Python неявно припускає, що кожен *фізичний рядок* відповідає *логічному рядку*.

Прикладом логічного рядка є statement `print('Привіт, Світ!')`- якщо воно в одному рядку(як ви бачите це в редакторі), то цей рядок також відповідає фізичному рядку.

Неявно Python заохочує використання одного statement на рядок, що робить код більш читабельним.

Якщо ви хочете вказати більше ніж один логічний рядок в одному фізичному рядку, ви повинні явно вказати це за допомогою крапки з комою (`;`), яка вказує на кінець логічного рядка/statement. Наприклад:

```
i = 5
print(i)
```

те саме, що

```
i = 5;
print(i);
```

і те саме може бути записано у вигляді

```
i = 5; print(i);
```

або

```
i = 5; print(i)
```

Однак я *настійно рекомендую* вам дотримуватися *написання максимум одного логічного рядка в кожному фізичному рядку*. Ідея полягає в тому, що ви можете обійтися без крапки з комою. Фактично, я *ніколи* не використовував і навіть не бачив крапку з комою в програмі Python.

Існує одна ситуація, коли ця концепція дійсно корисна: якщо у вас є довгий рядок коду, ви можете розбити його на кілька фізичних рядків, використовуючи зворотній слеш. Це називається *явним з'єднанням рядків (англ."explicit line joining")*:

```
s = 'Це рядок. \
Цей рядок продовжується.'
print(s)
```

Висновок:

```
Це рядок. Цей рядок продовжується.
```

Аналогічно,

```
i = \
5
```

те саме, що

```
i = 5
```

Іноді існує неявне припущення, що вам не потрібно використовувати зворотній слеш. Це випадок, коли в логічному рядку є відкриваюча кругла, квадратна або фігурна дужка, але немає закриваючої . Це називається *неявним з'єднанням ліній* (англ."implicit line joining") . Ви можете побачити це в дії, коли ми пишемо програми за допомогою *list* у наступних розділах.

3.7.12 Відступи

 англійська: *Indentation*

Пробіли (англ."Whitespace") важливі в Python. Точніше *пробіли на початку рядка важливі*. Це називається *відступами*. Передні відступи (пробіли та табуляції) на початку логічного рядка використовуються для визначення рівня відступу логічного рядка, який, у свою чергу, використовується для групування statement.

Це означає, що statement, які йдуть разом, *повинні* мати однаковий відступ. Кожен такий набір statement називається *блоком*. У наступних розділах ми побачимо приклади важливості блоків.

Пам'ятайте, що неправильний відступ може спричинити помилки. Наприклад:

код python

```
i = 5
# Помилка нижче! Зверніть увагу на один пробіл на початку рядка
 print('Значення є', i)
print('Я повторюю, значення є', i)
```

Коли ви запускаєте це, ви отримуєте таку помилку:

```
  File "whitespace.py", line 3
    print('Значення є', i) # Помилка! Пробіл на початку рядка
    ^
IndentationError: unexpected indent
```

Зверніть увагу, що на початку другого рядка є один пробіл. Помилка, відображена у Python, повідомляє нам, що синтаксис програми невірний, тобто програму було написано неправильно.Для вас це означає те, що *ви не можете довільно починати нові блоки statement* (за винятком головного блоку за замовчуванням, який ви використовували протягом всієї програми,). Випадки, коли ви можете використовувати нові блоки, будуть детально описані в наступних розділах, наприклад *control flow*.

Як зробити відступ

Використовуйте чотири пробіли для відступу. Це офіційна рекомендація щодо мови Python. Хороші редактори автоматично зроблять це за вас. Переконайтеся, що ви використовуєте однакову кількість пробілів для відступів, інакше ваша програма не працюватиме або матиме неочікувану поведінку.

Примітка для програмістів статичної мови програмування

Python завжди використовуватиме відступи для блоків і ніколи не використовуватиме дужки. Запустіть `from __future__ import braces`, щоб дізнатися більше.

3.7.13 Резюме

Тепер, коли ми пройшли через багато дрібних деталей, ми можемо перейти до більш цікавих речей, таких як оператори потоку керування. Переконайтеся, що ви зрозуміли те, що ви прочитали в цьому розділі.

3.8 Оператори та вирази

 англійська: *Operators and Expressions*

Більшість statements (логічних рядків), які ви пишете, міститимуть *вирази*. Простим прикладом виразу є `2 + 3`. Вираз можна розбити на оператори та операнди.

Оператори — це певний функціонал, який виконує певні дії та який може бути представлений такими символами, як наприклад «+», або спеціальними ключовими словами. Операторам потрібні деякі дані для роботи, і такі дані називаються *операндами*. У цьому випадку "2" і "3" є операндами.

3.8.1 Оператори

 англійська: *Operators*

Ми коротко розглянемо оператори та їх використання.

Зверніть увагу, що ви можете обчислити вирази, наведені у прикладах, використовуючи інтерпретатор інтерактивно. Наприклад, щоб перевірити вираз `2 + 3`, скористайтесь інтерактивним командним рядком інтерпретатора Python:

код Python (використовуючи idle)

```
>>>2 + 3
5
>>>3 * 5
15
>>>
```

Ось короткий огляд доступних операторів:

- `+` (Плюс,  англійська: *plus*)
  - Підсумовує два об'єкта
  - `3 + 5` дорівнює 8. `'a' + 'b'` дорівнює `'ab'`.
- `-` (Мінус,  англійська: *minus*)
  - Дає віднімання одного числа від іншого; якщо перший операнд відсутній, він вважається нульовим.
  - `-5.2` дасть негативне число, а `50 - 24` дасть 26.
- `*` (Множення,  англійська: *multiply*)
  - Видає множення двох чисел або повертає рядок, що повторюється задане число разів.
  - `2 * 3` дорівнює 6. `'la' * 3` дорівнює `'lalala'`.
- `**` (Піднесення до степеня,  англійська: *power*)
  - Повертає число x, зведене в ступінь y
  - `3 ** 4` дорівнює 81 (тобто `3 * 3 * 3 * 3`)
- `/` (Ділення,  англійська: *divide*)
  - Ділить x на y
  - `13 / 3` дорівнює `4.333333333333333`
- `//` (Цілочисельний поділ,  англійська: *divide and floor*)
  - Розділіть x на y та округліть відповідь *вниз* до найближчого цілого значення. Зауважте, що якщо одне зі значень є числом з плаваючою комою, ви отримаєте значення з плаваючою комою.
  - `13 // 3` дорівнює 4
  - `-13 // 3` дорівнює `-5`
  - `9//1.81` дорівнює `4.0`
- `%` (Поділ по модулю,  англійська: *modulo*)
  - Повертає залишок від ділення
  - `13 % 3` дорівнює 1. `-25.5 % 2.25` дорівнює `1.5`.
- `<<` (Оператори зсуву вліво на задану кількість біт,  англійська: *left shift*)

  - Зсуває біти числа вліво на задане число позицій. (Кожне число представлено в пам'яті бітами або двійковими цифрами, тобто 0 і 1)
  - `2 << 2` дорівнює `8`. У двійковій системи числення `2` представляє собою`10`.
  - Зрушення вліво на 2 біта дає «1000», що у десятковій системи числення означає 8.

- `>>` (Оператори зсуву вправо на задану кількість біт,  англійська: *right shift*)
  - Зсуває біти числа вправо на задане число позицій.
  - `11 >> 1` дорівнює `5`.
  - У двійковій системи числення `11` представлено в бітах як `1011` яке при зсуві вправо на 1 біт дорівнює `101` і яке є десятковим `5`.

- `&` (Побітове І,  англійська: *bit-wise AND*)
  - Побітова операція І над числами: якщо обидва біти дорівнюють `1`, то результат дорівнює `1`. В іншому випадку це `0`.
  - `5 & 3` дорівнює `1` (`0101 & 0011` дорівнює `0001`)

- `|` (Побітове АБО,  англійська: *bit-wise OR*)
  - Побітова операція АБО над числами: якщо обидва біти дорівнюють `0`, результат `0`. В іншому випадку це `1`.
  - `5 | 3` дорівнює `7` (`0101 | 0011` дорівнює `0111`)

- `^` (Побітове виключне АБО,  англійська: *bit-wise XOR*)
  - Побітова операція виключне АБО над числами: якщо обидва біти (`1 або 0`) однакові, результат дорівнює `0`. В іншому випадку це `1`.
  - `5 ^ 3` дорівнює `6` (`0101 ^ 0011` дорівнює `0110`)

- `~`(Побітове НЕ,  англійська: *bit-wise invert*)
  - Побітова операція НЕ для числа x відповідає -(x+1)
  - `~5` дорівнює `-6`. Детальніше на http://stackoverflow.com/a/11810203

- `<` (Менше ніж,  англійська: *less than*)
  - Визначає чи вірно те, що x менше за y. Усі оператори порівняння повертають`True` або `False`. Зверніть увагу на написання цих імен з великої літери.
  - `5 < 3` дорівнює `False`, а `3 < 5` дорівнює `True`.
  - Можна складати довільні ланцюжки: `3 < 5 < 7` дає `True`.

- `>` (Більше ніж,  англійська: *greater than*)
  - Визначає чи вірно те, що x більше за y
  - `5 > 3` повертає `True`. Якщо обидва операнди є числами, вони спочатку перетворюються на однаковий тип. В іншому випадку завжди повертається `False`.

- `<=` (менше або дорівнює,  англійська: *less than or equal to*)
  - Визначає чи вірно те, що x менше або дорівнює y
  - `x = 3; y = 6; x <= y` повертає `True`

- `>=` (більше або дорівнює, англ.»  англійська: *greater than or equal to*)
  - Визначає чи вірно те, що x більше або дорівнює y
  - `x = 4; y = 3; x >= 3` повертає `True`
- `==` (дорівнює,  англійська: *equal to*)
  - Порівнює, чи однакові об'єкти
  - `x = 2; y = 2; x == y` повертає`True`
  - `x = 'str'; y = 'stR'; x == y` повертає `False`
  - `x = 'str'; y = 'str'; x == y` повертає `True`
- `!=` (не дорівнює,  англійська: *not equal to*)
  - Порівнює, якщо об'єкти не рівні
  - `x = 2; y = 3; x != y` повертає `True`
- `not` (логічне НЕ,  англійська: *boolean NOT*)
  - Якщо x дорівнює `True`, оператор поверне `False`. Якщо ж x дорівнює `False`, отримаємо `True`.
  - `x = True; not x` повертає `False`.
- `and` (логічне І,  англійська: *boolean AND*)
  - `x and y` повертає `False`,якщо x дорівнює `False`,в протилежному випадку(x=True) повертає значення y
  - `x = False; y = True; x and y` повертає `False` оскільки x є False. У цьому випадку Python не обчислюватиме y, оскільки він знає, що ліва частина виразу «and» має значення «False», що означає, що весь вираз буде «False» незалежно від інших значень. Це називається скороченою оцінкою булевих (логічних) виразів (англ. "short-circuit evaluation").
- `or` (логічне АБО,  англійська: *boolean OR*)
  - Якщо x дорівнює `True`, він повертає True, в протилежному випадку отримаємо значення y
  - `x = True; y = False; x or y` повертає `True`.Тут також застосовується скорочена оцінка виразів.

3.8.2 Короткий запис математичних операцій та привласнення

Зазвичай виконується математична операція над змінною, а потім привласнюється результат цієї операції тій самій змінній, отже, для таких виразів існує скорочення:

```
a = 2
a = a * 3
```

можна записати як:

```
a = 2
a *= 3
```

Зверніть увагу, що вирази виду `var = var operation expression`("змінна = змінна операція вираз") стає `var operation= expression`("змінна операція = вираз").

3.8.3 Порядок обчислення

англійська: *evaluation Order*

Якщо у вас є такий вираз, як «2 + 3 * 4», спочатку виконується додавання чи множення? Шкільний курс математики говорить нам, що спочатку потрібно виконати множення. Це означає, що оператор множення має вищий пріоритет, ніж оператор додавання.

У наступній таблиці нижче наведено пріоритет операторів Python, починаючи з самого найнижчого пріоритету (найслабше зв'язування, англ."least binding") до найвищого пріоритету (найсильніше зв'язування, англ."most binding"). Це означає, що в заданому виразі Python спочатку обчислює оператори та вирази, розташовані в таблиці нижче, а вже потім ті, що розташовані вище.

Для повноти опису наведено наступну таблицю, взяту з Посібника мови Python (англійська: *Python reference manual*). Набагато краще використовувати дужки для відповідного групування операторів і операндів, щоб явно вказати порядок обчислення виразів. Це робить програму більш читабельною. Див. *Зміна порядку оцінювання (англійська:* Changing the Order of Evaluation*)* нижче для отримання додаткової інформації.

- `lambda` : Лямбда-вираз (англійська: *Lambda Expression*)
- `if - else`: Умовний вираз (англійська: *Conditional expression*)
- `or` : Логічне АБО (англійська: *Boolean OR*)
- `and` : Логічне І (англійська: *Boolean AND*)
- `not x` : Логічне НЕ (англійська: *Boolean NOT*)
- `in, not in, is, is not, <, <=, >, >=, !=, ==` : Порівняння, включаючи тести на приналежність і тести на тотожність
- `|` : Побітове АБО (англійська: *Bitwise OR*)
- `^` : Побітове ВИКЛЮЧНО АБО (англійська: *Bitwise XOR*)
- `&` : Побітове І (англійська: *Bitwise AND*)
- `<<, >>` : Зміщення (англійська: *Shifts*)
- `+, -` : Додавання і віднімання(англійська: *Addition and subtraction*)
- `*, /, //, %` : Множення, ділення, цілочисельне ділення та залишок від ділення (англійська: *Multiplication, Division, Floor Division and Remainder*)
- `+x, -x, ~x` : Позитивне, негативне, побітове НЕ (англійська: *Positive, Negative, bitwise NOT*)
- `**` : Піднесення до степеня (англійська: *Exponentiation*)
- `x[index], x[index:index], x(arguments...), x.attribute` : Звернення за індексом,зріз,виклик функції, посилання на атрибут (англійська: *Subscription, slicing, call, attribute reference*)

- `(expressions,..)`, `[expressions,..]`, `{key: value,..}`, `{expressions...}` : Зв'язка або кортеж, список, словник, множина (англійська: *Binding or tuple display, list display, dictionary display, set display*)

Оператори, які ми ще не зустрічали, будуть пояснені в наступних розділах.

Оператори з *однаковим пріоритетом* містяться в одному рядку у наведеній вище таблиці. Наприклад, «+» і «-» мають однаковий пріоритет.

3.8.4 Зміна порядку обчислення

англійська: *Changing the Order Of Evaluation*

Щоб зробити вирази більш зрозумілими, ми можемо використовувати круглі дужки.Наприклад, `2 + (3 * 4)` легше зрозуміти, ніж `2 + 3 * 4`, яке вимагає знання пріоритетів операторів. Як і в усьому іншому, дужки слід використовувати розумно (не перестарайтеся) і вони не повинні бути зайвими, як у `(2 + (3 * 4))`.

Є додаткова перевага використання круглих дужок - це допомагає нам змінити порядок обчислення. Наприклад, якщо ви хочете, щоб додавання обчислювалося перед множенням у виразі, ви можете написати щось на зразок `(2 + 3) * 4`.

3.8.5 Асоціативність

Оператори зазвичай обробляються зліва направо. Це означає, що оператори з однаковим пріоритетом обчислюються зліва направо. Наприклад, "2 + 3 + 4" обчислюється як "(2 + 3) + 4".

3.8.6 Вирази

англійська: *Expressions*

код Python expression_ukr.py

```
довжина = 5
ширина = 2

площа = довжина * ширина
print('Площа дорівнює', площа)
print('Периметр дорівнює', 2 * (довжина + ширина))
```

Висновок:

```
$ python expression_ukr.py
Площа дорівнює 10
Периметр дорівнює 14
```

Як це працює

Довжина і ширина прямокутника зберігаються в однойменних змінних (довжина та ширина) відповідно. Ми використовуємо їх для обчислення площі та периметра прямокутника за допомогою виразів. Ми зберігаємо результат виразу `довжина * ширина` у змінній `площа`, а потім друкуємо його за допомогою функції `print`. У другому випадку ми безпосередньо використовуємо значення виразу `2 * (довжина + ширина)` у функції друку.

Також зверніть увагу на те, як Python *"гарно друкує"* результат. Навіть незважаючи на те, що не вказано пробіл у кінці речення у лапках `'Площа дорівнює'`, Python розміщує його для нас, щоб ми отримали чистий гарний результат, і програма стала набагато читабельнішою таким чином (оскільки не потрібно турбуватися про пробіли в рядках, які ми використовуємо для виведення(друку)). Це приклад того, як Python полегшує життя програміста.

3.8.7 Резюме

Ми побачили, як використовувати оператори, операнди та вирази - це основні будівельні блоки будь-якої програми. Далі ми побачимо, як використовувати їх у наших програмах за допомогою statements (операторів).

– Від перекладача –

3.8.8 Розуміння двійкових цифр

Двійкове число ( англійська: *binary number*) - це число, виражене у двійковій системі числення, використовуючи лише два різні символи, як 0 і 1.

Щоб краще зрозуміти двійкову систему числення, давайте спочатку розглянемо більш популярну десяткову систему числення,  англійська: *decimal numeral system*:

Вона містить 10 різних символів (0,1,2,3,4,5,6,7,8 і 9).

Щоб записати число більше 9, потрібно використати 2 або більше таких символів.

Наприклад, щоб записати число тридцять шість у десятковій системі, потрібно написати:

Тридцять шість — це двозначне число, тобто для його опису потрібні два символи.

Якщо читати обидва символи зліва направо, вони утворюють інструкцію обчислення:

Цікава частина цього полягає в тому, що лівий символ (3) має бути обчислений на 10, а правий символ (6) має бути обчислений на одиницю, а сума обох операцій представляє число.

Щоб записати число, більше 99, потрібні три цифри. Наприклад, щоб записати число дев'ятсот сорок два, потрібно написати:

Оскільки число складається з трьох цифр, перший символ потрібно помножити на 100, другий на 10 і третій на 1.

Цю десяткову систему числення також називають системою з основою 10.

Коефіцієнти для множення символів на (1,10,100,...) можна записати у вигляді степенів основи 10:

Інструкцію з розрахунку числа 942 також можна записати так:

Двійкова система числення

Двійкова система має основу не 10, а 2:

Однозначні числа в двійковій системі можуть відображати тільки числа 0 і 1:

щоб виразити число нуль, ви пишете:

щоб виразити число один, ви пишете:

і для будь-якого числа, більшого за 1, необхідно більше цифр.

Для числа два ви пишете 10:

Для числа три ви пишете 11:

А для числа чотири потрібні вже три цифри:

Двійкові числа проти десяткових чисел

У цій таблиці порівнюються десяткова та двойкова системи числення

10-ва система	розрахунок	написання	2-ва система	розрахунок
0	0x100	нуль	0	0x20
1	1x100	один	1	1x20
2	2x100	два	10	1x21+0x20
3	3x100	три	11	1x21+1x20
4	4x100	чотири	100	1x22+0x21+0x20
5	5x100	п'ять	101	1x22+0x21+1x20
6	6x100	шість	110	1x22+1x21+0x20
7	7x100	сім	111	1x22+1x21+1x20
8	8x100	вісім	1000	1x23+0x22+0x21+0x20
9	9x100	дев'ять	1001	1x23+0x22+0x21+1x20
10	1x101+0x100	десять	1010	1x23+0x22+1x21+0x20
11	1x101+1x100	одинадцять	1011	1x23+0x22+1x21+1x20
12	1x101+2x100	дванадцять	1100	1x23+1x22+0x21+0x20
13	1x101+3x100	тринадцять	1101	1x23+1x22+0x21+1x20
14	1x101+4x100	чотирнадцять	1110	1x23+1x22+1x21+0x20
15	1x101+5x100	п'ятнадцять	1111	1x23+1x22+1x21+1x20
16	1x101+6x100	шістнадцять	10000	1x24+0x23+0x22+0x21+0x20

Як перетворити двійкову систему у десяткову

Необхідно записати основу (2) у степені (цифр):

як перетворити число з основою 2 `1001` на число з основою 10 (```9```):

число у двійковій системі:	1	0	0	1
степінь:	3	2	1	0
=	23	22	21	20
=	8	4	2	1
розрахунок:	8 +	0 +	0 +	1=
результат: 9				

У Pyhton ви можете просто написати префікс `0b` перед числом, щоб вказати, що ви використовуєте двійкову систему. Python миттєво перетворює число на десяткову систему:

```
>>>0b1001
9
```

Як перетворити десяткову систему у двійкову

Вам потрібно записати основу (2) у степені (цифри). Потім ви починаєте справа наліво встановлювати піднесення до степеня з основою 2 у порядку зростання, до того моменту поки сума всіх встановлених бітів є меншою або дорівнює числу з основою 10.

Приклад:

як перетворити число з основою 10 (37) у число з основою 2 (100101):

степінь:	5	4	3	2	1	0
	25	24	23	22	21	20
	32	16	8	4	2	1
встановлений біт:	1	0	0	1	0	1
37 =	32+	0+	0+	4+	0+	1
двійкова система: **100101**						

У Python ви просто використовуєте вбудовану функцію `bin()`, щоб перетворити число з основою 10 у число з основою 2 (з префіксом `0b`):

```
>>>bin(37)
'0b100101'
```

3.8.9 Оператори зсуву вліво на задану кількість біт

 англійська: *left shift operation*

Дивіться також статтю у Вікіпедії про Logical Shift

Операцію зсуву вліво можна виконати лише для двійкового числа (з основою 2). Простіше кажучи, усі числа переміщуються ліворуч, а крайні праві позиції заповнюються нулями. Знаком оператора є `<<`, і ви можете вказати, на скільки позицій ліворуч мають бути переміщені числа.

Приклад: виконайте операцію "зрушення вліво на 1" над двійковим числом `1001` (дев'ять за основою 10). Результат: `10010` (вісімнадцять за основою 10).

код Python `left_shift_ukr.py`

```
print("У двійковій системі:")
print("0b1001<<1")
print("=")
print(bin(0b1001<<1))
print()
print("У десятковій системі:")
print(0b1001, "<<1 =", 0b1001<<1)
```

Висновок:

```
У двійковій системі:
 0b1001<<1
=
0b10010

У десятковій системі:
9 <<1 = 18
```

Зображення нижче зі статті у Вікіпедії:

MSB = старший біт, LSB = молодший значущий біт

Зауважте, що біт, якби старший біт (MSB, число 7) був би 1, результуюче число було б більшим, як у прикладі вище.

left shift

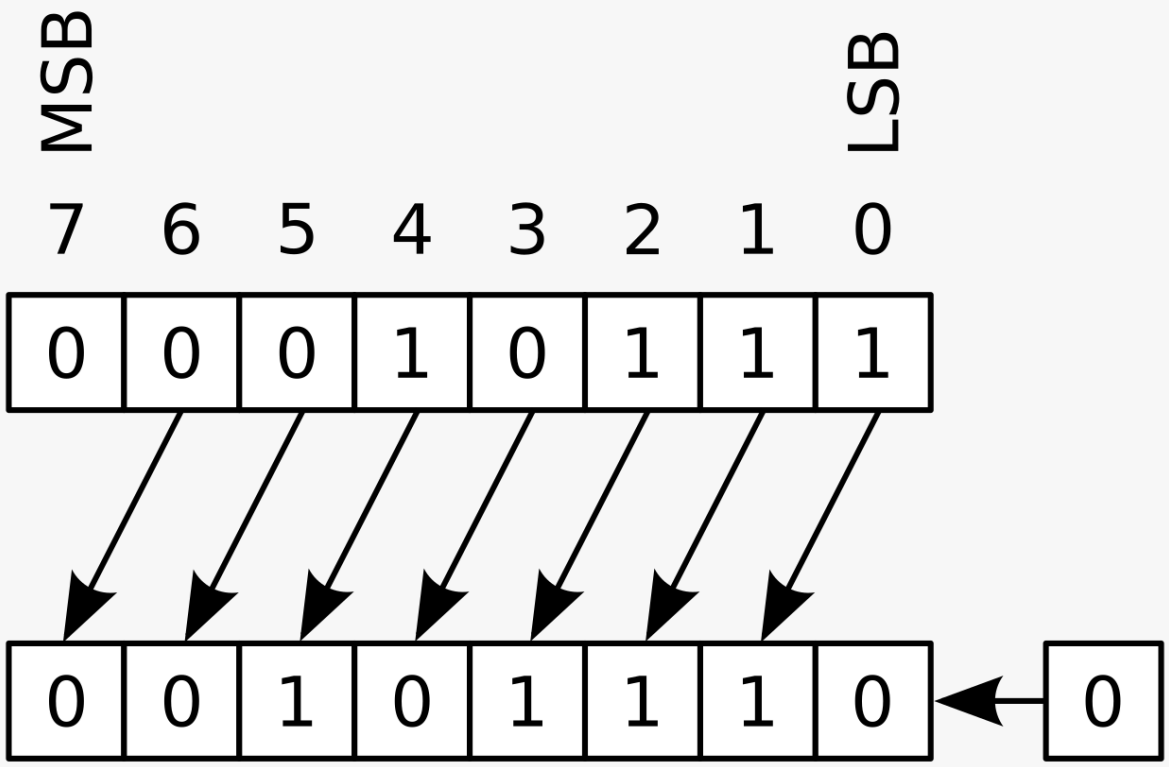


Fig. 1: Опис зображення: 0001 0111 (десяткове число 23) логічно зсувається на один біт вліво, результат 0010 1110 (десяткове число 46).
Image rights: By en:User:Cburnett - This W3C-unspecified vector image was created with Inkscape ., CC BY-SA 3.0, Link

3.8.10 Оператори зсуву вправо на задану кількість біт

англійська: *right shift operation*

Дивіться також статтю у Вікіпедії про Logical Shift

Операція зсуву вправо працює трохи подібно до операції зсуву вліво: усі числа переміщуються праворуч, крайні ліві позиції заповнюються нулями, а числа, які були в крайній правій позиції, просто видаляються.

Знаком оператора є `>>`, і ви можете вказати, на скільки позицій праворуч слід перемістити кожне число.

Приклад: виконайте операцію «зсув праворуч на 1» над двійковим числом `1001` (дев'ять за основою 10). Результатом є `0100` і зазвичай записується без нуля на початку як `100` (чотири за основою 10).

код Python `right_shift_ukr.py`

```
print("У двійковій системі:")
print("0b1001>>1")
```

```
print("=")
print(" "+bin(0b1001>>1))
print()
print("У десятковій системі:")
print(0b1001, ">>1 =", 0b1001>>1)
```

Висновок:

```
У двійковій системі:
0b1001>>1
=
 0b100

У десятковій системі:
9 >>1 = 4
```

Зображення нижче зі статті у Вікіпедії:

MSB = старший біт, LSB = молодший значущий біт

Зауважте, що на зображенні нижче молодший значущий біт (LSB) знищується

right shift

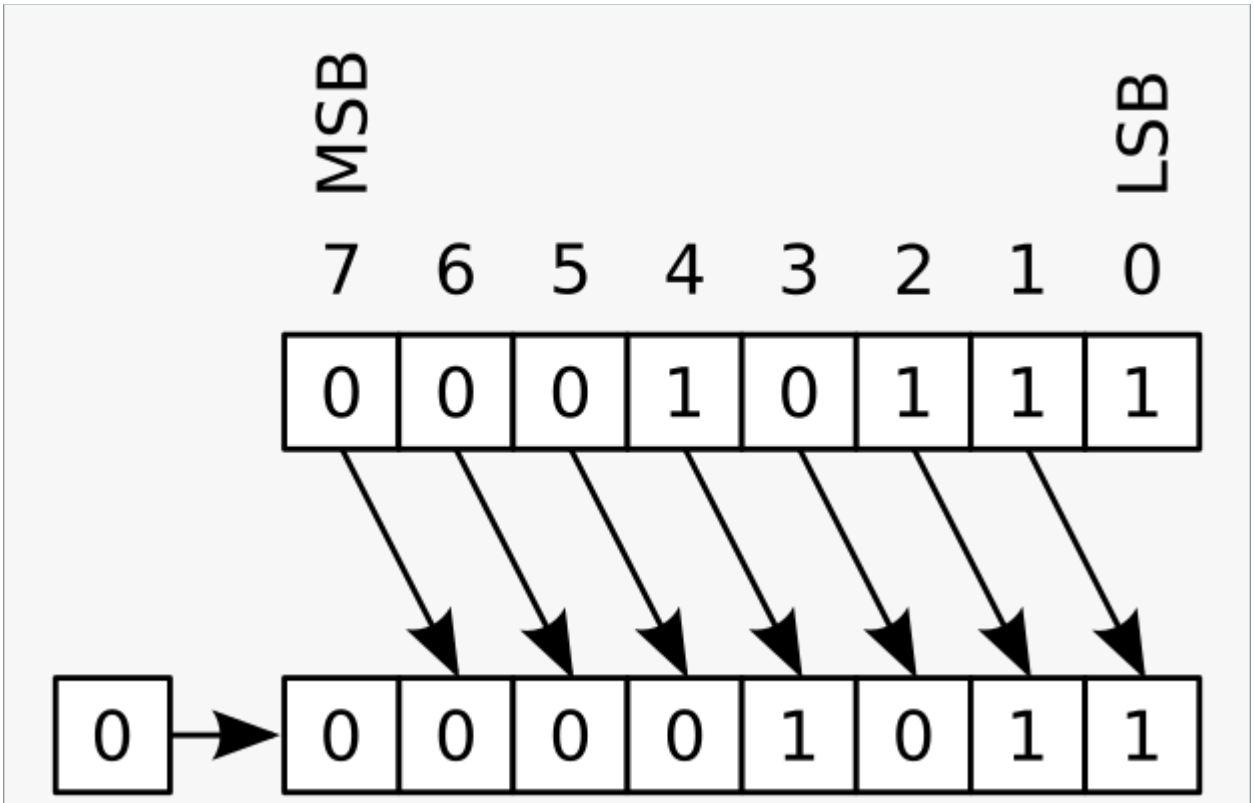


Fig. 2: Опис зображення: 0001 0111 (десяткове число 23) логічно зсувається на один біт вправо,результат буде: 0000 1011 (десяткове число 11)
Image rights: By en:User:Cburnett - This W3C-unspecified vector image was created with Inkscape ., CC BY-SA 3.0, Link

3.9 Логічні оператори python

англійська: *boolean operators*

значення A	значення B	результат для AND	результат для OR
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

3.9.1 Логічне НЕ

англійська: *NOT*

Оператор NOT дає значення інвертування: True стає False, а False стає True.

операція	результат
`not True`	`False`
`not False`	`True`

3.9.2 Логічне І

 англійська: *AND*

Оператор AND повинен бути розміщений між двома логічними значеннями. Результатом буде True, якщо обидва логічні значення True, інакше результат буде False

значення A	AND	значення B	результат
`True`	AND	`True`	= `True`
`True`	AND	`False`	=`False`
`False`	AND	`True`	=`False`
`False`	AND	`False`	=`False`

3.9.3 Логічне АБО

 англійська: *OR*

Оператор АБО має бути розміщено між двома логічними значеннями. Результат є True, якщо одне або обидва значення є True. Результатом буде False, якщо обидва значення є False.

значення A	OR	значення B	результат
`True`	OR	`True`	= `True`
`True`	OR	`False`	=`True`
`False`	OR	`True`	=`True`
`False`	OR	`False`	=`False`

3.9.4 Логічні операції з двійковими цифрами

 англійська: *boolean operations with binary digits*

Логічні операції (and, or, xor, not) можуть бути застосовані не лише до логічних значень (True і False), а й безпосередньо до бітів (де 1 означає True, а 0 означає False).

У Python є логічні команди `not or and` для логічних значень, але є також спеціальний набір команд для двійкових цифр:

```
~  Побітове НЕ (for bitwise inversion)
|  Побітове АБО (for bitwise OR)
&  Побітове І (for bitwise AND)
^  Побітове виключне АБО (for bitwise XOR)
```

Побітове НЕ

 англійська: *bitwise inversion*

~ (bit-wise invert) - Побітова операція НЕ для числа x відповідає -(x+1). Значення: до числа додається 0b01 і змінюється знак.

код Python `bitwise_inversion_ukr.py`

```
print("Приклад 1:")
print("Припустимо x становить 1001")
print("Побітове НЕ те саме, що -(x+1)")
print("Обчислювання x+1 наступне : 1001 + 0001")
print("Кінцевий результат є -(1001 + 0001)")
print("~0b1001 = ")
print(bin(~0b1001))
print("=-(x+1)=-(0b1001 + 0b0001) = -0b1010")
print("Приклад 2:")
print("~-0b1111 = ")
print("  "+bin(~-0b1111))
print("=-(x+1)=-(-0b1111+0b0001)=-0b1110")
```

Висновок:

```
Приклад 1:
Припустимо x становить 1001
Побітове НЕ те саме, що -(x+1)
Обчислювання x+1 наступне : 1001 + 0001
Кінцевий результат є -(1001 + 0001)
~0b1001 =
-0b1010
=-(x+1)=-(0b1001 + 0b0001) = -0b1010
Приклад 2:
~-0b1111 =
  0b1110
=-(x+1)=-(-0b1111+0b0001)=-0b1110
```

Побітове І

 англійська: *bitwise and*

- `&`(побітове І, англ."bit-wise AND")
  - Побітова операція І над числами: якщо обидва біти дорівнюють `1`, то результат дорівнює `1`. В іншому випадку це `0`.

Приклад A	Приклад B	Приклад C
0b1100	0b1111	0b1011
&	&	&
0b0101	0b0000	0b0001
=	=	=
0b0100	0b0000	0b0001

3.9.5 Побітне АБО

 англійська: *bitwise or, inclusive OR*

- | (bit-wise OR)
  - Побітова операція АБО над числами: якщо обидва біти дорівнюють `0`, результат `0`. В іншому випадку це `1`.

Приклад A	Приклад B	Приклад C
0b1100	0b1111	0b1011
|	|	|
0b0101	0b0000	0b0001
=	=	=
0b1101	0b1111	0b1011

3.9.6 Побітне ВИКЛЮЧНО АБО

 англійська: *bitwise xor, exclusive OR*

- ^ (bit-wise XOR)
  - Побітова операція виключне АБО над числами: якщо обидва біти (`1 або 0`) однакові, результат дорівнює `0`. В іншому випадку це `1`.

Приклад A	Приклад B	Приклад C
0b1100	0b1111	0b1011
^	^	^
0b0101	0b0000	0b0001
=	=	=
0b1001	0b1111	0b1010

3.10 Оператори та вирази

 англійська: *Operators and Expressions*

Більшість statements (логічних рядків), які ви пишете, міститимуть *вирази*. Простим прикладом виразу є `2 + 3`. Вираз можна розбити на оператори та операнди.

Оператори — це певний функціонал, який виконує певні дії та який може бути представлений такими символами, як наприклад «+», або спеціальними ключовими словами. Операторам потрібні деякі дані для роботи, і такі дані називаються *операндами*. У цьому випадку "2" і "3" є операндами.

3.10.1 Оператори

 англійська: *Operators*

Ми коротко розглянемо оператори та їх використання.

Зверніть увагу, що ви можете обчислити вирази, наведені у прикладах, використовуючи інтерпретатор інтерактивно. Наприклад, щоб перевірити вираз `2 + 3`, скористайтесь інтерактивним командним рядком інтерпретатора Python:

код Python (використовуючи idle)

```
>>>2 + 3
5
>>>3 * 5
15
>>>
```

Ось короткий огляд доступних операторів:

- `+` (Плюс,  англійська: *plus*)
  - Підсумовує два об'єкта
  - `3 + 5` дорівнює 8. `'a' + 'b'` дорівнює `'ab'`.
- `-` (Мінус,  англійська: *minus*)
  - Дає віднімання одного числа від іншого; якщо перший операнд відсутній, він вважається нульовим.
  - `-5.2` дасть негативне число, а `50 - 24` дасть 26.
- `*` (Множення,  англійська: *multiply*)
  - Видає множення двох чисел або повертає рядок, що повторюється задане число разів.
  - `2 * 3` дорівнює 6. `'la' * 3` дорівнює `'lalala'`.
- `**` (Піднесення до степеня,  англійська: *power*)
  - Повертає число x, зведене в ступінь y
  - `3 ** 4` дорівнює 81 (тобто `3 * 3 * 3 * 3`)
- `/` (Ділення,  англійська: *divide*)
  - Ділить x на y
  - `13 / 3` дорівнює `4.333333333333333`
- `//` (Цілочисельний поділ,  англійська: *divide and floor*)
  - Розділіть x на y та округліть відповідь *вниз* до найближчого цілого значення. Зауважте, що якщо одне зі значень є числом з плаваючою комою, ви отримаєте значення з плаваючою комою.
  - `13 // 3` дорівнює 4
  - `-13 // 3` дорівнює `-5`
  - `9//1.81` дорівнює `4.0`
- `%` (Поділ по модулю,  англійська: *modulo*)
  - Повертає залишок від ділення
  - `13 % 3` дорівнює 1. `-25.5 % 2.25` дорівнює `1.5`.
- `<<` (Оператори зсуву вліво на задану кількість біт,  англійська: *left shift*)

  - Зсуває біти числа вліво на задане число позицій. (Кожне число представлено в пам'яті бітами або двійковими цифрами, тобто 0 і 1)
  - `2 << 2` дорівнює `8`. У двійковій системи числення `2` представляє собою`10`.
  - Зрушення вліво на 2 біта дає «1000», що у десятковій системи числення означає 8.

- `>>` (Оператори зсуву вправо на задану кількість біт,  англійська: *right shift*)
  - Зсуває біти числа вправо на задане число позицій.
  - `11 >> 1` дорівнює `5`.
  - У двійковій системи числення `11` представлено в бітах як `1011` яке при зсуві вправо на 1 біт дорівнює `101` і яке є десятковим `5`.

- `&` (Побітове І,  англійська: *bit-wise AND*)
  - Побітова операція І над числами: якщо обидва біти дорівнюють `1`, то результат дорівнює `1`. В іншому випадку це `0`.
  - `5 & 3` дорівнює `1` (`0101 & 0011` дорівнює `0001`)

- `|` (Побітове АБО,  англійська: *bit-wise OR*)
  - Побітова операція АБО над числами: якщо обидва біти дорівнюють `0`, результат `0`. В іншому випадку це `1`.
  - `5 | 3` дорівнює `7` (`0101 | 0011` дорівнює `0111`)

- `^` (Побітове виключне АБО,  англійська: *bit-wise XOR*)
  - Побітова операція виключне АБО над числами: якщо обидва біти (`1 або 0`) однакові, результат дорівнює `0`. В іншому випадку це `1`.
  - `5 ^ 3` дорівнює `6` (`0101 ^ 0011` дорівнює `0110`)

- `~`(Побітове НЕ,  англійська: *bit-wise invert*)
  - Побітова операція НЕ для числа x відповідає -(x+1)
  - `~5` дорівнює `-6`. Детальніше на http://stackoverflow.com/a/11810203

- `<` (Менше ніж,  англійська: *less than*)
  - Визначає чи вірно те, що x менше за y. Усі оператори порівняння повертають`True` або `False`. Зверніть увагу на написання цих імен з великої літери.
  - `5 < 3` дорівнює `False`, а `3 < 5` дорівнює `True`.
  - Можна складати довільні ланцюжки: `3 < 5 < 7` дає `True`.

- `>` (Більше ніж,  англійська: *greater than*)
  - Визначає чи вірно те, що x більше за y
  - `5 > 3` повертає `True`. Якщо обидва операнди є числами, вони спочатку перетворюються на однаковий тип. В іншому випадку завжди повертається `False`.

- `<=` (менше або дорівнює,  англійська: *less than or equal to*)
  - Визначає чи вірно те, що x менше або дорівнює y
  - `x = 3; y = 6; x <= y` повертає `True`

- `>=` (більше або дорівнює, англ.»  англійська: *greater than or equal to*)
  - Визначає чи вірно те, що x більше або дорівнює y
  - `x = 4; y = 3; x >= 3` повертає `True`
- `==` (дорівнює,  англійська: *equal to*)
  - Порівнює, чи однакові об'єкти
  - `x = 2; y = 2; x == y` повертає`True`
  - `x = 'str'; y = 'stR'; x == y` повертає `False`
  - `x = 'str'; y = 'str'; x == y` повертає `True`
- `!=` (не дорівнює,  англійська: *not equal to*)
  - Порівнює, якщо об'єкти не рівні
  - `x = 2; y = 3; x != y` повертає `True`
- `not` (логічне НЕ,  англійська: *boolean NOT*)
  - Якщо x дорівнює `True`, оператор поверне `False`. Якщо ж x дорівнює `False`, отримаємо `True`.
  - `x = True; not x` повертає `False`.
- `and` (логічне І,  англійська: *boolean AND*)
  - `x and y` повертає `False`,якщо x дорівнює `False`,в протилежному випадку(x=True) повертає значення y
  - `x = False; y = True; x and y` повертає `False` оскільки x є False. У цьому випадку Python не обчислюватиме y, оскільки він знає, що ліва частина виразу «and» має значення «False», що означає, що весь вираз буде «False» незалежно від інших значень. Це називається скороченою оцінкою булевих (логічних) виразів (англ. "short-circuit evaluation").
- `or` (логічне АБО,  англійська: *boolean OR*)
  - Якщо x дорівнює `True`, він повертає True, в протилежному випадку отримаємо значення y
  - `x = True; y = False; x or y` повертає `True`.Тут також застосовується скорочена оцінка виразів.

3.10.2 Короткий запис математичних операцій та привласнення

Зазвичай виконується математична операція над змінною, а потім привласнюється результат цієї операції тій самій змінній, отже, для таких виразів існує скорочення:

```
a = 2
a = a * 3
```

можна записати як:

```
a = 2
a *= 3
```

Зверніть увагу, що вирази виду `var = var operation expression`("змінна = змінна операція вираз") стає `var operation= expression`("змінна операція = вираз").

3.10.3 Порядок обчислення

англійська: *evaluation Order*

Якщо у вас є такий вираз, як «2 + 3 * 4», спочатку виконується додавання чи множення? Шкільний курс математики говорить нам, що спочатку потрібно виконати множення. Це означає, що оператор множення має вищий пріоритет, ніж оператор додавання.

У наступній таблиці нижче наведено пріоритет операторів Python, починаючи з самого найнижчого пріоритету (найслабше зв'язування, англ."least binding") до найвищого пріоритету (найсильніше зв'язування, англ."most binding"). Це означає, що у будь-якому вираженні Python спочатку оцінить оператори та вирази, розташовані нижче в таблиці, а потім ті, що перераховані вище в таблиці.

Для повноти опису наведено наступну таблицю, взяту з Посібника мови Python (англійська: *Python reference manual*). Набагато краще використовувати дужки для відповідного групування операторів і операндів, щоб явно вказати порядок обчислення виразів. Це робить програму більш читабельною. Див. *Зміна порядку оцінювання (англійська:* Changing the Order of Evaluation*)* нижче для отримання додаткової інформації.

- `lambda` : Лямбда-вираз (англійська: *Lambda Expression*)
- `if - else`: Умовний вираз (англійська: *Conditional expression*)
- `or` : Логічне АБО (англійська: *Boolean OR*)
- `and` : Логічне І (англійська: *Boolean AND*)
- `not x` : Логічне НЕ (англійська: *Boolean NOT*)
- `in, not in, is, is not, <, <=, >, >=, !=, ==` : Порівняння, включаючи тести на приналежність і тести на тотожність
- `|` : Побітове АБО (англійська: *Bitwise OR*)
- `^` : Побітове ВИКЛЮЧНО АБО (англійська: *Bitwise XOR*)
- `&` : Побітове І (англійська: *Bitwise AND*)
- `<<, >>` : Зміщення (англійська: *Shifts*)
- `+, -` : Додавання і віднімання(англійська: *Addition and subtraction*)
- `*, /, //, %` : Множення, ділення, цілочисельне ділення та залишок від ділення (англійська: *Multiplication, Division, Floor Division and Remainder*)
- `+x, -x, ~x` : Позитивне, негативне, побітове НЕ (англійська: *Positive, Negative, bitwise NOT*)
- `**` : Піднесення до степеня (англійська: *Exponentiation*)
- `x[index], x[index:index], x(arguments...), x.attribute` : Звернення за індексом,зріз,виклик функції, посилання на атрибут (англійська: *Subscription, slicing, call, attribute reference*)

- `(expressions,..)`, `[expressions,..]`, `{key: value,..}`, `{expressions...}` : Зв'язка або кортеж, список, словник, множина (англійська: *Binding or tuple display, list display, dictionary display, set display*)

Оператори, які ми ще не зустрічали, будуть пояснені в наступних розділах.

Оператори з *однаковим пріоритетом* містяться в одному рядку у наведеній вище таблиці. Наприклад, «+» і «-» мають однаковий пріоритет.

3.10.4 Зміна порядку обчислення

англійська: *Changing the Order Of Evaluation*

Щоб зробити вирази більш зрозумілими, ми можемо використовувати круглі дужки.Наприклад, `2 + (3 * 4)` легше зрозуміти, ніж `2 + 3 * 4`, яке вимагає знання пріоритетів операторів. Як і в усьому іншому, дужки слід використовувати розумно (не перестарайтеся) і вони не повинні бути зайвими, як у `(2 + (3 * 4))`.

Є додаткова перевага використання круглих дужок - це допомагає нам змінити порядок обчислення. Наприклад, якщо ви хочете, щоб додавання обчислювалося перед множенням у виразі, ви можете написати щось на зразок `(2 + 3) * 4`.

3.10.5 Асоціативність

Операторы зазвичай обробляються зліва направо. Це означає, що оператори з однаковим пріоритетом обчислюються зліва направо. Наприклад, "2 + 3 + 4" обчислюється як "(2 + 3) + 4".

3.10.6 Вирази

англійська: *Expressions*

код Python expression_ukr.py

```
довжина = 5
ширина = 2

площа = довжина * ширина
print('Площа дорівнює', площа)
print('Периметр дорівнює', 2 * (довжина + ширина))
```

Висновок:

```
$ python expression_ukr.py
Площа дорівнює 10
Периметр дорівнює 14
```

Як це працює

Довжина і ширина прямокутника зберігаються в однойменних змінних (довжина та ширина) відповідно. Ми використовуємо їх для обчислення площі та периметра прямокутника за допомогою виразів. Ми зберігаємо результат виразу `довжина * ширина` у змінній `площа`, а потім друкуємо його за допомогою функції `print`. У другому випадку ми безпосередньо використовуємо значення виразу `2 * (довжина + ширина)` у функції друку.

Також зверніть увагу на те, як Python *"гарно друкує"* результат. Навіть незважаючи на те, що не вказано пробіл у кінці речення у лапках `'Площа дорівнює'`, Python розміщує його для нас, щоб ми отримали чистий гарний результат, і програма стала набагато читабельнішою таким чином (оскільки не потрібно турбуватися про пробіли в рядках, які ми використовуємо для виведення(друку)). Це приклад того, як Python полегшує життя програміста.

3.10.7 Резюме

Ми побачили, як використовувати оператори, операнди та вирази - це основні будівельні блоки будь-якої програми. Далі ми побачимо, як використовувати їх у наших програмах за допомогою statements (операторів).

– Від перекладача –

3.10.8 Розуміння двійкових цифр

Двійкове число ( англійська: *binary number*) - це число, виражене у двійковій системі числення, використовуючи лише два різні символи, як 0 і 1.

Щоб краще зрозуміти двійкову систему числення, давайте спочатку розглянемо більш популярну десяткову систему числення,  англійська: *decimal numeral system*:

Вона містить 10 різних символів (0,1,2,3,4,5,6,7,8 і 9).

Щоб записати число більше 9, потрібно використати 2 або більше таких символів.

Наприклад, щоб записати число тридцять шість у десятковій системі, потрібно написати:

Тридцять шість — це двозначне число, тобто для його опису потрібні два символи.

Якщо читати обидва символи зліва направо, вони утворюють інструкцію обчислення:

Цікава частина цього полягає в тому, що лівий символ (3) має бути обчислений на 10, а правий символ (6) має бути обчислений на одиницю, а сума обох операцій представляє число.

Щоб записати число, більше 99, потрібні три цифри. Наприклад, щоб записати число дев'ятсот сорок два, потрібно написати:

Оскільки число складається з трьох цифр, перший символ потрібно помножити на 100, другий на 10 і третій на 1.

Цю десяткову систему числення також називають системою з основою 10.

Коефіцієнти для множення символів на (1,10,100,...) можна записати у вигляді степенів основи 10:

Інструкцію з розрахунку числа 942 також можна записати так:

Двійкова система числення

Двійкова система має основу не 10, а 2:

Однозначні числа в двійковій системі можуть відображати тільки числа 0 і 1:

щоб виразити число нуль, ви пишете:

щоб виразити число один, ви пишете:

і для будь-якого числа, більшого за 1, необхідно більше цифр.

Для числа два ви пишете 10:

Для числа три ви пишете 11:

А для числа чотири потрібні вже три цифри:

Двійкові числа проти десяткових чисел

У цій таблиці порівнюються десяткова та двойкова системи числення

10-ва система	розрахунок	написання	2-ва система	розрахунок
0	0x100	нуль	0	0x20
1	1x100	один	1	1x20
2	2x100	два	10	1x21+0x20
3	3x100	три	11	1x21+1x20
4	4x100	чотири	100	1x22+0x21+0x20
5	5x100	п'ять	101	1x22+0x21+1x20
6	6x100	шість	110	1x22+1x21+0x20
7	7x100	сім	111	1x22+1x21+1x20
8	8x100	вісім	1000	1x23+0x22+0x21+0x20
9	9x100	дев'ять	1001	1x23+0x22+0x21+1x20
10	1x101+0x100	десять	1010	1x23+0x22+1x21+0x20
11	1x101+1x100	одинадцять	1011	1x23+0x22+1x21+1x20
12	1x101+2x100	дванадцять	1100	1x23+1x22+0x21+0x20
13	1x101+3x100	тринадцять	1101	1x23+1x22+0x21+1x20
14	1x101+4x100	чотирнадцять	1110	1x23+1x22+1x21+0x20
15	1x101+5x100	п'ятнадцять	1111	1x23+1x22+1x21+1x20
16	1x101+6x100	шістнадцять	10000	1x24+0x23+0x22+0x21+0x20

Як перетворити двійкову систему у десяткову

Необхідно записати основу (2) у степені (цифр):

як перетворити число з основою 2 `1001` на число з основою 10 (```9```):

число у двійковій системі:	1	0	0	1
степінь:	3	2	1	0
=	23	22	21	20
=	8	4	2	1
розрахунок:	8 +	0 +	0 +	1=
результат: 9				

У Pyhton ви можете просто написати префікс `0b` перед числом, щоб вказати, що ви використовуєте двійкову систему. Python миттєво перетворює число на десяткову систему:

```
>>>0b1001
9
```

Як перетворити десяткову систему у двійкову

Вам потрібно записати основу (2) у степені (цифри). Потім ви починаєте справа наліво встановлювати піднесення до степеня з основою 2 у порядку зростання, до того моменту поки сума всіх встановлених бітів є меншою або дорівнює числу з основою 10.

Приклад:

як перетворити число з основою 10 (37) у число з основою 2 (100101):

степінь:	5	4	3	2	1	0
	25	24	23	22	21	20
	32	16	8	4	2	1
встановлений біт:	1	0	0	1	0	1
37 =	32+	0+	0+	4+	0+	1
двійкова система: **100101**						

У Python ви просто використовуєте вбудовану функцію `bin()`, щоб перетворити число з основою 10 у число з основою 2 (з префіксом `0b`):

```
>>>bin(37)
'0b100101'
```

3.10.9 Оператори зсуву вліво на задану кількість біт

 англійська: *left shift operation*

Дивіться також статтю у Вікіпедії про Logical Shift

Операцію зсуву вліво можна виконати лише для двійкового числа (з основою 2). Простіше кажучи, усі числа переміщуються ліворуч, а крайні праві позиції заповнюються нулями. Знаком оператора є `<<`, і ви можете вказати, на скільки позицій ліворуч мають бути переміщені числа.

Приклад: виконайте операцію "зрушення вліво на 1" над двійковим числом `1001` (дев'ять за основою 10). Результат: `10010` (вісімнадцять за основою 10).

код Python `left_shift_ukr.py`

```
print("У двійковій системі:")
print("0b1001<<1")
print("=")
print(bin(0b1001<<1))
print()
print("У десятковій системі:")
print(0b1001, "<<1 =", 0b1001<<1)
```

Висновок:

```
У двійковій системі:
 0b1001<<1
=
0b10010

У десятковій системі:
9 <<1 = 18
```

Зображення нижче зі статті у Вікіпедії:

MSB = старший біт, LSB = молодший значущий біт

Зауважте, що біт, якби старший біт (MSB, число 7) був би 1, результуюче число було б більшим, як у прикладі вище.

left shift

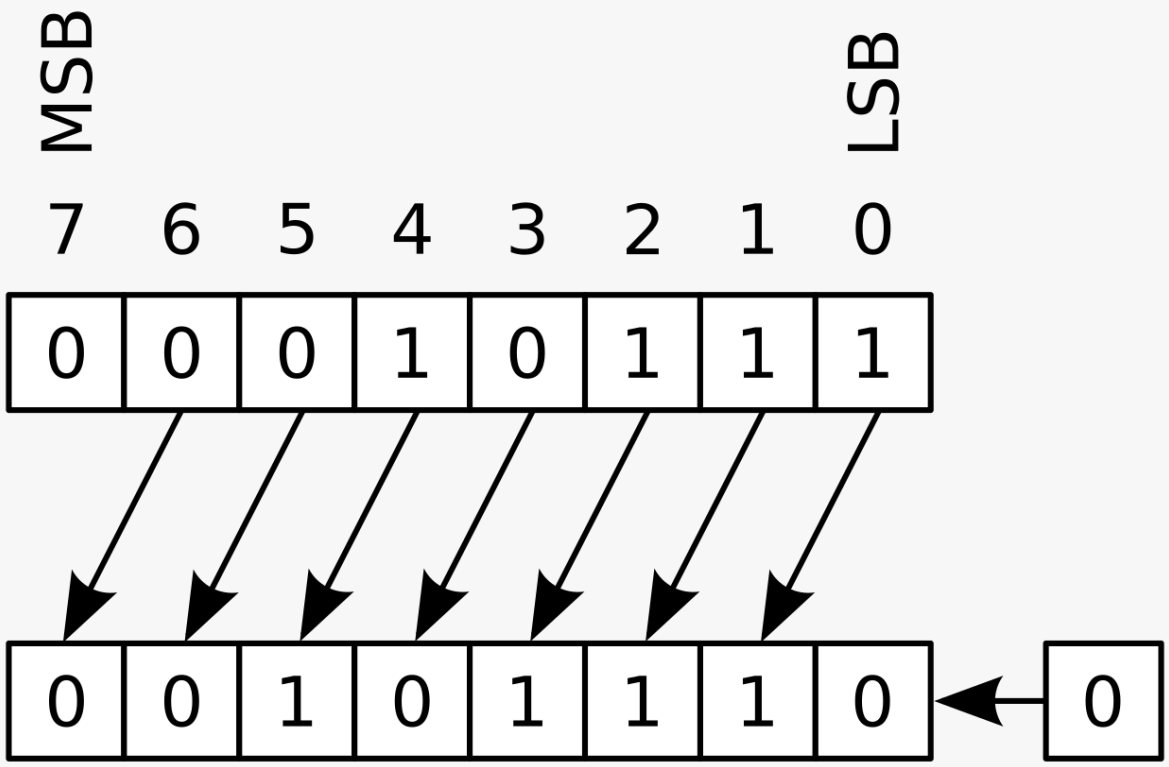


Fig. 3: Опис зображення: 0001 0111 (десяткове число 23) логічно зсувається на один біт вліво, результат 0010 1110 (десяткове число 46).
Image rights: By en:User:Cburnett - This W3C-unspecified vector image was created with Inkscape ., CC BY-SA 3.0, Link

3.10.10 Оператори зсуву вправо на задану кількість біт

англійська: *right shift operation*

Дивіться також статтю у Вікіпедії про Logical Shift

Операція зсуву вправо працює трохи подібно до операції зсуву вліво: усі числа переміщуються праворуч, крайні ліві позиції заповнюються нулями, а числа, які були в крайній правій позиції, просто видаляються.

Знаком оператора є `>>`, і ви можете вказати, на скільки позицій праворуч слід перемістити кожне число.

Приклад: виконайте операцію «зсув праворуч на 1» над двійковим числом `1001` (дев'ять за основою 10). Результатом є `0100` і зазвичай записується без нуля на початку як `100` (чотири за основою 10).

код Python `right_shift_ukr.py`

```
print("У двійковій системі:")
print("0b1001>>1")
```

```
print("=")
print(" "+bin(0b1001>>1))
print()
print("У десятковій системі:")
print(0b1001, ">>1 =", 0b1001>>1)
```

Висновок:

```
У двійковій системі:
0b1001>>1
=
 0b100

У десятковій системі:
9 >>1 = 4
```

Зображення нижче зі статті у Вікіпедії:

MSB = старший біт, LSB = молодший значущий біт

Зауважте, що на зображенні нижче молодший значущий біт (LSB) знищується

right shift

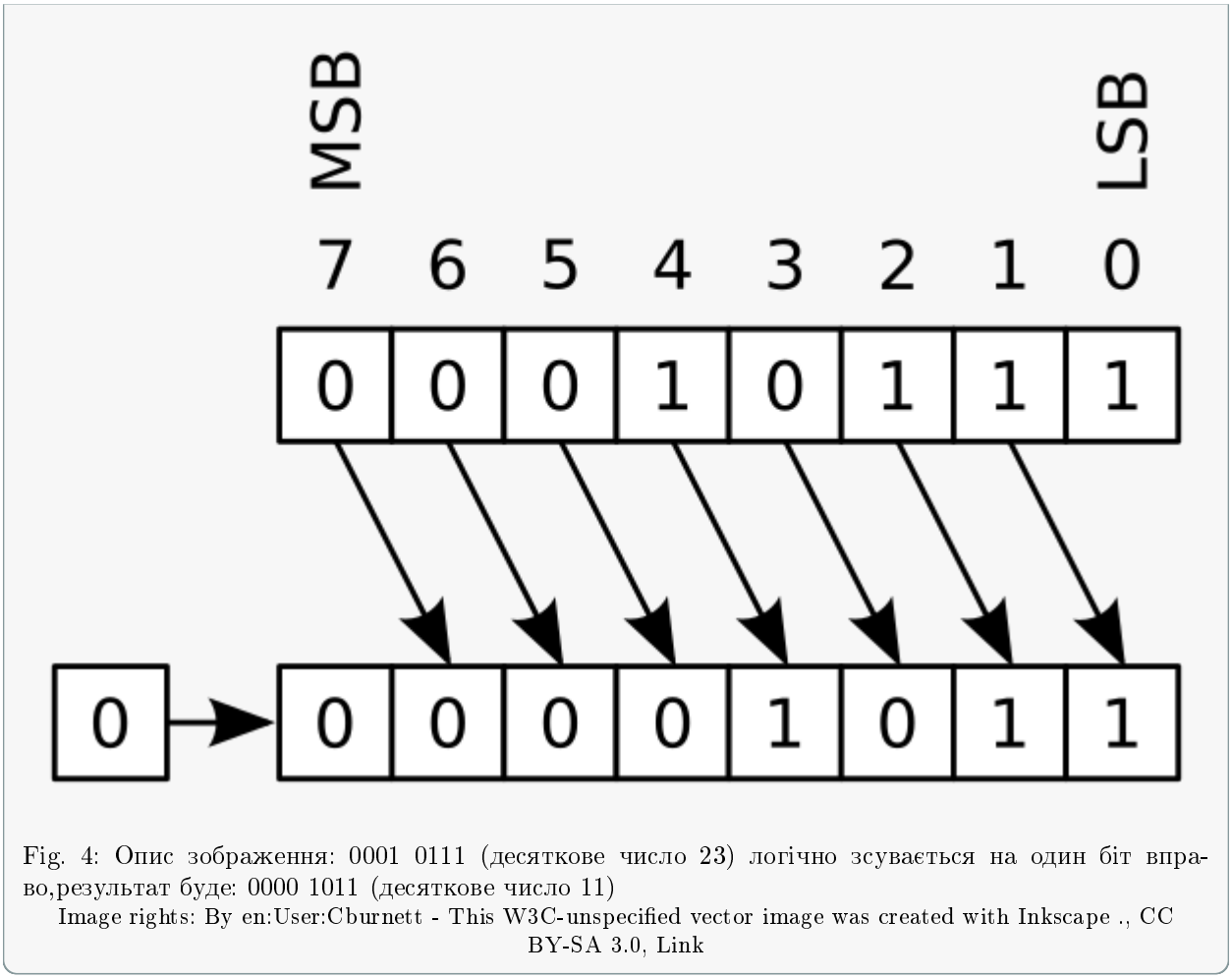


Fig. 4: Опис зображення: 0001 0111 (десяткове число 23) логічно зсувається на один біт вправо,результат буде: 0000 1011 (десяткове число 11)
Image rights: By en:User:Cburnett - This W3C-unspecified vector image was created with Inkscape ., CC BY-SA 3.0, Link

3.11 Логічні оператори python

англійська: *boolean operators*

значення A	значення B	результат для AND	результат для OR
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

3.11.1 Логічне НЕ

англійська: *NOT*

Оператор NOT дає значення інвертування: True стає False, а False стає True.

операція	результат
`not True`	`False`
`not False`	`True`

3.11.2 Логічне І

 англійська: *AND*

Оператор AND повинен бути розміщений між двома логічними значеннями. Результатом буде True, якщо обидва логічні значення True, інакше результат буде False

значення A	AND	значення B	результат
`True`	AND	`True`	= `True`
`True`	AND	`False`	=`False`
`False`	AND	`True`	=`False`
`False`	AND	`False`	=`False`

3.11.3 Логічне АБО

 англійська: *OR*

Оператор АБО має бути розміщено між двома логічними значеннями. Результат є True, якщо одне або обидва значення є True. Результатом буде False, якщо обидва значення є False.

значення A	OR	значення B	результат
`True`	OR	`True`	= `True`
`True`	OR	`False`	=`True`
`False`	OR	`True`	=`True`
`False`	OR	`False`	=`False`

3.11.4 Логічні операції з двійковими цифрами

 англійська: *boolean operations with binary digits*

Логічні операції (and, or, xor, not) можуть бути застосовані не лише до логічних значень (True і False), а й безпосередньо до бітів (де 1 означає True, а 0 означає False).

У Python є логічні команди `not or and` для логічних значень, але є також спеціальний набір команд для двійкових цифр:

```
~  Побітове НЕ (for bitwise inversion)
|  Побітове АБО (for bitwise OR)
&  Побітове І (for bitwise AND)
^  Побітове виключне АБО (for bitwise XOR)
```

Побітове НЕ

 англійська: *bitwise inversion*

~ (bit-wise invert) - Побітова операція НЕ для числа x відповідає -(x+1). Значення: до числа додається 0b01 і змінюється знак.

код Python `bitwise_inversion_ukr.py`

```
print("Приклад 1:")
print("Припустимо x становить 1001")
print("Побітове НЕ те саме, що -(x+1)")
print("Обчислювання x+1 наступне : 1001 + 0001")
print("Кінцевий результат є -(1001 + 0001)")
print("~0b1001 = ")
print(bin(~0b1001))
print("=-(x+1)=-(0b1001 + 0b0001) = -0b1010")
print("Приклад 2:")
print("~-0b1111 = ")
print("  "+bin(~-0b1111))
print("=-(x+1)=-(-0b1111+0b0001)=-0b1110")
```

Висновок:

```
Приклад 1:
Припустимо x становить 1001
Побітове НЕ те саме, що -(x+1)
Обчислювання x+1 наступне : 1001 + 0001
Кінцевий результат є -(1001 + 0001)
~0b1001 =
-0b1010
=-(x+1)=-(0b1001 + 0b0001) = -0b1010
Приклад 2:
~-0b1111 =
  0b1110
=-(x+1)=-(-0b1111+0b0001)=-0b1110
```

Побітове І

 англійська: *bitwise and*

- `&`(побітове І, англ."bit-wise AND")
  - Побітова операція І над числами: якщо обидва біти дорівнюють `1`, то результат дорівнює `1`. В іншому випадку це `0`.

Приклад A	Приклад B	Приклад C
0b1100	0b1111	0b1011
&	&	&
0b0101	0b0000	0b0001
=	=	=
0b0100	0b0000	0b0001

3.11.5 Побітне АБО

 англійська: *bitwise or, inclusive OR*

- | (bit-wise OR)
  - Побітова операція АБО над числами: якщо обидва біти дорівнюють `0`, результат `0`. В іншому випадку це `1`.

Приклад A	Приклад B	Приклад C
0b1100	0b1111	0b1011
|	|	|
0b0101	0b0000	0b0001
=	=	=
0b1101	0b1111	0b1011

3.11.6 Побітне ВИКЛЮЧНО АБО

 англійська: *bitwise xor, exclusive OR*

- ^ (bit-wise XOR)
  - Побітова операція виключне АБО над числами: якщо обидва біти (`1` або `0`) однакові, результат дорівнює `0`. В іншому випадку це `1`.

Приклад A	Приклад B	Приклад C
0b1100	0b1111	0b1011
^	^	^
0b0101	0b0000	0b0001
=	=	=
0b1001	0b1111	0b1010

3.12 Потік керування

 англійська: *Control Flow*

У програмах, які ми бачили дотепер, завжди була послідовність рядків коду, сумлінно виконаних Python у точному порядку зверху вниз. Що, якби ви хотіли змінити потік виконування рядків коду? Наприклад, ви хочете, щоб програма приймала деякі рішення та виконувала різні дії залежно від різних ситуацій, наприклад, друкувала "Доброго ранку" або "Доброго вечора" залежно від часу доби?

Як ви могли здогадатися, це досягається за допомогою операторів потоку керування (англ."control flow statements"). У Python є три оператора потоку керування: `if`, `for` і `while`.

3.12.1 Оператор `if`

 англійська: *The `if` statement*

Оператор `if` використовується для перевірки умови: *якщо* (англ."if") умова *вірна* (відповідає булевому значенню True), ми виконуємо блок рядків коду (званий *if-block*), *інакше* (англ."else") ми обробляємо інший блок рядків коду (званий *else-block*) . Блок *else* необов'язковий.

код python 'if_elif_else_ukr.py'

```
число = 23
здогадка = int(input('Введіть ціле число : '))

if здогадка == число:
    # Новий блок починається тут
    print('Вітаємо, ви вгадали,')
    print('(хоч і не виграли жодного призу!)')
    # Новий блок закінчується тут
elif здогадка < число:
    # Ще один блок
    print('Ні, задане число трохи вище, ніж здогадка')
    # Ви можете робити все, що завгодно в блоці ...
else:
    print('Ні, задане число трохи нижче, ніж здогадка')
    # щоб потрапити сюди здогадка повина бути більша, ніж число

print('Виконано')
# Цей останній рядок коду завжди виконується, після виконання оператора if.
```

Висновок (три рази):

```
$ python if_elif_else_ukr.py
Введіть ціле число: 50
Ні, задане число трохи нижче, ніж здогадка
Виконано

$ python if_elif_else.py
Введіть ціле число: 22
Ні, задане число трохи вище, ніж здогадка
Виконано

$ python if_elif_else.py
Введіть ціле число: 23
Вітаємо, ви вгадали.
(хоч і не виграли жодного призу!)
Виконано
```

Як це працює

У цій програмі ми приймаємо варіанти від користувача та перевіряємо, чи збігаються вони із заздалегідь заданим числом. Ми встановлюємо змінній `число` значення будь-якого цілого числа, якого хочемо, скажімо, 23. Потім ми приймаємо варіанти числа від користувача за допомогою функції `input()`. Функції — це лише багаторазово використовувані частини програм. Ми прочитаємо про них більше в *наступному розділі*.

Ми надаємо рядок вбудованій функції `input`, яка друкує його на екрані та чекає введення від користувача. Коли ми вводимо щось і натискаємо клавішу *Enter*, функція `input()` повертає те, що ми ввели, у вигляді рядка. Потім ми перетворюємо цей рядок на ціле число за допомогою `int`, та зберігаємо це значення в змінній `здогадка`. Насправді `int` — це клас, але на даному етапі вам достатньо знати лише, що ви можете використовувати його для перетворення рядка в ціле число (припускаючи, що рядок містить ціле число в тексті).

Далі ми порівнюємо здогадку користувача із заданим нами числом. Якщо вони рівні, ми друкуємо

повідомлення про успіх. Зверніть увагу, що ми використовуємо відповідні рівні відступу, щоб повідомити Python, які рядки коду належать до якого блоку. Ось чому відступи настільки важливі в Python. Сподіваюсь, ви дотримуєтесь правила "постійних відступів". Чи не так?

Зверніть увагу, що рядок коду, який починається з `if` мусить закінчуватися двокрапкою - ми вказуємо Python, що зараз буде слідувати рядок коду з відступом.

Потім ми перевіряємо, чи здогадка користувача є менша заданого числа, і якщо так, ми повідомляємо користувачу, що йому слід вибрати число трохи більші за це. Тут ми використали блок elif, який фактично об'єднує декілька блоків `if else-if else` в один об'єднаний блок `if-elif-else`. Це полегшує роботу з програмою та зменшує кількість необхідних відступів.

Блоки `elif` і `else` також повинні мати двокрапку в кінці логічного рядка, за якою слідує відповідні рядки коду (з відповідним числом відступів, звичайно).

Усередині if-блоку може бути інший блок `if` і так далі - це називається вкладеним блоком `if`.

Пам'ятайте, що блок `elif` і блок `else` необов'язкові. Мінімальний коректний запис блока `if` такий :

```
if True:
    print('Так, це вірно.')
```

Після того, як Python завершить виконання всього блока `if` разом із відповідними блоками `elif` і `else`, він переходить до наступного блока в блоці, що містить блок `if` (рядки коду з однаковим відступом). У нашому випадку це головний блок (в якому починається виконання програми), а наступним є вираз `print('Виконано')`. Після цього Python доходить до кінця програми та просто виходить із неї.

Незважаючи на те, що це дуже проста програма, я вказав на багато речей, на які ви повинні звернути увагу. Усе це досить просто (і напрочуд просто для тих із вас, хто має досвід C/C++). Спочатку вам потрібно буде запам'ятати всі ці речі, але після деякої практики з ними - ви звикнете, і все це буде для вас «природним».

Примітка для програмістів C/C++

Починаючи з версії 3.10, Python має оператор `"match - case"`, подібний до оператора `switch` в інших мовах програмування, таких як C, Java тощо. Для отримання додаткової інформації перегляньте офіційну документацію Python: https://docs.python.org/3/tutorial/controlflow.html#match-statements. Ви можете використовувати блок `if..elif..else` щоб зробити те саме,що і за допомогою оператора `switch`(і в деяких випадках використовуйте *словник*,щоб зробити це швидко)

3.12.2 Оператор `while`

 англійська: *The while Statement*

Оператор `while` дозволяє вам багаторазово виконувати блок рядків коду,поки умова істинна (англ."True"), або поки виконується деяка умова. Оператор `while` є прикладом так званого оператора *циклу*. Оператор `while` може мати необов'язковий блок `else`.

код python 'while_loop_ukr.py'

```
число = 23
гра_запуститься = True

while гра_запуститься == True:
    здогадка = int(input('Введіть ціле число : '))
```

```
    if здогадка == число:
        print('Вітаємо, ви вгадали.')
         # це призводить до зупинки циклу while
        гра_запуститься = False
    elif здогадка < число:
        print('Ні, задане число трохи вище, ніж здогадка.')
    else:
        print('Ні, задане число трохи нижче, ніж здогадка.')
else:
    print('Цикл while закінчено.')
    # Робіть тут усе, що хочете

print('Виконано')
```

Висновок:

```
$ python while_loop_ukr.py
Введіть ціле число: 50
Ні, задане число трохи нижче, ніж здогадка
Введіть ціле число: 22
Ні, задане число трохи вище, ніж здогадка
Введіть ціле число: 23
Вітаємо, ви вгадали.
Цикл while закінчено.
Виконано
```

Як це працює

У цій програмі ми все ще граємо у вгадування, але перевага полягає в тому, що користувачеві дозволено продовжувати вгадувати, доки він не вгадає правильно – немає потреби повторно запускати програму для кожного вгадування, як ми робили у попередньому прикладі. Це наочно демонструє використання оператора `while`.

Ми переміщуємо оператори `input` і `if` всередину циклу `while` і встановлюємо змінну `гра_запуститься` у значенні `True` перед циклом while. Спочатку ми перевіряємо, чи змінна `гра_запуститься`(гра_запуститься==True) має значення `True`, а потім продовжуємо виконувати відповідний *while-block*.

Після виконання цього блоку (*while-block*) знову перевіряється умова, яка в даному випадку є змінною `гра_запуститься`. Якщо вона істинна, ми знову виконуємо блок while, інакше ми продовжуємо виконувати необов'язковий блок else, а потім переходимо до наступного рядка коду.

Блок `else` виконується, коли умова циклу `while` стає хибною (англ.» `False`») - це може статися навіть при першій перевірці умови. Якщо у циклу `while` є додатковий блок else, він завжди виконується, якщо цикл не буде перерваний оператором break.

`True` і `False` називають булевим типом даних, і їх можна вважати еквівалентними значенням `1` і `0` відповідно.

Примітка для програмістів C/C++

Пам'ятайте, що цикл `while` може мати блок `else`.

– Від перекладача –

Після оператора `while` завжди йде вираз. Вираз — це щось, що може обчислити комп'ютер і завжди має результат `True` або `False`. (Наприклад, $3 > 5$ є `False`, але $3 == 3$ є `True`). У наведеному вище прикладі коду для змінної `гра_запуститься` було встановлено значення `True`.

```
гра_запуститься = True
```

Тепер Python має перевірити, чи значення *`гра_запуститься`* має значення `True`, щоб зрозуміти, чи потрібно виконувати блок while:

```
while гра_запуститься == True:
```

Частина між словами `while` і двокрапкою - є виразом, і Python може обчислити результат цього виразу. Результатом виразу є (на даний момент) `True`. Таким чином, буде виконано блок while.

Усередині блоку `while` є блок `if elif else`. Лише блок `if здогадка == число:` містить команду для зміни значення `гра_запуститься`:

```
гра_запуститься = False
```

Інші блоки (elif та else) не змінюють значення змінної `гра_запуститься`.

Після виконання всього блоку while програма Python повертається до початку блоку `while` і знову перевіряє вираз, щоб зрозуміти, чи потрібно блок `while` виконувати інший раз:

```
while гра_запуститься == True:
```

Якщо вираз має значення `False` (коли число було вгадано правильно та рядок `гра_запуститься` = `False` було виконано), тоді блок `while` НЕ буде виконано знову, і програма Python продовжує роботу в рядку коду після блоку `while` (у наведеному вище прикладі це рядок `else:`), за яким слідує `print("Цикл while закінчено")`

Блок `while` з блоком `else`

Блок `while` може мати додатковий блок `else` під собою, як у грі вгадування чисел вище.

Якщо вираз праворуч від команди `while` стає `False`, блок while не виконується, а замість нього виконується блок else. Якщо такого блоку else не існує, виконується наступний рядок коду після блоку while.

Однак існує особливий випадок: якщо виконується блок while і всередині блоку while виконується команда `break`, тоді блок else НЕ буде виконано. Команда `break` буде пояснена пізніше в цьому розділі.

– завершення доповнень від перекладача –

3.12.3 Цикл `for`

 англійська: *The `for` loop*

Оператор `for..in` — це ще один оператор циклу, який *ітерує* послідовність об'єктів, тобто проходить через кожен елемент у послідовності. Ми детально розглянемо про *послідовності* у наступних розділах. Зараз вам потрібно знати, що послідовність — це впорядкований набір елементів. Приклад (зберегти як `for.py`):

код python 'for_loop_ukr.py'

```
for i in range(1, 5):
    print(i)
else:
    print('Цикл for завершено')
```

Висновок:
```
$ python for_loop_ukr.py
1
2
3
4
Цикл for завершено
```

Як це працює

У цій програмі ми друкуємо *послідовність* чисел. Ми генеруємо цю послідовність чисел за допомогою вбудованої функції `range`.

Ми задаємо два числа, і `range` повертає послідовність чисел від першого числа до другого. Наприклад, `range(1,5)` дає послідовність `[1, 2, 3, 4]`. За замовчуванням `range` приймає кількість кроків, рівну 1. Якщо ми задамо також і третє число `range`, воно служитиме кроком. Наприклад, range (1,5,2) дає значення [1,3]. Пам'ятайте, інтервал простягається тільки до другого числа, тобто не включає його у себе.

Зауважте, що `range()` генерує послідовність чисел, але одне число за раз. Щоб побачити всю послідовність чисел відразу, напешіть `list(range())`, наприклад, `list(range(5))` призведе до "[0, 1, 2, 3, 4]". Списки (англ. "list") пояснюються в *главі про структури даних*.

Потім цикл `for` здійснює ітерацію з цього діапазону - *for i in range(1,5)* еквівалентно *for i in [1, 2, 3, 4]*, що схоже на привласнення змінної `i` за одним числом (або об'єктом) за раз, виконуючи блок команд для кожного значення `i`. У цьому випадку в блоці рядків коду ми просто друкуємо значення.

Пам'ятайте, що блок "else" необов'язковий. Якщо він присутній, він завжди виконується один раз після закінчення циклу for, якщо тільки не вказан оператор *break*.

Пам'ятайте також, що цикл `for..in` працює для будь-якої послідовності. У нашому випадку - це список чисел, згенерованих вбудованою функцією `range`, але загалом ми можемо використовувати будь-яку послідовність будь-яких типів об'єктів! Ми детально розглянемо цю ідею в наступних розділах.

Примітка для програмістів C/C++/Java/C#

Цикл `for` у Python радикально відрізняється від циклу `for` C/C++. Програмісти C# звернуть увагу на те, що цикл `for` у Python схожий на цикл `foreach` у C#. Java-програмістам це може нагадати конструкцію `for (int i : IntArray)` у Java 1.5.

У C/C++, якщо ви хочете написати `for (int i = 0; i < 5; i++)`, тоді у Python цьому відповідає вираз `for i in range(0,5)`. Як бачите, цикл for є простішим, виразнішим і менш схильним до помилок у Python.

3.12.4 Оператор break

 англійська: *The break Statement*

Команда `break` може бути використана лише всередині циклу (всередині блоку *for* чи блоку *while*). Що робить оператор `break`? Якщо Python читає команду break, Python миттєво припиняє виконання рядків коду у блоці команд цикла і переходить до першого командного рядка після циклу.

Важливе зауваження полягає в тому, що якщо цикли `for` або `while` *перервани* оператором break, відповідні їм блоки else виконуватися не будуть.

код python 'break_while_ukr.py'

```
while True:
    s = input('Введіть щось : ')
    if s == 'Вийти':
        break
    print('Довжина рядка становить :', len(s))
print('Виконано')
```

Висновок:

```
$ python break_while_ukr.py
Введіть щось : Програмування - це весело
Довжина рядка становить : 25
Введіть щось : Коли робота виконана
Довжина рядка становить : 20
Введіть щось : І якщо ви хочете повесилитися на роботі:
Довжина рядка становить : 40
Введіть щось : Використовуйте Python!
Довжина рядка становить : 22
Введіть щось : Вийти
Вийти
```

Як це працює

У цій програмі ми багаторазово зчитуємо введення користувача та друкуємо довжину кожного введення кожного разу. Ми надаємо спеціальну умову для зупинки програми, перевіряючи, чи є введені користувачем рядки коду `'Вийти'`. Для зупинення програми ми вводимо спеціальну умову, яка перевіряє, чи збігається введення користувача з рядком коду "вихід". Ми зупиняємо програму, *перериванням* циклу оператором break, і досягаємо кінця програми.

Довжина введеного рядка може бути знайдена за допомогою вбудованої функції `len`.

Пам'ятайте також, що оператор `break` може застосовуватись і в циклі `for`.

Поетичний Python Swaroop'a

Вхідні дані, які я використав тут, це міні-вірш, який я написав. Він називається Поетичний Python Swaroop'a:

```
Програмування - це весело.
Коли робота виконана,
і якщо ви хочете повесилитися на роботі:
    використовуйте Python!
```

Англійський варіант:

```
Programming is fun.
When the work is done,
if you wanna make your work also fun:
    use Python!
```

3.12.5 Оператор `continue`

 англійська: *The `continue` Statement*

Оператор `continue` використовується, щоб наказати Python пропустити решту операторів у поточному блоці циклу та *продовжити* (англ."continue") з наступної ітерації циклу.

код python 'continue_while_ukr.py'

```
while True:
    s = input('Введіть щось :')
    if s == 'вийти':
        break
    if len(s) < 3:
        print('Занадто мало')
        continue
    print('Введений рядок достатньої довжини')
    # Різні інші дії тут...
```

Висновок:

```
$ python continue_while_ukr.py
Введіть щось : a
Занадто мало
Введіть щось : 12
Занадто мало
Введіть щось : abc
Введений рядок достатньої довжини
Введіть щось : вийти
```

Як це працює

У цій програмі ми приймаємо введення від користувача, але обробляємо введений рядок, лише якщо він містить принаймні 3 символи. Отже, ми використовуємо вбудовану функцію `len`, щоб отримати довжину рядка, і якщо довжина менша за 3, ми пропускаємо решту операторів у блоці за допомогою оператора `continue`. В іншому випадку решта операторів у циклі виконується, роблячи будь-які маніпуляції, які нам потрібні.

Зауважте, що оператор continue також працює і з циклом for.

3.12.6 Резюме

Ми побачили, як використовувати три оператори потоку керування - `if`, `while` і `for` разом із відповідними операторами `break` і `continue`.Це одні з найбільш часто використовуваних конструкцій Python, тому оволодіти ними дуже важливо.

Далі ми побачимо, як створювати та використовувати функції.

3.13 Функції

Функції — є багаторазовими фрагментами програми. Вони дозволяють вам дати назву блоку рядків коду, щоб згодом запускати цей блок, використовуючи вказане ім'я будь-де у вашій програмі та будь-яку кількість разів. Це називається *викликом* функції. Ми вже використовували багато вбудованих функцій, таких як `len` і `range`.

Концепція функції є, ймовірно, *найважливішим* будівельним блоком будь-якого нетривіального програмного забезпечення (на будь-якій мові програмування), тому ми розглянемо різні аспекти функцій у цій главі.

Функції визначаються за допомогою ключового слова `def`. Для того, щоб створити функцію, потрібно розмістити ключове слово def перед ідентифікатором функції (її ім'ям), потім вказати пару дужок, всередині яких можуть міститися імена змінних, поставити в кінці рядка двокрапку, яка закінчує рядок. Далі йде блок рядків коду, які є частиною цієї функції. Приклад покаже, що це насправді дуже просто:

код python 'function1_ukr.py'

```
def скажи_привіт():
    # блок, що належить функції
    print('привіт, Світ!')
# Кінець функції

скажи_привіт()  # виклик функції
скажи_привіт()  # ще один виклик функції
```

Висновок:

```
$ python function1_ukr.py
привіт, Світ!
привіт, Світ!
```

Як це працює

Ми визначаємо функцію під назвою `скажи_привіт` використовуючи синтаксис, описаний вище. Ця функція не приймає параметрів і, отже, немає змінних, оголошених у дужках. Параметри функції – це деякі вхідні дані, які ми можемо передати функції, щоб отримати відповідний їм результат. Зверніть увагу, що ми можемо викликати ту саму функцію двічі, що означає, що нам не потрібно писати той самий код знову.

3.13.1 Параметри функції

Функції можуть приймати параметри, тобто деякі значення, що передаються функції,щоб вона щось *зробила* з ними. Ці параметри схожі на змінні, за винятком того, що значення цих змінних вказуються при виклику функції, та під час роботи функції їм вже надано їх значення.

Параметри вказуються в парі круглих дужок при визначенні функції, розділені символом коми. Коли ми викликаємо функцію, ми надаємо значення таким же чином. Зверніть увагу на термінологію: імена, наведені при визначенні функції, називаються *параметрами*, тоді як значення,які ви передаєте в функцію при її виклику, - називаються *аргументами*.

код python 'function_param_ukr.py'

```
def print_max(a, b):
    if a > b:
        print(a, 'є максимальним')
    elif a == b:
        print(a, 'дорівнює', b)
    else:
        print(b, 'є максимальним')
```

```
# пряма передача значень
print_max(3, 4)

x = 5
y = 7
# передача змінних як аргументи
print_max(x, y)
```

Висновок:

```
$ python function_param_ukr.py
4 є максимальним
7 є максимальним
```

Як це працює

Тут ми визначаємо функцію з назвою `print_max`, яка використовує два параметри з назвою `a` і `b`. Ми знаходимо більше число за допомогою простого блоку `if..else`, а потім друкуємо більше число.

Коли ми вперше викликаємо функцію `print_max`, ми безпосередньо передаємо числа як аргументи. У другому випадку ми викликаємо функцію зі змінними в якості аргументів. `print_max(x, y)` призначає значення аргумента `x` параметру `a`, а значення аргумента `y` - параметру `b`. Функція `print_max` працює однаково в обох випадках.

—- Від перекладача —-

код python 'function_param1_ukr.py'

```
def ноутбук(текст="привіт",кількість=2):
        print(текст*кількість)

ноутбук()               # виклик функції без аргументів (використовуються аргументи
↪за замовчуванням)
ноутбук("Toshiba")      #виклик функції лише з одним аргументом (аргумент за
↪замовчуванням
                        #використовується для другого параметра)
ноутбук("Lenovo",8)    # виклик функції з обома аргументами
```

Висновок:

```
$ function_param1_ukr.py

привіт привіт
Toshiba Toshiba
Lenovo Lenovo Lenovo Lenovo Lenovo Lenovo Lenovo Lenovo
```

де:

1. у рядку коду **def ноутбук (текст="привіт",кількість=2):** , *текст* та *кількість* є параметрами;
2. у рядкях коду **ноутбук("Toshiba")**, **ноутбук("Lenovo", 8)** , *Toshiba*, *Lenovo*, *8* є аргументами.

— завершення прикладу від перекладача —

3.13.2 Локальні змінні

При оголошенні змінних всередині визначення функції, вони жодним чином не пов'язані з іншими змінними з такими ж іменами, які використовуються поза функцією, тобто імена змінних є *локальними* для функції. Це називається *областю видимості* змінної. Область видимості всіх змінних обмежена блоком, де вони оголошені, починаючи з точки визначення імені.

код python function_local_ukr.py

```
x = 50


def func(x):
    print('x дорівнює', x)
    x = 2
    print('Заміна локального x на',x)


func(x)
print('x як і раніше', x)
```

Висновок:

```
$ python function_local_ukr.py
x дорівнює 50
Заміна локального x на 2
x як і раніше 50
```

Як це працює

Коли ми вперше друкуємо *значення*, наданого імені *x*, у першому рядку тіла функції (`print('x дорівнює', x)`), Python використовує значення параметра, оголошене в основному блоці над визначенням функції (x = 50).

Далі ми присвоюємо `x` значення 2. Лише в межах функції `x` тепер має значення 2. У глобальній області `x` все ще має значення `50`. Ім'я `x` є локальним для нашої функції. Отже, коли ми змінюємо значення `x` у функції, `x` визначене в основному блоці, залишається незмінним.

За допомогою останнього виклика функції `print` , ми відображаємо значення `x`, яке вказане в основному блоці, тим самим підтверджуючи, що воно не змінилося при локальному привласненні значення раніше викликаної функції.

3.13.3 `Global` оператор

 англійська: *The `global` statement*

Щоб призначити деяке значення змінній, визначеній на вищому рівні програми (тобто не в якійсь області видимості, як функції або класи), необхідно повідомити Python, що її ім'я не є локальним, а є `глобальним`. Ми робимо це за допомогою оператора `global`. Неможливо призначити значення змінній, визначеній поза функцією, без оператора `global`.

Можна використовувати вже існуючі значення змінних, визначених поза межами функції (за умови, що всередині функції не було оголошено змінної з таким же ім'ям). Однак це не заохочується, і цього слід уникати, оскільки читачеві програми стає незрозуміло, де знаходиться визначення цієї змінної. Використання оператора `global` дає зрозуміти, що змінна визначена в самому зовнішньому блоці.

код python function_global_ukr.py

```
x = 50


def func():
    global x

    print('x дорівнює', x)
    x = 2
    print('Глобальний x змінено на', x)


func()
print('Значення x є', x)
```

Висновок:

```
$ python function_global_ukr.py
x дорівнює 50
Глобальний x змінено на 2
Значення x є 2
```

Як це працює

Оператор `global` використовується для оголошення того, що `x` є глобальною змінною, отже, коли ми присвоюємо значення `x` всередині функції, це переінакшення позначиться на значенні змінної x в основному блоці програми.

Ви можете вказати більше однієї глобальної змінної за допомогою того самого оператора `global`, наприклад: `global x, y, z`.

3.13.4 Значення аргументів за замовчуванням

 англійська: *Default Argument Values*

Для деяких функцій ви можете зробити деякі параметри *необов'язковими* та використовувати значення за замовчуванням на випадок, якщо користувач не хоче надавати для них значення. Це робиться за допомогою значень аргументів за замовчуванням. Ви можете вказати значення аргументів за замовчуванням, додавши до назви параметра у визначенні функції оператор присвоєння (=), для подальших значень за замовчуванням.

Зауважте, що значення аргументу за замовчуванням має бути константою. Точніше, значення аргументу за замовчуванням має бути незмінним (англ."immutable")- це детально пояснюється в наступних розділах. Наразі просто запам'ятайте це.

код python function_default_ukr.py

```
def висловлювання(повідомлення, кількість=1):
    print(повідомлення * кількість)

висловлювання('Привіт')
висловлювання('Світ', 5)
```

Висновок:

```
$ python function_default_ukr.py
Привіт
СвітСвітСвітСвітСвітСвіт
```

3.14 Як це працює

Функція під назвою `висловлювання` використовується для друку рядка стільки разів, скільки вказано. Якщо ми не надаємо значення, то за замовчуванням рядок друкується лише один раз. Ми досягаємо цього, вказуючи значення аргументу за замовчуванням 1 для параметра `кількість`.

Під час першого виклика функції `висловлювання` ми вказуємо лише рядок ('Привіт'), і він друкує рядок один раз. Під час другого виклика функції `висловлювання` ми вказуємо і рядок ('Привіт'), і аргумент 5, показуючи, що ми хочемо *надрукувати* повідомлення рядка 5 разів.

Важливо

Значення за замовчуванням можуть мати лише параметри, що знаходяться в кінці списку параметрів. Таким чином, у списку параметрів функції, параметр зі значенням за замовчуванням не може передувати параметру без значення за замовчуванням.

Це пов'язано з тим, що значення призначаються параметрам за позицією. Наприклад, `def func(a, b=5)` є дійсним, але `def func(a=5, b)` є *недійсним*.

3.14.1 Аргументи ключових слів

 англійська: *Keyword Arguments*

Якщо у вас є функції з багатьма параметрами, і ви хочете вказати при виклику функції лише деякі з них, тоді ви можете надати значення таким параметрам, назвавши їх - це називається *аргументи ключового слова*. У цьому випадку для передачі аргументів функції використовується ім'я (ключ) замість позиції (як було досі).

Є дві переваги такого підходу: одна полягає в тому, що використовувати функцію легше, оскільки нам не потрібно турбуватися про порядок аргументів. По-друге, ми можемо надавати значення лише тим параметрам, яким ми хочемо, за умови, що інші параметри мають значення аргументів за замовчуванням.

код python function_keyword_ukr.py

```
def func(a, b=5, c=10):
    print('a дорівнює', a, ' b дорівнює', b, ' c дорівнює', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Висновок:

```
$ python function_keyword_ukr.py
a дорівнює 3, b дорівнює 7, c дорівнює 10
a дорівнює 25, b дорівнює 5, c дорівнює 24
a дорівнює 100, b дорівнює 5, c дорівнює 50
```

Як це працює

Функція з назвою `func` має один параметр без значення за замовчуванням, а потім два параметри зі значеннями за замовчуванням.

Під час першого виклику, `func(3, 7)`, параметр `a` отримує значення `3`, параметр `b` отримує значення `7`, а `c` отримує значення за умовчанням `10`.

При другому виклику `func(25, c=24)` змінна `a` отримує значення 25 через позицію аргументу. Тоді параметр `c` отримує значення `24` завдяки іменуванню, тобто ключовим аргументам. Змінна "b" отримує значення за замовчуванням, рівне "5".

У третьому випадку використання `func(c=50, a=100)` ми використовуємо ключові аргументи для всіх вказаних значень. Зверніть увагу, що ми вказуємо значення для параметра `c` перед значенням `a`, хоча `a` визначено перед `c` у визначенні функції.

3.14.2 Довільна кількість аргументів

 англійська: *VarArgs parameters or variable number of arguments*

Іноді може виникнути потреба визначити функцію, яка може приймати *будь-яку* кількість параметрів, тобто **змінну** кількість **аргументів**, цього можна досягти за допомогою зірочок (збережіть як `function_varargs.py`):

українська

код python function_varargs_ukr.py:

```
def підсумок(a=5, *номера, **телефонна_книга):
    print('a', a)

    #прохід по всіх елементах кортежу
    for один_елемент in номера:
        print('один_елемент', один_елемент)

    #прохід по всіх елементах словника
    for перша_частина, друга_частина in телефонна_книга.items():
        print(перша_частина,друга_частина)

підсумок(10,1,2,3,Джек=1123,Джон=2231,Інге=1560)
```

Висновок:

```
$ python function_varargs_ukr.py
a 10
один_елемент 1
один_елемент 2
один_елемент 3
Джек 1123
Джон 2231
Інге 1560
```

англійська

python code function_varargs_en.py:

```
def total(a=5, *numbers, **phonebook):
    print('a', a)

    #iterate through all the items in tuple
    for single_item in numbers:
        print('single_item', single_item)

    #iterate through all the items in dictionary
    for first_part, second_part in phonebook.items():
        print(first_part,second_part)

total(10,1,2,3,Jack=1123,John=2231,Inge=1560)
```

output:

```
$ python function_varargs_en.py
a 10
single_item 1
single_item 2
single_item 3
Jack 1123
John 2231
Inge 1560
```

Як це працює

Коли ми оголошуємо параметр із зірочкою, наприклад `*param`, тоді всі позиційні аргументи від цієї позиції і до кінця збираються в кортеж під назвою 'param'.

Подібним чином, коли ми оголошуємо параметр із подвійною зірочкою, наприклад `**param`, тоді всі аргументи ключового слова від цієї позиції і до кінця збираються як словник під назвою 'param'.

Ми досліджуватимемо кортежі та словники в *пізнішій главі*.

3.14.3 Оператор "return"

 англійська: *the* `return` *statement*

Оператор `return` використовується для *повернення* з функції, тобто для припинення її роботи та виходу з неї. За бажання ми також можемо *повернути значення* з функції.

код python function_return_ukr.py

```
def максимум(x, y):
    if x > y:
        return x
    elif x == y:
        return 'Числа рівні'
    else:
        return y

print(максимум(2, 3))
```

Висновок:

```
$ python function_return_ukr.py
3
```

Як це працює

Функція `максимум` повертає максимальний з двох параметрів, які в даному випадку передаються їй під час виклику. Вона використовує блок if... else для визначення найбільшого числа, а потім *повертає* це число.

Зауважте, що оператор `return` без значення еквівалентний виразу `return None`. `None` — це спеціальний тип даних у Python, який представляє ніщо. Наприклад, він використовується, щоб вказати, що змінна не має значення, якщо вона дорівнює «None».

Кожна функція неявно містить оператор `return None` в кінці, якщо ви не написали власний оператор `return`. Ви можете побачити це, запустивши `print(some_function())`, де функція `some_function` не використовує оператор `return` у явному вигляді, наприклад:

```
def some_function():
    pass
```

Оператор `pass` використовується в Python для позначення порожнього блоку команд.

Порада

Існує вбудована функція під назвою `max`, яка вже реалізує функцію 'знайти максимум', тому використовуйте цю вбудовану функцію, коли це можливо.

3.14.4 Рядки документації

 англійська: *DocStrings*

У Python є чудова функція під назвою *рядки документації*, яку зазвичай називають коротшою назвою *DocStrings*. DocStrings є важливим інструментом, який вам слід використовувати, оскільки він допомагає краще документувати програму та полегшує її розуміння. Дивовижно, але ми навіть можемо отримати рядок документації, скажімо, з функції, коли програма фактично запущена!

код python function_docstring_ukr.py

```
def print_max(x, y):
    '''Виводить більше із двох чисел.

    Обидва значення мають бути цілими числами.'''
    # конвертувати в цілі числа, якщо можливо
    x = int(x)
    y = int(y)

    if x > y:
        print(x, 'найбільше')
    else:
        print(y, 'найбільше')
```

```
print_max(3, 5)
print(print_max.__doc__)
```

Висновок:

```
$ python function_docstring_ukr.py
5 найбільше
Виводить максимальне із двох чисел.

    Ці два значення мають бути цілими числами.
```

Як це працює

Рядок у першому логічному рядку функції є *рядком документації* для цієї функції. Зверніть увагу, що DocStrings також застосовуються до *модулів* та *класів*, про які ми дізнаємося у відповідних розділах.

Загальноприйнятим стилем написання docstring є багаторядковий рядок, у якому: перший рядок починається з великої літери та закінчується крапкою, другий рядок — порожній, з третього рядка починається детальніше пояснення (за потреби). *Настійно рекомендовано* дотримуватися цього стилю для всіх ваших рядків документації зі всіма вашими нетривіальними функціями.

Ми можемо отримати доступ до рядка документації функції `print_max` за допомогою атрибута цієї функції (тобто імені, що належить їй) `__doc__` (зверніть увагу на *подвійне підкреслення*). Просто пам'ятайте, що Python розглядає *все* як об'єкт, включаючи функції. Ми дізнаємося більше про об'єкти в розділі про *класи*.

Якщо ви використовували функцію `help()` в Python, то ви вже бачили використання рядків документації! Ця функція просто зчитує атрибут `__doc__` відповідній функції та акуратно виводить його на екран.Ви можете перевірити її на розглянутій вище функції: просто включить `help(print_max)` у текст програми. Не забудьте натиснути клавішу `q`, щоб вийти з `довідки`.

Так само автоматизовані інструменти можуть отримувати документацію з вашої програми. Тому я *настійно рекомендую* використовувати рядки документів для будь-якої нетривіальної функції, яку ви пишете. Команда `pydoc`, яка постачається з пакетом Python, працює подібно до `help()`, використовуючи рядки документації.

Приклад для "автоматизованих інструментів" від перекладача:

Редактор коду «Visual Studio Code» може зрозуміти, що таке рядок документації функції (рядок 2), і відобразити його в маленькому полі над курсором, коли ім'я функції вводиться (рядок 8). Дивиться знімок екрана нижче:

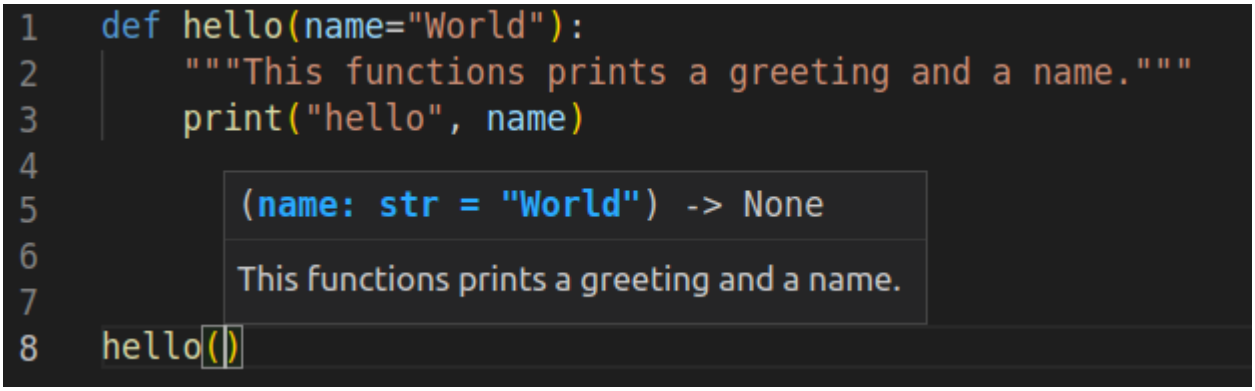


Fig. 5: Знімок екрана коду Visual Studio, що відображає рядок документації

3.14.5 Резюме

Ми розглянули доволі багато аспектів функцій, але зауважте, що ми ще не охопили всі їх аспекти. Однак ми вже розглянули більшість того, що ви будете використовувати щодо функцій Python щодня.

Далі ми побачимо, як використовувати та створювати модулі Python.

3.15 Модулі

 англійська: *Modules*

Ви вже бачили, як можна повторно використовувати код у своїй програмі, визначивши функції один раз. Що, якби ви хотіли повторно використати кілька функцій в інших програмах, які ви пишете? Як ви вже, напевно,здогадалися, відповідь — модулі.

Існують різні методи написання модулів, але найпростішим способом є створення файлу з розширенням `.py`, який містить функції та змінні.

Іншим методом є написання модулів тією мовою, якою був написаний сам інтерпретатор Python. Наприклад, ви можете писати модулі мовою програмування C ,які після компіляції можуть використовуватись стандартним інтерпретатором Python.

Модуль може бути *імпортований* в іншу програму, щоб програма використовувала функції з модуля. Таким чином ми також можемо використовувати стандартну бібліотеку Python. Спочатку ми розглянемо, як використовувати модулі стандартної бібліотеки.

Приклад (зберегти як `module_using_sys.py`):

код python module_using_sys_ukr.py

```
import sys

print('Аргументи командного рядка:')
for i in sys.argv:
    print(i)

print('\n\n Змінна  середовища PYTHONPATH містить', sys.path, '\n')
```

Висновок:

```
$ python module_using_sys_ukr.py ми є аргументами
Аргументи командного рядка:
module_using_sys.py
ми
є
аргументи

змінна середовища "Pythonpath" вашого комп'ютера: Змінна середовища PYTHONPATH
містить (тут ви побачите ВАШ Python path)
```

Як це працює

Спочатку ми *імпортуємо* модуль `sys` за допомогою команди `import`. По суті, це означає, що ми повідомляємо Python про використання цього модуля. Модуль `sys` містить функції, які стосуються інтерпретатора Python та його середовища, тобто системи (system).

Коли Python виконує команду `import sys`, він шукає модуль `sys`. У цьому випадку це один із вбудованих модулів, і, отже, Python знає, де його знайти.

Якби це не був скомпільований модуль, тобто модуль, написаний мовою Python,тоді інтерпретатор Python шукав би його у папках , указаних у змінній `sys.path`. Якщо модуль знайдено, команди в тілі цього модуля виконуються, і модуль стає *доступним* для використання. Зауважте, що ініціалізація (ряд дій, які проводяться при початковому завантаженні) виконується при *першому* імпорті модуля.

Доступ до змінної `argv` в модулі `sys` здійснюється за допомогою крапкової нотації (крапки), тобто `sys.argv`. Це чітко вказує на те, що це ім'я є частиною модуля `sys`. Ще одна перевага цього підходу полягає в тому, що ім'я не суперечить жодній змінній `argv`, яка використовується у вашій програмі.

Змінна `sys.argv` — це *список* рядків (списки докладно пояснюються в *пізнішій главі*). Зокрема, `sys.argv` містить список *аргументів командного рядка* (англ."*command line arguments*"),тобто аргументів, переданих програмі з командного рядка.

Якщо ви використовуєте середовище розробки(англ.IDE, "Integrated Development Environment") для написання та запуску цих програм, пошукайте десь у її меню можливість передавати параметри командного рядка.

У нашому прикладі, коли ми виконуємо `python module_using_sys.py ми є аргументами`, ми запускаємо модуль `module_using_sys.py` командою `python`,а все інше, що слідує далі, - є аргументами, які передаються програмі. Python зберігає аргументи командного рядка у змінній `sys.argv` для подальшого використання.

Пам'ятайте, ім'я запущеного сценарія (програму на інтерпретованій мові програмування також називають сценарієм або скриптом) завжди є першим аргументом у списку `sys.argv`. Отже, у цьому випадку ми матимемо `'module_using_sys.py'` як `sys.argv[0]`, `'ми'` як `sys.argv[1]`, `'є'` як `sys.argv[2]` та `'аргументами'` як `sys.argv[3]`. Зверніть увагу, що Python починає відлік з 0, а не з 1.

`sys.path`- це список шляхів (англ.» list of paths»), з яких імпортуються модулі. Зверніть увагу, що перший рядок у `sys.path` порожній - цей порожній рядок вказує на те, що поточна папка також є частиною `sys.path`. Це означає, що ви можете безпосередньо імпортувати модулі, розташовані в поточній папці. Якщо цього не буде, то вам доведеться розмістити свій модуль в одній із папок, указаних у `sys.path`.

Зверніть увагу, що поточна папка - це папка, з якої запускається програма. Запустіть `import os; print(os.getcwd())`, щоб дізнатися поточну папку вашої програми.

3.15.1 Байт-компільовані.pyc файли

 англійська: *Byte-compiled.pyc files*

Імпортування модуля є відносно дорогою справою, тому Python робить деякі хитрощі, щоб зробити це швидше. Одним із способів -створити *байт-скомпільовані* файли (або байткод) з розширенням `.pyc`, яке є проміжною формою, у яку Python перетворює програму (пам'ятаєте *вступний розділ* про те, як працює Python?). Цей файл `.pyc` корисний при імпорті модуля наступного разу в іншу програму - це буде набагато швидше, оскільки частину обробки, необхідної для імпортування модуля, уже виконано. Крім того, ці скомпільовані файли не залежать від платформи.

ПРИМІТКА. Зазвичай, файли `.pyc` створюються в тій ж самій папці, де розташовані і відповідні файли `.py`. Якщо Python не може отримати доступ до запису файлів у цій папці, файли `.pyc` *не* створюватимуться.

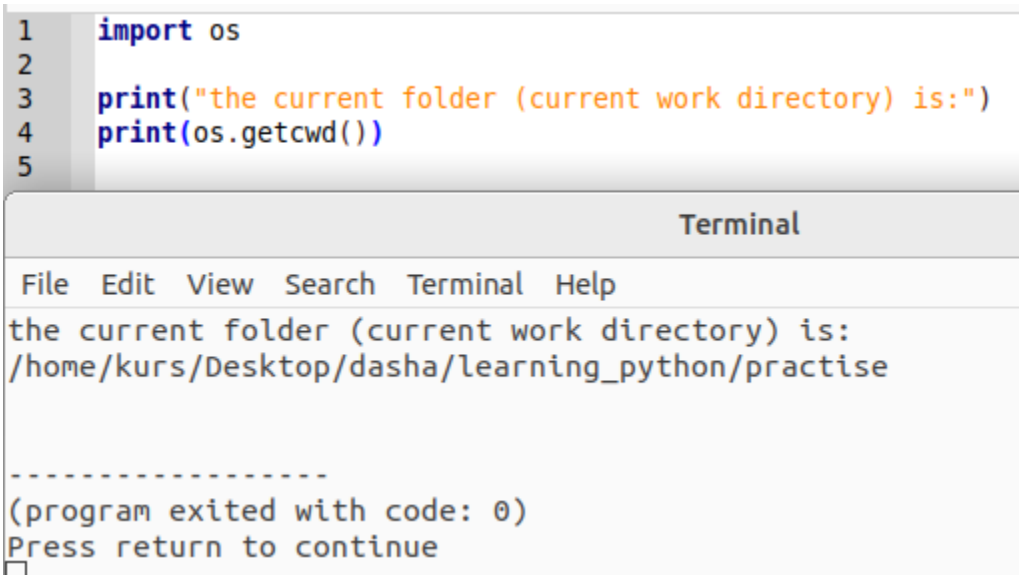


Fig. 6: Screenshot: Get current work directory

3.15.2 Оператор from...import

англійська: *The from...import statement*

Якщо ви хочете безпосередньо імпортувати змінну `argv` у свою програму (і не вводити для неї `sys.` при зверненні до неї, щоразу), ви можете використати оператор `from sys import argv`.

Примітка

загалом *уникайте* використання оператора `from..import`, використовуйте замість нього оператор `import`. Це пояснюється тим, що ваша програма уникатиме зіткнень імен і буде більш читабельною.

код python import_from1_ukr.py

```
# не така хороша версія (менше набирати на клавіатурі, але великий ризик заплутатися)
from math import sqrt
print("Квадратний корінь із 16 є", sqrt(16))
```

Висновок:

```
Квадратний корінь із 16 є 4
```

код python import_from2_ukr.py

```
# хороша версія (більше набирати на клавіатурі, але безпечніше)
import math
print("Квадратний корінь із 16 є", math.sqrt(16))
```

Висновок:

```
Квадратний корінь із 16 є 4
```

3.15.3 Ім'я модуля - __name__

 англійська: *A module's* `__name__`

Кожен модуль має ім'я. Існують команди Python, за допомогою яких можно дізнатися назву фактичного модуля Python. Це зручно, щоб з'ясувати, чи працює модуль окремо чи імпортується. Як згадувалося раніше, коли модуль імпортується вперше, код, який він містить, виконується. Ми можемо використовувати це, щоб змусити модуль поводитись по-різному, залежно від того, чи використовується він сам по собі, чи імпортується з іншої програми. Цього можна досягти за допомогою атрибута `__name__` модуля.

код python module_using_name_ukr.py

```
if __name__ == '__main__':
    print('Ця програма виконується самостійно')
else:
    print('Мене імпортують з іншого модуля')
```

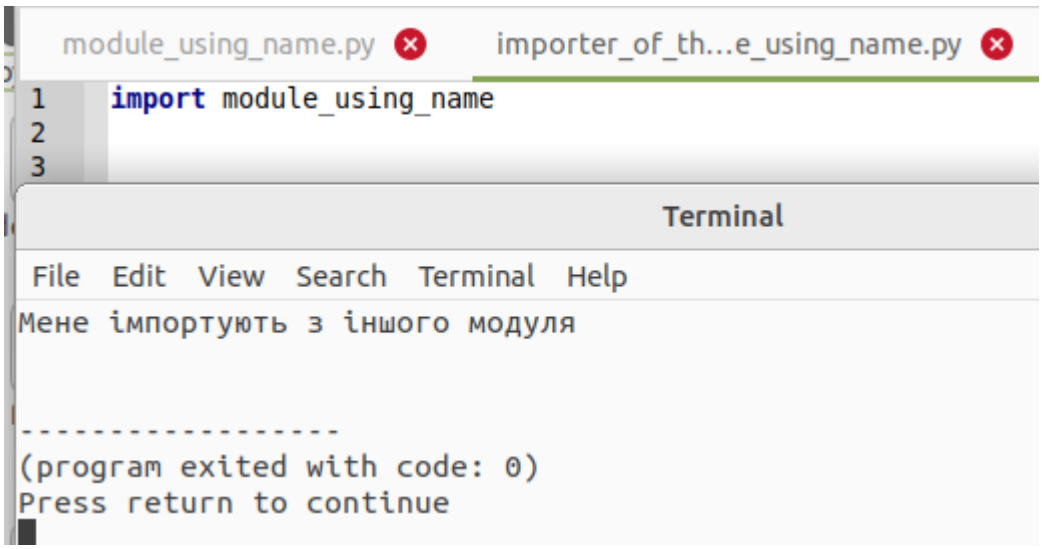
Висновок:

```
$ python module_using_name_ukr.py
Ця програма виконується самостійно
```

Якщо невелику програму вище запустити безпосередньо, ви побачите результат, як описано вище. Однак, якщо маленьку програму вище імпортовано з іншої програми, ви побачите наступний результат:

```
import module_using_name_ukr
```

Висновок:



Як це працює

Кожен модуль Python має своє ім´я `__name__`. Якщо воно дорівнює `'__main__'`, це означає, що модуль запущений самостійно користувачем, і ми можемо вжити відповідних дій.

3.15.4 Створення власних модулів

Створювати власні модулі легко, ви робити це весь час! Це тому, що кожна програма Python також є модулем. Вам просто потрібно переконатися, що вона має розширення `.py`. Наступний приклад повинен прояснити це.

код python my_module_ukr.py

```
def скажи_привіт():
    print('Привіт, говорить мій модуль.')

__version__ = '0.1'
```

Вище був зразок *модуля*. Як бачите, у ній немає нічого особливого порівняно з нашою звичайною програмою на Python. Далі ми побачимо, як використовувати цей модуль в інших наших програмах на Python.

Пам'ятайте, що модуль слід розмістити або в тій самій папці, що й програма, з якої ми його імпортуємо, або в одній з папок, указаних у `sys.path`.

код python mymodule_demo_ukr.py

```
import mymodule

mymodule.скажи_привіт()
print('Версія', mymodule.__version__)
```

Висновок

```
$ python mymodule_demo_ukr.py
Привіт, говорить мій модуль.
Версія 0.1
```

Як це працює

Зверніть увагу, що для доступу до елементів модуля ми використовуємо ту саму нотацію з крапкою. Python добре використовує ту саму нотацію, щоб надати їй характерного відчуття «Pythonic», щоб нам не довелося вивчати нові способи виконання завдань.

Ось версія, яка використовує синтаксис `from..import` (зберегти як`mymodule_demo2_ukr.py`):

код python mymodule_demo2_ukr.py

```
from mymodule import скажи_привіт, __version__

скажи_привіт()
print('Версія', __version__)
```

Висновок `mymodule_demo2_ukr.py` такий самий, як і висновок `mymodule_demo_ukr.py`.

Зауважте, що якби ім'я `__version__` вже було оголошено в модулі, який імпортує mymodule, виникне зіткнення. Це також ймовірно, тому що це звичайна практика для кожного модуля оголошувати но-

мер своєї версії за допомогою цієї назви. Тому, завжди рекомендується віддавати перевагу оператору `import`, навіть якщо це може зробити вашу програму трохи довшою.

Ви також можете використовувати:

```
from mymodule import *
```

Це імпортує всі загальнодоступні назви, такі як `скажи_привіт`, але не імпортує `__version__`, оскільки воно починається з подвійного підкреслення.

Попередження

пам'ятайте, що вам слід уникати використання import-star, тобто `from mymodule import *`.

Дзен of Python

Один із керівних принципів Python полягає в тому, що «Явне краще за неявне». Запустіть `import this` в Python, щоб дізнатися більше.

3.15.5 Функція `dir`

 англійська: *The `dir` function*

Вбудована функція `dir()` повертає список імен, визначених об'єктом. Якщо об'єкт є модулем, цей список включає функції, класи та змінні, визначені в цьому модулі.

Ця функція може приймати аргументи. Якщо аргументом є назва модуля, функція повертає список імен цього зазначеного модуля. Якщо аргументу немає, функція повертає список імен з поточного модуля.

приклад:

```
$ python
>>> import sys

# отримаємо список атрибутів модуля
>>> dir(sys)
['__displayhook__', '__doc__',
'argv', 'builtin_module_names',
'version', 'version_info']
# тут показано лише кілька записів

# отримати список атрибутів для поточного модуля
>>> dir()
['__builtins__', '__doc__',
'__name__', '__package__', 'sys']

# створити нову змінну 'a'
>>> a = 5

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys', 'a']
```

(continues on next page)

(continued from previous page)

```
# видалити/видалити ім'я
>>> del a

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
```

Як це працює

По-перше, ми бачимо використання `dir` к імпортованому модулі `sys`. Ми бачимо величезний список атрибутів, які він містить.

Далі ми викликаємо функцію `dir` без передачі їй параметрів. За замовчуванням вона повертає список атрибутів для поточного модуля. Зверніть увагу, що список імпортованих модулів також є частиною цього списку.

Щоб спостерігати за дією `dir`, ми визначаємо нову змінну `a` і призначаємо їй значення, а потім знову викликаємо `dir`. Бачимо, що в списку з такою ж назвою є додаткове значення. Ми видаляємо змінну/атрибут поточного модуля за допомогою оператора del, та зміни знову відобразяться на виведенні функції `dir`.

Примітка щодо `del`: цей оператор використовується для *видалення* змінної/назви, і після його виконання, у нашому випадку `del a`, ви більше не можете отримати доступ до змінної `a` - ніби вона раніше взагалі ніколи не існувала.

Зверніть увагу, що функція `dir()` працює з *будь-яким* об'єктом. Наприклад, запустіть `dir(str)` для атрибутів класу `str` (рядок).

Існує також функція `vars()`, яка потенційно може надати вам атрибути та їхні значення, але вона не працюватиме для всіх випадків.

3.15.6 Пакети

 англійська: *Packages*

На даний момент ви, мабуть, почали дотримуватись ієрархії організації ваших програм. Змінні зазвичай знаходяться всередині функцій. Функції та глобальні змінні зазвичай знаходяться всередині модулів. Що, якби ви хотіли організувати модулі? Ось тут і з'являються пакети.

Пакети — це просто папки модулів зі спеціальним файлом `__init__.py`, який вказує Python, що ця папка особлива, оскільки містить модулі Python.

Припустімо, ви хочете створити пакет під назвою «світ» із підпакетами «азія», «африка» тощо, а ці підпакети, у свою чергу, містять такі модулі, як «індія», «мадагаскар» тощо.

Ось як ви структуруєте папки:

```
- <деяка папка, наявна в sys.path>/
    - світ/
        - __init__.py
        - азія/
            - __init__.py
            - індія/
                - __init__.py
                - foo.py
        - африка/
```

(continues on next page)

(continued from previous page)

```
            - __init__.py
            - мадагаскар/
                - __init__.py
                - bar.py
```

Пакети – це просто зручність для ієрархічної організації модулів. Ви побачите багато прикладів цього в *стандартній бібліотеці*.

3.15.7 Резюме

Подібно до того, як функції є багаторазовими частинами програм, модулі є багаторазовими програмами. Пакети — це ще одна ієрархія для організації модулів. Стандартна бібліотека Python є прикладом такого набору пакетів і модулів.

Ми побачили, як використовувати ці модулі та створювати власні модулі.

Далі ми дізнаємося про деякі цікаві концепції, які називаються структурами даних.

3.15.8 – Доповнення від перекладача –

3.15.9 Модулі

код python dicethrow_ukr.py

```
import random

print("Імітація кидання двох кубиків:")
a = random.randint(1,6)
b = random.randint(1,6)
print("Перший кидок:", a)
print("Другий кидок:", b)
print("загальний бал:", a+b)
```

Висновок (ваш висновок може бути іншим!):

```
Імітація кидання двох кубиків:
Перший кидок: 5
Другий кидок: 3
Загальний бал: 8
```

Як це працює

По-перше, наведена вище програма Python *імпортує* модуль “random” зі стандартної бібліотеки Python. Стандартна бібліотека Python — це величезна колекція модулів Python, яка автоматично встановлюється, якщо ви інсталюєте Python на своєму комп’ютері. Перегляньте https://docs.python.org/3/library/index.html для повного списку модулів.

Кожен із цих модулів стандартної бібліотеки Python має декілька функцій.

Щоб скористатися однією з цих функцій, необхідно використовувати “крапкову нотацію”: написати ім’я модуля, потім крапку, а потім назву функції.

У наведеному вище прикладі функція *randint* модуля *random* викликається за допомогою:

```
random.randint(1,6)
```

Два параметри в дужках (1,6) вказують на те, що функція *randint* має створювати випадкове ціле число від 1 до 6 (подібно до кидання шестигранного кубика).

У прикладах від Swaroop використовуються поняття "PythonPath" і Змінні середовища (англ."Enviroment variables")

Оскільки не всі знають ці концепції, ось коротке пояснення (може бути корисним наступний сайт https://cpto.dp.ua/public_html/posibnyky/OSWin10/03/3_2.htm):

- У комп'ютерній термінології *каталог*, *директорія*(англ."directory") чи *тека*, *папка* (англ."folder") - це фізичне місце, де можна зберігати файли на комп'ютері. Такі файли, як програми на Python або сам інтерпретатор Python.
- *Підпапка* (англ."subfolder")- це папка всередині іншої папки.
- *Головна* або *кореневa* папка, каталог (англ."англ.root directory","root folder")- це перша папка в операційній системі. У Linux і MacOS це "/" або "/home/username". У Windows це зазвичай «C:»
- *Поточна папка* (англ."current folder") - це папка, в якій на даний момент працює комп'ютер (для збереження файлів)
- Шлях файлу (англ. "Path") - це рядок із іменами всіх папок (а також підпапок і підпідпапок...), починаючи з кореневої папки до поточної папки. Зауважте, що в WINDOWS назви папок відокремлюються зворотною скісною рискою(бекслеш), як-от "C:\carl\documents", але в Linux і MacOS назви папок відокремлюються скісною рискою(слеш): "/home/carl/documents".
- Змінна середовища (англ."enviroment variable") - це змінна, яку десь зберігає операційна система (Windows, Linux або MacOS), щоб вона могла шукати, де насправді встановлено різні програми. Наприклад, якщо ви запустите Microsoft Word із папки десь на вашому комп'ютері, комп'ютер дізнається, де встановлено «word.exe», і запустить його. Якщо бути більш точним, операційна система записує невеликий текстовий файл з одним рядком для кожної змінної середовища. Наприклад, один рядок скаже йому, де встановлено Microsoft Word, інший, де встановлено ваш веб-браузер, інший скаже, де встановлено Python. Усі ці записи називаються змінними середовища.
- PythonPath - це назва однієї спеціальної змінної середовища, яка повідомляє операційній системі (Windows, Linux або MacOS), де встановлено Python і де Python має шукати модулі. Насправді це список шляхів(a list of paths.).

3.15.10 Байт-компільовані.pyc файли

 англійська: *Byte-compiled.pyc files*

Якщо ви пишете програму на Python,використовуючи оператор `import`, як наприклад:

```
import turtle
```

тоді під час запуску програми відбуваються наступні речі.

Python намагається знайти модуль turtle.py. Python спочатку шукає в папці вашої програми, якщо Python не може знайти там модуль, він шукає в папці, де встановлено всі модулі Python (це називається шляхом python- Python path).

Потім всі функції цього модуля імпортуються, тобто ваша програма може їх використовувати. Цей процес імпорту займає відносно багато часу, тому Python робить маленьку хитрість: він намагається зберегти «скомпільовану» версію модуля з розширенням «.pyc» у тому самому місці, де й модуль.

Файл .pyc набагато швидше імпортувати для Python.

Наступного разу, коли одна з ваших програм намагатиметься імпортувати той самий модуль ("import"), Python не потрібно буде імпортувати відносно повільний модуль turtle.py, а натомість зможе використати набагато швидший скомпільований модуль turtle.pyc.

3.16 Функція dir

 англійська: *The `dir` function*

```
>>> a = 3.1255
>>> dir(a)
['__abs__', '__add__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__'
, '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__fo
rmat__', '__ge__', '__getattribute__', '__getformat__', '__getnewargs__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__int__', '__le__', '__lt__', '__mod__
', '__mul__', '__ne__', '__neg__', '__new__', '__pos__', '__pow__', '__radd__', '__r
divmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rmod__', '
__rmul__', '__round__', '__rpow__', '__rsub__', '__rtruediv__', '__setattr__', '__se
tformat__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '
__trunc__', 'as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'imag', 'is_integer',
'real']
>>> a.is_integer()
False
>>> a.hex()
'0x1.9010624dd2f1bp+1'
```

the value of the variable a is a number

The dir() functions accepts **any** object:
(numbers, strings and even modules)

Screenshot of dir function with a number

```
>>> b="Daria Jens"
>>> dir(b)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_
_eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs
__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__'
, '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__'
, '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'e
ncode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isal
num', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', '
isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lo
wer', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'repla
ce', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'spl
itlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfi
ll']
>>> b.swapcase()
'dARIA jENS'
>>> b.capitalize()
'Daria jens'
>>> b.__len__()
10
>>> b.upper()
'DARIA JENS'
```

The value of the variable b is a string

Screenshot of dir function with a string

```
>>> import sys
>>> dir(sys)
    ['__breakpointhook__', '__displayhook__', '__doc__', '__excepthook__', '__interactiv
    ehook__', '__loader__', '__name__', '__package__', '__spec__', '__stderr__', '__stdi
    n__', '__stdout__', '__unraisablehook__', '_base_executable', '_clear_type_cache', '
    _current_exceptions', '_current_frames', '_deactivate_opcache', '_debugmallocstats',
    '_framework', '_getframe', '_git', '_home', '_xoptions', 'abiflags', 'addaudithook',
    'api_version', 'argv', 'audit', 'base_exec_prefix', 'base_prefix', 'breakpointhook',
    'builtin_module_names', 'byteorder', 'call_tracing', 'copyright', 'displayhook', 'do
    nt_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix', 'executable', 'exit', '
    flags', 'float_info', 'float_repr_style', 'get_asyncgen_hooks', 'get_coroutine_origi
    n_tracking_depth', 'get_int_max_str_digits', 'getallocatedblocks', 'getdefaultencodi
    ng', 'getdlopenflags', 'getfilesystemencodeerrors', 'getfilesystemencoding', 'getpro
    file', 'getrecursionlimit', 'getrefcount', 'getsizeof', 'getswitchinterval', 'gettra
    ce', 'hash_info', 'hexversion', 'implementation', 'int_info', 'intern', 'is_finalizi
    ng', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'orig_argv', 'path', 'path_hoo
    ks', 'path_importer_cache', 'platform', 'platlibdir', 'prefix', 'pycache_prefix', 's
    et_asyncgen_hooks', 'set_coroutine_origin_tracking_depth', 'set_int_max_str_digits',
    'setdlopenflags', 'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace'
    , 'stderr', 'stdin', 'stdlib_module_names', 'stdout', 'thread_info', 'unraisablehook
    ', 'version', 'version_info', 'warnoptions']
```

Screenshot of using IDLE and displaying all the functionality of the sys module using dir().

— завершення доповнень від перекладача —

3.17 Структури даних

 англійська: *Data Structures*

Структури даних — це, по суті, і є *структури*, які можуть зберігати деякі *дані* разом. Іншими словами, вони використовуються для зберігання даних.

У Python є чотири вбудовані структури даних: *список (англ.“list”), кортеж (англ.“tuple”), словник (англ.“dictionary”) та множина (англ.“set”)*. Ми побачимо, як використовувати кожен із них і як вони полегшують нам життя.

3.17.1 Список

 англійська: *List*

Список — це структура даних, яка містить упорядкований набір елементів, тобто зберігає *послідовність* елементів у списку (англ.“*a sequence* of items in a list”). Це легко уявити, якщо згадати список покупок, в якому перераховується, що потрібно купити, з тим винятком, що у списку покупок кожен елемент зазвичай розміщується в окремому рядку, тоді як у Python вони розділяються комами.

Список елементів має бути укладений у квадратні дужки, щоб Python зрозумів, що ви вказуєте список. Створивши список, ви можете додавати, видаляти або шукати елементи у списку. Оскільки ми можемо додавати та видаляти елементи, ми говоримо, що список є *змінним* (англ.“*mutable*”) типом даних, тобто цей тип можна змінювати.

3.17.2 Короткий вступ до об’єктів і класів

 англійська: *Quick Introduction To Objects And Classes*

Хоч я і намагався досі відтягнути обговорення об’єктів і класів, на даному етапі все ж таки необхідне деяке пояснення, щоб ви краще зрозуміли ідею списків. Ми детально розглянемо цю тему у *пізнішому розділі*.

Список є прикладом використання об'єктів і класів. Коли ми присвоюємо деякій змінній `i` значення, скажімо, ціле число `5`, ви можете сприймати це як створення *об'єкта* (тобто екземпляра) `i` *класу* (тобто типу) `int`. Насправді, ви можете прочитати `help(int)`, щоб зрозуміти це краще.

Клас також може мати *методи*, тобто функції, визначені для використання лише стосовно цього класу. Ви можете використовувати ці частини функціональності, лише якщо у вас є об'єкт цього класу. Наприклад, Python надає метод `append` для класу `список`, який дозволяє вам додавати елемент у кінець списку. Наприклад, `мій_список.append('якийсь елемент')` додасть цей рядок до списку `мій_список`. Зверніть увагу на використання нотації з крапками для доступу до методів об'єктів.

Клас також може мати *поля*, які є нічим іншим, як змінними, визначеними для використання лише стосовно цього класу. Ви можете використовувати ці змінні/імена, лише якщо у вас є об'єкт цього класу. Поля також доступні за допомогою нотації з крапками, наприклад, `мій_список.field`.

код python ds_using_list_ukr.py

```
# Це мій список покупок
список_покупок = ['яблуко', 'манго', 'морква', 'банан']

print('Я маю', len(список_покупок), 'товари для покупок.')

print('Товари для покупок:', end=' ')
for елементи in список_покупок:
    print(елементи, end=' ')

print('\nЯ також маю купити рис.')
список_покупок.append('рис')
print('Мій список покупок зараз', список_покупок)

print('Я відсортую свій список зараз')
список_покупок.sort()
print('Сортований список покупок є', список_покупок)

print('Перший товар, який я куплю, це', список_покупок[0])
старий_елемент = список_покупок[0]
del список_покупок[0]
print('Я купив', старий_елемент )
print('Мій список покупок зараз', список_покупок)
```

Висновок:

```
Я маю 4 товари для покупок.
Товари для покупок: яблуко манго морква банан
Я також маю купити рис.
Мій список покупок зараз ['яблуко', 'манго', 'морква', 'банан', 'рис']
Я відсортую свій список зараз
Сортований список покупок є ['банан', 'манго', 'морква', 'рис', 'яблуко']
Перший товар, який я куплю, це банан
Я купив банан
Мій список покупок зараз ['манго', 'морква', 'рис', 'яблуко']
```

Як це працює

Змінна `список_покупок` - це список покупок для того, хто збирається на ринок. У `список_покупок` ми зберігаємо лише рядки назв предметів для покупки, але ви можете додавати *будь-які об'єкти* до

списку, включаючи номери та навіть інші списки.

Ми також використали цикл for... in для ітерації по елементах списку. Наразі ви вже зрозуміли, що список також є послідовністю. Особливості послідовностей буде обговорено у *пізнішому розділі*.

Зверніть увагу на використання параметра `end` у виклику функції `print`, який вказує, що ми хочемо закінчити рядок у висновку пробілом, замість звичайного розриву рядка.

Далі ми додаємо елемент до списку за допомогою методу `append`- методу об'єкта списку, як вже обговорювалося раніше. Потім ми перевіряємо, чи елемент справді додано до списку, друкуючи вміст списку за допомогою передачі цього списку функції`print`, яка старанно друкує його.

Потім ми сортуємо список за допомогою методу `sort`об'єкта списку. Важливо розуміти, що цей метод впливає на сам список і не повертає змінений список - це відрізняється від того, як працюють рядки. Це те, що ми маємо на увазі, кажучи, що списки *змінні* (англ."mutable"), а рядки *незмінні* (англ."immutable").

Потім, коли ми закінчимо купувати товар на ринку, ми хочемо видалити його зі списку. Ми досягаємо цього за допомогою оператора del. Тут ми згадуємо, який елемент зі списку ми хочемо видалити, і оператор del видаляє його зі списку за нас. Ми вказуємо, що ми хочемо видалити перший елемент зі списку, і тому ми використовуємо `del список_покупок[0]` (пам'ятайте, що Python починає відлік з 0).

Якщо ви хочете дізнатися про всі методи, визначені об'єктом списку, подробиці дивіться у `help(list)`.

3.17.3 Кортеж

 англійська: *Tuple*

Кортежі служать для зберігання кількох об'єктів разом. Вважайте їх подібними до списків, але без широкої функціональності, яку надає вам клас списку. Однією з головних особливостей кортежів є те, що вони *незмінні* (англ."*immutable*"),як і рядки, тобто ви не можете змінювати кортежі.

Кортежі позначаються визначенням елементів, розділених комами; за бажанням їх можна ще укласти в круглі дужки.

Кортежі зазвичай використовуються у випадках, коли оператор або визначена користувачем функція може безпечно припустити, що набір значень (тобто використаний кортеж значень) не зміниться.

код python ds_using_tuple_ukr.py

```
# Я рекомендую завжди використовувати круглі дужки
# позначати початок та кінець кортежу
# незважаючи на те, що дужки необов'язкові.
# Явне краще за неявне.
зоопарк  = ('пітон', 'слон', 'пінгвін')
print('Кількість тварин у зоопарку становить', len(зоопарк))

новий_зоопарк = 'мавпа', 'верблюд', зоопарк # дужки не потрібні, але це гарна ідея
print('Кількість клітин у новому зоопарку становить', len(новий_зоопарк))
print('Усі тварини в новому зоопарку є', новий_зоопарк)
print('Тварини, привезені зі старого зоопарку є', новий_зоопарк[2])
print('Остання тварина, привезена зі старого зоопарку', новий_зоопарко[2][2])
print('Кількість тварин у новому зоопарку становить',
      len(новий_зоопарк)-1+len(новий_зоопарк[2]))
```

Висновок:

```
$ python ds_using_tuple_ukr.py
Кількість тварин у зоопарку становить 3
Кількість клітин у новому зоопарку становить 3
Усі тварини в новому зоопарку є ('мавпа', 'верблюд', ('пітон', 'слон', 'пінгвін'))
Тварини, привезені зі старого зоопарку є ('пітон', 'слон', 'пінгвін')
Остання тварина, привезена зі старого зоопарку пінгвін
Кількість тварин у новому зоопарку становить 5
```

Як це працює

Змінна `зоопарк` означає кортеж елементів. Ми бачимо, що функцію `len` можна використовувати для отримання довжини кортежу. Це також вказує на те, що кортеж також є *послідовністю*.

Зараз ми переводимо цих тварин у новий зоопарк, оскільки старий зоопарк закривається. Таким чином, кортеж `новий_зоопарк` містить деяких тварин, які вже там разом із тваринами, привезеними зі старого зоопарку. Повертаючись до реальності, зауважте, що кортеж у кортежі не втрачає своєї ідентичності.

Ми можемо отримати доступ до елементів у кортежі, вказавши позицію елемента в парі квадратних дужок, як ми робили для списків. Це називається оператором *індексування* (англ.“*indexing*”). Ми отримуємо доступ до третього елемента в `новий_зоопарк`, вказуючи `новий_зоопарк[2]`, і отримуємо доступ до третього елемента в третьому елементі в кортежі `новий_зоопарк`, вказуючи `новий_зоопарк[2][2]`. Це досить просто, якщо ви зрозуміли ідіому.

Кортеж із 0 або 1 елементом

Порожній кортеж створюється пустою парою круглих дужок, наприклад `myempty = ()`. Однак кортеж з одним елементом не такий простий. Його потрібно вказати за допомогою коми після першого (і єдиного) елемента, щоб Python міг розрізнити кортеж і пару круглих дужок, які оточують об’єкт у виразі. Таким чином, щоб отримати кортеж, що містить елемент 2, вам потрібно буде вказати `singleton = (2 , )`.

Примітка для програмістів Perl

Список у списку не втрачає своєї ідентичності, тобто списки не розгортаються, як у Perl. Те саме стосується кортежу в кортежі, або кортежу в списку, або списку в кортежі тощо. Що стосується Python, це просто об’єкти, які зберігаються за допомогою іншого об’єкта, от і все.

3.17.4 Словник

 англійська: *Dictionary*

Словник схожий на адресну книгу, де ви можете знайти адресу або контактні дані особи, знаючи лише її/її ім’я; тобто ми пов’язуємо *ключі* (імена) із *значеннями* (інформацією). Зауважте, що ключ має бути унікальним, тому що ви не знаєте коректність наданої інформації, якщо у вас є дві людини з однаковими іменами.

Зауважте, що ви можете використовувати лише незмінні об’єкти (наприклад, рядки) для ключів словника, але ви можете використовувати як незмінні, так і змінні об’єкти для значень словника. По суті, це означає, що ви повинні використовувати лише прості об’єкти для ключів.

Пари ключ-значення вказуються в словнику за допомогою наступної нотації: `d = {key1 : value1,`

`key2 : value2 }`. Зверніть увагу, що ключ і значення відокремлені двокрапкою, а самі пари відокремлені комами, і все це взято у пару фігурних дужок.

Пам'ятайте, що пари ключ-значення в словнику не впорядковані жодним чином. Якщо вам потрібен певний порядок, вам доведеться самостійно відсортувати словник, перш ніж використовувати його.

Словники, які ви використовуватимете, є екземплярами/об'єктами класу `dict`.

код python ds_using_dict_ukr.py

```
# "ab" є скороченням від 'a'ddress'b'ook(address book-адресна книга).

ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotmail.com'
}

print("Адрес Swaroop'a:", ab['Swaroop'])

# Видалення пари ключ-значення
del ab['Spammer']

print('\nВ адресній книзі {} контакта\n'.format(len(ab)))

for ім_я, адреса in ab.items():
    print('Контакт {} за адресою: {}'.format(ім_я, адреса))

# Додавання пари ключ-значення
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print("\nАдрес Guido:", ab['Guido'])
```

Висновок:

```
$ python3 using_dict_ukr.py
Адрес Swaroop'a: swaroop@swaroopch.com

В адресній книзі 3 контакта

Контакт Swaroop за адресою: swaroop@swaroopch.com
Контакт Larry за адресою: larry@wall.org
Контакт Matsumoto за адресою: matz@ruby-lang.org

Адрес Guido: guido@python.org
```

Як це працює

Ми створюємо словник `ab`, використовуючи вже розглянуту нотацію. Потім ми отримуємо доступ до пар ключ-значення, вказуючи ключ за допомогою оператора індексування, як обговорювалося в контексті списків і кортежів. Зверніть увагу на простий синтаксис.

Ми можемо видалити пари ключ-значення за допомогою нашого старого друга - оператора `del`. Ми просто вказуємо ім'я словника та оператор індексування для ключа, який потрібно видалити, і передаємо його оператору del. Для цієї операції немає необхідності знати значення, яке відповідає ключу.

Далі ми звертаємося до всіх пар ключ-значення нашого словника ,використовуючи метод `items`, який повертає список кортежів, де кожен кортеж містить пару елементів — ключ та значення. Ми отримуємо цю пару та присвоюємо їй значення змінних `ім_я` та `адреса` відповідно до кожної з пар за допомогою циклу `for...in`, а потім друкуємо ці значення в блоці for.

Ми можемо додати нові пари ключ-значення, просто використовуючи оператор індексування для доступу до ключа та присвоєння йому деякого значення, як ми зробили для Guido у наведеному вище випадку.

Ми можемо перевірити, чи існує пара ключ-значення, використовуючи оператор `in`.

Щоб переглянути список методів класу `dict`, перегляньте `help(dict)`.

Ключові аргументи і словники

англійська: *Keyword Arguments and Dictionaries*

Якщо ви використовували ключові аргументи у своїх функціях, ви вже використовували словники! Просто подумайте про це: ви вказали пару ключ-значення серед параметрів функції при її визначенні, а коли звертаєтесь до змінних усередині функції, то це, власне, звернення за ключом до словника (який у термінах розробників компіляторів називається *таблицею імен* (англ.» symbol table»)).

3.17.5 Послідовність

англійська: *Sequence*

Списки, кортежі та рядки є прикладами послідовностей, але що таке послідовності і що в них такого особливого?

Основними функціями є *перевірка приналежності* (тобто вирази `in` та `not in`) і *операції індексування*, які дозволяють нам напряму отримати певний елемент послідовності.

Три типи послідовностей, згадані вище - списки, кортежі та рядки, також мають операцію зрізу (англ. *slicing*), яка дозволяє нам отримати зріз послідовності, тобто її фрагмент.

код python ds_seq_ukr.py

```
список_покупок = ['яблуко', 'манго', 'морква', 'банан']
ім_я = 'swaroop'

# Операція індексування
print('Елемент 0 є', список_покупок[0])
print('Елемент 1 є', список_покупок[1])
print('Елемент 2 є', список_покупок[2])
print('Елемент 3 є', список_покупок[3])
print('Елемент -1 є', список_покупок[-1])
print('Елемент -2 є', список_покупок[-2])
print('Символ 0 є', ім_я[0])

# Зріз зі списку
```

```
print('Елемент з 1 до 3 є', список_покупок[1:3])
print('Елемент з 2 до кінця є', список_покупок[2:])
print('Елемент з 1 до -1 є', список_покупок[1:-1])
print('Елемент від початку до кінця є',список_покупок[:])

# Зріз з рядка
print('Символи з 1 до 3 є', ім_я[1:3])
print('Символи з 2 до кінця є', ім_я[2:])
print('Символи з 1 до  -1 є', ім_я[1:-1])
print('Символи від початку до кінця є', ім_я[:])
```

Висновок:

```
$ python3 ds_seq_ukr.py
Елемент 0 є яблуко
Елемент 1 є манго
Елемент 2 є морква
Елемент 3 є банан
Елемент -1 є банан
Елемент -2 є морква
Символ 0 є s
Елемент з 1 до 3 є ['манго', 'морква']
Елемент з 2 до кінця є ['морква', 'банан']
Елемент з 1 до -1 є ['манго', 'морква']
Елемент від початку до кінця є ['яблуко', 'манго', 'морква', 'банан']
Символи з 1 по 3 є wa
Символи з 2 до кінця є aroop
Символи з 1 до  -1 є waroo
Символи від початку до кінця є swaroop
```

Як це працює

Спочатку ми бачимо, як використовувати індекси для отримання окремих елементів послідовності. Це також називається операцією *приписування індексу* (англ."subscription operation").

Коли ми вказуємо число у квадратних дужках після послідовності, як показано вище, Python витягує елемент, який відповідає зазначеній позиції в послідовності. Пам'ятайте, що Python починає рахувати числа з 0. Отже, `список_покупокт[0]` витягує перший елемент, а `список_покупокт[3]` витягує четвертий елемент у послідовності `список_покупок`.

Індекс також може бути від'ємним числом. У цьому випадку позиція обчислюється з кінця послідовності. Таким чином, `список_покупок[-1]` посилається на останній елемент у послідовності, а `список_покупок[-2]` витягує передостанній елемент у послідовності.

Операція зрізу використовується вказуючи ім'я послідовності, за якою слідують необов'язкові два числа, розділені двокрапкою всередині квадратних дужок. Зауважте, що це дуже схоже на операцію індексування, яку ви використовували досі. Пам'ятайте, що цифри необов'язкові, а двокрапка – обов'язкова.

Перше число (перед двокрапкою) в операції зрізу вказує позицію, з якої починається зріз, а друге число (після двокрапки) вказує, де зріз має закінчитися. Якщо перше число не вказано, Python розпочне з початку послідовності. Якщо друге число пропущено, Python зупиниться в кінці послідовності. Зверніть увагу,що отриманий зріз буде починатися із зазначеної початкової позиції (англ."*starts*"),а закінчуватися перед зазначеної кінцевої позицією (англ."*end*"), тобто, початкова позиція буде включена у зріз, а кінцева – ні.

Таким чином, `список_покупок[1:3]` повертає зріз із послідовності, починаючи з позиції 1, вклю-

чає позицію 2, але зупиняється на позиції 3, і тому повертає *зріз* з двох елементів. Так само `список_покупок[:]` повертає копію всієї послідовності.

Також можна робити зріз використовуючи від'ємні числа. Числа зі знаком мінус використовуються для позицій з кінця послідовності. Наприклад, `список_покупок[:-1]` поверне зріз послідовності, який виключає останній елемент послідовності, але містить усе інше.

Ви також можете надати третій аргумент для зрізу, який є *кроком* для зрізу (за замовчуванням розмір кроку дорівнює 1):

```
>>> список_покупок= ['яблуко', 'манго', 'морква', 'банан']
>>> список_покупок[::1]
['яблуко', 'манго', 'морква', 'банан']
>>> список_покупок[::2]
['яблуко', 'морква']
>>> список_покупок[::3]
['яблуко', 'банан']
>>> список_покупок[::-1]
['банан', 'морква', 'манго', 'яблуко']
```

Зауважте, що коли крок дорівнює 2, ми отримуємо елементи з позиціями 0, 2,... Коли розмір кроку дорівнює 3, ми отримуємо елементи з позиціями 0, 3,... тощо.

Спробуйте різні комбінації параметрів зрізу, використовуючи інтерактивний інтерпретатор Python, тобто підказку prompt, щоб ви могли негайно побачити результати. Чудова річ у послідовностях полягає в тому, що ви можете отримати доступ до кортежів, списків і рядків однаково!

3.17.6 Множина

 англійська: *Set*

Множини - це *невпорядковані* (англ.»*unordered* «) набори простих об'єктів. Вони необхідні тоді, коли присутність об'єкта в наборі важливіша за порядок або те, скільки разів даний об'єкт там зустрічається.

Використовуючи множини, ви можете перевірити приналежність, визначати, чи є ця множина підмножиною іншої множини, знайти перетин між двома множинами тощо.

```
Скандинавія = set(["Данія","Норвегія","Швеція"])
print("З країни Скандинавії:", Скандинавія)
print("Чи є Ісландія частиною Скандинавії?: ", "Ісландія" in Скандинавія)
print("Нордичні країни також включають Ісландію та Фінляндію")
Нордичні_країни = Скандинавія.copy()
Нордичні_країни.add("Ісландія")
Нордичні_країни.add("Фінляндія")
print("Нордичні країни", Нордичні_країни )
print("Чи Нордичні країни є супермножиною Скандинавії?", Нордичні_країни.
↪issuperset(Скандинавія))
print("Чи є Скандинавія підмножиною нордичних країн?", Скандинавія.issubset(Нордичні_
↪країни))
країни_сухопутного_кордону = Нордичні_країни.copy()
країни_сухопутного_кордону.remove("Ісландія")
print("Нордичні країни з сухопутними кордонами: ", країни_сухопутного_кордону)
print("Перетин Скандинавії та Нордичних країн: ", Скандинавія.intersection(Нордичні_
↪країни))
```

Як це працює

Якщо ви пам'ятаєте базову математику теорії множин зі школи, то цей приклад досить зрозумілий. Але якщо ні, ви можете пошукати в Google «теорію множин» і «діаграму Венна», щоб краще зрозуміти, як ми використовуємо множини в Python.

3.17.7 Посилання

 англійська: *References*

Коли ви створюєте об'єкт і присвоюєте його змінній, змінна лише *посилається* на об'єкт і не є цим об'єктом! Тобто ім'я змінної вказує на ту частину пам'яті комп'ютера, де зберігається об'єкт. Це називається *прив'язкою* (англ."*binding*") імені до об'єкта.

Загалом, вам не потрібно турбуватися про це, проте є деякий неочевидний ефект, про який потрібно пам'ятати:

код python ds_reference_ukr.py

```
print('Просте присвоєння')
список_покупок = ['яблуко', 'манго', 'морква', 'банан']
# мій_список - це просто інша назва, що вказує на той самий об'єкт!
мій_список = список_покупок

# Я купив перший товар, тому видаляю його зі списку
del список_покупок[0]

print('список_покупок :', список_покупок)
print('мій_список :', мій_список)
# Зауважте, що друкуються і список_покупок,і мій_список
# з однаковим списком без пункта «яблука»
# вони вказують на той самий об'єкт

print('Копіювати, зробивши повний зріз')
# Зробіть копію, зробивши повний зріз
мій_список = список_покупок[:]
# Видалити перший елемент
del мій_список[0]

print('список_покупок :', список_покупок)
print('мій_список :', мій_список)
# Зверніть увагу, що тепер два списки різні
```

Висновок:

```
$ python3 ds_reference_ukr.py
Просте присвоєння
список_покупок : ['манго', 'морква', 'банан']
мій_список : ['манго', 'морква', 'банан']
Копіювати, зробивши повний зріз
список_покупок : ['манго', 'морква', 'банан']
мій_список : ['морква', 'банан']
```

Як це працює

Більшість пояснень доступна в коментарях.

Пам'ятайте, що якщо ви хочете зробити копію списку або подібних типів послідовностей, або складних об'єктів (а не простих *об'єктів*, таких як цілі числа), тоді ви повинні використовувати операцію зріз, щоб зробити копію. Якщо ви просто присвоїте назву змінної іншій назві, обидві вони «посилатимуться» на той самий об'єкт, і це може спричинити проблеми, якщо ви не будете обережні.

Примітка для програмістів Perl

Пам'ятайте, що оператор присвоєння для списків **не** створює копію. Ви повинні використовувати операцію зріз, щоб створити копію послідовності.

3.17.8 Докладніше про рядки

 англійська: *More About Strings*

Ми вже детально обговорювали рядки раніше. Що ще можна про них дізнатися? Ну, чи знаєте ви, що рядки також є об'єктами та мають методи, які роблять усе, від перевірки частини рядка до видалення пробілів? Фактично, ви вже використовували рядковий метод... метод `format`!

Усі рядки, які ви використовуєте в програмах, є об'єктами класу `str`. Деякі корисні методи цього класу демонструються в наступному прикладі. Щоб отримати повний список таких методів, перегляньте `help(str)`.

код python ds_str_methods_ukr.py

```
# Це рядковий об'єкт
ім_я = 'Swaroop'

if ім_я.startswith('Swa'):
    print('Так, рядок починається з "Swa"')

if 'a' in ім_я:
    print('Так, він містить рядок "a"')

if ім_я.find('war') != -1:
    print('Так, він містить рядок "war"')

delimiter = '_*_'
мій_лист = ['Данія', 'Норвегія', 'Швеція', 'Ісланді','Фінляндія' ]
print(delimiter.join(мій_лист))
```

Висновок:

```
$ python3 ds_str_methods_ukr.py
Так, рядок починається з "Swa"
Так, він містить рядок "a"
Так, він містить рядок "war"
Данія_*_Норвегія_*_Швеція_*_Ісланді_*_Фінляндія
```

Як це працює

Тут бачимо відразу кілька методів рядків у дії. Метод `startswith` використовується, щоб дізнатися, чи починається рядок із заданого рядка. Оператор `in` використовується, щоб перевірити, чи є певний

рядок частиною цього рядка.

Метод `find` використовується для визначення позиції даного підрядка в рядку; `find` повертає -1, якщо не вдалося знайти підрядок. Клас `str` також має чудовий метод для `об'єднання`(англ. "`join`") елементів послідовності з рядком, який діє як роздільник між кожним елементом послідовності, і повертає більший рядок, згенерований із цього.

3.17.9 Резюме

Ми детально дослідили різноманітні вбудовані структури даних Python. Ці структури даних будуть необхідними для написання програм розумного розміру.

Тепер, коли ми маємо багато основ Python, ми далі побачимо, як розробити та написати реальну програму Python.

3.18 Структури даних

 англійська: *Data Structures*

Структури даних — це, по суті, і є *структури*, які можуть зберігати деякі *дані* разом. Іншими словами, вони використовуються для зберігання даних.

У Python є чотири вбудовані структури даних: *список (англ."list"), кортеж (англ."tuple"), словник (англ."dictionary") та множина (англ."set")*. Ми побачимо, як використовувати кожен із них і як вони полегшують нам життя.

3.18.1 Список

 англійська: *List*

Список — це структура даних, яка містить упорядкований набір елементів, тобто зберігає *послідовність* елементів у списку (англ."*a sequence* of items in a list"). Це легко уявити, якщо згадати список покупок, в якому перераховується, що потрібно купити, з тим винятком, що у списку покупок кожен елемент зазвичай розміщується в окремому рядку, тоді як у Python вони розділяються комами.

Список елементів має бути укладений у квадратні дужки, щоб Python зрозумів, що ви вказуєте список. Створивши список, ви можете додавати, видаляти або шукати елементи у списку. Оскільки ми можемо додавати та видаляти елементи, ми говоримо, що список є *змінним* (англ."*mutable*") типом даних, тобто цей тип можна змінювати.

3.18.2 Короткий вступ до об'єктів і класів

 англійська: *Quick Introduction To Objects And Classes*

Хоч я і намагався досі відтягнути обговорення об'єктів і класів, на даному етапі все ж таки необхідне деяке пояснення, щоб ви краще зрозуміли ідею списків. Ми детально розглянемо цю тему у *пізнішому розділі*.

Список є прикладом використання об'єктів і класів. Коли ми присвоюємо деякій змінній `i` значення, скажімо, ціле число `5`, ви можете сприймати це як створення *об'єкта* (тобто екземпляра) `i` *класу* (тобто типу) `int`. Насправді, ви можете прочитати `help(int)`, щоб зрозуміти це краще.

Клас також може мати *методи*, тобто функції, визначені для використання лише стосовно цього класу. Ви можете використовувати ці частини функціональності, лише якщо у вас є об'єкт цього класу. Наприклад, Python надає метод `append` для класу `список`, який дозволяє вам додавати елемент у кінець

списку. Наприклад, `мій_список.append('якийсь елемент')` додасть цей рядок до списку `мій_список`. Зверніть увагу на використання нотації з крапками для доступу до методів об'єктів.

Клас також може мати *поля*, які є нічим іншим, як змінними, визначеними для використання лише стосовно цього класу. Ви можете використовувати ці змінні/імена, лише якщо у вас є об'єкт цього класу. Поля також доступні за допомогою нотації з крапками, наприклад, `мій_список.field`.

код python ds_using_list_ukr.py

```
# Це мій список покупок
список_покупок = ['яблуко', 'манго', 'морква', 'банан']

print('Я маю', len(список_покупок), 'товари для покупок.')

print('Товари для покупок:', end=' ')
for елементи in список_покупок:
    print(елементи, end=' ')

print('\nЯ також маю купити рис.')
список_покупок.append('рис')
print('Мій список покупок зараз', список_покупок)

print('Я відсортую свій список зараз')
список_покупок.sort()
print('Сортований список покупок є', список_покупок)

print('Перший товар, який я куплю, це', список_покупок[0])
старий_елемент = список_покупок[0]
del список_покупок[0]
print('Я купив', старий_елемент )
print('Мій список покупок зараз', список_покупок)
```

Висновок:

```
Я маю 4 товари для покупок.
Товари для покупок: яблуко манго морква банан
Я також маю купити рис.
Мій список покупок зараз ['яблуко', 'манго', 'морква', 'банан', 'рис']
Я відсортую свій список зараз
Сортований список покупок є ['банан', 'манго', 'морква', 'рис', 'яблуко']
Перший товар, який я куплю, це банан
Я купив банан
Мій список покупок зараз ['манго', 'морква', 'рис', 'яблуко']
```

Як це працює

Змінна `список_покупок` - це список покупок для того, хто збирається на ринок. У `список_покупок` ми зберігаємо лише рядки назв предметів для покупки, але ви можете додавати *будь-які об'єкти* до списку, включаючи номери та навіть інші списки.

Ми також використали цикл for... in для ітерації по елементах списку. Наразі ви вже зрозуміли, що список також є послідовністю. Особливості послідовностей буде обговорено у *пізнішому розділі*.

Зверніть увагу на використання параметра `end` у виклику функції `print`, який вказує, що ми хочемо закінчити рядок у висновку пробілом, замість звичайного розриву рядка.

Далі ми додаємо елемент до списку за допомогою методу `append`- методу об'єкта списку, як вже обговорювалося раніше. Потім ми перевіряємо, чи елемент справді додано до списку, друкуючи вміст списку за допомогою передачі цього списку функції`print`, яка старанно друкує його.

Потім ми сортуємо список за допомогою методу `sort`об'єкта списку. Важливо розуміти, що цей метод впливає на сам список і не повертає змінений список - це відрізняється від того, як працюють рядки. Це те, що ми маємо на увазі, кажучи, що списки *змінні* (англ."mutable"), а рядки *незмінні* (англ."immutable").

Потім, коли ми закінчимо купувати товар на ринку, ми хочемо видалити його зі списку. Ми досягаємо цього за допомогою оператора del. Тут ми згадуємо, який елемент зі списку ми хочемо видалити, і оператор del видаляє його зі списку за нас. Ми вказуємо, що ми хочемо видалити перший елемент зі списку, і тому ми використовуємо `del список_покупок[0]` (пам'ятайте, що Python починає відлік з 0).

Якщо ви хочете дізнатися про всі методи, визначені об'єктом списку, подробиці дивіться у `help(list)`.

3.18.3 Кортеж

 англійська: *Tuple*

Кортежі служать для зберігання кількох об'єктів разом. Вважайте їх подібними до списків, але без широкої функціональності, яку надає вам клас списку. Однією з головних особливостей кортежів є те, що вони *незмінні* (англ."*immutable*"),як і рядки, тобто ви не можете змінювати кортежі.

Кортежі позначаються визначенням елементів, розділених комами; за бажанням їх можна ще укласти в круглі дужки.

Кортежі зазвичай використовуються у випадках, коли оператор або визначена користувачем функція може безпечно припустити, що набір значень (тобто використаний кортеж значень) не зміниться.

код python ds_using_tuple_ukr.py

```
# Я рекомендую завжди використовувати круглі дужки
# позначати початок та кінець кортежу
# незважаючи на те, що дужки необов'язкові.
# Явне краще за неявне.
зоопарк  = ('пітон', 'слон', 'пінгвін')
print('Кількість тварин у зоопарку становить', len(зоопарк))

новий_зоопарк = 'мавпа', 'верблюд', зоопарк # дужки не потрібні, але це гарна ідея
print('Кількість клітин у новому зоопарку становить', len(новий_зоопарк))
print('Усі тварини в новому зоопарку є', новий_зоопарк)
print('Тварини, привезені зі старого зоопарку є', новий_зоопарк[2])
print('Остання тварина, привезена зі старого зоопарку', новий_зоопарко[2][2])
print('Кількість тварин у новому зоопарку становить',
      len(новий_зоопарк)-1+len(новий_зоопарк[2]))
```

Висновок:

```
$ python ds_using_tuple_ukr.py
Кількість тварин у зоопарку становить 3
Кількість клітин у новому зоопарку становить 3
Усі тварини в новому зоопарку є ('мавпа', 'верблюд', ('пітон', 'слон', 'пінгвін'))
Тварини, привезені зі старого зоопарку є ('пітон', 'слон', 'пінгвін')
Остання тварина, привезена зі старого зоопарку пінгвін
Кількість тварин у новому зоопарку становить 5
```

Як це працює

Змінна `зоопарк` означає кортеж елементів. Ми бачимо, що функцію `len` можна використовувати для отримання довжини кортежу. Це також вказує на те, що кортеж також є *послідовністю*.

Зараз ми переводимо цих тварин у новий зоопарк, оскільки старий зоопарк закривається. Таким чином, кортеж `новий_зоопарк` містить деяких тварин, які вже там разом із тваринами, привезеними зі старого зоопарку. Повертаючись до реальності, зауважте, що кортеж у кортежі не втрачає своєї ідентичності.

Ми можемо отримати доступ до елементів у кортежі, вказавши позицію елемента в парі квадратних дужок, як ми робили для списків. Це називається оператором *індексування* (англ.“*indexing*”). Ми отримуємо доступ до третього елемента в `новий_зоопарк`, вказуючи `новий_зоопарк[2]`, і отримуємо доступ до третього елемента в третьому елементі в кортежі `новий_зоопарк`, вказуючи `новий_зоопарк[2][2]`. Це досить просто, якщо ви зрозуміли ідіому.

Кортеж із 0 або 1 елементом

Порожній кортеж створюється пустою парою круглих дужок, наприклад `myempty = ()`. Однак кортеж з одним елементом не такий простий. Його потрібно вказати за допомогою коми після першого (і єдиного) елемента, щоб Python міг розрізнити кортеж і пару круглих дужок, які оточують об’єкт у виразі. Таким чином, щоб отримати кортеж, що містить елемент 2, вам потрібно буде вказати `singleton = (2 , )`.

Примітка для програмістів Perl

Список у списку не втрачає своєї ідентичності, тобто списки не розгортаються, як у Perl. Те саме стосується кортежу в кортежі, або кортежу в списку, або списку в кортежі тощо. Що стосується Python, це просто об’єкти, які зберігаються за допомогою іншого об’єкта, от і все.

3.18.4 Словник

 англійська: *Dictionary*

Словник схожий на адресну книгу, де ви можете знайти адресу або контактні дані особи, знаючи лише її/її ім’я; тобто ми пов’язуємо *ключі* (імена) із *значеннями* (інформацією). Зауважте, що ключ має бути унікальним, тому що ви не знаєте коректність наданої інформації, якщо у вас є дві людини з однаковими іменами.

Зауважте, що ви можете використовувати лише незмінні об’єкти (наприклад, рядки) для ключів словника, але ви можете використовувати як незмінні, так і змінні об’єкти для значень словника. По суті, це означає, що ви повинні використовувати лише прості об’єкти для ключів.

Пари ключ-значення вказуються в словнику за допомогою наступної нотації: `d = {key1 : value1, key2 : value2 }`. Зверніть увагу, що ключ і значення відокремлені двокрапкою, а самі пари відокремлені комами, і все це взято у пару фігурних дужок.

Пам’ятайте, що пари ключ-значення в словнику не впорядковані жодним чином. Якщо вам потрібен певний порядок, вам доведеться самостійно відсортувати словник, перш ніж використовувати його.

Словники, які ви використовуватимете, є екземплярами/об’єктами класу `dict`.

код python ds_using_dict_ukr.py

```
# "ab" є скороченням від 'a'ddress'b'ook(address book-адресна книга).

ab = {
    'Swaroop': 'swaroop@swaroopch.com',
    'Larry': 'larry@wall.org',
    'Matsumoto': 'matz@ruby-lang.org',
    'Spammer': 'spammer@hotmail.com'
}

print("Адрес Swaroop'a:", ab['Swaroop'])

# Видалення пари ключ-значення
del ab['Spammer']

print('\nВ адресній книзі {} контакта\n'.format(len(ab)))

for ім_я, адреса in ab.items():
    print('Контакт {} за адресою: {}'.format(ім_я, адреса))

# Додавання пари ключ-значення
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print("\nАдрес Guido:", ab['Guido'])
```

Висновок:

```
$ python3 using_dict_ukr.py
Адрес Swaroop'a: swaroop@swaroopch.com

В адресній книзі 3 контакта

Контакт Swaroop за адресою: swaroop@swaroopch.com
Контакт Larry за адресою: larry@wall.org
Контакт Matsumoto за адресою: matz@ruby-lang.org

Адрес Guido: guido@python.org
```

Як це працює

Ми створюємо словник `ab`, використовуючи вже розглянуту нотацію. Потім ми отримуємо доступ до пар ключ-значення, вказуючи ключ за допомогою оператора індексування, як обговорювалося в контексті списків і кортежів. Зверніть увагу на простий синтаксис.

Ми можемо видалити пари ключ-значення за допомогою нашого старого друга - оператора `del`. Ми просто вказуємо ім'я словника та оператор індексування для ключа, який потрібно видалити, і передаємо його оператору del. Для цієї операції немає необхідності знати значення, яке відповідає ключу.

Далі ми звертаємося до всіх пар ключ-значення нашого словника ,використовуючи метод `items`, який повертає список кортежів, де кожен кортеж містить пару елементів — ключ та значення. Ми отримуємо цю пару та присвоюємо їй значення змінних `ім_я` та `адреса` відповідно до кожної з пар за допомогою циклу `for...in`, а потім друкуємо ці значення в блоці for.

Ми можемо додати нові пари ключ-значення, просто використовуючи оператор індексування для доступу до ключа та присвоєння йому деякого значення, як ми зробили для Guido у наведеному вище

випадку.

Ми можемо перевірити, чи існує пара ключ-значення, використовуючи оператор `in`.

Щоб переглянути список методів класу `dict`, перегляньте `help(dict)`.

Ключові аргументи і словники

англійська: *Keyword Arguments and Dictionaries*

Якщо ви використовували ключові аргументи у своїх функціях, ви вже використовували словники! Просто подумайте про це: ви вказали пару ключ-значення серед параметрів функції при її визначенні, а коли звертаєтесь до змінних усередині функції, то це, власне, звернення за ключом до словника (який у термінах розробників компіляторів називається *таблицею імен* (англ.» symbol table»)).

3.18.5 Послідовність

англійська: *Sequence*

Списки, кортежі та рядки є прикладами послідовностей, але що таке послідовності і що в них такого особливого?

Основними функціями є *перевірка приналежності* (тобто вирази `in` та `not in`) і *операції індексування*, які дозволяють нам напряму отримати певний елемент послідовності.

Три типи послідовностей, згадані вище - списки, кортежі та рядки, також мають операцію зрізу (англ. *slicing*), яка дозволяє нам отримати зріз послідовності, тобто її фрагмент.

код python ds_seq_ukr.py

```
список_покупок = ['яблуко', 'манго', 'морква', 'банан']
ім_я = 'swaroop'

# Операція індексування
print('Елемент 0 є', список_покупок[0])
print('Елемент 1 є', список_покупок[1])
print('Елемент 2 є', список_покупок[2])
print('Елемент 3 є', список_покупок[3])
print('Елемент -1 є', список_покупок[-1])
print('Елемент -2 є', список_покупок[-2])
print('Символ 0 є', ім_я[0])

# Зріз зі списку
print('Елемент з 1 до 3 є', список_покупок[1:3])
print('Елемент з 2 до кінця є', список_покупок[2:])
print('Елемент з 1 до -1 є', список_покупок[1:-1])
print('Елемент від початку до кінця є',список_покупок[:])

# Зріз з рядка
print('Символи з 1 до 3 є', ім_я[1:3])
print('Символи з 2 до кінця є', ім_я[2:])
print('Символи з 1 до  -1 є', ім_я[1:-1])
print('Символи від початку до кінця є', ім_я[:])
```

Висновок:

```
$ python3 ds_seq_ukr.py
Елемент 0 є яблуко
Елемент 1 є манго
Елемент 2 є морква
Елемент 3 є банан
Елемент -1 є банан
Елемент -2 є морква
Символ 0 є s
Елемент з 1 до 3 є ['манго', 'морква']
Елемент з 2 до кінця є ['морква', 'банан']
Елемент з 1 до -1 є ['манго', 'морква']
Елемент від початку до кінця є ['яблуко', 'манго', 'морква', 'банан']
Символи з 1 по 3 є wa
Символи з 2 до кінця є aroop
Символи з 1 до  -1 є waroo
Символи від початку до кінця є swaroop
```

Як це працює

Спочатку ми бачимо, як використовувати індекси для отримання окремих елементів послідовності. Це також називається операцією *приписування індексу* (англ."subscription operation").

Коли ми вказуємо число у квадратних дужках після послідовності, як показано вище, Python витягує елемент, який відповідає зазначеній позиції в послідовності. Пам'ятайте, що Python починає рахувати числа з 0. Отже, `список_покупокт[0]` витягує перший елемент, а `список_покупокт[3]` витягує четвертий елемент у послідовності `список_покупок`.

Індекс також може бути від'ємним числом. У цьому випадку позиція обчислюється з кінця послідовності. Таким чином, `список_покупок[-1]` посилається на останній елемент у послідовності, а `список_покупок[-2]` витягує передостанній елемент у послідовності.

Операція зрізу використовується вказуючи ім'я послідовності, за якою слідують необов'язкові два числа, розділені двокрапкою всередині квадратних дужок. Зауважте, що це дуже схоже на операцію індексування, яку ви використовували досі. Пам'ятайте, що цифри необов'язкові, а двокрапка – обов'язкова.

Перше число (перед двокрапкою) в операції зрізу вказує позицію, з якої починається зріз, а друге число (після двокрапки) вказує, де зріз має закінчитися. Якщо перше число не вказано, Python розпочне з початку послідовності. Якщо друге число пропущено, Python зупиниться в кінці послідовності. Зверніть увагу,що отриманий зріз буде починатися із зазначеної початкової позиції (англ."*starts*"),а закінчуватися перед зазначеної кінцевої позицією (англ."*end*"), тобто, початкова позиція буде включена у зріз, а кінцева – ні.

Таким чином, `список_покупок[1:3]` повертає зріз із послідовності, починаючи з позиції 1, включає позицію 2, але зупиняється на позиції 3, і тому повертає *зріз* з двох елементів. Так само `список_покупок[:]` повертає копію всієї послідовності.

Також можна робити зріз використовуючи від'ємні числа. Числа зі знаком мінус використовуються для позицій з кінця послідовності. Наприклад, `список_покупок[:-1]` поверне зріз послідовності, який виключає останній елемент послідовності, але містить усе інше.

Ви також можете надати третій аргумент для зрізу, який є *кроком* для зрізу (за замовчуванням розмір кроку дорівнює 1):

```
>>> список_покупок= ['яблуко', 'манго', 'морква', 'банан']
>>> список_покупок[::1]
['яблуко', 'манго', 'морква', 'банан']
>>> список_покупок[::2]
['яблуко', 'морква']
>>> список_покупок[::3]
['яблуко', 'банан']
>>> список_покупок[::-1]
['банан', 'морква', 'манго', 'яблуко']
```

Зауважте, що коли крок дорівнює 2, ми отримуємо елементи з позиціями 0, 2,... Коли розмір кроку дорівнює 3, ми отримуємо елементи з позиціями 0, 3,... тощо.

Спробуйте різні комбінації параметрів зрізу, використовуючи інтерактивний інтерпретатор Python, тобто підказку prompt, щоб ви могли негайно побачити результати. Чудова річ у послідовностях полягає в тому, що ви можете отримати доступ до кортежів, списків і рядків однаково!

3.18.6 Множина

 англійська: *Set*

Множини - це *невпорядковані* (англ.»*unordered* «) набори простих об'єктів. Вони необхідні тоді, коли присутність об'єкта в наборі важливіша за порядок або те, скільки разів даний об'єкт там зустрічається.

Використовуючи множини, ви можете перевірити приналежність, визначати, чи є ця множина підмножиною іншої множини, знайти перетин між двома множинами тощо.

```
Скандинавія = set(["Данія","Норвегія","Швеція"])
print("З країни Скандинавії:", Скандинавія)
print("Чи є Ісландія частиною Скандинавії?: ", "Ісландія" in Скандинавія)
print("Нордичні країни також включають Ісландію та Фінляндію")
Нордичні_країни = Скандинавія.copy()
Нордичні_країни.add("Ісландія")
Нордичні_країни.add("Фінляндія")
print("Нордичні країни", Нордичні_країни )
print("Чи Нордичні країни є супермножиною Скандинавії?", Нордичні_країни.
↪issuperset(Скандинавія))
print("Чи є Скандинавія підмножиною нордичних країн?", Скандинавія.issubset(Нордичні_
↪країни))
країни_сухопутного_кордону = Нордичні_країни.copy()
країни_сухопутного_кордону.remove("Ісландія")
print("Нордичні країни з сухопутними кордонами: ", країни_сухопутного_кордону)
print("Перетин Скандинавії та Нордичних країн: ", Скандинавія.intersection(Нордичні_
↪країни))
```

Як це працює

Якщо ви пам'ятаєте базову математику теорії множин зі школи, то цей приклад досить зрозумілий. Але якщо ні, ви можете пошукати в Google «теорію множин» і «діаграму Венна», щоб краще зрозуміти, як ми використовуємо множини в Python.

3.18.7 Посилання

 англійська: *References*

Коли ви створюєте об'єкт і присвоюєте його змінній, змінна лише *посилається* на об'єкт і не є цим об'єктом! Тобто ім'я змінної вказує на ту частину пам'яті комп'ютера, де зберігається об'єкт. Це називається *прив'язкою* (англ."*binding*") імені до об'єкта.

Загалом, вам не потрібно турбуватися про це, проте є деякий неочевидний ефект, про який потрібно пам'ятати:

код python ds_reference_ukr.py

```
print('Просте присвоєння')
список_покупок = ['яблуко', 'манго', 'морква', 'банан']
# мій_список - це просто інша назва, що вказує на той самий об'єкт!
мій_список = список_покупок

# Я купив перший товар, тому видаляю його зі списку
del список_покупок[0]

print('список_покупок :', список_покупок)
print('мій_список :', мій_список)
# Зауважте, що друкуються і список_покупок,і мій_список
# з однаковим списком без пункта «яблука»
# вони вказують на той самий об'єкт

print('Копіювати, зробивши повний зріз')
# Зробіть копію, зробивши повний зріз
мій_список = список_покупок[:]
# Видалити перший елемент
del мій_список[0]

print('список_покупок :', список_покупок)
print('мій_список :', мій_список)
# Зверніть увагу, що тепер два списки різні
```

Висновок:

```
$ python3 ds_reference_ukr.py
Просте присвоєння
список_покупок : ['манго', 'морква', 'банан']
мій_список : ['манго', 'морква', 'банан']
Копіювати, зробивши повний зріз
список_покупок : ['манго', 'морква', 'банан']
мій_список : ['морква', 'банан']
```

Як це працює

Більшість пояснень доступна в коментарях.

Пам'ятайте, що якщо ви хочете зробити копію списку або подібних типів послідовностей, або складних об'єктів (а не простих *об'єктів*, таких як цілі числа), тоді ви повинні використовувати операцію зріз, щоб зробити копію. Якщо ви просто присвоїте назву змінної іншій назві, обидві вони «посилатимуться»

на той самий об'єкт, і це може спричинити проблеми, якщо ви не будете обережні.

Примітка для програмістів Perl

Пам'ятайте, що оператор присвоєння для списків **не** створює копію. Ви повинні використовувати операцію зріз, щоб створити копію послідовності.

3.18.8 Докладніше про рядки

 англійська: *More About Strings*

Ми вже детально обговорювали рядки раніше. Що ще можна про них дізнатися? Ну, чи знаєте ви, що рядки також є об'єктами та мають методи, які роблять усе, від перевірки частини рядка до видалення пробілів? Фактично, ви вже використовували рядковий метод... метод `format`!

Усі рядки, які ви використовуєте в програмах, є об'єктами класу `str`. Деякі корисні методи цього класу демонструються в наступному прикладі. Щоб отримати повний список таких методів, перегляньте `help(str)`.

код python ds_str_methods_ukr.py

```
# Це рядковий об'єкт
ім_я = 'Swaroop'

if ім_я.startswith('Swa'):
    print('Так, рядок починається з "Swa"')

if 'a' in ім_я:
    print('Так, він містить рядок "a"')

if ім_я.find('war') != -1:
    print('Так, він містить рядок "war"')

delimiter = '_*_'
мій_лист = ['Данія', 'Норвегія', 'Швеція', 'Ісланді','Фінляндія' ]
print(delimiter.join(мій_лист))
```

Висновок:

```
$ python3 ds_str_methods_ukr.py
Так, рядок починається з "Swa"
Так, він містить рядок "a"
Так, він містить рядок "war"
Данія_*_Норвегія_*_Швеція_*_Ісланді_*_Фінляндія
```

Як це працює

Тут бачимо відразу кілька методів рядків у дії. Метод `startswith` використовується, щоб дізнатися, чи починається рядок із заданого рядка. Оператор `in` використовується, щоб перевірити, чи є певний рядок частиною цього рядка.

Метод `find` використовується для визначення позиції даного підрядка в рядку; `find` повертає -1, якщо не вдалося знайти підрядок. Клас `str` також має чудовий метод для `об'єднання`(англ. "`join`") елемен-

тів послідовності з рядком, який діє як роздільник між кожним елементом послідовності, і повертає більший рядок, згенерований із цього.

3.18.9 Резюме

Ми детально дослідили різноманітні вбудовані структури даних Python. Ці структури даних будуть необхідними для написання програм розумного розміру.

Тепер, коли ми маємо багато основ Python, ми далі побачимо, як розробити та написати реальну програму Python.

3.19 Вирішення проблем

 англійська: *Problem Solving*

Ми розглянули різні частини мови Python, а тепер ми подивимося, як усі ці частини поєднуються між собою, розробляючи та пишучи програму, яка *робить* щось корисне. Мета полягає в тому, щоб навчитися писати сценарії мовою Python самостійно.

3.19.1 Проблема

Перед нами стоїть наступна проблема:

Увага

Скласти програму, яка створює резервні копії всіх важливих файлів.

Хоча це проста проблема (задача), нам недостатньо інформації, щоб почати її вирішення. Необхідний додатковий *аналіз*. Наприклад, як ми виберемо, *які* файли копіювати? *Як* їх зберігати? *Де* їх зберігати?

Після належного аналізу проблеми, ми *проєктуємо * нашу програму. Ми складаємо список того, що має зробити наша програма. У цьому випадку я створив наступний список того, як *я* уявляю її роботу. Коли ви проєктуєте програму, у вас може вийти інший результат, оскільки кожна людина уявляє собі це по-своєму, тож це цілком нормально.

- Файли та папки, які потрібно скопіювати, збираються до списку.
- Резервні копії повинні зберігатися у головній папці резервних копій.
- Резервні копії файлів зберігаються у файлі zip.
- Ім'я для zip-архіву - поточна дата та час.
- Будемо використовувати стандартну команду `zip`, доступну за замовчуванням у будь-якому стандартному дистрибутиві GNU/Linux або Unix. Зауважте, що ви можете використовувати будь-яку команду архівування, якщо вона має інтерфейс командного рядка.

Для користувачів Windows

Користувачі Windows можуть встановити команду `zip` зі сторінки проекту GnuWin32 і додати `C:\Program Files\GnuWin32\bin` до набору змінних середовища `PATH`, подібно до того *що ми зробили для розпізнавання самої команди python*.

3.19.2 Вирішення

 англійська: *The Solution*

Оскільки проєкт нашої програми зараз досить стабільний, ми можемо написати код, який є *реалізацією* нашого вирішення.

код python backup_ver1_ukr.py

```
import os
import time

# 1.Файли та папки, які потрібно скопіювати, збираються до списку.
# Приклад у  Windows:
# джерело = ['"C:\\My Documents"']
# Приклад у  Mac OS X and Linux:
джерело = ['/Users/swa/notes']
# Зауважте, що ми повинні використовувати подвійні лапки всередині рядка для імен із␣
↪пробілами.
# Ми також могли використати необроблений рядок,
# написавши [r'C:\My Documents'].

# 2. Резервні копії повинні зберігатися у головній папці резервних копій
# Приклад у  Windows:
# цільова_папка (англ."target_dir",target directory) = 'E:\\Backup'
# Приклад у  Mac OS X and Linux:
цільова_папка = '/Users/swa/backup'
# Не забудьте замінити шлях `/Users/swa/backup` вашими власними назвами папок.

# 3. Резервні копії файлів зберігаються у файлі zip.
# 4. Ім'я для zip-архіву - поточна дата та час.
ціль = цільова_папка + os.sep + \
         time.strftime('%Y%m%d%H%M%S') + '.zip'

# Створіть цільову папку, якщо її немає
if not os.path.exists(цільова_папка):
    os.mkdir(цільова_папка)  #створити папку

# 5. Ми використовуємо команду zip, щоб помістити файли в zip-архів
zip_command = 'zip -r {0} {1}'.format(ціль,
                                      ' '.join(джерело))

# Запускаємо створення резервної копії
print('Zip command є:')
print(zip_command)
print('Запуск:')
if os.system(zip_command) == 0:
    print('Резервна копія успішно створена', ціль)
else:
    print('Створення резервної копії НЕ ВДАЛОСЯ')
```

Висновок:

```
$ python backup_ver1_ukr.py
Zip command є:
zip -r /Users/swa/backup/20140328084844.zip /Users/swa/notes
Запуск:
  adding: Users/swa/notes/ (stored 0%)
  adding: Users/swa/notes/blah1.txt (stored 0%)
  adding: Users/swa/notes/blah2.txt (stored 0%)
  adding: Users/swa/notes/blah3.txt (stored 0%)
Резервна копія успішно створена в /Users/swa/backup/20140328084844.zip
```

Зараз ми знаходимося на етапі *тестування*, де ми перевіряємо, чи наша програма працює належним чином. Якщо вона не поводиться так, як очікувалося, ми повинні *налагодити* (англ."debug") нашу програму, тобто видалити *помилки* (англ.»bugs (errors) «) з програми.

Якщо наведена вище програма не працює для вас, скопіюйте рядок, надрукований після рядка `Zip command є` у висновку (zip -r /Users/swa/backup/20140328084844.zip /Users/swa/notes), вставте його в оболонку (у GNU/Linux і Mac OS X) / `cmd` (у Windows), подивіться, в чому полягає помилка, і спробуйте її виправити. Також перевірте посібник з команди "zip", щоб дізнатися, що може бути не так. Якщо ця команда виконується успішно, проблема може бути в самій програмі Python, тому перевірте, чи вона точно відповідає програмі, написаній вище.

Як це працює

Ви вже помітили, як ми крок за кроком перетворили наш *проєкт* на *код*.

Ми використовуємо модулі `os` і `time`, попередньо імпортувавши їх. Потім ми вказуємо файли та папки для резервного копіювання у списку `джерело`. Цільова папка (англ."The target directory") – це місце, де ми зберігаємо всі файли резервних копій, і це вказано у змінній `цільова_папка`. Ім'я zip-архіву, який ми збираємося створити, — це поточна дата й час, які ми генеруємо за допомогою функції `time.strftime()`. Він (архів) також матиме розширення `.zip` і зберігатиметься в папці `цільова_папка`.

Зверніть увагу на використання змінної `os.sep` - вона має роздільник шляху для папки,відповідно до вашої операційної системи, тобто це буде `'/'` у GNU/Linux, Unix, macOS, і буде `'\\'` у Windows. Використання `os.sep` замість цих символів безпосередньо зробить нашу програму переносимою та працюватиме в усіх цих системах.

Функція `time.strftime()` приймає як аргумент формат виводу часу (рядок певного вигляду), подібну до того, який ми використовували у наведеній вище програмі. Символ формата `%Y` буде замінено роком та століттям. Символ формата `%m` буде замінено місяцем у формі числа від `01` до `12` і так далі. Повний список таких символів формата можна знайти в Довідковому посібнику Python.

Ми створюємо назву цільового zip-файлу за допомогою додаткового оператора, який *конкатенує* (англ."*concatenates*") рядки, тобто об'єднує два рядки разом і повертає новий. Потім ми створюємо рядок `zip_command`, який містить команду, яку ми збираємося виконати. Ви можете перевірити, чи ця команда працює, запустивши її в оболонці (термінал GNU/Linux або командний рядок DOS).

Команда `zip`, яку ми використовуємо, має кілька доступних параметрів, і одним із цих параметрів є `-r`. Параметр `-r` вказує, що команда zip має працювати **р**екурсивно (англ."recursive") для папок, тобто вона має включати всі підпапки та файли. За параметрами слідує ім'я zip-архіву, який потрібно створити, за ним вказується список файлів і папок для резервного копіювання. Ми перетворюємо список джерел (англ."sourse") у рядок за допомогою методу об'єднання рядків (англ."join"), який ми вже бачили, як використовувати.

Потім ми нарешті *запускаємо* команду за допомогою функції `os.system`, яка запускає команду так, ніби її було запущено з *системи*, тобто із командної оболонки - вона повертає `0`, якщо команда виконана успішно, інакше вона повертає номер помилки.

Залежно від результату виконання команди ми друкуємо відповідне повідомлення про те, чи успішним було створення резервних копій чи ні.

Ось і все, ми створили сценарій для збереження резервних копій наших важливих файлів!

Примітка для користувачів Windows

Замість подвійної зворотної скісної риски ви також можете використовувати необроблені рядки. Наприклад, використовуйте `'C:\\Documents'` або `r'C:\Documents'`. Однак *не* використовуйте `'C:\Documents'`, оскільки в результаті ви використовуєте невідому екрановану послідовність `\D`.

Тепер, коли у нас є робочий сценарій резервного копіювання, ми можемо використовувати його, коли захочемо зробити резервну копію файлів. Це називається *операційною* фазою (англ.“operation phase”) або фазою *розгортання* (англ.“deployment phase”) програмного забезпечення.

Наведена вище програма працює належним чином, але (зазвичай) перші програми працюють не зовсім так, як ви очікуєте. Наприклад, можуть виникнути проблеми, якщо ви неправильно розробили програму або якщо ви зробили помилку під час введення коду тощо. Відповідно вам доведеться повернутися до фази проєктування або вам доведеться налагодити свою програму.

3.19.3 Друга версія

Перша версія нашого сценарію працює. Однак ми можемо внести в нього деякі вдосконалення, щоб вона працювала краще щодня. Це називається фазою *супроводу* (англ.“maintenance phase”) програмного забезпечення.

Одне із вдосконалень, яке, як мені здається, буде корисними - це кращий механізм іменування файлів: використання *часу* (англ.“*time*”) в якості імені файлу, що зберігається в папці з поточною *датою* (англ.“*date*”) в якості імені, яке, у свою чергу, розташоване у головній папці для збереження резервного копіювання. Перша перевага полягає в тому, що ваші резервні копії зберігаються в ієрархічному порядку, тому ними набагато легше керувати. Друга перевага полягає в тому, що імена файлів набагато коротші. Третя перевага полягає в тому, що за іменами папок можна легко визначити, в які дні створювалися резервні копії, оскільки папка створюється лише у разі резервного копіювання даних у цей день.

Зберегти як `backup_ver2.py`:

код python backup_ver2_ukr.py

```
import os
import time

# 1. Файли та папки, які потрібно скопіювати, збираються до списку.
# Приклад у Windows:
# джерело = ['"C:\\My Documents"', 'C:\\Code']
# Приклад на Mac OS X та Linux:
джерело = ['/Users/swa/notes']
# # Зауважте, що ми повинні використовувати подвійні лапки всередині рядка
# для імен із пробілами.

# 2. Резервні копії повинні зберігатися у головній папці резервних копій
# Приклад у Windows:
# цільова_папка (англ."target_dir",target directory) = 'E:\\Backup'
# Приклад у  Mac OS X та Linux:
```

```
цільова_папка = '/Users/swa/backup'
# Не забудьте замінити шлях `/Users/swa/backup` вашими власними назвами папок.

# Створіть цільову папку,якщо її не має
if not os.path.exists(цільова_папка):
    os.mkdir(цільова_папка)  # зробити папку

# 3. Резервні копії файлів зберігаються у файлі zip.
# 4. Поточна дата - це ім'я підпапки у головній папці.
сьогодні = цільова_папка  + os.sep + time.strftime('%Y%m%d')
# Поточний час є ім'ям zip-архіву.
зараз = time.strftime('%H%M%S')

# Ім'я для zip-архіву
ціль = сьогодні + os.sep + зараз + '.zip'

# Створіть підпапку,якщо її не має
if not os.path.exists(сьогодні):
    os.mkdir(сьогодні)
    print('Папку успішно створено', сьогодні)

# 5. Ми використовуємо команду zip, щоб помістити файли в zip-архів
zip_command = 'zip -r {0} {1}'.format(ціль,
                                      ' '.join(джерело))

# Запустіть резервне копіювання
print('Zip command є:')
print(zip_command)
print('Запуск:')
if os.system(zip_command) == 0:
    print('Резервна копія успішно створена', ціль)
else:
    print('Створення резервної копії НЕ ВДАЛОСЯ')
```

Висновок:

```
$ python backup_ver2_ukr.py
Успішно створена папка /Users/swa/backup/20140329
Zip command є:
zip -r /Users/swa/backup/20140329/073201.zip /Users/swa/notes
Запуск:
  adding: Users/swa/notes/ (stored 0%)
  adding: Users/swa/notes/blah1.txt (stored 0%)
  adding: Users/swa/notes/blah2.txt (stored 0%)
  adding: Users/swa/notes/blah3.txt (stored 0%)
Резервна копія успішно створена /Users/swa/backup/20140329/073201.zip
```

Як це працює

Більша частина програми залишається незмінною. Зміни полягають у тому, що ми перевіряємо, чи є папка з ім'ям, що відповідає поточній даті, в головній папці зберігання резервних копій за допомогою функції `os.path.exists`. Якщо її не існує, ми створюємо її за допомогою функції `os.mkdir`.

3.19.4 Третя версія

Друга версія працює добре з великою кількістю резервних копій, але коли резервних копій стає багато, стає важко відрізнити, яка копія для чого! Наприклад, ми могли внести серйозні зміни в програму чи презентацію, тепер ми хочемо вказати суть цих змін в назві архіву zip. Цього можна легко досягти, додавши наданий користувачем коментар до назви zip-архіву.

ПОПЕРЕДЖЕННЯ: наступна програма не працює, тому не лякайтеся, просто пройдіть по ній, тому що в ній міститься урок.

код python backup_ver3_ukr.py

```
import os
import time

# 1. Файли та папки, які потрібно скопіювати, збираються до списку.
# Приклад у Windows:
# джерело = ['"C:\\My Documents"', 'C:\\Code']
# Приклад на Mac OS X та Linux:
джерело = ['/Users/swa/notes']
# Зауважте, що ми повинні використовувати подвійні лапки всередині рядка для імен із
↪пробілами.

# 2. Резервні копії повинні зберігатися у головній папці резервних копій
# Приклад у Windows:
# цільова_папка (англ."target_dir",target directory) = 'E:\\Backup'
# Приклад у  Mac OS X та Linux:
цільова_папка = '/Users/swa/backup'
# Не забудьте замінити шлях `/Users/swa/backup` вашими власними назвами папок.

# Створіть цільову папку,якщо її не має
if not os.path.exists(цільова_папка):
    os.mkdir(цільова_папка)  # зробити папку

# 3. Резервні копії файлів зберігаються у файлі zip.
# 4. Поточна дата - це ім'я підпапки у головній папці.
сьогодні = цільова_папка + os.sep + time.strftime('%Y%m%d')
# Поточний час є ім'ям zip-архіву.
зараз = time.strftime('%H%M%S')

# Прийміть коментар від користувача для
# створення назви zip-файлу
коментар = input('Введіть коментар  --> ')
# Перевірте, чи було введено коментар
if len(коментар) == 0:
    ціль = сьогодні + os.sep + зараз + '.zip'
else:
    ціль = сьогодні + os.sep + зараз + '_' +
        коментар.replace(' ', '_') + '.zip'


# Створіть підпапку,якщо її не має
if not os.path.exists(сьогодні):
    os.mkdir(сьогодні)
```

```
    print('Папку успішно створено', сьогодні)


# 5. Ми використовуємо команду zip, щоб помістити файли в zip-архів
zip_command = 'zip -r {0} {1}'.format(ціль,
                                      ' '.join(джерело))

# Запустіть резервне копіювання
print('Zip command є:')
print(zip_command)
print('Запуск:')
if os.system(zip_command) == 0:
    print('Резервна копія успішно створена', ціль)
else:
    print('Створення резервної копії НЕ ВДАЛОСЯ')
```

Висновок:

```
$ python backup_ver3_ukr.py
  File "backup_ver3.py", line 39
    ціль = сьогодні + os.sep + зараз + '_' +
                                       ^
SyntaxError: invalid syntax
```

Як це (не) працює

Ця програма не працює! Python повідомляє, що є синтаксична помилка, яка означає, що сценарій не задовольняє структурі, яку очікує побачити Python. Коли ми спостерігаємо помилку, видану Python, він також повідомляє нам місце, де виявив помилку. Отже, ми починаємо *налагодження* (англ."*debugging*") нашої програми з цього рядка.

Уважно спостерігаючи, ми бачимо, що один логічний рядок був розділений на два фізичних рядка, але ми не вказали, що ці два фізичні рядки є частиною одного. По суті, Python знайшов оператор додавання (+) без жодного операнда в цьому логічному рядку, і тому він не знає, як продовжити. Пам'ятайте, що ми можемо вказати, що логічний рядок продовжується в наступному фізичному рядку, використовуючи зворотну косу риску в кінці фізичного рядка. Отже, ми вносимо цю поправку в нашу програму. Це виправлення програми, коли ми знаходимо помилки, називається *виправленням помилок* (англ."*bug fixing*").

3.19.5 Четверта версія

код python backup_ver4_ukr.py

```
import os
import time

# 1. Файли та папки, які потрібно скопіювати, збираються до списку.
# Приклад у Windows:
# джерело = ['"C:\\My Documents"', 'C:\\Code']
# Приклад на Mac OS X та Linux:
джерело = ['/Users/swa/notes']
# Зауважте, що ми повинні використовувати подвійні лапки всередині рядка для імен із␣
↪пробілами.
```

```
# 2. Резервні копії повинні зберігатися в головній папці резервних копій
# Приклад у Windows:
# цільова_папка (англ."target_dir",target directory) = 'E:\\Backup'
# Приклад у  Mac OS X та Linux:
цільова_папка = '/Users/swa/backup'
# Не забудьте замінити шлях `/Users/swa/backup` вашими власними назвами папок.

# Створіть цільову папку,якщо її не має
if not os.path.exists(цільова_папка):
    os.mkdir(цільова_папка)  # зробити папку

# 3. Резервні копії файлів зберігаються у файлі zip.
# 4. Поточна дата - це ім'я підпапки у головній папці.
сьогодні = цільова_папка + os.sep + time.strftime('%Y%m%d')
# Поточний час є ім'ям zip-архіву.
зараз = time.strftime('%H%M%S')

# Прийміть коментар від користувача для
# створення назви zip-файлу
коментар = input('Введіть коментар  --> ')
# Перевірте, чи було введено коментар
if len(коментар) == 0:
    ціль = сьогодні + os.sep + зараз + '.zip'
else:
    ціль = сьогодні + os.sep + зараз + '_' +\
        коментар.replace(' ', '_') + '.zip'


# Створіть підпапку,якщо її не має
if not os.path.exists(сьогодні):
    os.mkdir(сьогодні)
    print('Папку успішно створено', сьогодні)


# 5. Ми використовуємо команду zip, щоб помістити файли в zip-архів
zip_command = 'zip -r {0} {1}'.format(ціль,
                                      ' '.join(джерело))

# Запустіть резервне копіювання
print('Zip command є:')
print(zip_command)
print('Запуск:')
if os.system(zip_command) == 0:
    print('Резервна копія успішно створена', ціль)
else:
    print('Створення резервної копії НЕ ВДАЛОСЯ')
```

Висновок:

```
$ python backup_ver4_ukr.py
Введіть коментар  --> додав нові приклади
Zip command є:
zip -r /Users/swa/backup/20140329/074122_додав_нові_приклади.zip /Users/swa/notes
```

```
Запуск:
  adding: Users/swa/notes/ (stored 0%)
  adding: Users/swa/notes/blah1.txt (stored 0%)
  adding: Users/swa/notes/blah2.txt (stored 0%)
  adding: Users/swa/notes/blah3.txt (stored 0%)
Резервна копія успішно створена /Users/swa/backup/20140329/074122_додав_нові_приклади.
↪zip
```

Як це працює

Ця програма тепер працює! Розглянемо фактичні вдосконалення, які ми зробили у версії 3. Ми беремо коментарі користувача за допомогою функції `input`, а потім перевіряємо, чи дійсно користувач щось ввів, з'ясовуючи довжину введення за допомогою функції `len`. Якщо користувач щойно натиснув `enter`, не вводячи нічого (можливо, це було просто звичайне резервне копіювання або не було зроблено жодних спеціальних змін), тоді ми продовжуємо, як робили раніше.

Однак, якщо було надано коментар, він додається до імені zip-архіву безпосередньо перед розширенням `.zip`. Зверніть увагу, що ми замінюємо пробіли в коментарях на підкреслення - це тому, що керувати іменами файлів без пробілів набагато легше.

3.19.6 Більше уточнень

Четверта версія є задовільно робочим сценарієм для більшості користувачів, але завжди є місце для вдосконалення. Наприклад, ви можете включити рівень *дослівності* (англ."*verbosity* level") для команди zip, вказавши опцію `-v`, щоб зробити вашу програму більш балакучою (англ."talkative"), або опцію `-q`, щоб зробити її *тихою* (англ."quiet").

Ще одним можливим покращенням була б можливість передавати сценарію інші файли та папки прямо у командному рядку. Ми можемо отримати ці імена зі списку `sys.argv` і додати їх до нашого списку `джерело` за допомогою методу `extend`, який надається класом `список`(англ."list").

Найважливішим удосконаленням було б не використовувати спосіб створення архівів `os.system`, а натомість використовувати zipfile або tarfile - вбудовані модулі для створення цих архівів. Вони є частиною стандартної бібліотеки та вже доступні для використання без зовнішніх залежностей від програми zip, доступної на вашому комп'ютері.

Однак я використовував спосіб `os.system` для створення резервної копії у наведених вище прикладах суто з педагогічною метою, так що приклад достатньо простий, щоб його зрозуміли всі, але достатньо реальний, щоб бути корисним.

Чи можете ви спробувати написати п'яту версію, яка використовує модуль zipfile замість виклику `os.system`?

3.19.7 Процес розробки програмного забезпечення

Зараз ми пройшли різні *етапи* в процесі написання програмного забезпечення. Ці фази можна підсумувати таким чином:

1. Що (Аналіз)
2. Як (Проєктування)
3. Зроби це (Реалізація)
4. Тестування (Тестування та Налагодження)
5. Використання (Оперування та Розгортання)
6. Супровід (Вдосконалення)

Рекомендований спосіб написання програм — це процедура, якої ми дотримувалися під час створення сценарію резервного копіювання: виконайте аналіз і проєктування. Почніть реалізацію з простого варіанту. Протестуйте та налагодьте його. Використовуйте його, щоб переконатися, що він працює належним чином. Тепер додайте будь-які функції, які вам потрібні, і продовжуйте повторювати цикл «Зроби-тест-використай» стільки разів, скільки потрібно.

Пам'ятайте:

> Програмне забезпечення вирощують, а не створюють. – Bill de hÓra

3.19.8 Резюме

Ми побачили, як створювати власні програми/сценарії на Python, а також різні етапи написання таких програм. Вам може бути корисно створити власну програму, як ми робили в цьому розділі, щоб ви навчилися працювати з Python, а також вирішувати проблеми.

Далі ми обговоримо об'єктно-орієнтоване програмування.

3.19.9 – Доповнення від перекладача –

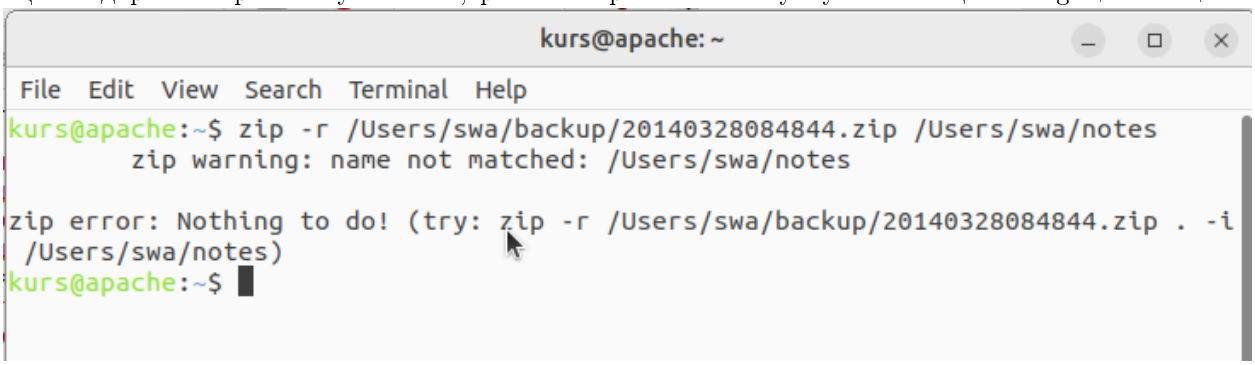
Перший приклад не буде працювати на вашему комп'ютері,якщо ви не змінили Swaroop Path (/Users/swa/notes) на ваш Path.

- *Головна* або *кореневa* папка, каталог (англ."англ.root directory","root folder")- це перша папка в операційній системі. У Linux і MacOS це "/" або "/home/username". У Windows це зазвичай «C:»
- Шлях файлу (англ. "Path") - це рядок із іменами всіх папок (а також підпапок і підпідпапок. . .), починаючи з кореневої папки до поточної папки. Зауважте, що в WINDOWS назви папок відокремлюються зворотною скісною рискою(бекслеш), як-от "C:\carl\documents", але в Linux і MacOS назви папок відокремлюються скісною рискою(слеш): "/home/carl/documents".

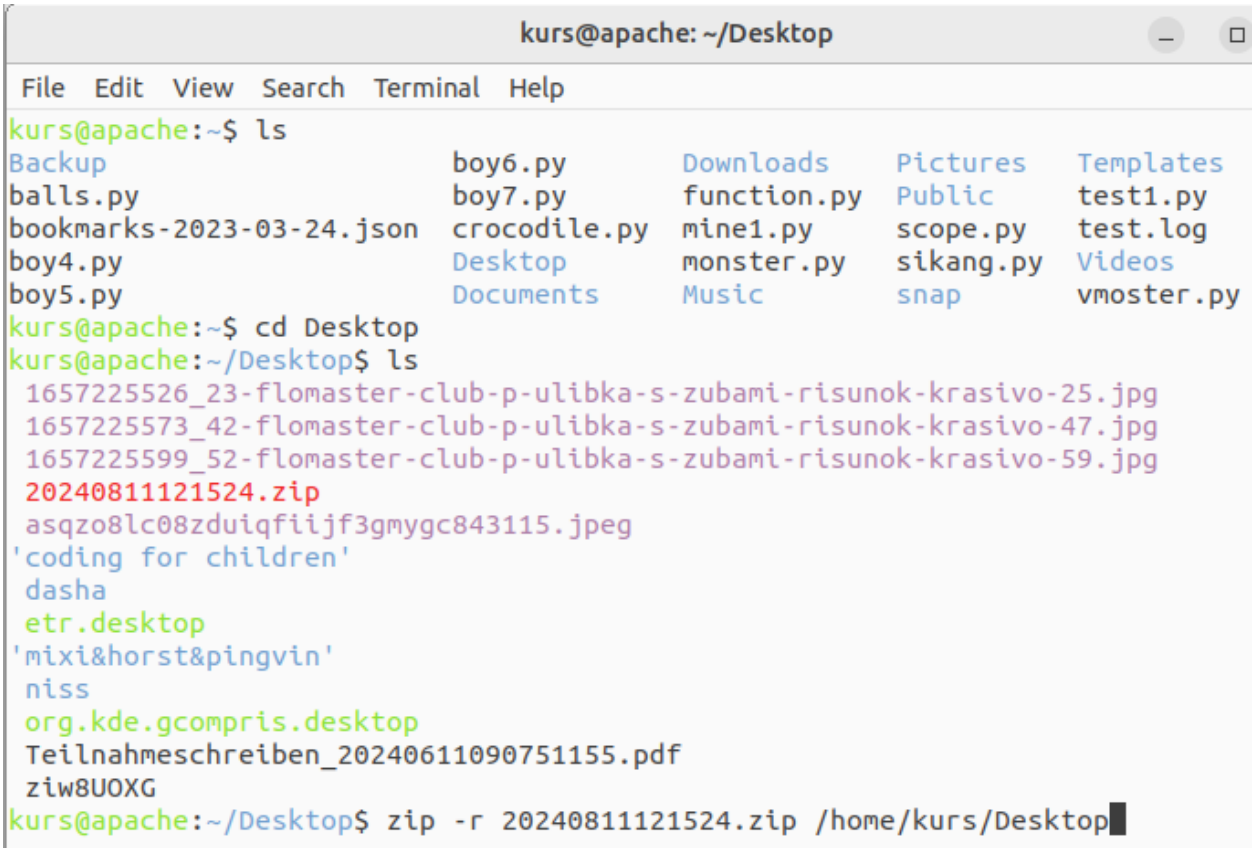
Swaroop пояснює,що якщо наведена вище програма не працює для вас, скопіюйте рядок, надрукований після рядка `Zip command є` у висновку (zip -r /Users/swa/backup/20140328084844.zip /Users/swa/notes), вставте його в оболонку (у GNU/Linux і Mac OS X) / `cmd` (у Windows), подивіться, в чому полягає помилка, і спробуйте її виправити.

Як це зробити?

Щоб відкрити термінал у Linux ,треба використати наступну комбінацію: Strg + Alt + T



Якщо ви хочете перевірити,які файли у вас у головній папці,треба ввести ls (англ."list files"). Якщо хочете перейти в якусь субпапку треба ввести cd (chance directory) та назву субпапки.



На моєму комп'ютері перший приклад працює у наступному виконанні:

```
import os
import time

# 1.Файли та папки, які потрібно скопіювати, збираються до списку.
# Приклад у  Windows:
# джерело = ['"C:\\My Documents"']
# Приклад у  Mac OS X and Linux:
джерело = ['/home/kurs/Documents']
# Зауважте, що ми повинні використовувати подвійні лапки всередині рядка для імен із␣
↪пробілами.
# Ми також могли використати необроблений рядок,
# написавши [r'C:\My Documents'].

# 2. Резервні копії повинні зберігатися у головній папці резервних копій
# Приклад у  Windows:
# цільова_папка (англ."target_dir",target directory = 'E:\\Backup'
# Приклад у  Mac OS X and Linux:
цільова_папка = '/home/kurs/Desktop/byte_of_python_ps'
# Не забудьте замінити шлях `/Users/swa/backup` вашими власними назвами папок.

# 3. Резервні копії файлів зберігаються у файлі zip.
# 4. Ім'я для zip-архіву - поточна дата та час.
ціль = цільова_папка + os.sep + \
        time.strftime('%Y%m%d%H%M%S') + '.zip'
```

(continues on next page)

(continued from previous page)

```
# Створіть цільову папку, якщо її немає
if not os.path.exists(цільова_папка):
    os.mkdir(цільова_папка)  #створити папку

# 5. Ми використовуємо команду zip, щоб помістити файли в zip-архів
zip_command = 'zip -r {0} {1}'.format(ціль,
                                      ' '.join(джерело))

# Запускаємо створення резервної копії
print('Zip command є:')
print(zip_command)
print('Запуск:')
if os.system(zip_command) == 0:
    print('Резервна копія успішно створена', ціль)
else:
    print('Створення резервної копії НЕ ВДАЛОСЯ')
```

Висновок:

```
$ python backup_ver1.py
Zip command є:
zip -r /Users/swa/backup/20140328084844.zip /Users/swa/notes
Запуск:
  adding: home/kurs/Documents/dasha_learning_python/vpython/presentation/fotos/demo/
↪reveal.js-master/examples/math.html (deflated 74%)
  adding: home/kurs/Documents/dasha_learning_python/vpython/presentation/fotos/demo/
↪reveal.js-master/examples/media.html (deflated 64%)

Резервна копія успішно створена  /home/kurs/Desktop/byte_of_python_ps/20240813213524.zip
```

— завершення доповнень від перекладача —

3.20 Об'єктно-орієнтоване програмування

 англійська: *Object Oriented Programming*

Досі наші програми складалися з функцій, тобто блоків операторів, які маніпулюють даними. Такий підхід до створення програм називається процедурно-орієнтованим програмуванням. Існує ще один спосіб організації вашої програми, який полягає в тому, щоб об'єднати дані та функціональність усередині якогось об'єкта. Це називається парадигмою об'єктно-орієнтованого програмування. У більшості випадків ви можете використовувати процедурне програмування, але коли ви пишете великі програми або маєте проблему, яка краще підходить для цього методу, ви можете використовувати методи об'єктно-орієнтованого програмування.

Класи та об'єкти є двома основними аспектами об'єктно-орієнтованого програмування. **Клас** (англ."class") створює новий *тип* (англ."type"), тим часом як **об'єкти** (англ."objects") є **екземплярами** (англ."instances") класу. Аналогічно,коли ми говоримо про змінні типи `int`, це означає, що змінні, які зберігають цілочисельні значення, є екземплярами (об'єктами) класу `int`.

Примітка для програмістів статичної мови

Зауважте, що навіть цілі числа розглядаються як об'єкти (класу `int`), на відміну від C++ і Java (до версії 1.5), де цілі числа є примітивами.

Перегляньте `help(int)` для отримання додаткової інформації про клас.

Програмісти C# і Java 1.5 знайдуть це схожим з концепцією *упаковки* та *распаковки* (англ.» boxing та unboxing»).

Об'єкти можуть зберігати дані за допомогою звичайних змінних, які *належать* об'єкту. Змінні, які належать об'єкту або класу, називаються **полями** (англ."fields"). Об'єкти також можуть мати функціонал, тобто мати функції, які належать до класу. Такі функції називаються **методами** (англ."methods") класу. Ця термінологія важлива, оскільки вона допомагає нам розрізняти незалежні функції і змінні,і ті, що належать до класу чи об'єкта. У сукупності поля та методи можна назвати **атрибутами** (англ."attributess") цього класу.

Поля бувають двох типів - вони можуть належати кожному екземпляру/об'єкту класа або вони можуть належати лише самому класу. Вони називаються **змінними екземпляра** (англ."instance variables") і **змінними класа** (англ."class variables") відповідно.

Клас створюється за допомогою ключового слова class. Поля та методи класу записуються в блоці кода з відступом.

3.20.1 Self

Методи класу мають лише одну конкретну відмінність від звичайних функцій — вони повинні мати додаткове ім'я, яке має бути додано на початку списку параметрів. Однак ви **не надаєте** значення цьому параметру під час виклику методу, це надасть Python. Ця конкретна змінна посилається на сам об'єкт екземпляра класу, і за домовленістю їй має назву `self`.

Незважаючи на те, що ви можете дати будь-яку назву для цього параметра, *наполегливо рекомендовано* використовувати назву `self` - будь-яка інша назва однозначно неприйнятна. Є багато переваг використання стандартної назви - будь-який читач вашої програми одразу впізнає її, і навіть спеціалізовані IDE (інтегровані середовища розробки) можуть допомогти вам, якщо ви використовуєте `self`.

Примітка для програмістів C++/Java/C#

`self` у Python еквівалентний вказівнику `this` у C++ і посиланню `this` у Java та C#.

Вам, мабуть, цікаво, як Python присвоює значення для `self` і чому вам не потрібно давати йому значення. Приклад прояснить це. Скажімо, у вас є клас під назвою `MyClass` і екземпляр цього класу під назвою `myobject`. Коли ви викликаєте метод цього об'єкта як `myobject.method(arg1, arg2)`, Python автоматично перетворює його на `MyClass.method(myobject, arg1, arg2)` — це все, що стосується спеціального `self`.

Це також означає, що якщо у вас є метод, який не приймає аргументів, ви все одно повинні мати один аргумент – `self`.

3.20.2 Класи

 англійська: *Classes*

Найпростіший можливий клас показано в наступному прикладі (збережіть як oop_simplestclass.py).

```
class Person:
    pass  # Порожній блок

p = Person()
print(p)
```

Висновок:

```
<__main__.Person object at 0x10171f518>
```

Як це працює

Ми створюємо новий клас, використовуючи оператор `class` і назву класу. Далі йде блок рядків коду з відступом, які утворюють тіло класу. У цьому випадку ми маємо порожній блок, який позначається оператором `pass`.

Далі ми створюємо об'єкт/екземпляр цього класу, використовуючи ім'я класу, після якого йде пара круглих дужок. (Ми дізнаємося докладніше про інстанціювання у наступному розділі). Для нашої перевірки ми підтверджуємо тип змінної, просто друкуючи її. Вивод на екран повідомляє нам, що ми маємо екземпляр класу `Person` в модулі `__ main __`.

Зверніть увагу, що також друкується адреса пам'яті комп'ютера, де зберігається ваш об'єкт. Адреса матиме інше значення на вашому комп'ютері, оскільки Python може зберігати об'єкт у будь-якому місці.

3.20.3 Методи

Ми вже обговорювали, що класи/об'єкти можуть мати методи, що являють собою функції, за винятком додаткової змінної self. Тепер ми побачимо приклад (збережіть як oop_method.py).

код python oop1_ukr.py

```
class Person:
    def скажи_привіт(self):
        print('Привіт,ти хто?')

p = Person()
p.скажи_привіт()
```

Висновок:

```
Привіт,ти хто?
```

Як це працює

Тут ми бачимо `self` в дії. Зверніть увагу, що метод `скажи_привіт` не приймає параметрів, але все ще має `self` у визначенні функції.

3.20.4 Метод __init__

 англійська: *The __init__ method*

Існує багато методів, які мають особливе значення у класах Python. Зараз ми побачимо значення методу `__init__`.

Метод `__init__` запускається, як тільки об'єкт класу створюється (тобто реалізується). Цей метод корисний для будь-якої *ініціалізації* (тобто передачі початкових значень вашому об'єкту), яку ви хочете зробити з вашим об'єктом. Зверніть увагу на подвійне підкреслення як на початку, так і в кінці імені.

код python oop_init_ukr.py

```
class Person:
    def __init__(self, ім_я):
        self.ім_я = ім_я

    def скажи_привіт(self):
        print('Привіт,моє ім_я', self.ім_я)

p = Person('Swaroop')
p.скажи_привіт()
# Попередні 2 рядки також можна записати як
# Person().скажи_привіт()
```

Висновок:

```
Привіт,моє ім_я Swaroop
```

Як це працює

Тут ми визначаємо метод `__init__` так, щоб він приймав параметр `ім_я`(разом із звичайним `self`). Тут ми просто створюємо нове поле, яке також називається `ім_я`. Зауважте, що це дві різні змінні, хоча обидві мають назву 'ім_я'. Немає жодних проблем, оскільки нотація з крапкою `self.ім_я` означає, що існує щось під назвою "ім_я", яке є частиною об'єкта під назвою "self", а інше `ім_я` є локальною змінною. Оскільки ми чітко вказуємо, яке ім'я маємо на увазі, плутанини немає.

Створюючи новий екземпляр `p` класу `Person`, ми вказуємо ім'я класу, після якого- аргументи в дужках: p = Person('Swaroop').

Ми явно не викликаємо метод `__init__`. У цьому полягає особлива значимість цього методу.

Тепер ми можемо використовувати поле `self.ім_я` у наших методах, що продемонстровано в методі `скажи_привіт`.

3.20.5 Змінні класу та об'єкту

 англійська: *Class And Object Variables*

Ми вже обговорювали функціональну частину класів і об'єктів (тобто методів), тепер давайте дізнаємося про частину даних. Дані, тобто поля, є не чим іншим, як звичайними змінними, які *прив'язані* (англ."bound") до **просторів імен** (англ."namespaces") класів і об'єктів. Це означає, що ці імена дійсні лише в контексті цих класів та об'єктів. Ось чому їх називають `просторами імен` (name spaces).

Існує два типи `полів` - змінні класу та змінні об'єкта, які різняться залежно від того,`належить` змінна класу чи об'єкту *відповідно*.

Змінні класу (англ."Class variables") є спільними – до них можуть отримати доступ усі екземпляри цього класу. Існує лише одна копія змінної класу, і коли будь-який об'єкт вносить зміни до змінної класу, цю зміну побачать усі інші екземпляри.

Змінні об'єкту (англ."Object variables") не є спільними, змінні об'єкту належать кожному окремому об'єкту/екземпляру класу. У цьому випадку кожен об'єкт має власну копію поля, тобто вони не є спільними та жодним чином не пов'язані з полем з тим самим ім'ям в іншому екземплярі. Приклад допоможе зрозуміти це (збережіть як oop_objvar.py):

код python oop_objvar_ukr.py

```
class Робот:
    """Представляє робота з ім'ям."""

    # Змінна класу,яка підраховує кількість роботів
    населення = 0

    def __init__(self, ім_я):
        """Створення (initialization -ініціалізація) даних."""
        self.ім_я = ім_я
        print("(Ініціалізація {})".format(self.ім_я))

        # При створенні цієї особи, робот
        # додається до змінної 'населення'
        Робот.населення += 1

    def вмирати(self):
        """Я вмираю."""
        print("{} знищується".format(self.ім_я))

        Робот.населення -= 1

        if Робот.населення == 0:
            print("{} я був останній.".format(self.ім_я))
        else:
            print("Все ще є {:d} робочий робот.".format(
                Робот.населення))

    def скажи_привіт(self):
        """Привітання від робота..

        Так, вони можуть це зробити."""
        print("Вітаю, мої господарі називають мене  {}.".format(self.ім_я))

    @classmethod
    def скільки(cls):
        """Друкує поточне населення."""
        print("У нас є {:d} робот.".format(cls.населення))


droid1 = Робот("R2-D2")
droid1.скажи_привіт()
Робот.скільки()
```

```
droid2 = Робот("C-3PO")
droid2.скажи_привіт()
Робот.скільки()

print("\nРоботи можуть виконувати тут певну роботу.\n")

print("Роботи закінчили свою роботу. Тож давайте їх знищимо.")
droid1.вмирати()
droid2.вмирати()
Робот.скільки()
```

Висновок:

```
(Створення R2-D2)
Вітаю, мої господарі називають мене  R2-D2.
У нас є 1 робот.
(Створення C-3PO)
Вітаю, мої господарі називають мене  C-3PO.
У нас є 2 робот.

Роботи можуть виконувати тут певну роботу.

Роботи закінчили свою роботу. Тож давайте їх знищимо.
R2-D2 знищується
Все ще є 1 робочий робот.
C-3PO знищується
C-3PO я був останній.
У нас є 0 робот.
```

Як це працює

Це довгий приклад, але він допомагає продемонструвати природу змінних класу та об'єкта. Тут `населення` належить до класу `Робот` і, отже, є змінною класу. Змінна `ім_я` належить об'єкту (їй присвоюється значення за допомогою `self`) і, отже, є змінною об'єкта.

Таким чином, ми звертаємося до змінної класу населення як `Робот.населення`, а `не self.населення`. До змінної ж об'єкта `ім_я` у всіх методах цього об'єкта ми звертаємося за допомогою позначення `self.ім_я`. Запам'ятайте цю просту різницю між змінними класу та об'єкта. Також зауважте, що змінна об'єкта з тим же іменем, що й змінна класу, приховає змінну класу!

Замість `Робот.населення` ми також могли б використати `self.__class__.населення`, оскільки кожен об'єкт посилається на свій клас через атрибут `self.__class__`.

`Скільки` насправді є методом, який належить до класу, а не до об'єкта. Це означає, що ми можемо визначити його як `classmethod` або `staticmethod` залежно від того, чи потрібно нам знати, у якому класі ми знаходимося. Оскільки ми посилаємося на змінну класу, давайте використаємо `classmethod`.

Ми позначили метод `скільки` як метод класу за допомогою `decorator`.

Декоратори можна уявити як ярлик для виклику функції-обгортки (англ."wrapper function") (тобто функції, яка «обгортається» навколо іншої функції, щоб вона могла робити щось до або після внутрішньої функції), тому застосування декоратора `@classmethod` є таким самим, як і функція виклику:

```
скільки = classmethod(скільки)
```

Зверніть увагу, що метод `__init__` використовується для ініціалізації екземпляра `Робот` з ім'ям. У цьому методі ми збільшуємо кількість `населення` на 1, оскільки ми додаємо ще одного робота. Також

зауважте, що значення `self.ім_я` для кожного об'єкта свої, що свідчить про природу змінних об'єкта.

Пам'ятайте, що ви повинні звертатися до змінних і методів того самого об'єкта, використовуючи тільки `self`. Це називається `посиланням на атрибут` (англ."attribute reference.").

У цій програмі ми також бачимо використання рядків `документації` (англ."docstrings") для класів, а також для методів. Під час виконання ми можемо звертатись до рядка документації класу за допомогою `Робот.__doc__` ,а до рядка документації методу – за допомогою `Robot.скажи_привіт.__doc__`

У методі `вмирати` ми просто зменшуємо кількість `Робот.населення` на 1.

Усі члени класу є відкритими. Один виняток: якщо ви використовуєте елементи даних з іменами, що використовують префікс *подвійного підкреслення*,(англ.»double underscore prefix «) наприклад `__privatevar`, Python використовує спотворення імен (англ."name-mangling"), щоб ефективно зробити їх приватною змінною.

Таким чином, висновок полягає в тому, що будь-яка змінна, яка має використовуватися лише в межах класу чи об'єкта, повинна починатися з підкреслення, а всі інші імена є загальнодоступними та можуть використовуватися іншими класами/об'єктами. Пам'ятайте, що це лише домовленість, і Python її не вимагає (за винятком подвійного префікса підкреслення).

Примітка для програмістів C++/Java/C#

Усі члени класу (включно з членами даних) є *загальнодоступними* (англ."public"), а всі методи — *віртуальними* (англ."virtual") у Python.

3.20.6 Наслідування

 англійська: *Inheritance*

Однією з головних переваг об'єктно-орієнтованого програмування є **багаторазове використання** (англ."reuse") одного і того ж коду, і один із способів цього досягти - за допомогою механізму **наслідування** (англ. "inheritance"). Наслідування найкраще можна уявити у вигляді відношення між класами як **тип та підтип** (англ.» type and subtype»).

Припустімо, ви хочете написати програму, яка повинна відстежувати вчителів і студентів у коледжі. Вони мають деякі спільні характеристики, такі як ім'я, вік та адреса. Вони також мають певні характеристики, такі як зарплата, курси та відпустки для вчителів, а також оцінки та гонорари для студентів.

Можна створити для них незалежні класи та працювати з ними, але тоді додавання будь-якої нової загальної характеристики вимагатиме додавання її до кожного з цих незалежних класів окремо.Це швидко стає громіздким.

Кращим способом було б створити загальний клас під назвою `УчастникШколи`, а потім зробити так, щоб класи викладача і студента `успадкували` цей клас, тобто вони стануть підтипами цього типу (класу), і тоді ми зможемо додати певні характеристики до цих підтипів -типів.

Цей підхід має багато переваг. Якщо ми додаємо/змінюємо будь-яку функціональність в `УчастникШколи`, це також автоматично відображається в підтипах. Наприклад, ви можете додати нове поле ідентифікаційної картки як для вчителів, так і для студентів, просто додавши його до класу УчастникШколи. Однак зміни в підтипах не впливають на інші підтипи. Ще одна перевага полягає в тому, що ви можете звертатися до об'єкта «вчитель» або «студент» як до об'єкта `УчастникШколи`, що може бути корисним у деяких ситуаціях, наприклад підрахунок кількості учасників школи. Це називається **поліморфізмом** (англ."polymorphism"), коли підтип можна підставити у місці, де очікується батьківський тип, тобто об'єкт вважається екземпляром батьківського класу.

Також зауважте, що ми повторно використовуємо код батьківського класу, і нам не потрібно повторювати його в різних класах, як нам довелося б, якби ми використовували незалежні класи.

Клас `УчастникШколу` цій ситуації відомий як **базовий клас** (англ.“base class”) або **суперклас** (англ.» superclass»). Класи `Вчитель` і `Студент` називаються **похідними класами** (англ.“derived classes”) або **підкласами** (англ.» subclasses»).

Тепер ми побачимо цей приклад на англійській мові як програму (збережіть як oop_subclass.py):

українська

код python oop_subclass_ukr.py

```
class УчастникШколи:
    '''Представляє будь-якого участника школи.'''
    def __init__(self, ім_я, вік):
        self.ім_я = ім_я
        self.вік = вік
        print('(Створено УчастникШколи: {})'.format(self.ім_я))

    def повідомити(self):
        '''Повідомити деталі.'''
        print('Ім_я:"{}" Вік:"{}"'.format(self.ім_я, self.вік), end=" ")


class Викладач(УчастникШколи):
    '''Представляє викладача.'''
    def __init__(self, ім_я, вік, зарплата):
        УчастникШколи.__init__(self, ім_я, вік)
        self.зарплата = зарплата
        print('(Створено Викладач: {})'.format(self.ім_я))

    def повідомити(self):
        УчастникШколи.повідомити(self)
        print('зарплата: "{:d}"'.format(self.зарплата))


class Студент(УчастникШколи):
    '''Представляє студента.'''
    def __init__(self, ім_я, вік, оцінки):
        УчастникШколи.__init__(self, ім_я, вік)
        self.оцінки = оцінки
        print('(Створено студент: {})'.format(self.ім_я))

    def повідомити(self):
        УчастникШколи.повідомити(self)
        print('Оцінки: "{:d}"'.format(self.оцінки))

t = Викладач('Mrs. Shrividya', 40, 30000)
s = Студент('Swaroop', 25, 75)

# друкує порожній рядок
print()

учасники = [t, s]
```

(continues on next page)

(continued from previous page)

```
for учасники in учасники:
    # Працює як для вчителів, так і для студентів
    учасники.повідомити()
```

Висновок:

```
(Створено УчастникШколи: Mrs. Shrividya)
(Створено Викладач: Mrs. Shrividya)
(Створено УчастникШколи: Swaroop)
(Створено студент: Swaroop)

Ім_я:"Mrs. Shrividya" Вік:"40" зарплата: "30000"
Ім_я:"Swaroop" Вік:"25" Оцінки: "75"
```

англійська

python code *oop_subclass_en.py*

```
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('(Initialized SchoolMember: {})'.format(self.name))

    def tell(self):
        '''Tell my details.'''
        print('Name:"{}" Age:"{}"'.format(self.name, self.age), end=" ")


class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('(Initialized Teacher: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Salary: "{:d}"'.format(self.salary))


class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print('(Initialized Student: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Marks: "{:d}"'.format(self.marks))
```

(continues on next page)

(continued from previous page)

```
t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 25, 75)

# prints a blank line
print()

members = [t, s]
for member in members:
    # Works for both Teachers and Students
    member.tell()
```

output:

```
(Initialized SchoolMember: Mrs. Shrividya)
(Initialized Teacher: Mrs. Shrividya)
(Initialized SchoolMember: Swaroop)
(Initialized Student: Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"25" Marks: "75"
```

Як це працює

Щоб використовувати наслідування,при визначенні класу ми вказуємо імена його базових класів у вигляді кортежу,який йде відразу за його назвою (наприклад,`class Викладач (УчастникШколи)`. Далі ми спостерігаємо, що метод `__init__` базового класу явно викликається за допомогою змінної `self`, щоб ми могли ініціалізувати частину об'єкта, що відноситься до базового класу. Це дуже важливо пам'ятати,оскільки ми визначаємо метод `__init__` у підкласах Викладач та Студент, Python не викликає автоматично конструктор базового класу `УчастникШколи`, ви повинні викликати його самостійно у явному вигляді.

Навпаки, якщо ми не визначили метод `__init__` у підкласі, Python автоматично викличе конструктор базового класу.

Хоча ми могли б обробляти екземпляри `Викладач` або `Студент` так само, як екземпляр `УчастникШколи`, і отримати доступ до методу `повідомити` класа `УчастникШколи`, просто ввівши `Викладач.повідомити` або `Студент.повідомити`.Натомість ми визначаємо інший метод `повідомити` у кожному підкласі (використовуючи метод `повідомити` в класі `УчастникШколи` для його частини), щоб пристосувати його для цього підкласу. Оскільки ми це зробили, написавши `Викладач.повідомити`, Python використовує метод `повідомити` для цього підкласу проти суперкласу. Однак, якби у нас не було методу `повідомити` у підкласі, Python використовував би метод `повідомити` у суперкласі. Python завжди спочатку починає шукати методи у фактичному типі підкласу, і якщо він нічого не знаходить, він починає шукати методи в базових класах підкласу, один за іншим у тому порядку, в якому вони вказані в кортежі (тут у нас є лише 1 базовий клас, але ви можете мати кілька базових класів) у визначенні класу.

Примітка щодо термінології: якщо в кортежі наслідування зазначено більше одного класу, це називається **множинним наслідуванням** (англ."multiple inheritance").

Параметр `end` використовується у функції `print` у методі `повідомити()` суперкласу, щоб надрукувати рядок і дозволити наступному друку продовжуватись у тому самому рядку. Це трюк, щоб змусити `print` не друкувати символ `\n` (новий рядок) у кінці друку.

3.20.7 Резюме

Зараз ми дослідили різні аспекти класів і об'єктів, а також різну термінологію, пов'язану з ними. Ми також побачили переваги та підводні камені об'єктно-орієнтованого програмування. Python дуже об'єктно-орієнтований, і ретельне розуміння цих концепцій дуже допоможе вам у довгостроковій перспективі.

Далі ми дізнаємося, як працювати з введенням/виведенням і як отримувати доступ до файлів у Python.

3.21 Об'єктно-орієнтоване програмування

 англійська: *Object Oriented Programming*

Досі наші програми складалися з функцій, тобто блоків операторів, які маніпулюють даними. Такий підхід до створення програм називається процедурно-орієнтованим програмуванням. Існує ще один спосіб організації вашої програми, який полягає в тому, щоб об'єднати дані та функціональність усередині якогось об'єкта. Це називається парадигмою об'єктно-орієнтованого програмування. У більшості випадків ви можете використовувати процедурне програмування, але коли ви пишете великі програми або маєте проблему, яка краще підходить для цього методу, ви можете використовувати методи об'єктно-орієнтованого програмування.

Класи та об'єкти є двома основними аспектами об'єктно-орієнтованого програмування. **Клас** (англ."class") створює новий *тип* (англ."type"), тим часом як **об'єкти** (англ."objects") є **екземплярами** (англ."instances") класу. Аналогічно,коли ми говоримо про змінні типи `int`, це означає, що змінні, які зберігають цілочисельні значення, є екземплярами (об'єктами) класу `int`.

Примітка для програмістів статичної мови

Зауважте, що навіть цілі числа розглядаються як об'єкти (класу `int`), на відміну від C++ і Java (до версії 1.5), де цілі числа є примітивами.

Перегляньте `help(int)` для отримання додаткової інформації про клас.

Програмісти C# і Java 1.5 знайдуть це схожим з концепцією *упаковки* та *распаковки* (англ.» boxing та unboxing»).

Об'єкти можуть зберігати дані за допомогою звичайних змінних, які *належать* об'єкту. Змінні, які належать об'єкту або класу, називаються **полями** (англ."fields"). Об'єкти також можуть мати функціонал, тобто мати функції, які належать до класу. Такі функції називаються **методами** (англ."methods") класу. Ця термінологія важлива, оскільки вона допомагає нам розрізняти незалежні функції і змінні,і ті, що належать до класу чи об'єкта. У сукупності поля та методи можна назвати **атрибутами** (англ."attributess") цього класу.

Поля бувають двох типів - вони можуть належати кожному екземпляру/об'єкту класа або вони можуть належати лише самому класу. Вони називаються **змінними екземпляра** (англ."instance variables") і **змінними класа** (англ."class variables") відповідно.

Клас створюється за допомогою ключового слова class. Поля та методи класу записуються в блоці кода з відступом.

3.21.1 Self

Методи класу мають лише одну конкретну відмінність від звичайних функцій — вони повинні мати додаткове ім'я, яке має бути додано на початку списку параметрів. Однак ви **не надаєте** значення

цьому параметру під час виклику методу, це надасть Python. Ця конкретна змінна посилається на сам об'єкт екземпляра класу, і за домовленістю їй має назву `self`.

Незважаючи на те, що ви можете дати будь-яку назву для цього параметра, *наполегливо рекомендовано* використовувати назву `self` - будь-яка інша назва однозначно неприйнятна. Є багато переваг використання стандартної назви - будь-який читач вашої програми одразу впізнає її, і навіть спеціалізовані IDE (інтегровані середовища розробки) можуть допомогти вам, якщо ви використовуєте `self`.

Примітка для програмістів C++/Java/C#

`self` у Python еквівалентний вказівнику `this` у C++ і посиланню `this` у Java та C#.

Вам, мабуть, цікаво, як Python присвоює значення для `self` і чому вам не потрібно давати йому значення. Приклад прояснить це. Скажімо, у вас є клас під назвою `MyClass` і екземпляр цього класу під назвою `myobject`. Коли ви викликаєте метод цього об'єкта як `myobject.method(arg1, arg2)`, Python автоматично перетворює його на `MyClass.method(myobject, arg1, arg2)` — це все, що стосується спеціального `self`.

Це також означає, що якщо у вас є метод, який не приймає аргументів, ви все одно повинні мати один аргумент – `self`.

3.21.2 Класи

 англійська: *Classes*

Найпростіший можливий клас показано в наступному прикладі (збережіть як oop_simplestclass.py).

```
class Person:
    pass  # Порожній блок

p = Person()
print(p)
```

Висновок:

```
<__main__.Person object at 0x10171f518>
```

Як це працює

Ми створюємо новий клас, використовуючи оператор `class` і назву класу. Далі йде блок рядків коду з відступом, які утворюють тіло класу. У цьому випадку ми маємо порожній блок, який позначається оператором `pass`.

Далі ми створюємо об'єкт/екземпляр цього класу, використовуючи ім'я класу, після якого йде пара круглих дужок. (Ми дізнаємося докладніше про інстанціювання у наступному розділі). Для нашої перевірки ми підтверджуємо тип змінної, просто друкуючи її. Вивод на екран повідомляє нам, що ми маємо екземпляр класу `Person` в модулі `__ main __`.

Зверніть увагу, що також друкується адреса пам'яті комп'ютера, де зберігається ваш об'єкт. Адреса матиме інше значення на вашому комп'ютері, оскільки Python може зберігати об'єкт у будь-якому місці.

3.21.3 Методи

Ми вже обговорювали, що класи/об'єкти можуть мати методи, що являють собою функції, за винятком додаткової змінної self. Тепер ми побачимо приклад (збережіть як oop_method.py).

код python oop1_ukr.py

```
class Person:
    def скажи_привіт(self):
        print('Привіт,ти хто?')

p = Person()
p.скажи_привіт()
```

Висновок:

```
Привіт,ти хто?
```

Як це працює

Тут ми бачимо `self` в дії. Зверніть увагу, що метод `скажи_привіт` не приймає параметрів, але все ще має `self` у визначенні функції.

3.21.4 Метод __init__

 англійська: *The __init__ method*

Існує багато методів, які мають особливе значення у класах Python. Зараз ми побачимо значення методу `__init__`.

Метод `__init__` запускається, як тільки об'єкт класу створюється (тобто реалізується). Цей метод корисний для будь-якої *ініціалізації* (тобто передачі початкових значень вашому об'єкту), яку ви хочете зробити з вашим об'єктом. Зверніть увагу на подвійне підкреслення як на початку, так і в кінці імені.

код python oop_init_ukr.py

```
class Person:
    def __init__(self, ім_я):
        self.ім_я = ім_я

    def скажи_привіт(self):
        print('Привіт,моє ім_я', self.ім_я)

p = Person('Swaroop')
p.скажи_привіт()
# Попередні 2 рядки також можна записати як
# Person().скажи_привіт()
```

Висновок:

```
Привіт,моє ім_я Swaroop
```

Як це працює

Тут ми визначаємо метод `__init__` так, щоб він приймав параметр `ім_я`(разом із звичайним `self`). Тут ми просто створюємо нове поле, яке також називається `ім_я`. Зауважте, що це дві різні змінні, хоча обидві мають назву 'ім_я'. Немає жодних проблем, оскільки нотація з крапкою `self.ім_я` означає, що існує щось під назвою "ім_я", яке є частиною об'єкта під назвою "self", а інше `ім_я` є локальною змінною. Оскільки ми чітко вказуємо, яке ім'я маємо на увазі, плутанини немає.

Створюючи новий екземпляр `p` класу `Person`, ми вказуємо ім'я класу, після якого- аргументи в дужках: p = Person('Swaroop').

Ми явно не викликаємо метод `__init__`. У цьому полягає особлива значимість цього методу.

Тепер ми можемо використовувати поле `self.ім_я` у наших методах, що продемонстровано в методі `скажи_привіт`.

3.21.5 Змінні класу та об'єкту

 англійська: *Class And Object Variables*

Ми вже обговорювали функціональну частину класів і об'єктів (тобто методів), тепер давайте дізнаємося про частину даних. Дані, тобто поля, є не чим іншим, як звичайними змінними, які *прив'язані* (англ."bound") до **просторів імен** (англ."namespaces") класів і об'єктів. Це означає, що ці імена дійсні лише в контексті цих класів та об'єктів. Ось чому їх називають `просторами імен` (name spaces).

Існує два типи `полів` - змінні класу та змінні об'єкта, які різняться залежно від того,`належить` змінна класу чи об'єкту *відповідно*.

Змінні класу (англ."Class variables") є спільними – до них можуть отримати доступ усі екземпляри цього класу. Існує лише одна копія змінної класу, і коли будь-який об'єкт вносить зміни до змінної класу, цю зміну побачать усі інші екземпляри.

Змінні об'єкту (англ."Object variables") не є спільними, змінні об'єкту належать кожному окремому об'єкту/екземпляру класу. У цьому випадку кожен об'єкт має власну копію поля, тобто вони не є спільними та жодним чином не пов'язані з полем з тим самим ім'ям в іншому екземплярі. Приклад допоможе зрозуміти це (збережіть як oop_objvar.py):

код python oop_objvar_ukr.py

```
class Робот:
    """Представляє робота з ім'ям."""

    # Змінна класу,яка підраховує кількість роботів
    населення = 0

    def __init__(self, ім_я):
        """Створення (initialization -ініціалізація) даних."""
        self.ім_я = ім_я
        print("(Ініціалізація {})".format(self.ім_я))

        # При створенні цієї особи, робот
        # додається до змінної 'населення'
        Робот.населення += 1

    def вмирати(self):
        """Я вмираю."""
        print("{} знищується".format(self.ім_я))
```

```
        Робот.населення -= 1

        if Робот.населення == 0:
            print("{} я був останній.".format(self.ім_я))
        else:
            print("Все ще є {:d} робочий робот.".format(
                Робот.населення))

    def скажи_привіт(self):
        """Привітання від робота..

        Так, вони можуть це зробити."""
        print("Вітаю, мої господарі називають мене  {}.".format(self.ім_я))

    @classmethod
    def скільки(cls):
        """Друкує поточне населення."""
        print("У нас є {:d} робот.".format(cls.населення))


droid1 = Робот("R2-D2")
droid1.скажи_привіт()
Робот.скільки()

droid2 = Робот("C-3PO")
droid2.скажи_привіт()
Робот.скільки()

print("\nРоботи можуть виконувати тут певну роботу.\n")

print("Роботи закінчили свою роботу. Тож давайте їх знищимо.")
droid1.вмирати()
droid2.вмирати()
Робот.скільки()
```

Висновок:

```
(Створення R2-D2)
Вітаю, мої господарі називають мене  R2-D2.
У нас є 1 робот.
(Створення C-3PO)
Вітаю, мої господарі називають мене  C-3PO.
У нас є 2 робот.

Роботи можуть виконувати тут певну роботу.

Роботи закінчили свою роботу. Тож давайте їх знищимо.
R2-D2 знищується
Все ще є 1 робочий робот.
C-3PO знищується
C-3PO я був останній.
У нас є 0 робот.
```

Як це працює

Це довгий приклад, але він допомагає продемонструвати природу змінних класу та об'єкта. Тут `населення` належить до класу `Робот` і, отже, є змінною класу. Змінна `ім_я` належить об'єкту (їй присвоюється значення за допомогою `self`) і, отже, є змінною об'єкта.

Таким чином, ми звертаємося до змінної класу населення як `Робот.населення`, а `не self.населення`. До змінної ж об'єкта `ім_я` у всіх методах цього об'єкта ми звертаємося за допомогою позначення `self.ім_я`. Запам'ятайте цю просту різницю між змінними класу та об'єкта. Також зауважте, що змінна об'єкта з тим же іменем, що й змінна класу, приховає змінну класу!

Замість `Робот.населення` ми також могли б використати `self.__class__.населення`, оскільки кожен об'єкт посилається на свій клас через атрибут `self.__class__`.

`Скільки` насправді є методом, який належить до класу, а не до об'єкта. Це означає, що ми можемо визначити його як `classmethod` або `staticmethod` залежно від того, чи потрібно нам знати, у якому класі ми знаходимося. Оскільки ми посилаємося на змінну класу, давайте використаємо `classmethod`.

Ми позначили метод `скільки` як метод класу за допомогою `decorator`.

Декоратори можна уявити як ярлик для виклику функції-обгортки (англ."wrapper function") (тобто функції, яка «обгортається» навколо іншої функції, щоб вона могла робити щось до або після внутрішньої функції), тому застосування декоратора `@classmethod` є таким самим, як і функція виклику:

```
скільки = classmethod(скільки)
```

Зверніть увагу, що метод `__init__` використовується для ініціалізації екземпляра `Робот` з ім'ям. У цьому методі ми збільшуємо кількість `населення` на 1, оскільки ми додаємо ще одного робота. Також зауважте, що значення `self.ім_я` для кожного об'єкта свої, що свідчить про природу змінних об'єкта.

Пам'ятайте, що ви повинні звертатися до змінних і методів того самого об'єкта, використовуючи тільки `self`. Це називається `посиланням на атрибут` (англ."attribute reference.").

У цій програмі ми також бачимо використання рядків `документації` (англ."docstrings") для класів, а також для методів. Під час виконання ми можемо звертатись до рядка документації класу за допомогою `Робот.__doc__` ,а до рядка документації методу – за допомогою `Robot.скажи_привіт.__doc__`

У методі `вмирати` ми просто зменшуємо кількість `Робот.населення` на 1.

Усі члени класу є відкритими. Один виняток: якщо ви використовуєте елементи даних з іменами, що використовують префікс *подвійного підкреслення*,(англ.»double underscore prefix «) наприклад `__privatevar`, Python використовує спотворення імен (англ."name-mangling"), щоб ефективно зробити їх приватною змінною.

Таким чином, висновок полягає в тому, що будь-яка змінна, яка має використовуватися лише в межах класу чи об'єкта, повинна починатися з підкреслення, а всі інші імена є загальнодоступними та можуть використовуватися іншими класами/об'єктами. Пам'ятайте, що це лише домовленість, і Python її не вимагає (за винятком подвійного префікса підкреслення).

Примітка для програмістів C++/Java/C#

Усі члени класу (включно з членами даних) є *загальнодоступними* (англ.“public”), а всі методи — *віртуальними* (англ.“virtual”) у Python.

3.21.6 Наслідування

 англійська: *Inheritance*

Однією з головних переваг об’єктно-орієнтованого програмування є **багаторазове використання** (англ.“reuse”) одного і того ж коду, і один із способів цього досягти - за допомогою механізму **наслідування** (англ. “inheritance”). Наслідування найкраще можна уявити у вигляді відношення між класами як **тип та підтип** (англ.» type and subtype»).

Припустімо, ви хочете написати програму, яка повинна відстежувати вчителів і студентів у коледжі. Вони мають деякі спільні характеристики, такі як ім’я, вік та адреса. Вони також мають певні характеристики, такі як зарплата, курси та відпустки для вчителів, а також оцінки та гонорари для студентів.

Можна створити для них незалежні класи та працювати з ними, але тоді додавання будь-якої нової загальної характеристики вимагатиме додавання її до кожного з цих незалежних класів окремо.Це швидко стає громіздким.

Кращим способом було б створити загальний клас під назвою `УчастникШколи`, а потім зробити так, щоб класи викладача і студента `успадкували` цей клас, тобто вони стануть підтипами цього типу (класу), і тоді ми зможемо додати певні характеристики до цих підтипів -типів.

Цей підхід має багато переваг. Якщо ми додаємо/змінюємо будь-яку функціональність в `УчастникШколи`, це також автоматично відображається в підтипах. Наприклад, ви можете додати нове поле ідентифікаційної картки як для вчителів, так і для студентів, просто додавши його до класу УчастникШколи. Однак зміни в підтипах не впливають на інші підтипи. Ще одна перевага полягає в тому, що ви можете звертатися до об’єкта «вчитель» або «студент» як до об’єкта `УчастникШколи`, що може бути корисним у деяких ситуаціях, наприклад підрахунок кількості учасників школи. Це називається **поліморфізмом** (англ.“polymorphism”), коли підтип можна підставити у місці, де очікується батьківський тип, тобто об’єкт вважається екземпляром батьківського класу.

Також зауважте, що ми повторно використовуємо код батьківського класу, і нам не потрібно повторювати його в різних класах, як нам довелося б, якби ми використовували незалежні класи.

Клас `УчастникШколу` цій ситуації відомий як **базовий клас** (англ.“base class”) або **суперклас** (англ.» superclass»). Класи `Вчитель` і `Студент` називаються **похідними класами** (англ.“derived classes”) або **підкласами** (англ.» subclasses»).

Тепер ми побачимо цей приклад на англійській мові як програму (збережіть як oop_subclass.py):

українська

код python oop_subclass_ukr.py

```
class УчастникШколи:
    '''Представляє будь-якого участника школи.'''
    def __init__(self, ім_я, вік):
        self.ім_я = ім_я
        self.вік = вік
        print('(Створено УчастникШколи: {})'.format(self.ім_я))
```

(continues on next page)

(continued from previous page)

```
    def повідомити(self):
        '''Повідомити деталі.'''
        print('Ім_я:"{}" Вік:"{}"'.format(self.ім_я, self.вік), end=" ")


class Викладач(УчастникШколи):
    '''Представляє викладача.'''
    def __init__(self, ім_я, вік, зарплата):
        УчастникШколи.__init__(self, ім_я, вік)
        self.зарплата = зарплата
        print('(Створено Викладач: {})'.format(self.ім_я))

    def повідомити(self):
        УчастникШколи.повідомити(self)
        print('зарплата: "{:d}"'.format(self.зарплата))


class Студент(УчастникШколи):
    '''Представляє студента.'''
    def __init__(self, ім_я, вік, оцінки):
        УчастникШколи.__init__(self, ім_я, вік)
        self.оцінки = оцінки
        print('(Створено студент: {})'.format(self.ім_я))

    def повідомити(self):
        УчастникШколи.повідомити(self)
        print('Оцінки: "{:d}"'.format(self.оцінки))

t = Викладач('Mrs. Shrividya', 40, 30000)
s = Студент('Swaroop', 25, 75)

# друкує порожній рядок
print()

учасники = [t, s]
for учасники in учасники:
    # Працює як для вчителів, так і для студентів
    учасники.повідомити()
```

Висновок:

```
(Створено УчастникШколи: Mrs. Shrividya)
(Створено Викладач: Mrs. Shrividya)
(Створено УчастникШколи: Swaroop)
(Створено студент: Swaroop)

Ім_я:"Mrs. Shrividya" Вік:"40" зарплата: "30000"
Ім_я:"Swaroop" Вік:"25" Оцінки: "75"
```

англійська

python code *oop_subclass_en.py*

```
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('(Initialized SchoolMember: {})'.format(self.name))

    def tell(self):
        '''Tell my details.'''
        print('Name:"{}" Age:"{}"'.format(self.name, self.age), end=" ")


class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('(Initialized Teacher: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Salary: "{:d}"'.format(self.salary))


class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print('(Initialized Student: {})'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Marks: "{:d}"'.format(self.marks))

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 25, 75)

# prints a blank line
print()

members = [t, s]
for member in members:
    # Works for both Teachers and Students
    member.tell()
```

output:

```
(Initialized SchoolMember: Mrs. Shrividya)
(Initialized Teacher: Mrs. Shrividya)
```

(continues on next page)

(continued from previous page)

```
(Initialized SchoolMember: Swaroop)
(Initialized Student: Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000"
Name:"Swaroop" Age:"25" Marks: "75"
```

Як це працює

Щоб використовувати наслідування,при визначенні класу ми вказуємо імена його базових класів у вигляді кортежу,який йде відразу за його назвою (наприклад,`class Викладач (УчастникШколи)`). Далі ми спостерігаємо, що метод `__init__` базового класу явно викликається за допомогою змінної `self`, щоб ми могли ініціалізувати частину об'єкта, що відноситься до базового класу. Це дуже важливо пам'ятати,оскільки ми визначаємо метод `__init__` у підкласах Викладач та Студент, Python не викликає автоматично конструктор базового класу `УчастникШколи`, ви повинні викликати його самостійно у явному вигляді.

Навпаки, якщо ми не визначили метод `__init__` у підкласі, Python автоматично викличе конструктор базового класу.

Хоча ми могли б обробляти екземпляри `Викладач` або `Студент` так само, як екземпляр `УчастникШколи`, і отримати доступ до методу `повідомити` класа `УчастникШколи`, просто ввівши `Викладач.повідомити` або `Студент.повідомити`.Натомість ми визначаємо інший метод `повідомити` у кожному підкласі (використовуючи метод `повідомити` в класі `УчастникШколи` для його частини), щоб пристосувати його для цього підкласу. Оскільки ми це зробили, написавши `Викладач.повідомити`, Python використовує метод `повідомити` для цього підкласу проти суперкласу. Однак, якби у нас не було методу `повідомити` у підкласі, Python використовував би метод `повідомити` у суперкласі. Python завжди спочатку починає шукати методи у фактичному типі підкласу, і якщо він нічого не знаходить, він починає шукати методи в базових класах підкласу, один за іншим у тому порядку, в якому вони вказані в кортежі (тут у нас є лише 1 базовий клас, але ви можете мати кілька базових класів) у визначенні класу.

Примітка щодо термінології: якщо в кортежі наслідування зазначено більше одного класу, це називається **множинним наслідуванням** (англ."multiple inheritance").

Параметр `end` використовується у функції `print` у методі `повідомити()` суперкласу, щоб надрукувати рядок і дозволити наступному друку продовжуватись у тому самому рядку. Це трюк, щоб змусити `print` не друкувати символ `\n` (новий рядок) у кінці друку.

3.21.7 Резюме

Зараз ми дослідили різні аспекти класів і об'єктів, а також різну термінологію, пов'язану з ними. Ми також побачили переваги та підводні камені об'єктно-орієнтованого програмування. Python дуже об'єктно-орієнтований, і ретельне розуміння цих концепцій дуже допоможе вам у довгостроковій перспективі.

Далі ми дізнаємося, як працювати з введенням/виведенням і як отримувати доступ до файлів у Python.

3.22 Введення-виведення

 англійська: *Input and Output*

Будуть ситуації, коли ваша програма повинна взаємодіяти з користувачем. Наприклад, ви хотіли б прийняти дані від користувача, а потім надрукувати деякі результати. Ми можемо досягти цього за допомогою функції `input()` і функції `print` відповідно.

Для виведення ми також можемо використовувати різні методи класу `str` (рядок). Наприклад, ви можете використати метод `rjust`, щоб отримати рядок, вирівняний по правому краю до зазначеної ширини. Дивіться `help(str)` для отримання додаткової інформації.

Іншим поширеним типом введення/виведення є робота з файлами. Можливість створювати, читати та записувати файли є важливою для багатьох програм, і ми дослідимо цей аспект у цій главі.

3.22.1 Введення від користувача

 англійська: *Input from user*

код python io_input_ukr.py

```
def реверс(текст):
    return текст[::-1]


def є_паліндром(текст):
    return текст == реверс(текст)

щось = input("Введіть текст: ")
if є_паліндром(щось):
    print("Так, це паліндром")
else:
    print("Ні, це не паліндром")
```

Висновок (3 рази):

```
$ python3 io_input_ukr.py
Введіть текст: пан
Ні, це не паліндром

$ python3 io_input.py
Введіть текст: мадам
Так, це паліндром

$ python3 io_input.py
Введіть текст: радар
Так, це паліндром
```

Як це працює

Ми використовуємо функцію зріз, щоб перевернути текст. Ми вже бачили, як можна зробити *зрізи послідовностей* за допомогою коду `seq[a:b]`, починаючи з позиції `a` до позиції `b`. Ми також можемо надати третій аргумент, який визначає *крок* (англ."*step*"), за яким виконується зріз. Крок за замовчуванням дорівнює `1`, через що він повертає безперервну частину тексту. Введення від'ємного кроку, тобто `-1`, поверне текст у зворотному порядку.

Функція `input()` приймає рядок як аргумент і відображає його користувачеві. Потім вона чекає, поки користувач щось введе та натисне клавішу введення. Коли користувач введе та натисне клавішу введення, функція `input()` поверне текст, який ввів користувач.

Ми беремо цей текст і перевертаємо його. Якщо вихідний текст і реверсований текст рівні, тоді текст є паліндромом.

Домашнє завдання

Перевірка того, чи є текст паліндромом, також має ігнорувати пунктуацію, пробіли та регістр літер. Наприклад, «Далі буде дуб і лад», також є паліндромом,але наша поточна програма так не вважає.Чи можете ви вдосконалити наведену вище програму, щоб програма розпізнала цей паліндром?

Підказка:

Скористайтеся кортежем (список усіх символів пунктуації можна знайти тут), що містять усі заборонені символи, та застосуйте тест на приналежність, щоб виявити символи, що підлягають видаленню, тобто forbidden = („!“, „?“, „.“, . . .).

Варіант вирешення:

```
# покращена версія, буде ігнорувати всі
# знаки пунктуації, такі як пробіли, крапки, коми тощо.

ігнорувати_рядок = " .,;:!?"

def чистий_текст(чернетка):
    чистий = ""
    for x in чернетка:
        if x not in ігнорувати_рядок:
            чистий += x
    return чистий


def reverse(текст):
    return текст[::-1]


def є_паліндром(текст):
    return текст.lower() == reverse(текст).lower()

# "Далі буде дуб і лад"
щось = input("Введіть текст:")
if є_паліндром(чистий_текст(щось)):
    print("Так, це паліндром")
else:
    print("Ні, це не паліндром")
```

3.22.2 Файли

Ви можете відкривати та використовувати файли для читання або запису, створивши об'єкт класу `file` ,а читати/записувати у файл - використовуючи його методи `read`, `readline` або `write` відповідно. Можливість читання або запису у файл залежить від режиму, який ви вказали для відкриття файлу. Після роботи з файлом, потрібно викликати метод `close`, щоб повідомити Python, що ми закінчили використовувати файл.

код python io_using_file_ukr.py

```
вірш = """\
Програмування - це весело.
Коли робота виконана,
```

```
і якщо ви хочете повесилитися на роботі:
    використовуйте Python!

"""

# Відкрито для написання(англ."'w'riting")
f = open('вірш.txt', 'w')
# Записати текст у файл
f.write(вірш)
# Закрийте файл
f.close()

# Якщо режим не вказано,
# режим читання (англ."'r'eading") передбачається за замовчуванням
f = open('вірш.txt')
while True:
    лінія = f.readline()
    # Нульова довжина вказує на кінец файлу(EOF)
    if len(лінія) == 0:
        break
    # У `рядку`(`line`) вже є новий рядок
    # у кінці кожного рядка,
    # оскільки він читає  файл.
    print(лінія, end='')
# закрити файл
f.close()
```

Висновок:

```
Програмування - це весело.
Коли робота виконана,
і якщо ви хочете повесилитися на роботі:
    використовуйте Python!
```

Як це працює

Зауважте, що ми можемо створити новий файловий об'єкт просто за допомогою методу `open`. Ми відкриваємо цей файл (або створюємо його, якщо він ще не існує) за допомогою вбудованої функції `open` і вказуємо назву файлу та режим, у якому ми хочемо відкрити файл. Режим може бути режимом читання (`'r'`), режимом запису (`'w'`) або режимом додавання (`'a'`). Ми також можемо вказати, чи ми читаємо, записуємо або додаємо дані: в текстовому режимі (`'t'`) чи двійковому (бінарному) режимі (`'b'`). Насправді існує набагато більше доступних режимів, і `help (open)` надасть вам більше інформації про них. За замовчуванням `open()` вважає файл текстовим файлом і відкриває його в режимі читання.

У нашому прикладі ми спочатку відкриваємо/створюємо файл у режимі запису тексту та використовуємо метод `write` файлового об'єкта, щоб записати нашу рядкову змінну `вірш` у файл, а потім ми закриваємо файл за допомогою `close`.

Далі знову відкриваємо цей же файл для читання. Нам не потрібно вказувати режим, оскільки «читати текстовий файл» є режимом за замовчуванням. Ми читаємо кожен рядок файлу за допомогою методу `readline` у циклі. Цей метод повертає повний рядок, включаючи символ нового рядка в кінці рядка. Коли повертається *порожній* рядок, це означає, що ми перериваємо цикл за допомогою break.

Зрештою, ми остаточно закриваємо файл за домопогою `close`.

Ми бачимо з наших результатів `readline`, що ця програма справді записувала та читала наш новий

файл `вірш.txt`.

3.22.3 Pickle

Python надає стандартний модуль під назвою `pickle`(pickle-англ.“маринувати”,“солити”), який можна використовувати для зберігання *будь-якого* простого об’єкта Python у файлі, а потім отримати його назад. Це називається *тривалим* (англ.“*persistently*”) збереженням об’єкта.

код python io_pickle_ukr.py

```
import pickle

# Ім'я файлу, де ми будемо зберігати об'єкт
файл_список_покупок = 'список_покупок.data'
# Список фруктів для покупки
список_покупок_фрукти = ['яблуко', 'манго', 'морква']

# Запис у файл
f = open(файл_список_покупок, 'wb')
# Вивести об'єкт у файл
pickle.dump(список_покупок_фрукти, f)
f.close()

# Знищити змінну список_покупок_фрукти
del список_покупок_фрукти

# Прочитати зі сховища
f = open(файл_список_покупок, 'rb')
# Завантажити об'єкт із файлу
збережений_список = pickle.load(f)
print(збережений_список)
f.close()
```

Висновок:

```
```$ python io_pickle_ukr.py
['яблуко', 'манго', 'морква']
```
```

Як це працює

Щоб зберегти об’єкт у файлі, ми маємо спочатку відкрити файл за допомогою `open` в режимі бінарного запису (‘wb’), а потім викликати функцію `dump` із модуля `pickle`. Цей процес називається *консервацією*(англ.“*pickling*”).

Далі ми отримуємо об’єкт за допомогою функції `load` із модуля `pickle`, яка повертає об’єкт. Цей процес називається “розконсервацією”(англ.“*unpickling*”).

3.22.4 Unicode

Досі, коли ми писали та використовували рядки або читали та записували у файл, ми використовували лише прості англійські символи. Як англійські, так і неанглійські символи можуть бути представлені в Unicode (будь ласка, перегляньте статті в кінці цього розділу для отримання додаткової інформації), а Python 3 за замовчуванням зберігає рядкові змінні (подумайте про весь той текст, який ми написали, використовуючи одиничні, подвійні або потрійні лапки) в Unicode.

Примітка

Якщо ви використовуєте Python 2, і ми хочемо мати можливість читати та писати іншими мовами, відмінними від англійської, нам потрібно використовувати тип `unicode`, і все починається з символу `u`, напр. `u"Привіт, Світ!"`

```
>>> "Привіт, Світ!"
'Привіт, Світ!'
>>> type("Привіт, Світ!")
<class 'str'>
>>> u"Привіт, Світ!"
'Привіт, Світ!'
>>> type(u"Привіт, Світ!")
<class 'str'>
```

Коли дані надсилаються через Інтернет, нам потрібно надсилати їх у байтах... те, що ваш комп'ютер легко розуміє. Правила перекладу Unicode (це те, що Python використовує, коли зберігає рядок) у байти, називаються encoding (encoding- англ."кодування","шифрування"). Популярним encoding є UTF-8. Ми можемо читати та писати в UTF-8, використовуючи простий ключовий аргумент у нашій функції `open`.

код python io_unicode_ukr.py

```
import io

f = io.open("abc.txt", "wt", encoding="utf-8")
f.write(u"Уявіть собі неанглійську мову")
f.close()

text = io.open("abc.txt", encoding="utf-8").read()
print(text)
```

Як це працює

Ми використовуємо `io.open`, а потім використовуємо аргумент `encoding` у першому рядку коду (in the first open statement) для кодування повідомлення, а потім знову в другому рядку коду open під час декодування повідомлення. Зауважте, що ми повинні використовувати кодування лише в рядку коду open у текстовому режимі.

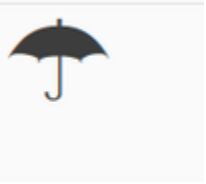
Кожного разу, коли ми пишемо програму, яка використовує літерали Unicode (ставляючи `u` перед рядком), як ми використовували вище, ми повинні переконатися, що самому Python повідомляється, що наша програма використовує UTF-8, і ми повинні поставити `# encoding=utf-8` коментар у верхній частині нашої програми.

Ви повинні дізнатися більше про цю тему, прочитавши:

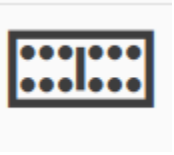
- "Абсолютний мінімум, який кожен розробник програмного забезпечення має знати про Unicode та набори символів", англ."The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets"
- Python Unicode Howto
- Pragmatic Unicode talk by Nat Batchelder

Від перекладача:

Як працювати з unicode: 1.знайдіть перелік символів у wiki: https://en.wikipedia.org/wiki/List_of_Unicode_characters ; 2.знайдіть один знак (наприклад: надрукувати парасольку), у вікіпедії сказано, що юнікод: U+2602; 3.якщо unicode містить 4 цифри або менше: print("\u2602")



4.у випадку друку знака плитки доміно 6, unicode - U+1F061 5.даний Юнікод містить більше 4 цифр: заповнити рядок початковими нулями, поки не буде 8 цифр: print("\U0001F061")



3.22.5 Резюме

Ми обговорили різні типи введення-виведення, обробку файлів, модуль pickle і Unicode.

Далі ми розглянемо концепцію винятків.

3.23 Винятки

англійська: *Exceptions*

Винятки трапляються, коли у вашій програмі виникають *виняткові* (*exceptional*) ситуації. Наприклад,якщо ви збираєтеся прочитати файл, а файл не існує? Або, якщо ви випадково видалили файл під час роботи програми? Такі ситуації обробляються за допомогою **винятків** (англ."**exceptions**").

Подібним чином, якби ваша програма мала деякі неприпустимі команди? У цьому випадку Python **піднімає** (англ."**raises**")руки та повідомляє, що виявив **помилку** (англ."**error**").

3.23.1 Помилки

англійська: *Errors*

Розглянемо простий виклик функції `print`. Що, якщо ми помилково напишемо `print` як `Print`? Зверніть увагу на використання великих літер (англ."capitalization"). У цьому випадку Python *піднімає* синтаксичну помилку.

Приклад англійською:

```
>>> Print("Hello World")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>> print("Hello World")
Hello World
```

Приклад українською (скріншот, зроблений в оболонці Python за допомогою IDLE):

```
    Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
    Type "help", "copyright", "credits" or "license()" for more information.
>>> Print("Привіт,Світ!")
    Traceback (most recent call last):
      File "/usr/lib/python3.10/idlelib/run.py", line 578, in runcode
        exec(code, self.locals)
      File "<pyshell#0>", line 1, in <module>
    NameError: name 'Print' is not defined. Did you mean: 'print'?
>>> print("Привіт,Світ!")
    Привіт,Світ!
```

де: name 'Print' is not defined.Did you mean:'print'? - ім'я «Print» не визначено. Ви мали на увазі :'print'?

Зверніть увагу, що була підіймана помилка`NameError`,а також друкується місце, де було виявлено помилку. Так у цьому випадку діє **обробник помилок** (англ."**error handler**").

3.23.2 Винятки

 англійська: *Exceptions*

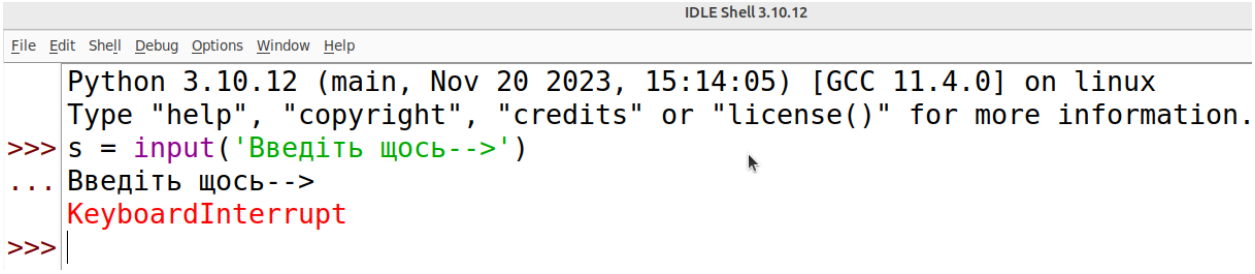
Ми **спробуємо** (англ."**try**") прочитати щось від користувача. Введіть перший рядок нижче та натисніть клавішу `Enter`. Коли ваш комп'ютер запропонує вам ввести дані, натомість натисніть `[ctrl-d]` на Mac або `[ctrl-z]` на Windows і подивіться, що станеться. (Якщо ви користуєтеся Windows і жоден із варіантів не працює, ви можете спробувати `[ctrl-c]` у командному рядку, тобто створити KeyboardInterrupt error).

Приклад англійською, автор використовує `[ctrl-d]` чи `[ctrl-z]`:

```
>>> s = input('Enter something --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

Python підіймає помилку під назвою end-of-file (`EOFError`)(для користувачів,які використовують `[ctrl-d]` чи `[ctrl-z]`), яка в основному означає, що він знайшов символ *кінця файлу*.

Приклад українською (скріншот `[ctrl-c]`, зроблений в оболонці Python за допомогою IDLE):



Python підіймає помилку під назвою `KeyboardInterrupt Error`(для користувачів,які використовують `[ctrl-c]`).

І в англійському, і в українському варіанті сталася помилка.

3.23.3 Обробка винятків

 англійська: *Handling Exceptions*

Ми можемо обробляти винятки за допомогою оператора `try..except`. По суті, ми розміщуємо наші звичайні команди в блоці try, а всі наші обробники винятків помилок – у блоці except.

код python exceptions_handle_ukr.py

```
try:
    text = input('Введіть щось --> ')
except EOFError:
    print('Чому ви прислалі мені символ кінця файлу?')
except KeyboardInterrupt:
    print('Ви скасували операцію.')
else:
    print('Ви увійшли{}'.format(text))
```

Висновок:

```
$ python exceptions_handle_ukr.py
Введіть щось -->   # натисніть ctrl + d
Чому ви прислалі мені сигнал кінец файлу?

$ python exceptions_handle_ukr.py
Введіть щось -->  # натисніть ctrl + c
Ви скасували операцію.

$ python exceptions_handle_ukr.py
Введіть щось --> Без винятків
Ви увійшли Без винятків
```

Підказка: якщо CTRL+c закриває вікно терміналу замість того, щоб створити очікуване повідомлення про помилку, спробуйте написати цю програму за допомогою IDLE (Меню: File -> New File, потім натисніть Run -> Run module)

```
IDLE Shell 3.10.12
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    = RESTART: /home/kurs/Desktop/dasha/learning_python/byte-of-python-master/programs/e
    xceptions_handle2.py
    Введіть щось -->
    Чому ви прислалі мені символ кінця файлу?
>>>
    = RESTART: /home/kurs/Desktop/dasha/learning_python/byte-of-python-master/programs/e
    xceptions_handle2.py
    Введіть щось -->
    Ви скасували операцію.
>>>
    = RESTART: /home/kurs/Desktop/dasha/learning_python/byte-of-python-master/programs/e
    xceptions_handle2.py
    Введіть щось -->  Без винятків
    Ви увійшли Без винятків
```

Як це працює

Ми розміщуємо всі командии, які можуть спричиняти винятки/помилки у блоці `try`, а потім розміщуємо обробники відповідних помилок/винятків у блоці `except`. Вираз `except` може обробляти як одиночну помилку або виняток, так і список помилок/винятків у дужках. Якщо не надано назви помилок чи винятків, він оброблятиме *всі* помилки та винятки.

Зауважте, що для кожного виразу `try` має бути принаймні одне речення `except`. Інакше який сенс мати блок try?

Якщо будь-яка помилка чи виняток не оброблені, тоді викликається обробник Python за замовчуванням, який просто зупиняє виконання програми та друкує повідомлення про помилку. Ми вже бачили це в дії вище.

Також можна додати пункт `else` до відповідного блоку `try..except`. Пункт `else` виконується, якщо не відбувається винятків.

У наступному прикладі ми також побачимо, як отримати об'єкт винятку, щоб ми могли отримати додаткову інформацію.

3.23.4 Виклик винятків

 англійська: *Raising Exceptions*

Ви можете *викликати* винятки за допомогою оператора `raise`, передавши йому ім'я помилки або винятку, а також об'єкт винятку, який потрібно *викинути.*

Помилка або виняток, який ви можете викликати, має бути класом, який прямо чи опосередковано є похідним від класу `Exception`.

українська

код python exceptions_raise_ukr.py:

```
class Виняток_короткого_введення(Exception):
    '''Визначений користувачем клас винятків.'''
    def __init__(self, довжина, як_мінімум):
        Exception.__init__(self)
        self.довжина = довжина
        self.як_мінімум = як_мінімум

try:
    текст = input('Введіть щось --> ')
    if len(текст) < 3:
        raise Виняток_короткого_введення(len(текст), 3)
    # Тут може відбуватися звичайна робота
except EOFError:
    print('Чому ви прислалі мені символ кінця файлу?')
except Виняток_короткого_введенн as вн:
    print(('Виняток_короткого_введення: Довжина_введеного_рядка ' +
           '{0} очікувалося_як_мінімум {1}')
          .format(вн.довжина, вн.як_мінімум))
else:
    print('Винятків не було.')
```

Висновок:

```
$ python exceptions_raise_ukr.py
Введіть щось --> a
Виняток_короткого_введення: Довжина_введеного_рядка очікувалося_як_мінімум 3

$ python exceptions_raise_ukr.py
Введіть щось --> abc
Винятків не було.
```

англійська

python code exceptions_raise_en.py:

```
class ShortInputException(Exception):
    '''A user-defined exception class.'''
    def __init__(self, length, atleast):
        Exception.__init__(self)
        self.length = length
        self.atleast = atleast

try:
    text = input('Enter something --> ')
    if len(text) < 3:
        raise ShortInputException(len(text), 3)
    # Other work can continue as usual here
except EOFError:
    print('Why did you do an EOF on me?')
except ShortInputException as ex:
    print(('ShortInputException: The input was ' +
           '{0} long, expected at least {1}')
          .format(ex.length, ex.atleast))
else:
    print('No exception was raised.')
```

output:

```
$ python exceptions_raise_en.py
Enter something --> a
ShortInputException: The input was 1 long, expected at least 3

$ python exceptions_raise_en.py
Enter something --> abc
No exception was raised.
```

Як це працює

Тут ми створюємо власний тип винятку. Цей новий тип винятку називається `Виняток_короткого_введення`. У ньому є два поля: `довжина`, що зберігає довжину введеного тексту, і `як_мінімум`, що вказує, яку мінімальну довжину тексту очікувала програма.

У пункті `except` ми вказуємо клас помилки, який зберігатиметься `як`(англ.`as`) змінна `вн` , що містить відповідний об'єкт помилки/виключення. Це аналогічно параметрам і аргументам у виклику функції. Всередині цього пункту `except` ми використовуємо поля `довжину` і `як_мінімум` об'єкта винятку, щоб надрукувати відповідне повідомлення для користувача.

3.23.5 Try ... Finally

Припустимо, ви читаєте файл у своїй програмі. Як переконатися, що об'єкт файлу був коректно закритий і що не виникло жодного винятку? Це можна зробити за допомогою блоку `finally`.

українська

код python exceptions_finally_ukr.py:

```
import sys
import time

f = None
try:
    f = open("вірш.txt")
    # наш звичайний спосіб читати файли
    while True:
        лінія = f.readline()
        if len(лінія) == 0:
            break
        print(лінія, end='')
        sys.stdout.flush()
        print("Натисніть ctrl+c зараз")
        # Щоб переконатися, що він працює деякий час
        time.sleep(2)
except IOError:
    print("Не вдалося знайти файл вірш.txt")
except Клавіатура_переривання:
    print("!! Ви скасували читання з файлу.")
finally:
    if f:
        f.close()
    print("(Очищення: файл закрито)")
```

Висновок:

```
$ python exceptions_finally_ukr.py
Програмування - це весело.
Натисніть ctrl+c зараз

^C!! Ви скасували читання з файлу.
(Очищення: файл закрито)
```

англійська

python code exceptions_finally_en.py:

```
import sys
import time

f = None
try:
    f = open("poem.txt")
    # Our usual file-reading idiom
```

(continues on next page)

(continued from previous page)

```
    while True:
        line = f.readline()
        if len(line) == 0:
            break
        print(line, end='')
        sys.stdout.flush()
        print("Press ctrl+c now")
        # To make sure it runs for a while
        time.sleep(2)
except IOError:
    print("Could not find file poem.txt")
except KeyboardInterrupt:
    print("!! You cancelled the reading from the file.")
finally:
    if f:
        f.close()
    print("(Cleaning up: Closed the file)")
```

output:

```
$ python exceptions_finally.py
Programming is fun
Press ctrl+c now
^C!! You cancelled the reading from the file.
(Cleaning up: Closed the file)
```

Як це працює

Ми виконуємо звичайне читання файлів, але ми довільно ввели сплячий режим протягом 2 секунд після друку кожного рядка за допомогою функції `time.sleep`, щоб програма працювала повільно (Python дуже швидкий за своєю природою). Коли програма все ще працює, натисніть `ctrl + c`, щоб перервати/скасувати програму.

Зверніть увагу, що виникає виняток `Клавіатура_переривання` і програма завершує роботу. Однак перед завершенням роботи програми виконується пункт finally, і файловий об'єкт завжди закривається.

Зауважте, що змінна, якій присвоєно значення 0 або `None`, або змінна, яка є порожньою послідовністю чи колекцією, вважається `False` у Python. Ось чому ми можемо використовувати `if f:` у коді вище.

Також зауважте, що ми використовуємо `sys.stdout.flush()` після `print`, щоб він негайно друкувався на екрані.

3.23.6 Оператор with

 англійська: *The with statement*

Типовою схемою є запит деякого ресурсу в блоці `try` і подальше звільнення цього ресурсу в блоці `finally`. Отже, є також оператор `with`, який дозволяє це зробити більш "чисто":

Зберегти як`exceptions_using_with.py`:

код python exceptions_using_with_ukr.py

```
with open("вірш.txt") as f:
    for лінія in f:
```

```
        print(лінія, end='')
```

Як це працює

Результат має бути таким самим, як у попередньому прикладі. Різниця тут полягає в тому, що ми використовуємо функцію `open` з оператором `with` - цим ми залишаємо автоматичне закриття файлу під відповідальність `with open`.

Те, що відбувається за кулісами, полягає в тому, що існує такий собі протокол, який використовується оператором with. Він зчитує об'єкт, який повертається оператором `open`, назвемо його в даному випадку "thefile".

Перед запуском блоку коду, що міститься в ньому, оператор `with` *завжди* викликає функцію із файлу`thefile.__enter__`,а також *завжди* викликає `thefile.__exit__` після завершення цього блоку кода.

Таким чином, код, який ми б написали в блоці `finally`, повинен автоматично оброблятися методом `__exit__`. Це те, що допомагає нам уникнути повторного використання явних операторів `try..finally`.

Додаткове обговорення цієї теми виходить за рамки цієї книги, тому, будь ласка, зверніться до PEP 343 для вичерпного пояснення.

3.24 Що робить оператор with (від перекладача):

Кожен файловий об'єкт має дві функції: **enter** та **exit**. Оператор `with` спочатку викликає функцію **enter** об'єкта файлу, потім виконує всі рядки коду всередині блоку `with`, а після завершення викликає функцію **exit** цього ж об'єкта.

Без блоку with вам потрібно написати код так:

українська

код python poem_ukr.py:

```
f = open("вірш.txt")
# ще кілька команд Python
f.close()
# інший код
```

англійська

python code poem_en.py

```
f = open("poem.txt")
# some more python commands
f.close()
# other code
```

Використовуючи блок with, ви можете писати елегантніше і не турбуватись про закриття файлу:

українська

код python poem2_ukr.py:

```
with open("вірш.txt") as f:
     print("файл зараз відкрит")
# ще кілька команд Python
# тепер автоматично закривається!
print("файл зараз закрит")
# інший код
```

англійська

python code poem2_en.py

```
with open("poem.txt") as f:
    print("file is now open")
    # some more python commands
# now automatically closed!
print("file is now closed")
# other code
```

3.24.1 Резюме

Ми обговорили використання операторів `try..except` і `try..finally`. Ми побачили, як створювати власні типи винятків, а також як викликати винятки.

Далі ми вивчимо стандартну бібліотеку Python.

3.25 Стандартна бібліотека

 англійська: *Standard Library*

Стандартна бібліотека Python містить величезну кількість корисних модулів і є частиною кожної стандартної інсталяції Python. Важливо ознайомитися зі стандартною бібліотекою Python, оскільки багато проблем можна швидко вирішити, якщо ви знайомі з можливостями цих бібліотек.

Ми розглянемо деякі з часто використовуваних модулів у цій бібліотеці. Ви можете знайти повну інформацію про всі модулі стандартної бібліотеки Python у розділі «Довідник бібліотеки» (англ."Library Reference") документації, яка постачається разом із інсталяцією Python.

Давайте розглянемо кілька корисних модулів.

Попередження

якщо ви вважаєте теми цього розділу занадто складними, ви можете пропустити цей розділ. Однак я настійно рекомендую повернутися до цього розділу, коли вам стане зручніше програмувати на Python.

3.25.1 модуль `sys`

 англійська: *`sys` module*)

Модуль `sys` містить функціональність, характерну для системи. Ми вже бачили, що список `sys.argv` містить аргументи командного рядка.

Припустімо, ми хочемо перевірити версію використовуваного програмного забезпечення Python, модуль `sys` надає нам цю інформацію.

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=0, releaselevel='final', serial=0)
>>> sys.version_info.major == 3
True
```

Як це працює

Модуль `sys` має кортеж `version_info`, який надає нам інформацію про версію. Перший запис (major=3) — основна версія. Ми можемо отримати цю інформацію, щоб використати її.

3.25.2 Модуль `logging`

 англійська: *`logging` module*

Уявіть ситуацію, коли необхідно зберегти деякі налагоджувальні або інші важливі повідомлення де-небудь,щоб мати можливість пізніше перевірити, чи ваша програма працює так, як ви цього очікували? Як саме «зберігати десь» ці повідомлення ? Цього можна досягти за допомогою модуля `logging`.

код python stdlib_logging_ukr.py

```
import os
import platform
import logging

if platform.platform().startswith('Windows'):
    logging_file = os.path.join(os.getenv('HOMEDRIVE'),
                                os.getenv('HOMEPATH'),
                                'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
                                'test.log')

print("Logging to", logging_file)

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename=logging_file,
    filemode='w',
)

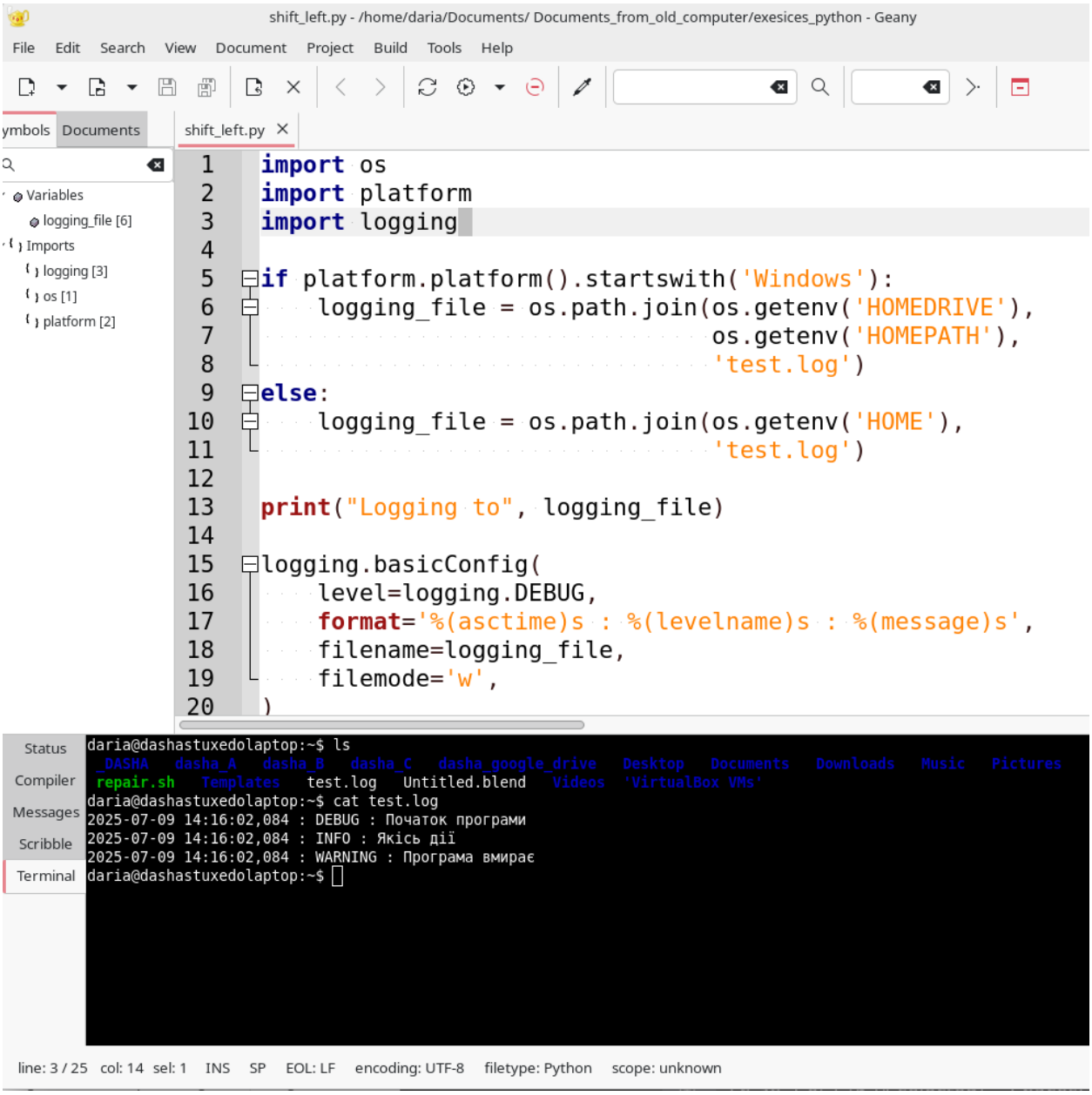
logging.debug("Початок програми")
logging.info("Якісь дії")
logging.warning("Програма вмирає")
```

Висновок:

```
$ python stdlib_logging_ukr.py
Logging to /Users/swa/test.log

$ cat /Users/swa/test.log
2014-03-29 09:27:36,660 : DEBUG : Початок програми
2014-03-29 09:27:36,660 : INFO : Якісь дії
2014-03-29 09:27:36,660 : WARNING : Програма вмирає
```

Щоб переглянути та вивчити файл test.log, просто відкрийте його в будь-якому текстовому редакторі. Якщо вам зручно працювати в терміналі, у вашій операційній системі зазвичай є команда для відображення текстового файлу. У Linux та MacOS ця команда називається cat.



Як це працює

Ми використовуємо три модулі зі стандартної бібліотеки: модуль `os` для взаємодії з операційною системою, модуль `platform` для отримання інформації про платформу, тобто операційну систему, і модуль `logging` для *логування* інформації (англ.» to log information»).

Спочатку ми перевіряємо, яку операційну систему ми використовуємо, перевіряючи рядок, який повертає функція `platform.platform()` (для отримання додаткової інформації див. `import platform; help (platform)`). Якщо це Windows, ми визначаємо home drive (диск) ,який містить домашню папку та

назву файлу, де ми хочемо зберігати інформацію. Зібравши ці три частини разом, ми отримаємо повне розташування файлу. Для інших платформ нам потрібно знати лише домашню папку користувача, і ми отримаємо повний шлях до файлу.

Ми використовуємо функцію `os.path.join()`, щоб поєднати ці три частини шляху. Причина використання спеціальної функції, а не простого додавання рядків разом, полягає в тому, що ця функція гарантує, що повний шлях відповідає формату, очікуваному операційною системою.

Примітка : метод `join()`, який ми використовуємо тут є частиною модуля `os`,він відрізняється від рядкового методу `join()`, який ми використовували в інших частинах цієї книги.

Ми налаштовуємо модуль `logging` для запису всіх повідомлень у певному форматі у вказаний файл.

Нарешті, ми можемо виводити повідомлення, призначені для налагодження, інформування, попередження та навіть критичні повідомлення. Після запуску програми ми можемо перевірити цей файл і знати, що сталося в програмі, навіть якщо користувач, який запускає програму, не показує жодної інформації.

3.25.3 Модуль серії тижня

 англійська: *Module of the Week Series*

У стандартній бібліотеці, як-от налагодження (англ."debugging"), обробка параметрів командного рядка(англ."handling command line options"), регулярні вирази(англ."regular expressions") є ще багато чого для вивчення.

Найкращий спосіб глибше вивчити стандартну бібліотеку — це прочитати чудову серію Doug Hellmann "Модуль тижня" (англ."Python Module of the Week") (також доступна як книга)або Офіційну документацію Python.

3.25.4 Резюме

Ми дослідили деякі функції багатьох модулів стандартної бібліотеки Python. Настійно рекомендуємо переглянути документацію стандартної бібліотеки Python, щоб отримати уявлення про всі доступні модулі.

Далі ми розглянемо різні аспекти Python, які зроблять наш огляд Python більш *повним*.

3.26 Стандартна бібліотека

 англійська: *Standard Library*

Стандартна бібліотека Python містить величезну кількість корисних модулів і є частиною кожної стандартної інсталяції Python. Важливо ознайомитися зі стандартною бібліотекою Python, оскільки багато проблем можна швидко вирішити, якщо ви знайомі з можливостями цих бібліотек.

Ми розглянемо деякі з часто використовуваних модулів у цій бібліотеці. Ви можете знайти повну інформацію про всі модулі стандартної бібліотеки Python у розділі «Довідник бібліотеки» (англ."Library Reference") документації, яка постачається разом із інсталяцією Python.

Давайте розглянемо кілька корисних модулів.

Попередження

якщо ви вважаєте теми цього розділу занадто складними, ви можете пропустити цей розділ. Однак я настійно рекомендую повернутися до цього розділу, коли вам стане зручніше програмувати на Python.

3.26.1 модуль sys

 англійська: *sys module*)

Модуль `sys` містить функціональність, характерну для системи. Ми вже бачили, що список `sys.argv` містить аргументи командного рядка.

Припустімо, ми хочемо перевірити версію використовуваного програмного забезпечення Python, модуль `sys` надає нам цю інформацію.

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=0, releaselevel='final', serial=0)
>>> sys.version_info.major == 3
True
```

Як це працює

Модуль `sys` має кортеж `version_info`, який надає нам інформацію про версію. Перший запис (major=3) — основна версія. Ми можемо отримати цю інформацію, щоб використати її.

3.26.2 Модуль logging

 англійська: *logging module*

Уявіть ситуацію, коли необхідно зберегти деякі налагоджувальні або інші важливі повідомлення денебудь,щоб мати можливість пізніше перевірити, чи ваша програма працює так, як ви цього очікували? Як саме «зберігати десь» ці повідомлення ? Цього можна досягти за допомогою модуля `logging`.

код python stdlib_logging_ukr.py

```
import os
import platform
import logging

if platform.platform().startswith('Windows'):
    logging_file = os.path.join(os.getenv('HOMEDRIVE'),
                                os.getenv('HOMEPATH'),
                                'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
                                'test.log')

print("Logging to", logging_file)

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename=logging_file,
```

```
    filemode='w',
)

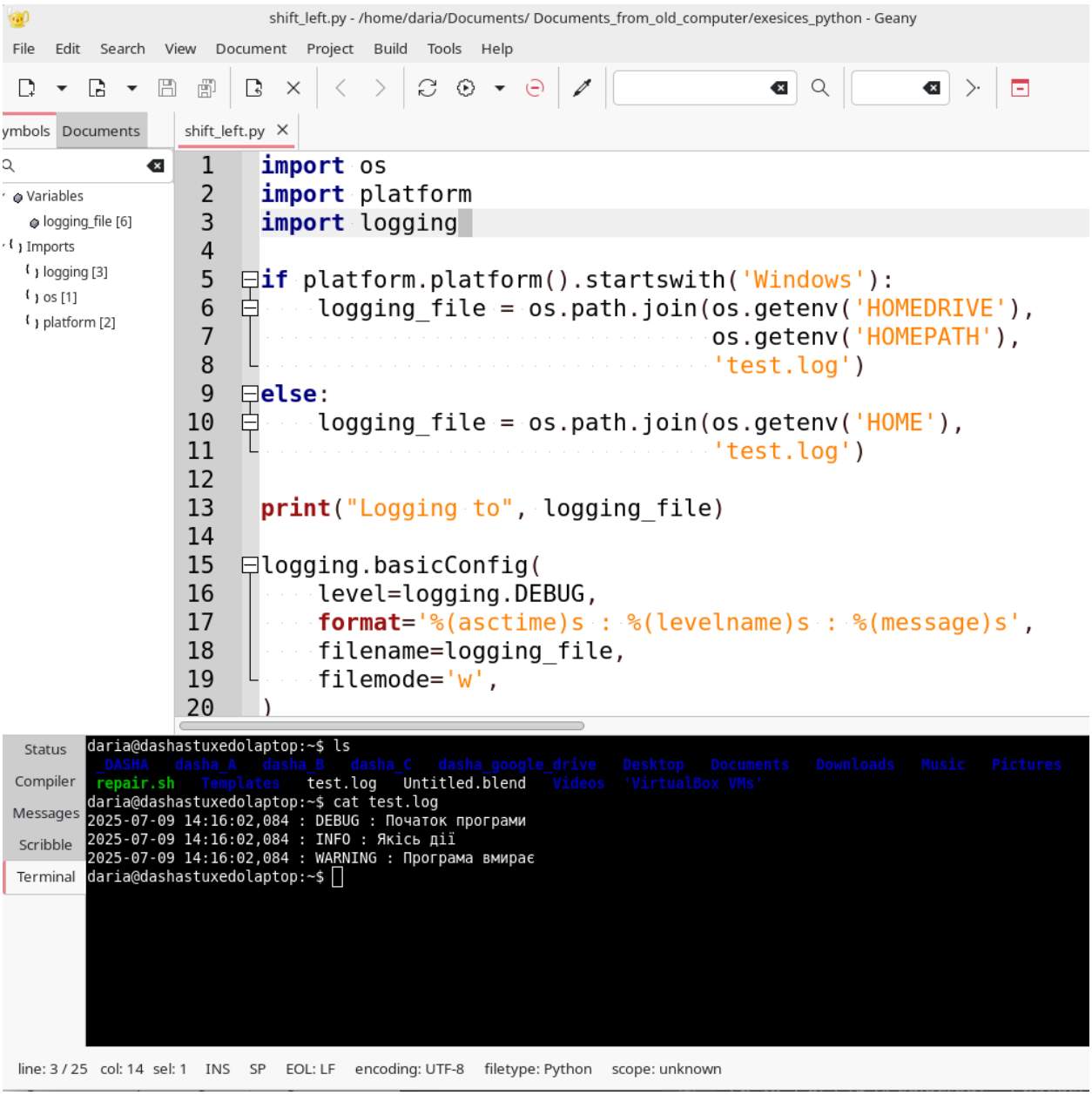
logging.debug("Початок програми")
logging.info("Якісь дії")
logging.warning("Програма вмирає")
```

Висновок:

```
$ python stdlib_logging_ukr.py
Logging to /Users/swa/test.log

$ cat /Users/swa/test.log
2014-03-29 09:27:36,660 : DEBUG : Початок програми
2014-03-29 09:27:36,660 : INFO : Якісь дії
2014-03-29 09:27:36,660 : WARNING : Програма вмирає
```

Щоб переглянути та вивчити файл test.log, просто відкрийте його в будь-якому текстовому редакторі. Якщо вам зручно працювати в терміналі, у вашій операційній системі зазвичай є команда для відображення текстового файлу. У Linux та MacOS ця команда називається cat.



Як це працює

Ми використовуємо три модулі зі стандартної бібліотеки: модуль `os` для взаємодії з операційною системою, модуль `platform` для отримання інформації про платформу, тобто операційну систему, і модуль `logging` для *логування* інформації (англ.» to log information»).

Спочатку ми перевіряємо, яку операційну систему ми використовуємо, перевіряючи рядок, який повертає функція `platform.platform()` (для отримання додаткової інформації див. `import platform; help(platform)`). Якщо це Windows, ми визначаємо home drive (диск) ,який містить домашню папку та назву файлу, де ми хочемо зберігати інформацію. Зібравши ці три частини разом, ми отримаємо повне розташування файлу. Для інших платформ нам потрібно знати лише домашню папку користувача, і ми отримаємо повний шлях до файлу.

Ми використовуємо функцію `os.path.join()`, щоб поєднати ці три частини шляху. Причина використання спеціальної функції, а не простого додавання рядків разом, полягає в тому, що ця функція гарантує, що повний шлях відповідає формату, очікуваному операційною системою.

Примітка : метод `join()`, який ми використовуємо тут є частиною модуля `os`,він відрізняється від рядкового методу `join()`, який ми використовували в інших частинах цієї книги.

Ми налаштовуємо модуль `logging` для запису всіх повідомлень у певному форматі у вказаний файл.

Нарешті, ми можемо виводити повідомлення, призначені для налагодження, інформування, попередження та навіть критичні повідомлення. Після запуску програми ми можемо перевірити цей файл і знати, що сталося в програмі, навіть якщо користувач, який запускає програму, не показує жодної інформації.

3.26.3 Модуль серії тижня

 англійська: *Module of the Week Series*

У стандартній бібліотеці, як-от налагодження (англ."debugging"), обробка параметрів командного рядка(англ."handling command line options"), регулярні вирази(англ."regular expressions") є ще багато чого для вивчення.

Найкращий спосіб глибше вивчити стандартну бібліотеку — це прочитати чудову серію Doug Hellmann "Модуль тижня" (англ."Python Module of the Week") (також доступна як книга)або Офіційну документацію Python.

3.26.4 Резюме

Ми дослідили деякі функції багатьох модулів стандартної бібліотеки Python. Настійно рекомендуємо переглянути документацію стандартної бібліотеки Python, щоб отримати уявлення про всі доступні модулі.

Далі ми розглянемо різні аспекти Python, які зроблять наш огляд Python більш *повним*.

3.27 Стандартна бібліотека

 англійська: *Standard Library*

Стандартна бібліотека Python містить величезну кількість корисних модулів і є частиною кожної стандартної інсталяції Python. Важливо ознайомитися зі стандартною бібліотекою Python, оскільки багато проблем можна швидко вирішити, якщо ви знайомі з можливостями цих бібліотек.

Ми розглянемо деякі з часто використовуваних модулів у цій бібліотеці. Ви можете знайти повну інформацію про всі модулі стандартної бібліотеки Python у розділі «Довідник бібліотеки» (англ."Library Reference") документації, яка постачається разом із інсталяцією Python.

Давайте розглянемо кілька корисних модулів.

Попередження

якщо ви вважаєте теми цього розділу занадто складними, ви можете пропустити цей розділ. Однак я настійно рекомендую повернутися до цього розділу, коли вам стане зручніше програмувати на Python.

3.27.1 модуль sys

 англійська: *sys module*)

Модуль `sys` містить функціональність, характерну для системи. Ми вже бачили, що список `sys.argv` містить аргументи командного рядка.

Припустімо, ми хочемо перевірити версію використовуваного програмного забезпечення Python, модуль `sys` надає нам цю інформацію.

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=0, releaselevel='final', serial=0)
>>> sys.version_info.major == 3
True
```

Як це працює

Модуль `sys` має кортеж `version_info`, який надає нам інформацію про версію. Перший запис (major=3) — основна версія. Ми можемо отримати цю інформацію, щоб використати її.

3.27.2 Модуль `logging`

 англійська: *`logging` module*

Уявіть ситуацію, коли необхідно зберегти деякі налагоджувальні або інші важливі повідомлення денебудь,щоб мати можливість пізніше перевірити, чи ваша програма працює так, як ви цього очікували? Як саме «зберігати десь» ці повідомлення ? Цього можна досягти за допомогою модуля `logging`.

код python stdlib_logging_ukr.py

```
import os
import platform
import logging

if platform.platform().startswith('Windows'):
    logging_file = os.path.join(os.getenv('HOMEDRIVE'),
                                os.getenv('HOMEPATH'),
                                'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
                                'test.log')

print("Logging to", logging_file)

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename=logging_file,
    filemode='w',
)

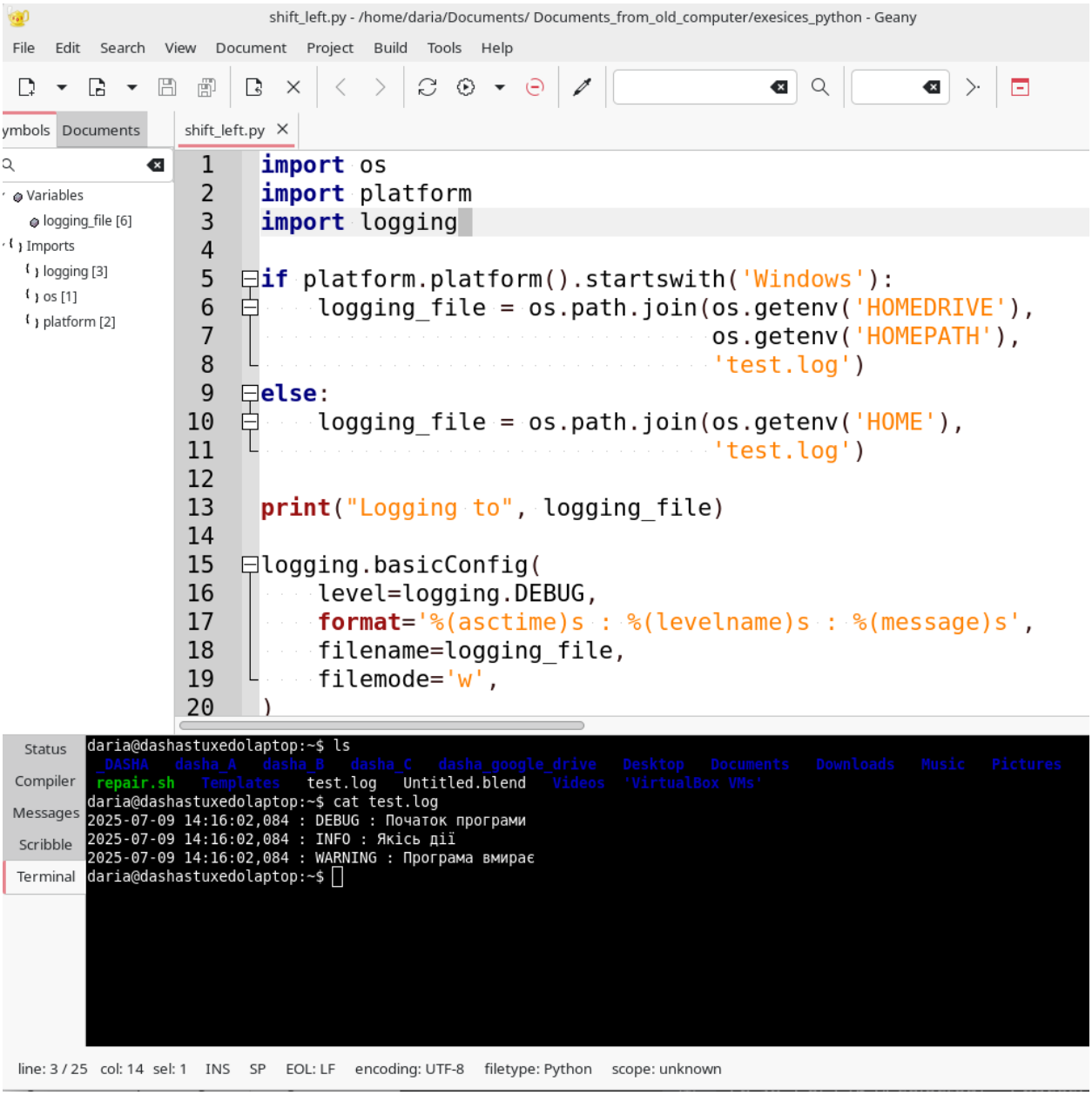
logging.debug("Початок програми")
logging.info("Якісь дії")
logging.warning("Програма вмирає")
```

Висновок:

```
$ python stdlib_logging_ukr.py
Logging to /Users/swa/test.log

$ cat /Users/swa/test.log
2014-03-29 09:27:36,660 : DEBUG : Початок програми
2014-03-29 09:27:36,660 : INFO : Якісь дії
2014-03-29 09:27:36,660 : WARNING : Програма вмирає
```

Щоб переглянути та вивчити файл test.log, просто відкрийте його в будь-якому текстовому редакторі. Якщо вам зручно працювати в терміналі, у вашій операційній системі зазвичай є команда для відображення текстового файлу. У Linux та MacOS ця команда називається cat.



Як це працює

Ми використовуємо три модулі зі стандартної бібліотеки: модуль `os` для взаємодії з операційною системою, модуль `platform` для отримання інформації про платформу, тобто операційну систему, і модуль `logging` для *логування* інформації (англ.» to log information»).

Спочатку ми перевіряємо, яку операційну систему ми використовуємо, перевіряючи рядок, який повертає функція `platform.platform()` (для отримання додаткової інформації див. `import platform; help (platform)`). Якщо це Windows, ми визначаємо home drive (диск) ,який містить домашню папку та назву файлу, де ми хочемо зберігати інформацію. Зібравши ці три частини разом, ми отримаємо повне розташування файлу. Для інших платформ нам потрібно знати лише домашню папку користувача, і ми отримаємо повний шлях до файлу.

Ми використовуємо функцію `os.path.join()`, щоб поєднати ці три частини шляху. Причина використання спеціальної функції, а не простого додавання рядків разом, полягає в тому, що ця функція гарантує, що повний шлях відповідає формату, очікуваному операційною системою.

Примітка : метод `join()`, який ми використовуємо тут є частиною модуля `os`,він відрізняється від рядкового методу `join()`, який ми використовували в інших частинах цієї книги.

Ми налаштовуємо модуль `logging` для запису всіх повідомлень у певному форматі у вказаний файл.

Нарешті, ми можемо виводити повідомлення, призначені для налагодження, інформування, попередження та навіть критичні повідомлення. Після запуску програми ми можемо перевірити цей файл і знати, що сталося в програмі, навіть якщо користувач, який запускає програму, не показує жодної інформації.

3.27.3 Модуль серії тижня

 англійська: *Module of the Week Series*

У стандартній бібліотеці, як-от налагодження (англ."debugging"), обробка параметрів командного рядка(англ."handling command line options"), регулярні вирази(англ."regular expressions") є ще багато чого для вивчення.

Найкращий спосіб глибше вивчити стандартну бібліотеку — це прочитати чудову серію Doug Hellmann "Модуль тижня" (англ."Python Module of the Week") (також доступна як книга)або Офіційну документацію Python.

3.27.4 Резюме

Ми дослідили деякі функції багатьох модулів стандартної бібліотеки Python. Настійно рекомендуємо переглянути документацію стандартної бібліотеки Python, щоб отримати уявлення про всі доступні модулі.

Далі ми розглянемо різні аспекти Python, які зроблять наш огляд Python більш *повним*.

3.28 Більше

Наразі ми розглянули більшість різноманітних аспектів Python, які ви використовуватимете. У цьому розділі ми розглянемо ще деякі аспекти, які зроблять наші знання про Python більш повними.

3.28.1 Передача кортежів

 англійська: *Passing tuples around*

Ви коли-небудь хотіли, щоб функція повернула не один результат, а два? Це можливо. Все, що для цього потрібно, – використовувати кортеж.

українська

код python (idle)

```
>>> def отримати_опис_помилки():
...     return (2,"опис")
...
>>> номер_помилки,рядок_помилки = отримати_опис_помилки()
>>> номер_помилки
2
>>> рядок_помилки
'опис'
```

англійська

python code (idle)

```
>>> def get_error_details():
...     return (2, 'details')
...
>>> errnum, errstr = get_error_details()
>>> errnum
2
>>> errstr
'details'
```

Зауважте, що використання `a, b = <деякий вираз>` (англ.» `a, b = <some expression>`») інтерпретує результат виразу як кортеж із двома значеннями.

Це також означає, що найшвидший спосіб поміняти дві змінні в Python можна наступним чином:

```
>>> a = 5; b = 8
>>> a, b
(5, 8)
>>> a, b = b, a
>>> a, b
(8, 5)
```

3.28.2 Спеціальні методи

Існують певні методи, такі як `__init__` і `__del__`, які мають особливе значення в класах.

Для імітації певної поведінки вбудованих типів даних використовуються спеціальні методи. Наприклад, якщо ви хочете використовувати операцію індексування `x[індекс]` (англ. `x[key]`) для свого класу (так само, як ви використовуєте її для списків і кортежів), тоді все, що вам потрібно зробити, це реалізувати метод `__getitem__()` і ваше завдання зроблено. До речі, саме цей метод Python використовує для класу`list`!

Деякі корисні спеціальні методи перераховані в наступній таблиці. Якщо ви хочете дізнатися про всі спеціальні методи, перегляньте посібник.

- `__init__(self, ...)`

  - Цей метод викликається безпосередньо перед тим, як новостворений об'єкт повертається для використання.

- `__del__(self)`
  - Викликається безпосередньо перед знищенням об'єкта (що має непередбачуваний час, тому уникайте цього)
- `__str__(self)`
  - Викликається, коли ми використовуємо функцію `print` або `str()`.
- `__lt__(self, other)`
  - Викликається, коли використовується оператор *менше ніж* (<). Так само існують спеціальні методи для всіх операторів (+, > тощо)
- `__getitem__(self, key)`
  - Викликається, коли використовується операція індексування `x[індекс]`.
- `__len__(self)`
  - Викликається, коли для об'єкта послідовності використовується вбудована функція `len()`.

3.28.3 Блоки окремих операторів Блоки в один вираз

 англійська: *Single Statement Blocks*

Ми бачили, що кожен блок рядків коду відокремлений від решти власним рівнем відступу. Що ж, є одне застереження. Якщо ваш блок рядків коду містить лише один вираз, ви можете вказати його в одному рядку, скажімо, умовного оператора або оператора циклу. Наступний приклад повинен прояснити це:

```
>>> flag = True
>>> if flag: print('Yes')
...
Yes
```

Зверніть увагу, що єдиний оператор використовується в рядку, а не як окремий блок. Хоча ви можете використовувати це, щоб зробити свою програму *меншою*, я наполегливо рекомендую уникати цього скороченого методу, за винятком перевірки помилок, головним чином тому, що буде набагато легше додати додатковий оператор, якщо ви використовуєте належний відступ.

3.28.4 Лямбда-форми

 англійська: *Lambda Forms*

Ключове слово `lambda` використовується для створення нових функціональних об'єктів (нових функцій та повернення їх значення у час виконання програми). По суті, "лямбда" приймає параметр, за яким слідує один вираз. Лямбда стає тілом функції. Значення цього виразу повертається новою функцією.

код python more_lambda_ukr.py

```
бали = [{'x': 2, 'y': 3},
        {'x': 4, 'y': 1}]
бали.sort(key=lambda i: i['y'])
print(бали)
```

Висновок:

```
[{'x': 4, 'y': 1}, {'x': 2, 'y': 3}]
```

Як це працює

Зверніть увагу, що метод `sort` класа `list` може приймати параметр `key`, який визначає спосіб сортування списку (зазвичай ми думаємо тільки про сортування за зростанням або за спаданням). У нашому випадку ми хочемо виконати спеціальне сортування, і для цього нам потрібно написати функцію. Замість написання окремого блоку `def` для функції, яка використовуватиметься лише в цьому місці, ми використовуємо лямбда-вираз для створення нової функції.

3.28.5 Генератори списків

 англійська: *List Comprehension*

Генератори списків використовується для отримання нового списку з існуючого списку. Уявіть, що є список чисел, на основі якого потрібно отримати новий список, що складається з усіх чисел, помножених на 2, але тільки за умови, що саме число більше 2. Генератори списків ідеально підходить для таких ситуацій.

Приклад англійською (зберегти як `more_list_comprehension.py`):

код python more_list_comprehension_ukr.py

```
список_перший = [2, 3, 4]
список_другий = [2*i for i in список_перший if i > 2]
print(список_другий)
```

Висновок:

```
$ python more_list_comprehension_ukr.py
[6, 8]
```

Як це працює

Тут ми отримуємо новий список, вказуючи маніпуляцію, яку потрібно виконати (`2*i`), коли виконується певна умова (`if i > 2`). Зауважте, що вихідний список залишається без змін.

Перевага використання генераторів списків полягає в тому, що воно зменшує кількість шаблонного коду, необхідного, коли ми використовуємо цикли для обробки кожного елемента списку та збереження його в новому списку.

3.28.6 Передача кортежів та словників у функції

 англійська: *Receiving Tuples and Dictionaries in Functions*

Існує спеціальний спосіб для отримання параметрів переданих функції, у вигляді кортежу або словника за допомогою префікса `*` або `**` відповідно. Це корисно, коли функція приймає змінну кількість аргументів.

українська

код python (idle)

```
>>> def сума_зведення_у_ступінь(зведення_у_ступінь, *args):
...     '''Повертає суму кожного аргументу, зведеного до вказаного степеня.'''
...     результат = 0
...     for i in args:
...         результат += pow(i, зведення_у_ступінь)
...     return результат
...
>>> сума_зведення_у_ступінь(2, 3, 4)
25
>>> сума_зведення_у_ступінь(2, 10)
100
```

англійська

python code (idle)

```
>>> def powersum(power, *args):
...     '''Return the sum of each argument raised to the specified power.'''
...     total = 0
...     for i in args:
...         total += pow(i, power)
...     return total
...
>>> powersum(2, 3, 4)
25
>>> powersum(2, 10)
100
```

Оскільки ми маємо префікс `*` у змінній `args`, усі додаткові аргументи, передані функції, зберігаються в `args` як кортеж. Якби замість нього використовувався префікс `**`, додаткові параметри вважалися б парами ключ/значення словника.

3.28.7 Інструкція assert

 англійська: *The assert statement*

Інструкція `assert` використовується для підтвердження того, чи є задана умова істинною чи ні. Якщо умова виконується, нічого не відбувається, але якщо вона не відповідає дійсності, інструкція `assert` є ідеальним у цій ситуації. Коли задана умова хибна, виникає помилка AssertionError. Метод `pop()` видаляє та повертає останній елемент зі списку.

українська

код python (idle)

```
>>> мій_лист=["елемент"]
>>> assert len(мій_лист) >= 1
>>> мій_лист.pop()
'елемент'
>>> assert len(мій_лист) >= 1
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
  File "/usr/lib/python3.10/idlelib/run.py", line 578, in runcode
    exec(code, self.locals)
  File "<pyshell#4>", line 1, in <module>
AssertionError
```

англійська

python code (idle)

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> assert len(mylist) >= 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

Інструкцію `assert` слід використовувати з обережністю. У більшості випадків краще перехоплювати винятки або вирішувати відповідну проблему автоматично, або відображати повідомлення про помилку користувачеві, а потім вийти із програми.

3.28.8 Декоратори

 англійська: *Decorators*

Декоратори — це швидкий шлях до застосування функцій- обгорток (англ."wrapper functions"). Це функція, яка дозволяє обгортнути іншу функцію для розширення її функціональності без безпосередньої зміни коду. «Обгорнути» функціональність тим самим кодом знову і знову- це корисно. Наприклад, я створив для себе "retry decorator", який я можу просто застосувати до будь-якої функції, і якщо під час запуску викидається(виникає) будь-який виняток, він повторюється ще раз, максимум 5 разів із затримкою між кожною повторною спробою. Це особливо корисно в ситуаціях, коли ви намагаєтеся здійснити мережеве з'єднання з віддаленим комп'ютером:

код python decorators_en.py

```
from time import sleep
from functools import wraps
import logging
logging.basicConfig()
log = logging.getLogger("retry")


def retry(f):
    @wraps(f)
    def wrapper_function(*args, **kwargs):
        MAX_ATTEMPTS = 5
        for attempt in range(1, MAX_ATTEMPTS + 1):
            try:
                return f(*args, **kwargs)
            except Exception:
```

```
                log.exception("Attempt %s/%s failed : %s",
                              attempt,
                              MAX_ATTEMPTS,
                              (args, kwargs))
                sleep(10 * attempt)
        log.critical("All %s attempts failed : %s",
                     MAX_ATTEMPTS,
                     (args, kwargs))
    return wrapper_function


counter = 0


@retry
def save_to_database(arg):
    print("Write to a database or make a network call or etc.")
    print("This will be automatically retried if exception is thrown.")
    global counter
    counter += 1
    # This will throw an exception in the first call
    # And will work fine in the second call (i.e. a retry)
    if counter < 2:
        raise ValueError(arg)


if __name__ == '__main__':
    save_to_database("Some bad value")
```

Output:

```
$ python decorators_en.py
Write to a database or make a network call or etc.
This will be automatically retried if exception is thrown.
ERROR:retry:Attempt 1/5 failed : (('Some bad value',), {})
Traceback (most recent call last):
  File "more_decorator.py", line 14, in wrapper_function
    return f(*args, **kwargs)
  File "more_decorator.py", line 39, in save_to_database
    raise ValueError(arg)
ValueError: Some bad value
Write to a database or make a network call or etc.
This will be automatically retried if exception is thrown.
```

Як це працює

Подивитися:

- Відео: Декоратори Python робляться легко (англ."Video : Python Decorators Made Easy")
- http://www.ibm.com/developerworks/linux/library/l-cpdecor.html
- http://toumorokoshi.github.io/dry-principles-through-python-decorators.html

3.28.9 Відмінності між Python 2 і Python 3

Подивитися:

- "Шість" бібліотек (англ."Six" library»)
- Перенесення на Python 3 Redux від Armin(англ."Porting to Python 3 Redux by Armin")
- Досвід Python 3 від PyDanny(англ."Python 3 experience by PyDanny")
- Офіційний посібник Django з портування на Python 3(англ."Official Django Guide to Porting to Python 3")
- Обговорення на тему «Які переваги має python 3.x?» (англ."Discussion on What are the advantages to python 3.x?"

3.28.10 Резюме

У цій главі ми розглянули ще деякі функції Python, але не всі можливості Python. Однак на цьому етапі ми розглянули більшість того, що ви збираєтеся використовувати на практиці. Цього достатньо, щоб почати роботу з будь-якою програмою, яку ви збираєтеся створювати.

Далі ми обговоримо, як далі досліджувати Python.

3.29 Що далі

Якщо ви до цього часу уважно прочитали цю книгу та практикували написання багатьох програм, то вам, мабуть, стало зручніше з Python. Ви, напевно, створили кілька програм на Python, щоб випробувати щось, а також потренувати свої навички Python. Якщо ви ще цього не зробили, то повинні. Зараз постає питання «Що далі?».

Я пропоную вам вирішити цю проблему (завдання):

> Створіть власну програму *Адресна книга*, що працює з командного рядка. За допомогою даної програми ви можете переглядати, додавати, змінювати, видаляти або шукати контактні дані, такі як дані друзів, родини, колег, а також інформацію про них, наприклад адресу електронної пошти та/або номер телефону. Деталі необхідно зберегти для подальшого пошуку.

Це досить легко, якщо подумати про це в термінах, з якими ми стикалися досі. Якщо ви все ще хочете отримати вказівки, як діяти далі, ось підказка[1].

Як тільки ви зможете це зробити, ви можете претендувати на те, щоб бути програмістом Python. Тепер негайно надішліть мені електронний лист із подякою за цю чудову книгу ;-). Цей крок необов'язковий, але рекомендований. Крім того, розгляньте купівлю друкованої копії, щоб підтримати продовження розробки цієї книги.

Якщо вам ця проблема (завдання) здалася легкою, ось ще одна:

> Виконайте команду replace (укр."замінити"). Ця команда замінить один рядок іншим у списку наданих файлів.

Команда replace може бути будь-якої бажаної складності: від простої заміни рядків до пошуку за шаблоном (регулярним виразом).

[1] Створіть клас для збереження персональних даних. Об'єкти візитних карток зберігайте у словнику, в якому імена контактів будуть ключами. Використовуйте модуль pickle для тривалого зберігання об'єктів на жорсткому диску. Використовуйте вбудовані методи словника, щоб додавати, видаляти та змінювати контакти.

3.29.1 Наступні проекти

Якщо ви виявили, що вищезазначені програми легко створювати, перегляньте цей вичерпний список проектів і спробуйте написати власні програми:

- https://github.com/thekarangoel/Projects#numbers (той самий список також є в Martyr2's Mega Project List).

Дивіться також:

- Exercises for Programmers: 57 Challenges to Develop Your Coding Skills)
- Intermediate Python Projects.

3.29.2 Приклад коду

 англійська: *Example Code*

Найкращий спосіб вивчити мову програмування – це писати багато коду та читати багато коду:

- Python Cookbook — це надзвичайно цінна колекція рецептів (порад) щодо вирішення певних проблем за допомогою Python. Це обов'язково для прочитання для кожного користувача Python.
- Python Module of the Week — це ще один чудовий посібник із *Стандартної бібліотеки*, який потрібно прочитати.

3.29.3 Порада

- The Hitchhiker's Guide to Python!
- The Elements of Python Style
- Python Big Picture
- Writing Idiomatic Python» ebook (платно)

3.29.4 Відео

- Full Stack Web Development with Flask
- PyVideo

3.29.5 Питання та відповіді

- Official Python Dos and Don'ts
- Official Python FAQ
- Norvig's list of Infrequently Asked Questions
- Python Interview Q & A
- StackOverflow questions tagged with python

3.29.6 Підручники

 англійська: *Tutorials*

- Hidden features of Python
- What's the one code snippet/python trick/etc did you wish you knew when you learned python?
- Awaretek's comprehensive list of Python tutorials

3.29.7 Обговорення

Якщо ви застрягли з проблемою Python і не знаєте, кого запитати, то перелік розсилки python-tutor (англ."python-tutor list") — найкраще місце, щоб поставити свої запитання.

Переконайтеся, що ви виконали домашнє завдання, спробувавши спершу розв'язати задачу самостійно, і тільки потім поставити розумні запитання (англ."ask smart questions").

3.29.8 Новини

Якщо вас цікавлять останні новини світу Python, відстежуйте їх на офіційній планеті Python.

3.29.9 Встановлення бібліотек

 англійська: *Installing libraries*

У Каталогу пакетів Python (англ."Python Package Index") є величезна кількість бібліотек з відкритим кодом, які ви можете використовувати у своїх програмах.

Щоб установити та використовувати ці бібліотеки, ви можете використовувати pip.

3.29.10 Створення сайту

Вивчіть Flask, щоб створити власний веб-сайт. Деякі ресурси для початку:

- Flask Official Quickstart
- The Flask Mega-Tutorial
- Example Flask Projects

3.29.11 Графічне програмне забезпечення

 англійська: *Graphical Software*

Припустімо, ви хочете створити власні графічні програми за допомогою Python. Це можна зробити за допомогою бібліотеки ГІК (графічного інтерфейсу користувача),  англійська: *GUI - Graphical User Interface* зі своїми прив'язками (англ."bindings") до Python.Прив'язки (англ."bindings") – це те, що дозволяє вам писати програми на Python і використовувати бібліотеки, які самі написані на C або C++ або іншими мовами.

Є багато варіантів для GUI з використанням Python:

- Kivy
  - http://kivy.org
- PyGTK
  - Це прив'язка Python до набору інструментів GTK+, на основі якого побудовано GNOME. GTK+ має багато особливостей у використанні, але як тільки ви навчитеся, ви зможете швидко створювати програми з графічним інтерфейсом. Конструктор графічного інтерфейсу Glade незамінний. Документацію ще потрібно покращити. GTK+ добре працює на GNU/Linux, але його перенесення на Windows не завершено. За допомогою GTK+ можна створювати як вільні, так і пропрієтарні програми. Щоб почати, прочитайте PyGTK tutorial.
- PyQt

  - Це прив'язка Python для набору інструментів Qt, який є основою, на якій побудована KDE. Qt надзвичайно простий у використанні та дуже потужний, особливо завдяки Qt Designer та чудовій документації Qt. PyQt безкоштовно, якщо використовується для створення вільних програм (з ліцензією GPL). Для створення закритих пропрієтарних програм вам доведеться його купити.Починаючи з Qt 4.5, дозволяється створювати за допомогою нього не лише GPL'ні програми. Щоб почати, прочитайте про PySide.

- wxPython -Це прив'язки Python для набору інструментів wxWidgets. wxPython не такий простий в освоєнні. Однак він дуже портативний і працює на GNU/Linux, Windows, Mac і навіть на вбудованих платформах. Багато середовищ розробки для wxPython, такі як SPE (Stani's Python Editor) та wxGlade включають дизайнери графічного інтерфейсу. За допомогою wxPython можна створювати як вільні, так і пропрієтарні програми. Для початку прочитайте підручник з wxPython tutorial.

Резюме з інструментів ГІК

 англійська: *GUI Summary of GUI Tools*

Щоб дізнатися більше, перегляньте wiki-сторінці GuiProgramming офіційного сайту Python .

На жаль, для Python не існує жодного стандартного графічного інструменту. Я пропоную вам вибрати один із наведених вище інструментів залежно від вашої ситуації. Перший фактор полягає в тому, чи готові ви платити за використання будь-якого інструменту GUI. Другий фактор полягає в тому, чи хочете ви, щоб програма запускалася лише на Windows, чи на Mac і GNU/Linux, чи на всіх. І по-третє, якщо ви вибрали платформу GNU/Linux,то в якому середовищі ви працюєте: у KDE чи у GNOME.

Для більш детального та всебічного аналізу див. сторінку 26 'The Python Papers, Volume 3, Issue 1' (PDF).

3.29.12 Різні реалізації

 англійська: *Various Implementations*

Зазвичай мова програмування складається з двох частин - мови та програмного забезпечення. Мова - це *як* ви щось пишете. Програмне забезпечення – це те, *що* запускає наші програми.

Ми використовували програмне забезпечення *CPython* для запуску наших програм. Його називають CPython, оскільки він написаний мовою C і є *класичним інтерпретатором Python*(англ."*Classical Python interpreter*").

Але існує й інше програмне забезпечення, здатне виконувати програми на Python:

- Jython
  - Реалізація Python, яка працює на платформі Java. Це означає, що ви можете використовувати бібліотеки та класи Java в програмі Python і навпаки.
- IronPython
  - Реалізація Python, яка працює на платформі .NET,що означає можливість використання бібліотек та класів .NET в програмах Python і навпаки.
- PyPy
  - Реалізація Python, написана на Python! Це дослідницький проект, спрямований на те, щоб швидко та легко вдосконалити інтерпретатор, оскільки сам інтерпретатор написаний динамічною мовою (на відміну від статичних мов, таких як C, Java або C# у трьох наведених вище реалізаціях).

Існують також інші реалізації, такі як CLPython – реалізація Python, написана мовою Common Lisp і Brython, яка є реалізацію поверх інтерпретатора JavaScript, що може означати, що ви можете використовувати Python (замість JavaScript) для написання програм веб-браузера («Ajax»).

Кожна з цих реалізацій має свої спеціальні області, де вони корисні.

3.29.13 Функціональне програмування (для досвідчених читачів)

Коли ви починаєте писати більші програми, вам обов'язково слід дізнатися більше про функціональний підхід до програмування на відміну від підходу до програмування, що базується на класах, про який ми дізналися в *розділі про об'єктно-орієнтоване програмування*:

- Functional Programming Howto by A.M. Kuchling
- Functional programming chapter in 'Dive Into Python' book
- Functional Programming with Python presentation
- Funcy library
- PyToolz library

3.29.14 Резюме

Ми підійшли до кінця цієї книги, але, як кажуть, це *початок кінця*! Тепер ви затятий користувач Python і, без сумніву, готові вирішити багато проблем за допомогою Python. Ви можете почати автоматизувати свій комп'ютер, щоб робити всі види раніше немислимих речей або писати власні ігри та багато іншого. Отже, починайте!

3.30 Що далі

Якщо ви до цього часу уважно прочитали цю книгу та практикували написання багатьох програм, то вам, мабуть, стало зручніше з Python. Ви, напевно, створили кілька програм на Python, щоб випробувати щось, а також потренувати свої навички Python. Якщо ви ще цього не зробили, то повинні. Зараз постає питання «Що далі?».

Я пропоную вам вирішити цю проблему (завдання):

> Створіть власну програму *Адресна книга*, що працює з командного рядка. За допомогою даної програми ви можете переглядати, додавати, змінювати, видаляти або шукати контактні дані, такі як дані друзів, родини, колег, а також інформацію про них, наприклад адресу електронної пошти та/або номер телефону. Деталі необхідно зберегти для подальшого пошуку.

Це досить легко, якщо подумати про це в термінах, з якими ми стикалися досі. Якщо ви все ще хочете отримати вказівки, як діяти далі, ось підказка[с. 175, 1].

Як тільки ви зможете це зробити, ви можете претендувати на те, щоб бути програмістом Python. Тепер негайно надішліть мені електронний лист із подякою за цю чудову книгу ;-). Цей крок необов'язковий, але рекомендований. Крім того, розгляньте купівлю друкованої копії, щоб підтримати продовження розробки цієї книги.

Якщо вам ця проблема (завдання) здалася легкою, ось ще одна:

> Виконайте команду replace (укр."замінити"). Ця команда замінить один рядок іншим у списку наданих файлів.

Команда replace може бути будь-якої бажаної складності: від простої заміни рядків до пошуку за шаблоном (регулярним виразом).

3.30.1 Наступні проекти

Якщо ви виявили, що вищезазначені програми легко створювати, перегляньте цей вичерпний список проектів і спробуйте написати власні програми:https://github.com/thekarangoel/Projects#numbers (той самий список також є в Martyr2's Mega Project List).

Дивіться також:

- Exercises for Programmers: 57 Challenges to Develop Your Coding Skills)
- Intermediate Python Projects.

3.30.2 Приклад коду

 англійська: *Example Code*

Найкращий спосіб вивчити мову програмування – це писати багато коду та читати багато коду:

- Python Cookbook — це надзвичайно цінна колекція рецептів (порад) щодо вирішення певних проблем за допомогою Python. Це обов'язково для прочитання для кожного користувача Python.
- Python Module of the Week — це ще один чудовий посібник із *Стандартної бібліотеки*, який потрібно прочитати.

3.30.3 Порада

- The Hitchhiker's Guide to Python!
- The Elements of Python Style
- Python Big Picture
- Writing Idiomatic Python» ebook (платно)

3.30.4 Відео

- Full Stack Web Development with Flask
- PyVideo

3.30.5 Питання та відповіді

- Official Python Dos and Don'ts
- Official Python FAQ
- Norvig's list of Infrequently Asked Questions
- Python Interview Q & A
- StackOverflow questions tagged with python

3.30.6 Підручники

 англійська: *Tutorials*

- Hidden features of Python
- What's the one code snippet/python trick/etc did you wish you knew when you learned python?
- Awaretek's comprehensive list of Python tutorials

3.30.7 Обговорення

Якщо ви застрягли з проблемою Python і не знаєте, кого запитати, то перелік розсилки python-tutor (англ."python-tutor list") — найкраще місце, щоб поставити свої запитання.

Переконайтеся, що ви виконали домашнє завдання, спробувавши спершу розв'язати задачу самостійно, і тільки потім поставити розумні запитання (англ."ask smart questions").

3.30.8 Новини

Якщо вас цікавлять останні новини світу Python, відстежуйте їх на офіційній планеті Python.

3.30.9 Встановлення бібліотек

 англійська: *Installing libraries*

У Каталогу пакетів Python (англ."Python Package Index") є величезна кількість бібліотек з відкритим кодом, які ви можете використовувати у своїх програмах.

Щоб установити та використовувати ці бібліотеки, ви можете використовувати pip.

3.30.10 Створення сайту

Вивчіть Flask, щоб створити власний веб-сайт. Деякі ресурси для початку:

- Flask Official Quickstart
- The Flask Mega-Tutorial
- Example Flask Projects

3.30.11 Графічне програмне забезпечення

 англійська: *Graphical Software*

Припустімо, ви хочете створити власні графічні програми за допомогою Python. Це можна зробити за допомогою бібліотеки ГІК (графічного інтерфейсу користувача),  англійська: *GUI - Graphical User Interface* зі своїми прив'язками (англ."bindings") до Python.Прив'язки (англ."bindings") – це те, що дозволяє вам писати програми на Python і використовувати бібліотеки, які самі написані на C або C++ або іншими мовами.

Є багато варіантів для GUI з використанням Python:

- Kivy
  - http://kivy.org
- PyGTK
  - Це прив'язка Python до набору інструментів GTK+, на основі якого побудовано GNOME. GTK+ має багато особливостей у використанні, але як тільки ви навчитеся, ви зможете швидко створювати програми з графічним інтерфейсом. Конструктор графічного інтерфейсу Glade незамінний. Документацію ще потрібно покращити. GTK+ добре працює на GNU/Linux, але його перенесення на Windows не завершено. За допомогою GTK+ можна створювати як вільні, так і пропрієтарні програми. Щоб почати, прочитайте PyGTK tutorial.
- PyQt

  - Це прив'язка Python для набору інструментів Qt, який є основою, на якій побудована KDE. Qt надзвичайно простий у використанні та дуже потужний, особливо завдяки Qt Designer та чудовій документації Qt. PyQt безкоштовно, якщо використовується для створення вільних програм (з ліцензією GPL). Для створення закритих пропрієтарних програм вам доведеться його купити.Починаючи з Qt 4.5, дозволяється створювати за допомогою нього не лише GPL'ні програми. Щоб почати, прочитайте про PySide.

- wxPython -Це прив'язки Python для набору інструментів wxWidgets. wxPython не такий простий в освоєнні. Однак він дуже портативний і працює на GNU/Linux, Windows, Mac і навіть на вбудованих платформах. Багато середовищ розробки для wxPython, такі як SPE (Stani's Python Editor) та wxGlade включають дизайнери графічного інтерфейсу. За допомогою wxPython можна створювати як вільні, так і пропрієтарні програми. Для початку прочитайте підручник з wxPython tutorial.

Резюме з інструментів ГІК

 англійська: *GUI Summary of GUI Tools*

Щоб дізнатися більше, перегляньте wiki-сторінці GuiProgramming офіційного сайту Python .

На жаль, для Python не існує жодного стандартного графічного інструменту. Я пропоную вам вибрати один із наведених вище інструментів залежно від вашої ситуації. Перший фактор полягає в тому, чи готові ви платити за використання будь-якого інструменту GUI. Другий фактор полягає в тому, чи хочете ви, щоб програма запускалася лише на Windows, чи на Mac і GNU/Linux, чи на всіх. І по-третє, якщо ви вибрали платформу GNU/Linux,то в якому середовищі ви працюєте: у KDE чи у GNOME.

Для більш детального та всебічного аналізу див. сторінку 26 'The Python Papers, Volume 3, Issue 1' (PDF).

3.30.12 Різні реалізації

 англійська: *Various Implementations*

Зазвичай мова програмування складається з двох частин - мови та програмного забезпечення. Мова - це *як* ви щось пишете. Програмне забезпечення – це те, *що* запускає наші програми.

Ми використовували програмне забезпечення *CPython* для запуску наших програм. Його називають CPython, оскільки він написаний мовою C і є *класичним інтерпретатором Python*(англ."*Classical Python interpreter*").

Але існує й інше програмне забезпечення, здатне виконувати програми на Python:

- Jython
  - Реалізація Python, яка працює на платформі Java. Це означає, що ви можете використовувати бібліотеки та класи Java в програмі Python і навпаки.
- IronPython
  - Реалізація Python, яка працює на платформі .NET,що означає можливість використання бібліотек та класів .NET в програмах Python і навпаки.
- PyPy
  - Реалізація Python, написана на Python! Це дослідницький проект, спрямований на те, щоб швидко та легко вдосконалити інтерпретатор, оскільки сам інтерпретатор написаний динамічною мовою (на відміну від статичних мов, таких як C, Java або C# у трьох наведених вище реалізаціях).

Існують також інші реалізації, такі як CLPython – реалізація Python, написана мовою Common Lisp і Brython, яка є реалізацію поверх інтерпретатора JavaScript, що може означати, що ви можете використовувати Python (замість JavaScript) для написання програм веб-браузера («Ajax»).

Кожна з цих реалізацій має свої спеціальні області, де вони корисні.

3.30.13 Функціональне програмування (для досвідчених читачів)

Коли ви починаєте писати більші програми, вам обов'язково слід дізнатися більше про функціональний підхід до програмування на відміну від підходу до програмування, що базується на класах, про який ми дізналися в *розділі про об'єктно-орієнтоване програмування*:

- Functional Programming Howto by A.M. Kuchling
- Functional programming chapter in 'Dive Into Python' book
- Functional Programming with Python presentation
- Funcy library
- PyToolz library

3.30.14 Резюме

Ми підійшли до кінця цієї книги, але, як кажуть, це *початок кінця*! Тепер ви затятий користувач Python і, без сумніву, готові вирішити багато проблем за допомогою Python. Ви можете почати автоматизувати свій комп'ютер, щоб робити всі види раніше немислимих речей або писати власні ігри та багато іншого. Отже, починайте!

3.31 Додаток: Про книгу

3.31.1 Колофон

Майже все програмне забезпечення, яке я використовував для створення цієї книги, є *ВВПЗ (англ."FLOSS")*.

3.31.2 Народження Книги

При написанні первинного начерку цієї книги в основі моєї системи була Red Hat 9.0 Linux, але вже шосту версію чернетки я писав на Fedora Core 3 Linux. Спочатку я використовував KWord для написання книги (як пояснюється в *уроці історії(англ."history lesson")*.

3.31.3 Підліткові роки

Пізніше я перейшов на формат DocBook XML та використовував Kate,але це здалося мені надто нудним. Отже, я перейшов на OpenOffice,який чудово підходив зі своїм рівнем управління форматуванням та можливістю генерувати PDF, але він створював дуже неохайний HTML із документа.

Нарешті я відкрив для себе XEmacs і переписав книгу з нуля в форматі DocBook XML (знову), після того як вирішив, що цей формат є довгостроковим рішенням.

Для шостої версії чернетки я вирішив використати Quanta+.При цьому я використав стандартні таблиці стилів XSL, які йшли у комплекті з Fedora Core 3 Linux.Потім я написав документ CSS, щоб додати колір і стиль сторінкам HTML. Я також написав грубий лексичний аналізатор, звісно на Python, який автоматично надавав підсвічування синтаксису в прикладах програм.

Для сьомої редакції я використовував MediaWiki як основу мого сайта. Тепер я все редагую прямо на сайті, і читачі можуть безпосередньо читати/редагувати/обговорювати вміст на вікі-сторінці, але зрештою я витратив більше часу на боротьбу зі спамом, ніж на написання.

Для восьмої версії чернетки я використовував Vim, Pandoc, and Mac OS X.

Для дев'ятої версії чернетки я перейшов на AsciiDoc format і використав Emacs 24.3, tomorrow theme, Fira Mono font та adoc-mode для написання книги.

3.31.4 Зараз

2016: Я втомився від кількох дрібних проблем із відтворенням у AsciiDoctor, як-от «++» у «C/C++» зникав, і мені було важко відслідковувати такі незначні речі. Крім того, мені не хотілося редагувати текст через складний формат Asciidoc.

Для десятої чернетки я перейшов до написання у форматі Markdown + GitBook, використовуючи редактор Spacemacs.

Листопад 2020: оскільки Gitbook відмовився від свого програмного забезпечення з відкритим кодом, перейшов на Honkit, a community-maintained fork of Gitbook legacy.

3.31.5 Про автора

Перегляньте # Додаток : FLOSS

3.31.6 Вільне та Відкрите Програмне Забезпечення (*ВВПЗ*)

англійська: *Free/Libre and Open Source Software (FLOSS)*

Примітка

Будь ласка, зверніть увагу, що цей розділ був написаний у 2003 році, тому деякі речі можуть здатися вам трохи застарілими :-)

ВВПЗ ґрунтується на концепції спільноти, в якій прийнято ділитися і обмінюватися знаннями. ВВПЗ безкоштовне для використання, модифікації та розповсюдження.

Якщо ви вже прочитали цю книгу, ви вже знайомі з ВВПЗ, так як ви вивчали Python весь цей час, а Python є вільним програмним забезпеченням!

Ось кілька прикладів ВВПЗ, за якими можна скласти деяке уявлення про те, що здатне створити така спільнота:

Linux: це вільне ядро операційної системи, що використовується в операційній системі GNU/Linux. Розробку ядра «Linux» було створено Лінусом Торвальдсом ще студентом. Android базується на Linux. Будь-який веб-сайт, яким ви користуєтеся сьогодні, здебільшого працює на Linux.

Ubuntu: це дистрибутив, керований спільнотою, який спонсорує Canonical, і це найпопулярніший дистрибутив GNU/Linux на сьогодні. Це дозволяє встановити безліч доступних ВВПЗ, і все це простим у використанні та легким у встановленні способом. Найкраще те, що ви можете просто перезавантажити комп'ютер і запустити GNU/Linux із компакт-диска! Це дає змогу повністю випробувати нову ОС перед встановленням її на комп'ютері. Однак Ubuntu не є повністю безкоштовним програмним забезпеченням; він містить пропрієтарні драйвери, мікропрограми та додатки.

LibreOffice: це чудовий офісний пакет, розроблений і підтримуваний спільнотою. Він містить текстовий редактор, засіб для створення презентацій, табличний процесор, інструмент для малювання та інші

компоненти. Він без проблем відкриває та редагує файли MS Word і MS PowerPoint. Працює практично на всіх платформах і є повністю вільним, відкритим та безкоштовним програмним забезпеченням.

Mozilla Firefox: це *найкращий* веб-переглядач. Він неймовірно швидкий і отримав визнання критиків за свої розумні та вражаючі функції. Концепція розширень дозволяє використовувати будь-які плагіни.

Mono: це безкоштовна реалізація платформи Microsoft .NET. Вона дозволяє створювати та запускати програми .NET на GNU/Linux, Windows, FreeBSD, Mac OS та багатьох інших платформах.

Apache web server: це популярний відкритий веб-сервер. Фактично, це *найпопулярніший* веб-сервер на планеті! Він керує майже половиною веб-сайтів. Так, саме так – Apache обробляє більше веб-сайтів, ніж решта веб-серверів (включаючи Microsoft IIS) разом узяті.

VLC Player: це програвач, який може відтворювати все, починаючи від DivX і MP3, Ogg, VCD, DVD, ... і хто сказав, що це не забавно? ;-)

Цей список призначений лише для того, щоб коротко передати вам думку, насправді існує ще безліч вільного ПЗ, такого як мова Perl (Language), мова PHP (Language), система керування вмістом вебсайтів Drupal, сервер баз даних PostgreSQL (Database Server), гра TORCS (Racing Game),середовище розробки KDevelop (IDE), програвач Xine (X the movie player),редактор VIM editor,редактор Quanta+ editor,аудіоплеєр Banshee audio player,графічний редактор GIMP (image editing program), ... Цей список можна продовжувати вічно.

Щоб відстежити свіжі чутки у світі вільного програмного забезпечення, відвідайте такі сайти:

- OMG! Ubuntu!
- Web Upd8
- DistroWatch
- Planet Debian

Дізнатися більше про вільне програмне забезпечення можна на наступних сайтах:

- GitHub Explore
- Code Triage
- SourceForge
- FreshMeat

Отже, вперед і досліджуйте величезний, вільний і відкритий світ ВВПЗ!

3.32 Додаток: про книгу

3.32.1 Колофон

Майже все програмне забезпечення, яке я використовував для створення цієї книги, є *ВВПЗ (англ."FLOSS")*.

3.32.2 Народження Книги

При написанні первинного начерку цієї книги в основі моєї системи була Red Hat 9.0 Linux, але вже шосту версію чернетки я писав на Fedora Core 3 Linux. Спочатку я використовував KWord для написання книги (як пояснюється в *уроці історії(англ."history lesson")*.

3.32.3 Підліткові роки

Пізніше я перейшов на формат DocBook XML та використовував Kate,але це здалося мені надто нудним. Отже, я перейшов на OpenOffice,який чудово підходив зі своїм рівнем управління форматуванням та можливістю генерувати PDF, але він створював дуже неохайний HTML із документа.

Нарешті я відкрив для себе XEmacs і переписав книгу з нуля в форматі DocBook XML (знову), після того як вирішив, що цей формат є довгостроковим рішенням.

Для шостої версії чернетки я вирішив використати Quanta+.При цьому я використав стандартні таблиці стилів XSL, які йшли у комплекті з Fedora Core 3 Linux.Потім я написав документ CSS, щоб додати колір і стиль сторінкам HTML. Я також написав грубий лексичний аналізатор, звісно на Python, який автоматично надавав підсвічування синтаксису в прикладах програм.

Для сьомої редакції я використовував MediaWiki як основу мого сайта. Тепер я все редагую прямо на сайті, і читачі можуть безпосередньо читати/редагувати/обговорювати вміст на вікі-сторінці, але зрештою я витратив більше часу на боротьбу зі спамом, ніж на написання.

Для восьмої версії чернетки я використовував Vim, Pandoc, and Mac OS X.

Для дев'ятої версії чернетки я перейшов на AsciiDoc format і використав Emacs 24.3, tomorrow theme, Fira Mono font та adoc-mode для написання книги.

3.32.4 Зараз

2016: Я втомився від кількох дрібних проблем із відтворенням у AsciiDoctor, як-от «++» у «C/C++» зникав, і мені було важко відслідковувати такі незначні речі. Крім того, мені не хотілося редагувати текст через складний формат Asciidoc.

Для десятої чернетки я перейшов до написання у форматі Markdown + GitBook, використовуючи редактор Spacemacs.

Листопад 2020: оскільки Gitbook відмовився від свого програмного забезпечення з відкритим кодом, перейшов на Honkit, a community-maintained fork of Gitbook legacy.

3.32.5 Про автора

Перегляньте

3.33 Додаток: Урок історії

Я вперше почав працювати з Python, коли мені потрібно було написати інсталятор для програмного забезпечення під назвою «Diamond», щоб я міг спростити інсталяцію. Мені довелося вибирати між прив'язками Python і Perl для бібліотеки Qt. Я трохи дослідив Інтернет і натрапив на статтю Еріка С. Реймонда, відомого та шанованого хакера, де він розповідав про те, як Python став його улюбленою мовою програмування. Я також дізнався, що прив'язки PyQt були більш зрілими порівняно з Perl-Qt. Отже, я вирішив, що мова Python для мене.

Тоді я почав шукати хорошу книгу про Python. І не знайшов! Я знайшов деякі книги O'Reilly, але вони або були занадто дорогими, або більше нагадували довідник, ніж путівник. Так що мені довелося задовольнятися документацією, яка постачається в комплекті з Python. Однак вона була занадто короткою та малою. Безперечно, вона дала мені деяке уявлення про те, що таке Python, але цього було явно замало. Я впорався з документацією, оскільки мав попередній досвід програмування, але вона не підходила для новачків.

Приблизно через шість місяців після мого першого знайомства з Python я встановив (на той час) найновішу версію Red Hat 9.0 Linux і бавився з KWord. Я був у захваті від цього, і раптом у мене виникла ідея написати щось о Python. Я почав писати кілька сторінок, але швидко це стало 30 сторінок.

Тоді я вирішив серйозно надати цьому тексту форму книги. Після *великої* кількості переписувань вона досягла такого етапу, коли стала корисним посібником для вивчення мови Python. Я вважаю цю книгу своїм внеском і даниною спільноті відкритого коду.

Ця книга почалась як мої особисті нотатки про Python, і я досі розглядаю її так само, хоча я доклав багато зусиль, щоб зробити її більш приємною для інших :)

У справжньому дусі відкритого коду я отримав багато конструктивних пропозицій, критики та *відгуків* від захоплених читачів, що дуже допомогло мені покращити цю книгу.

3.33.1 Статус книги

Книзі потрібна допомога таких читачів, як ви, щоб вказати на будь-які частини книги, які є поганими, незрозумілими або просто неправильними. Будь ласка, напишіть головному автору або відповідним *перекладачам* свої коментарії та пропозиції.

3.34 Додаток: Історія версій

- Без змін версії
  - 06 Nov 2020 06/11/2020
  - Перенесено із залишеного на GitBook на community-maintained Honkit, a fork of GitBook legacy
- 4.0
  - 19 Jan 2016 19/01/2016
  - Повернувся до Python 3
  - Повернувся до Markdown, використовуючи GitBook та Spacemacs
- 3.0
  - 31 Mar 2014 31/03/2014
  - Переписано для Python 2 з використанням AsciiDoc та adoc-mode.
- 2.1
  - 03 Aug 2013 03/08/2013
  - Переписано за допомогою Markdown та Jason Blevins' Markdown Mode
- 2.0
  - 20 Oct 2012 20/10/2012
  - Переписано у форматі Pandoc format, дякую моїй дружині, яка перевела велику частина тексту з формату Mediawiki
  - Спрощення тексту, видалення несуттєвих розділів, таких як `nonlocal` і метакласи
- 1.90
  - 04 Sep 2008 і все ще триває
  - Відродження після перерви в 3,5 роки!
  - Оновлення для Python 3.0
  - Переписано у форматі http://www.mediawiki.org[MediaWiki] (ще раз)
- 1.20

  - 13 Jan 2005
  - Повне переписування за допомогою Quanta+ на Fedora Core 3 із багатьма виправленнями та оновленнями. Багато нових прикладів. Переписав налаштування DocBook з нуля.
- 1.15
  - 28 Mar 2004
  - Незначні доопрацювання
- 1.12
  - 16 Mar 2004
  - Доповнення та виправлення
- 1.10
  - 09 Mar 2004
  - Виправлення помилок, завдяки безлічі відгуків зацікавлених читачів.
- 1.00
  - 08 Mar 2004
  - Після колосальної кількості відгуків та пропозицій від читачів я зробив значну переробку тексту поряд із виправленням друкарських помилок.
- 0.99
  - 22 Feb 2004
  - Додано нову главу про модулі. Додано інформацію про змінну кількість аргументів у функціях.
- 0.98
  - 16 Feb 2004
  - Написав сценарій Python і таблицю стилів CSS для покращення вихідних даних XHTML, включаючи грубий, але функціональний лексичний аналізатор для автоматичного підсвічування синтаксису, подібного до VIM, у прикладах програм.
- 0.97
  - 13 Feb 2004
  - Ще одна повністю переписана чернетка в DocBook XML (знову). Книжка значно покращилася - стала більш зв'язною та читабельною.
- 0.93
  - 25 Jan 2004
  - Додано опис IDLE та багато іншого, що стосується Windows
- 0.92
  - 05 Jan 2004
  - Зміни в кількох прикладах.
- 0.91
  - 30 Dec 2003
  - Виправлені помилки. Зроблено начерки багатьох розділів.

- 0.90
  - 18 Dec 2003
  - Додано ще 2 розділи. Формат OpenOffice із змінами.
- 0.60
  - 21 Nov 2003
  - Повністю переписано та розширено.
- 0.20
  - 20 Nov 2003
  - Виправлено деякі опечатки та помилки.
- 0.15
  - 20 Nov 2003
  - Переведено на форматDocBook XML за допомогою XEmacs.
- 0.10
  - 14 Nov 2003
  - Первинний начерк у редакторі KWord.
  -

3.35 Додаток: Переклади

Завдяки багатьом невтомним волонтерам доступно багато перекладів книги різними мовами!

Якщо ви хочете допомогти з цими перекладами, будь ласка, перегляньте список волонтерів для відповідних мов нижче та вирішіть, чи хочете ви розпочати новий переклад чи допомогти в уже існуючих проектах перекладу.

Якщо ви плануєте почати новий переклад, будь ласка, прочитайте *Інструкцію з перекладу*.

3.35.1 Арабська

Нижче наведено посилання на арабську версію. Дякуємо Ashraf Ali Khalaf за переклад книги, ви можете прочитати всю книгу онлайн за адресою http://www.khaledhosny.org/byte-of-python/index.html або завантажити її з [sourceforge.net](http://downloads.sourceforge.net/omlx/byteofpython_arabic.pdf?use_mirror=osdn). Для отримання додаткової інформації див. http://itwadi.com/byteofpython_arabi.

3.35.2 Азербайджанська

Jahangir Shabiyev (c.shabiev@gmail.com) зголосився перекласти книгу азербайджанською мовою. Переклад виконується за адресою https://www.gitbook.com/book/jahangir-sh/piton-sancmasi

3.35.3 Бразильська Португальська

Є два переклади з різними рівнями завершеності та доступності. Старіший переклад зараз відсутній/втрачений, а новіший переклад неповний.

Samuel Dias Neto (samuel.arataca@gmail.com) зробив перший бразильський португальський переклад (старіший переклад) цієї книги, коли версія Python була 2.3.5. Переклад більше не є загальнодоступним.

Rodrigo Amaral (rodrigoamaral@gmail.com) зголосився перекласти книгу бразильською португальською мовою (новіший переклад), який ще не завершено.

Каталонська

Moises Gomez (moisesgomezgiron@gmail.com) зголосився перекласти книгу каталонською мовою. Переклад триває.

> Moisès Gómez - Я розробник, а також викладач програмування (як правило, для людей без попереднього досвіду).
>
> Деякий час тому мені потрібно було навчитися програмувати на Python, і робота Swaroop'a була справді корисною. Чітко, лаконічно та досить повно. Саме те, що мені було потрібно.
>
> Після цього досвіду я подумав, що деякі інші люди в моїй країні також можуть скористатися цим. Але англійська мова може стати перешкодою.
>
> Отже, чому б не спробувати це перекласти? І я зробив для попередньої версії книги.
>
> У моїй країні є дві державні мови. Я вибрав каталонську, тому що подумав, що на більш поширену іспанську її напевно переведе хтось інший.

3.35.4 Китайський

У 2017 році, тобто через 11 років, Mo Lun (i@molun.net) повторно переклав книгу з самого початку, на основі версії 4.0. А переклад зберігається на GitHub і Gitbook. Він стежить за цим перекладеним виданням і готовий виправити його, якщо в перекладеній версії книги є якась помилка.

Видання перекладу 2017 року доступне на https://bop.molun.net.

Mo Lun каже:

> Я звичайний студент журналістики Китайського молодіжного університета політичних наук (англ."China Youth University for Political Sciences (CYU)"), Пекін. І фактично, коли я починав перекладати цю книгу, я був абсолютний новачок у програмуванні на Python. Спочатку це була просто примха, але коли я закінчив переклад, я зрозумів, що рішення, викликане інтересом, змусило мене зайти так далеко.
>
> За допомогою перекладів моїх попередників,великої кількості інформації в Інтернет, і за допомогою моїх друзів я зміг представити це видання. Я просто сподіваюся, що моя перекладацька робота допоможе іншим новачкам у вивченні Python.
>
> Водночас я завжди чекаю зауважень і пропозицій до мого перекладу і готовий змінити або вдосконалити цю поверхневу роботу.

Попередній китайський переклад

В 2005, Shen Jieyuan переклав цю книгу з версією 1.20 китайською мовою та опублікував її в Інтернеті. Це перше китайське видання. На офіційному сайті книги його звали Juan Shen,за адресою електронної пошти orion_val@163.comЦе видання було широко розповсюджене в мережі, а посилання, надані на офіційному сайті книги, більше не доступні, тому його оригінальне джерело неможливо знайти. Тому тут не можна вказати певну адресу. Але ви можете спробувати знайти копію за ключовими словами, наприклад. . .

Juan Shen каже:

> Я аспірант відділу бездротових телекомунікацій у Пекінському технологічному університеті, Китай. Наразі мої дослідницькі інтереси стосуються синхронізації, регулюванням каналу передачі даних і багатокористувацьким визначенням системи з багатьма несучими частотами CDMA. Python — моя основна мова програмування для щоденного моделювання та дослідницької роботи. Здебільшого з використанням Python Numeric. Я познайомився з Python

лише півроку тому, але, як ви бачите, він дійсно простий у використанні, простий у використанні та ефективний. Як і попереджав у своїй книзі Swaroop, «Це тепер моя улюблена мова програмування».

'A Byte of Python' був моїм посібником із вивчення Python. Він зрозуміло та ефективно, вводить вас у світ Python за найкоротший час. Він не надто довгий, але ефективно охоплює майже всі важливі теми в Python. Я думаю, що «A Byte of Python» слід настійно рекомендувати новачкам як перший підручник з Python. Я присвячую мій переклад потенційним мільйонам користувачів Python у Китаї.

3.35.5 Китайський традиційний

Fred Lin (gasolin@gmail.com) зголосився перекласти книгу китайською мовою.

Книга доступна на: http://code.google.com/p/zhpy/wiki/ByteOfZhpy.

Захоплююча особливість цього перекладу полягає в тому, що він також містить *вихідні тексти китайського Python* поруч із оригінальними текстами на Python.

Fred Lin - Я працюю інженером мережевого мікропрограмного забезпечення в Delta Network, а також роблю внесок у веб-платформу TurboGears.

Як євангеліст Python (:-p), мені потрібен матеріал для просування мови Python. Я виявив, що «A Byte of Python»... hhjmjkjhg найкраще підходить як для новачків, так і для досвідчених програмістів. «A Byte of Python» детально викладає основи Python в розумних обсягах.

Переклад спочатку базувався на спрощеній китайській версії, і незабаром було зроблено багато переписань, щоб відповідати поточній версії книги та якості читання.

Остання китайська традиційна версія також містить вихідні коди програм китайського Python, які досягнуті за допомогою мого нового проекту «zhpy» (китайською — python) (запуск 7 серпня).

zhpy(вимовляти як "Зед.Аш.Пі", або "Зіппі") є такою собі надбудовою над Python, що переводить Python на традиційну або спрощену китайську. Цей проект існує насамперед з освітньою метою.

3.35.6 Французька

Gregory (coulix@ozforces.com.au) зголосився перекласти книгу французькою мовою.

Gérard Labadie (gerard.labadie@gmail.com) завершив переклад книги французькою мовою.

Пізніше цей переклад було перенесено у формат markdown, оновлено відповідно до останньої версії книги та опубліковано на GitBook Romain Gilliotte (rgilliotte@gmail.com).

Його можна знайти за адресою https://rgilliotte.gitbook.io/byte-of-python/

3.35.7 Німецький

Lutz Horn (lutz.horn@gmx.de), Bernd Hengelein (bernd.hengelein@gmail.com) та Christoph Zwerschke (cito@online.de) зголосилися перекласти книгу німецькою мовою.

Переклад можна знайти за адресою: http://cito.github.io/byte_of_python/

Lutz Horn каже:

Мені 32 роки, я маю ступінь з математики в Гейдельберзькому університеті, Німеччина. Наразі я працюю інженером-програмістом в проекті, який фінансується державою, щоб створити веб-портал комп'ютерних наук у Німеччині. Основною мовою, яку я використовую як професіонал, є Java, але я намагаюся робити якомога більше з Python за кадром. Зокрема,

> з Python дуже легко аналізувати та конвертувати текст. Я не дуже знайомий із наборами інструментів графічного інтерфейсу користувача, оскільки більшість мого програмування стосується веб-додатків, де інтерфейс користувача формується такими Java інструментами як Struts. Зараз я намагаюся більше використовувати функції функціонального програмування Python і генераторів. Після короткого вивчення Ruby я був дуже вражений використанням блоків у цій мові. Загалом мені подобається динамічний характер таких мов, як Python і Ruby, оскільки це дозволяє мені робити те, що неможливо в більш статичних мовах, таких як Java. Я шукав якийсь вступ до програмування, придатний для навчання повного непрограміста. Я знайшов книжку 'How to Think Like a Computer Scientist: Learning with Python', та „Dive into Python. Перша книга добра для початківців, але її потрібно довго перекладати. Друга - не підходить новачкам. Я думаю, що «A Byte of Python» чудово вписується між ними, оскільки вона не надто довга, написана по суті, і водночас досить докладна для навчання новачка. Крім того, мені подобається проста структура DocBook, яка дозволяє перекладати текст, а також генерувати результуючий текст у різних форматах як за помахом чарівної палички.

Bernd Hengelein каже:

> Lutz і я збираємося зробити німецький переклад разом. Ми тільки почали зі вступу та передмови, але ми будемо інформувати вас про прогрес, якого ми досягаємо. Гаразд, тепер трохи особистого про мене. Мені 34 роки, і я граю з комп'ютерами з 1980-х років, коли «Commodore C64» панував у дитячих кімнатах. Після вивчення інформатики я почав працювати інженером-програмістом. Зараз я працюю у сфері медичної візуалізації у великій німецькій компанії. Хоча C++ є основною мовою, яку я (маю) використовувати для щоденної роботи, я постійно шукаю щось нове для вивчення. Минулого року я закохався у Python, яка є чудовою мовою як за своїми можливостями, так і за своєю красою. Я читав десь в мережі про хлопця, який сказав, що йому подобається Python, тому що код виглядає дуже красиво. Як на мене, він абсолютно правий. У той час, коли я вирішив вивчити python, я помітив, що є дуже мало хорошої документації німецькою мовою. Коли я натрапив на вашу книгу, мені спала на думку спонтанна ідея німецького перекладу. На щастя, у Lutz була та сама ідея, і тепер ми можемо розділити роботу. Я з нетерпінням чекаю на плідну співпрацю!

3.35.8 Грецька

Грецька спільнота Ubuntu [переклала книгу грецькою] (http://wiki.ubuntu-gr.org/byte-of-python-el), для використання в наших онлайн уроках Python, які проводяться на наших форумах. Щоб дізнатися більше, зв'яжіться з @savvasradevic.

3.35.9 Індонезійська

Daniel (daniel.mirror@gmail.com) перекладає книгу індонезійською мовою http://python.or.id/moin.cgi/ByteofPython.

Wisnu Priyambodo (cibermen@gmail.com) також зголосився перекласти книгу індонезійською мовою.

Також, Bagus Aji Santoso (baguzzzaji@gmail.com) зголосився.

3.35.10 Італійська (перша)

Enrico Morelli (mr.mlucci@gmail.com) і Massimo Lucci (morelli@cerm.unifi.it) зголосилися перекласти книгу італійською мовою.

Переклад італійською доступний на http://www.gentoo.it/Programmazione/byteofpython.

> *Massimo Lucci and Enrico Morelli* - ми працюємо в університеті Флоренції (Італія) - хімічний факультет. Я (Massimo) як сервісний інженер та системний адміністратор спектрометрів ядерного магнітного резонансу; Enrico як сервісний інженер і системний адміністратор

паралельних/кластерних систем. Ми програмуємо на python близько семи років, маємо досвід роботи з платформами Linux з десяти років. В Італії ми адмініструємо веб-сторінку www.gentoo.it для дистрибутива Gentoo/Linux та www.nmr.it (зараз у розробці) для додатків ядерного магнітного резонансу та організації конгресу та управління. Ось і все! Ми вражені розумною мовою, використаною у вашій книзі, і ми вважаємо, що це важливо для наближення Python до нових користувачів (ми маємо на увазі сотні студентів і дослідників, які працюють у наших лабораторіях)...

3.35.11 Італійська (друга)

Італійський переклад був створений Calvina Bice та колегами з http://besthcgdropswebsite.com/translate/a-byte-of-python/.

3.35.12 Японська

Shunro Dozono (dozono@gmail.com) перекладає книгу японською мовою.

3.35.13 Корейська

Epsimatt (2019)

Epsimatt розпочав новий корейський переклад:

- Читайте онлайн на https://epsimatt.gitbook.io/byte-of-python/
- Слідкуйте за прогресом на https://github.com/epsimatt/byte-of-python/issues/16

Старіша

Jeongbin Park (pjb7687@gmail.com) переклав книгу на корейську - https://github.com/pjb7687/byte_of_python

> Я Jeongbin Park, зараз працюю дослідником у галузі біофізики та біоінформатики в Кореї.
>
> Рік тому я шукав гарний підручник/посібник на Python, щоб представити його своїм колегам, тому що використання Python у галузях досліджень зростає через те, що кількість користувачів стає дедалі більшою.
>
> Але на той час було лише кілька книг про Python доступні корейською мовою, тому я вирішив перекласти вашу електронну книгу, оскільки вона виглядає як один із найкращих посібників, які я коли-небудь читав!
>
> Наразі книга майже повністю перекладена корейською мовою, за винятком частини тексту у вступному розділі та додатків.
>
> Ще раз дякую, що написали такий гарний посібник!

3.35.14 Монгольська

Ariunsanaa Tunjin (luftballons2010@gmail.com) зголосився перекласти книгу монгольською мовою.

Оновлення від 22 листопада 2009 : Ariunsanaa на порозі завершення перекладу.

3.35.15 Норвезька (bokmål)

Eirik Vågeskar студент Sandvika videregående skole у Норвегії, блогер і зараз перекладає книгу норвезькою (bokmål).

> *Eirik Vågeskar*: Я завжди хотів програмувати, але оскільки я розмовляю малопоширеною мовою, процес навчання був набагато важчим. Більшість підручників і книг написані дуже технічною англійською мовою, тому більшість випускників середньої школи навіть не матимуть словникового запасу, щоб зрозуміти, про що підручник. Коли я відкрив цю книгу, усі мої проблеми були вирішені. «A Byte of Python» використав просту нетехнічну мову, щоб пояснити таку ж просту мову програмування, і ці дві речі роблять вивчення Python цікавим. Прочитавши половину книги, я вирішил, що книга варта перекладу. Я сподіваюся, що переклад допоможе людям, які опинилися в такій самій ситуації, як я (особливо молоді), і, можливо, допоможе поширити інтерес до мови серед людей з меншими технічними знаннями.

3.35.16 Польська

Dominik Kozaczko (dominik@kozaczko.info) зголосився перекласти книгу польською мовою. Переклад триває, і його головна сторінка доступна тут: Ukąś Pythona.

Оновлення : переклад завершено та готовий станом на 2 жовтня 2009 р. Дякуємо Dominik та двом його студентам та їхньому другові за їх час і зусилля!

> *Dominik Kozaczko* - Я вчитель інформатики та інформаційних технологій.

3.35.17 Португальська

Artur Weber (arturweberguimaraes@gmail.com) завершив переклад цієї книги португальською (станом на 21 лютого 2018 р.) на https://www.homeyou.com/~edu/introducao.

> *Artur Weber*: Мої студенти навчаються на політехнічному факультеті Екологічного університету в місті Куритиба (Бразилія), і деякі з них цікавляться різними роботами.
>
> Оскільки вони пишуть курсові та академічні роботи, вони завжди шукають цікаві статті та сторінки. Також я намагаюся знайти цікаві матеріали, які можуть бути джерелами для їхніх університетських робіт.
>
> Я знайшов матеріали з вашого сайту корисними для деяких моїх студентів, які пишуть роботи на основі програмування на Python. Власне, тому я вирішив зробити португальський переклад, щоб мої студенти, які не знають англійської, могли читати захоплюючі статті рідною мовою (португальською).

3.35.18 Російська

Vladimir Smolyar (v_2e@ukr.net) завершив переклад російською на http://wombat.org.ua/AByteOfPython/.

3.35.19 Українська

Averkiev Andrey (averkiyev@ukr.net) зголосився перекласти книгу російською, і, можливо, українською (якщо дозволить час).

Daria Jens (jensdarya@gmail.com) у 2025 році завершила переклад українською мовою: https://spielend-programmieren.at/byte_of_python_ukraine/

3.35.20 Сербська

“BugSpice” (amortizerka@gmail.com) завершив сербський переклад:

> Це посилання для завантаження більше не доступне.

Докладніше на http://forum.ubuntu-rs.org/Thread-zagrljaj-pitona.

3.35.21 Словацька

Albertio Ward (albertioward@gmail.com) переклав книгу словацькою мовою на http://www.fatcow.com/edu/python-swaroopch-sl/ :

> Ми є некомерційною організацією «Переклад для освіти». Ми представляємо групу людей, переважно студентів і викладачів Слов'янського університету. Тут зібрані студенти різних факультетів: лінгвістики, хімії, біології та ін. Ми намагаємося знаходити в Інтернеті цікаві публікації, які можуть бути актуальними для нас та наших колег з університету. Іноді ми самі знаходимо статті; іноді наші викладачі допомагають нам підібрати матеріал для перекладу. Після отримання дозволу авторів ми перекладаємо статті та розміщуємо їх у своєму блозі, який доступний для наших колег і друзів. Ці перекладені публікації часто допомагають студентам у щоденній рутині навчання.

3.35.22 Іспанська

Alfonso de la Guarda Reyes (alfonsodg@ictechperu.net), Gustavo Echeverria (gustavo.echeverria@gmail.com), David Crespo Arroyo (davidcrespoarroyo@hotmail.com) і Cristian Bermudez Serna (crisbermud@hotmail.com) зголосилися перекладати книгу на іспанську.

Gustavo Echeverria каже:

> Я працюю інженером-програмістом в Аргентині. На роботі я використовую здебільшого технології C# і .Net, але в особистих проектах – виключно Python або Ruby. Я знав про Python багато років тому і відразу ж зупинився на Python. Незабаром після знайомства з Python я відкрив цю книгу, і вона допомогла мені вивчити мову. Тоді я зголосився перекласти книгу іспанською мовою. Тепер, отримавши кілька запитів, я почав перекладати «A Byte of Python» разом з Maximiliano Soler.

Cristian Bermudez Serna каже:

> Я студент телекомунікаційної інженерії в Університеті Антіокії (Колумбія). Кілька місяців тому я почав вивчати Python і знайшов цю чудову книгу, тому я зголосився приєднатися до перекладу на іспанську мову.

3.35.23 Шведська

Mikael Jacobsson (leochingkwake@gmail.com) зголосився перекласти книгу шведською мовою.

3.35.24 Турецька

Türker SEZER (tsezer@btturk.net) та Bugra Cakir (bugracakir@gmail.com) зголосилися перекласти книгу турецькою мовою. "Де турецька версія? Bitse de okusak."

3.36 Переклади

Завдяки багатьом невтомним волонтерам доступно багато перекладів книги різними мовами!

Якщо ви хочете допомогти з цими перекладами, будь ласка, перегляньте список волонтерів для відповідних мов нижче та вирішіть, чи хочете ви розпочати новий переклад чи допомогти в уже існуючих проектах перекладу.

Якщо ви плануєте почати новий переклад, будь ласка, прочитайте *Інструкцію з перекладу*.

3.36.1 Арабська

Нижче наведено посилання на арабську версію. Дякуємо Ashraf Ali Khalaf за переклад книги, ви можете прочитати всю книгу онлайн за адресою http://www.khaledhosny.org/byte-of-python/index.html або завантажити її з [sourceforge.net](http://downloads.sourceforge.net/omlx/byteofpython_arabic.pdf?use_mirror=osdn). Для отримання додаткової інформації див. http://itwadi.com/byteofpython_arabi.

3.36.2 Азербайджанська

Jahangir Shabiyev (c.shabiev@gmail.com) зголосився перекласти книгу азербайджанською мовою. Переклад виконується за адресою https://www.gitbook.com/book/jahangir-sh/piton-sancmasi

3.36.3 Бразильська Португальська

Є два переклади з різними рівнями завершеності та доступності. Старіший переклад зараз відсутній/втрачений, а новіший переклад неповний.

Samuel Dias Neto (samuel.arataca@gmail.com) зробив перший бразильський португальський переклад (старіший переклад) цієї книги, коли версія Python була 2.3.5. Переклад більше не є загальнодоступним.

Rodrigo Amaral (rodrigoamaral@gmail.com) зголосився перекласти книгу бразильською португальською мовою (новіший переклад), який ще не завершено.

Каталонська

Moises Gomez (moisesgomezgiron@gmail.com) зголосився перекласти книгу каталонською мовою. Переклад триває.

> Moisès Gómez - Я розробник, а також викладач програмування (як правило, для людей без попереднього досвіду).
>
> Деякий час тому мені потрібно було навчитися програмувати на Python, і робота Swaroop'a була справді корисною. Чітко, лаконічно та досить повно. Саме те, що мені було потрібно.
>
> Після цього досвіду я подумав, що деякі інші люди в моїй країні також можуть скористатися цим. Але англійська мова може стати перешкодою.
>
> Отже, чому б не спробувати це перекласти? І я зробив для попередньої версії книги.
>
> У моїй країні є дві державні мови. Я вибрав каталонську, тому що подумав, що на більш поширену іспанську її напевно переведе хтось інший.

3.36.4 Китайський

У 2017 році, тобто через 11 років, Mo Lun (i@molun.net) повторно переклав книгу з самого початку, на основі версії 4.0. А переклад зберігається на GitHub і Gitbook. Він стежить за цим перекладеним виданням і готовий виправити його, якщо в перекладеній версії книги є якась помилка.

Видання перекладу 2017 року доступне на https://bop.molun.net.

Mo Lun каже:

> Я звичайний студент журналістики Китайського молодіжного університета політичних наук (англ."China Youth University for Political Sciences (CYU)"), Пекін. І фактично, коли я починав перекладати цю книгу, я був абсолютний новачок у програмуванні на Python. Спочатку це була просто примха, але коли я закінчив переклад, я зрозумів, що рішення, викликане інтересом, змусило мене зайти так далеко.

За допомогою перекладів моїх попередників,великої кількості інформації в Інтернет, і за допомогою моїх друзів я зміг представити це видання. Я просто сподіваюся, що моя перекладацька робота допоможе іншим новачкам у вивченні Python.

Водночас я завжди чекаю зауважень і пропозицій до мого перекладу і готовий змінити або вдосконалити цю поверхневу роботу.

Попередній китайський переклад

В 2005, Shen Jieyuan переклав цю книгу з версією 1.20 китайською мовою та опублікував її в Інтернеті. Це перше китайське видання. На офіційному сайті книги його звали Juan Shen,за адресою електронної пошти orion_val@163.comЦе видання було широко розповсюджене в мережі, а посилання, надані на офіційному сайті книги, більше не доступні, тому його оригінальне джерело неможливо знайти. Тому тут не можна вказати певну адресу. Але ви можете спробувати знайти копію за ключовими словами, наприклад...

Juan Shen каже:

Я аспірант відділу бездротових телекомунікацій у Пекінському технологічному університеті, Китай. Наразі мої дослідницькі інтереси стосуються синхронізації, регулюванням каналу передачі даних і багатокористувацьким визначенням системи з багатьма несучими частотами CDMA. Python — моя основна мова програмування для щоденного моделювання та дослідницької роботи. Здебільшого з використанням Python Numeric. Я познайомився з Python лише півроку тому, але, як ви бачите, він дійсно простий у використанні, простий у використанні та ефективний. Як і попереджав у своїй книзі Swaroop, «Це тепер моя улюблена мова програмування».

'A Byte of Python' був моїм посібником із вивчення Python. Він зрозуміло та ефективно, вводить вас у світ Python за найкоротший час. Він не надто довгий, але ефективно охоплює майже всі важливі теми в Python. Я думаю, що «A Byte of Python» слід настійно рекомендувати новачкам як перший підручник з Python. Я присвячую мій переклад потенційним мільйонам користувачів Python у Китаї.

3.36.5 Китайський традиційний

Fred Lin (gasolin@gmail.com) зголосився перекласти книгу китайською мовою.

Книга доступна на: http://code.google.com/p/zhpy/wiki/ByteOfZhpy.

Захоплююча особливість цього перекладу полягає в тому, що він також містить *вихідні тексти китайського Python* поруч із оригінальними текстами на Python.

Fred Lin - Я працюю інженером мережевого мікропрограмного забезпечення в Delta Network, а також роблю внесок у веб-платформу TurboGears.

Як євангеліст Python (:-p), мені потрібен матеріал для просування мови Python. Я виявив, що «A Byte of Python»... hhjmjkjhg найкраще підходить як для новачків, так і для досвідчених програмістів. «A Byte of Python» детально викладає основи Python в розумних обсягах.

Переклад спочатку базувався на спрощеній китайській версії, і незабаром було зроблено багато переписань, щоб відповідати поточній версії книги та якості читання.

Остання китайська традиційна версія також містить вихідні коди програм китайського Python, які досягнуті за допомогою мого нового проекту «zhpy» (китайською — python) (запуск 7 серпня).

zhpy(вимовляти як "Зед.Аш.Пі", або "Зіппі") є такою собі надбудовою над Python, що переводить Python на традиційну або спрощену китайську. Цей проект існує насамперед з освітньою метою.

3.36.6 Французька

Gregory (coulix@ozforces.com.au) зголосився перекласти книгу французькою мовою.

Gérard Labadie (gerard.labadie@gmail.com) завершив переклад книги французькою мовою.

Пізніше цей переклад було перенесено у формат markdown, оновлено відповідно до останньої версії книги та опубліковано на GitBook Romain Gilliotte (rgilliotte@gmail.com).

Його можна знайти за адресою https://rgilliotte.gitbook.io/byte-of-python/

3.36.7 Німецький

Lutz Horn (lutz.horn@gmx.de), Bernd Hengelein (bernd.hengelein@gmail.com) та Christoph Zwerschke (cito@online.de) зголосилися перекласти книгу німецькою мовою.

Переклад можна знайти за адресою: http://cito.github.io/byte_of_python/

Lutz Horn каже:

> Мені 32 роки, я маю ступінь з математики в Гейдельберзькому університеті, Німеччина. Наразі я працюю інженером-програмістом в проекті, який фінансується державою, щоб створити веб-портал комп'ютерних наук у Німеччині. Основною мовою, яку я використовую як професіонал, є Java, але я намагаюся робити якомога більше з Python за кадром. Зокрема, з Python дуже легко аналізувати та конвертувати текст. Я не дуже знайомий із наборами інструментів графічного інтерфейсу користувача, оскільки більшість мого програмування стосується веб-додатків, де інтерфейс користувача формується такими Java інструментами як Struts. Зараз я намагаюся більше використовувати функції функціонального програмування Python і генераторів. Після короткого вивчення Ruby я був дуже вражений використанням блоків у цій мові. Загалом мені подобається динамічний характер таких мов, як Python і Ruby, оскільки це дозволяє мені робити те, що неможливо в більш статичних мовах, таких як Java. Я шукав якийсь вступ до програмування, придатний для навчання повного непрограміста. Я знайшов книжку 'How to Think Like a Computer Scientist: Learning with Python', та „Dive into Python. Перша книга добра для початківців, але її потрібно довго перекладати. Друга - не підходить новачкам. Я думаю, що «A Byte of Python» чудово вписується між ними, оскільки вона не надто довга, написана по суті, і водночас досить докладна для навчання новачка. Крім того, мені подобається проста структура DocBook, яка дозволяє перекладати текст, а також генерувати результуючий текст у різних форматах як за помахом чарівної палички.

Bernd Hengelein каже:

> Lutz і я збираємося зробити німецький переклад разом. Ми тільки почали зі вступу та передмови, але ми будемо інформувати вас про прогрес, якого ми досягаємо. Гаразд, тепер трохи особистого про мене. Мені 34 роки, і я граю з комп'ютерами з 1980-х років, коли «Commodore C64» панував у дитячих кімнатах. Після вивчення інформатики я почав працювати інженером-програмістом. Зараз я працюю у сфері медичної візуалізації у великій німецькій компанії. Хоча C++ є основною мовою, яку я (маю) використовувати для щоденної роботи, я постійно шукаю щось нове для вивчення. Минулого року я закохався у Python, яка є чудовою мовою як за своїми можливостями, так і за своєю красою. Я читав десь в мережі про хлопця, який сказав, що йому подобається Python, тому що код виглядає дуже красиво. Як на мене, він абсолютно правий. У той час, коли я вирішив вивчити python, я помітив, що є дуже мало хорошої документації німецькою мовою. Коли я натрапив на вашу книгу, мені спала на думку спонтанна ідея німецького перекладу. На щастя, у Lutz була та сама ідея, і тепер ми можемо розділити роботу. Я з нетерпінням чекаю на плідну співпрацю!

3.36.8 Грецька

Грецька спільнота Ubuntu [переклала книгу грецькою] (http://wiki.ubuntu-gr.org/byte-of-python-el), для використання в наших онлайн уроках Python, які проводяться на наших форумах. Щоб дізнатися більше, зв'яжіться з @savvasradevic.

3.36.9 Індонезійська

Daniel (daniel.mirror@gmail.com) перекладає книгу індонезійською мовою http://python.or.id/moin.cgi/ByteofPython.

Wisnu Priyambodo (cibermen@gmail.com) також зголосився перекласти книгу індонезійською мовою.

Також, Bagus Aji Santoso (baguzzzaji@gmail.com) зголосився.

3.36.10 Італійська (перша)

Enrico Morelli (mr.mlucci@gmail.com) і Massimo Lucci (morelli@cerm.unifi.it) зголосилися перекласти книгу італійською мовою.

Переклад італійською доступний на http://www.gentoo.it/Programmazione/byteofpython.

> *Massimo Lucci and Enrico Morelli* - ми працюємо в університеті Флоренції (Італія) - хімічний факультет. Я (Massimo) як сервісний інженер та системний адміністратор спектрометрів ядерного магнітного резонансу; Enrico як сервісний інженер і системний адміністратор паралельних/кластерних систем. Ми програмуємо на python близько семи років, маємо досвід роботи з платформами Linux з десяти років. В Італії ми адмініструємо веб-сторінку www.gentoo.it для дистрибутива Gentoo/Linux та www.nmr.it (зараз у розробці) для додатків ядерного магнітного резонансу та організації конгресу та управління. Ось і все! Ми вражені розумною мовою, використаною у вашій книзі, і ми вважаємо, що це важливо для наближення Python до нових користувачів (ми маємо на увазі сотні студентів і дослідників, які працюють у наших лабораторіях)...

3.36.11 Італійська (друга)

Італійський переклад був створений Calvina Bice та колегами з http://besthcgdropswebsite.com/translate/a-byte-of-python/.

3.36.12 Японська

Shunro Dozono (dozono@gmail.com) перекладає книгу японською мовою.

3.36.13 Корейська

Epsimatt (2019)

Epsimatt розпочав новий корейський переклад:

- Читайте онлайн на https://epsimatt.gitbook.io/byte-of-python/
- Слідкуйте за прогресом на https://github.com/epsimatt/byte-of-python/issues/16

Старіша

Jeongbin Park (pjb7687@gmail.com) переклав книгу на корейську - https://github.com/pjb7687/byte_of_python

Я Jeongbin Park, зараз працюю дослідником у галузі біофізики та біоінформатики в Кореї.

Рік тому я шукав гарний підручник/посібник на Python, щоб представити його своїм колегам, тому що використання Python у галузях досліджень зростає через те, що кількість користувачів стає дедалі більшою.

Але на той час було лише кілька книг про Python доступні корейською мовою, тому я вирішив перекласти вашу електронну книгу, оскільки вона виглядає як один із найкращих посібників, які я коли-небудь читав!

Наразі книга майже повністю перекладена корейською мовою, за винятком частини тексту у вступному розділі та додатків.

Ще раз дякую, що написали такий гарний посібник!

3.36.14 Монгольська

Ariunsanaa Tunjin (luftballons2010@gmail.com) зголосився перекласти книгу монгольською мовою.

Оновлення від 22 листопада 2009 : Ariunsanaa на порозі завершення перекладу.

3.36.15 Норвезька (bokmål)

Eirik Vågeskar студент Sandvika videregående skole у Норвегії, блогер і зараз перекладає книгу норвезькою (bokmål).

Eirik Vågeskar: Я завжди хотів програмувати, але оскільки я розмовляю малопоширеною мовою, процес навчання був набагато важчим. Більшість підручників і книг написані дуже технічною англійською мовою, тому більшість випускників середньої школи навіть не матимуть словникового запасу, щоб зрозуміти, про що підручник. Коли я відкрив цю книгу, усі мої проблеми були вирішені. «A Byte of Python» використав просту нетехнічну мову, щоб пояснити таку ж просту мову програмування, і ці дві речі роблять вивчення Python цікавим. Прочитавши половину книги, я вирішил, що книга варта перекладу. Я сподіваюся, що переклад допоможе людям, які опинилися в такій самій ситуації, як я (особливо молоді), і, можливо, допоможе поширити інтерес до мови серед людей з меншими технічними знаннями.

3.36.16 Польська

Dominik Kozaczko (dominik@kozaczko.info) зголосився перекласти книгу польською мовою. Переклад триває, і його головна сторінка доступна тут: Ukąś Pythona.

Оновлення : переклад завершено та готовий станом на 2 жовтня 2009 р. Дякуємо Dominik та двом його студентам та їхньому другові за їх час і зусилля!

Dominik Kozaczko - Я вчитель інформатики та інформаційних технологій.

3.36.17 Португальська

Artur Weber (arturweberguimaraes@gmail.com) завершив переклад цієї книги португальською (станом на 21 лютого 2018 р.) на https://www.homeyou.com/~edu/introducao.

Artur Weber: Мої студенти навчаються на політехнічному факультеті Екологічного університету в місті Куритиба (Бразилія), і деякі з них цікавляться різними роботами.

Оскільки вони пишуть курсові та академічні роботи, вони завжди шукають цікаві статті та сторінки. Також я намагаюся знайти цікаві матеріали, які можуть бути джерелами для їхніх університетських робіт.

> Я знайшов матеріали з вашого сайту корисними для деяких моїх студентів, які пишуть роботи на основі програмування на Python. Власне, тому я вирішив зробити португальський переклад, щоб мої студенти, які не знають англійської, могли читати захоплюючі статті рідною мовою (португальською).

3.36.18 Російська

Vladimir Smolyar (v_2e@ukr.net) завершив переклад російською на http://wombat.org.ua/AByteOfPython/.

3.36.19 Українська

Averkiev Andrey (averkiyev@ukr.net) зголосився перекласти книгу російською, і, можливо, українською (якщо дозволить час).

Daria Jens (jensdarya@gmail.com) 2024 році завершила переклад українською мовою.

3.36.20 Сербська

“BugSpice” (amortizerka@gmail.com) завершив сербський переклад:

> Це посилання для завантаження більше не доступне.

Докладніше на http://forum.ubuntu-rs.org/Thread-zagrljaj-pitona.

3.36.21 Словацька

Albertio Ward (albertioward@gmail.com) переклав книгу словацькою мовою на http://www.fatcow.com/edu/python-swaroopch-sl/ :

> Ми є некомерційною організацією «Переклад для освіти». Ми представляємо групу людей, переважно студентів і викладачів Слов'янського університету. Тут зібрані студенти різних факультетів: лінгвістики, хімії, біології та ін. Ми намагаємося знаходити в Інтернеті цікаві публікації, які можуть бути актуальними для нас та наших колег з університету. Іноді ми самі знаходимо статті; іноді наші викладачі допомагають нам підібрати матеріал для перекладу. Після отримання дозволу авторів ми перекладаємо статті та розміщуємо їх у своєму блозі, який доступний для наших колег і друзів. Ці перекладені публікації часто допомагають студентам у щоденній рутині навчання.

3.36.22 Іспанська

Alfonso de la Guarda Reyes (alfonsodg@ictechperu.net), Gustavo Echeverria (gustavo.echeverria@gmail.com), David Crespo Arroyo (davidcrespoarroyo@hotmail.com) і Cristian Bermudez Serna (crisbermud@hotmail.com) зголосилися перекладати книгу на іспанську.

Gustavo Echeverria каже:

> Я працюю інженером-програмістом в Аргентині. На роботі я використовую здебільшого технології C# і .Net, але в особистих проектах – виключно Python або Ruby. Я знав про Python багато років тому і відразу ж зупинився на Python. Незабаром після знайомства з Python я відкрив цю книгу, і вона допомогла мені вивчити мову. Тоді я зголосився перекласти книгу іспанською мовою. Тепер, отримавши кілька запитів, я почав перекладати «A Byte of Python» разом з Maximiliano Soler.

Cristian Bermudez Serna каже:

> Я студент телекомунікаційної інженерії в Університеті Антіокії (Колумбія). Кілька місяців тому я почав вивчати Python і знайшов цю чудову книгу, тому я зголосився приєднатися до перекладу на іспанську мову.

3.36.23 Шведська

Mikael Jacobsson (leochingkwake@gmail.com) зголосився перекласти книгу шведською мовою.

3.36.24 Турецька

Türker SEZER (tsezer@btturk.net) та Bugra Cakir (bugracakir@gmail.com) зголосилися перекласти книгу турецькою мовою. “Де турецька версія? Bitse de okusak.”

3.37 Додаток: Інструкція з перекладу

1. Повний вихідний текст книги доступний за посиланням .
2. Будь ласка створіть відгалуження репозиторію.
3. Потім завантажте репозиторій на свій комп’ютер. Для цього вам потрібно знати, як використовувати Git.
4. Прочитайте Honkit documentation, особливо Markdown section.
5. Почніть редагувати файли `.md` на своїй рідній мові.
6. Перегляньте INSTALL.md про те, як створити веб-сайт, PDF, EPUB.

3.38 Зворотній зв’язок

Книзі потрібна допомога таких читачів, як ви, щоб вказати на будь-які частини книги, які є поганими, незрозумілими або просто неправильними. Будь ласка, напишіть основному автору або *відповідним перекладачам* свої коментарі та пропозиції.

3.39 Вікторіна

3.39.1 Завдання до розділу Основи:

 англійська: *Basic*

Завдання 1

- Створіть змінну з назвою `речення` і присвойте їй значення — рядок, який містить одинадцять слів (будь-яких).
- Виведіть на екран значення змінної `речення`.
- Додайте цей рядок коду в кінці своєї програми:

```
print(len(sentence.split()))
```

Підказка: Функція split() у Python рахує кількість слів у рядку, підраховуючи кількість пробілів між словами. Тож завдання звучить так: «напиши слова, між якими є 10 пробілів».

Можливе розв’язання

```
# Ось приклад розв'язання для завдання 1 із розділу Основи ("")
речення = "Мова Python дуже цікава, проста і легка для вивчення кожному щодня"
print(речення)
print(len(речення.split()))
```

Завдання 2

- Створіть змінну з назвою `вірш` і присвойте їй рядок із кількома словами.
- Рядок повинен складатися з трьох рядків тексту.
- Виведіть на екран значення змінної `вірш`.
- Додайте цей рядок коду в кінці своєї програми:

```
print(len(poem.splitlines()))
```

Чи можеш розв'язати це завдання (щоб рядок охоплював кілька рядків тексту) різними способами?

Можливі розв'язання:

```
# Розв'язання для розділу "Основи" Basic, завдання 2, варіант A.
вірш="""Код,
рядок за рядком я пишу,
світ новий у полі творю."""
print(вірш)
print(len(вірш.splitlines()))
```

```
# Розв'язання для розділу "Основи" Basic, завдання 2, варіант B
вірш='Код,\nрядок за рядком я пишу,\nсвіт новий у полі творю.'
print(вірш)
print(len(вірш.splitlines()))
```

```
# Розв'язання для розділу "Основи" Basic, завдання 2, варіант C
вірш="\n".join(["Код,","рядок за рядком я пишу,","світ новий у полі творю."])
print(вірш)
print(len(вірш.splitlines()))
```

Завдання 3

- Створіть змінну з назвою `заробітна плата` і присвойте їй значення 4000.
- Виведіть на екран значення змінної `заробітна плата`

Можливі розв'язання:

```
# Розв'язання для розділу "Основи" Basic, завдання 3, варіант A
# найпоширеніший варіант
заробітна_плата = 4000
print(заробітна_плата)
```

```
# Розв'язання для розділу "Основи" Basic, завдання 3, варіант B
# використовуйте підкреслення (underscores) для покращення читабельності
заробітна_плата = 4_000
print(заробітна_плата)
```

Завдання 4

- Створіть змінну з назвою `Дохід` і надайте їй значення 4000.
- Створіть змінну з назвою `Висновок`. Значення змінної `Висновок` має бути наступним рядком: "Мій дохід становить X євро на місяць".

- Змініть код так, щоб Python підставляв замість `X` значення змінної `Дохід`.
- Надрукуюте значення змінної `Висновок`

Чи можеш ти розв'язати це завдання кількома способами?

Можливі розв'язання:

```
# Розв'язання для розділу «Основи» Basic, завдання 4, варіант A.
# використовувайте .format()
Дохід = 4000
Висновок = 'Мій дохід становить {} євро на місяць'.format(дохід)
print(висновок)
```

```
# Розв'язання для розділу «Основи» Basic, завдання 4, варіант B
# використовувайте f-strings
Дохід = 4000
Висновок = f'Мій дохід становить {дохід} євро на місяць'
print(висновок)
```

```
# Розв'язання для розділу «Основи» Basic, завдання 4, варіант C
# використовувайте str()
Дохід = 4000
Висновок = "Мій дохід становить " + str(дохід)  + " євро на місяць"
print(висновок)
```

```
# Розв'язання для розділу «Основи» Basic, завдання 4, варіант D
# використовувайте %
Дохід = 4000
Висновок = 'Мій дохід становить %i євро на місяць' % дохід
print(висновок)
```

Завдання 5

Ідентифікатори для змінних

Які з наведених назв є допустимими *ідентифікаторами* для *змінних* у Python?

1) `a`
2) `A`
3) `aaaa`
4) `a123`
5) `123a`
6) `_123a`
7) `_a123`
8) `a_123`
9) `1_a23`
10) `!abc`
11) `a-b`
12) `a_minus_b`

Правильні відповіді:

1,2,3,4,6,7,8,12

3.39.2 Завдання для розділу *Оператори та вирази* :

 англійська: *Operators and Expressions*

Завдання 1

Які з наведених виразів у Python набувають значення `True`?

1) `True == False`
2) `True == True`
3) `False == False`
4) `5 > 2`
5) `len("Michael") > len("Mike")`
6) `5 != 7`
7) `6 >= 6`
8) `"abc" * 3 == "abcabcabc"`
9) `5**2 == 25`
10) `0 == False`
11) `1 == True`
12) `2 == True`
13) `2 == False`
14) `True == 1`
15) `None == None`
16) `None != 0`
17) `None == ""`

Правильні відповіді:

2,3,4,5,6,7,8,9,10,11,14,15,16

Завдання 2

Які з наведених *виразів* у Python набувають значення `False` ?

1) `10 / 5 == 2.0`
2) `10 // 5 == 2`
3) `10 % 3 == 1`
4) `( 5 > 1) and ( 5 > 7)`
5) `( 5 > 1) or (5 > 7)`
6) `not (5 > 7)`

Правильна відповідь:

4

Завдання 3

- Яким буде вивід (значення змінної `x`) у цій програмі Python?

```
# розділ _Оператори та вирази_ Завдання 3
x = 5
x = 5+1
x += 1
x = x * 2
x /= 2
print(x)
```

Правильна відповідь:

```
7.0
```

3.39.3 Завдання для розділу: "Потік керування":

 англійська: *Control Flow*

Завдання 1

У завданнях цього розділу використовуйте (деякі з) *операторів* Python, описаних у розділі "Керування потоком виконання" (таких як `if`, `elif`, `else`, `for`, `while`, `continue`, `break`)

- Напишіть програму, яка дозволяє користувачу ввести пароль і надає відповідь залежно від введеного значення:
  - Якщо пароль — `SeCrEt`, програма має вивести `Правильно` і завершити роботу.
  - Якщо користувач вводить неправильний пароль, програма повинна вивести `Неправильно` і знову запитати пароль.
- Після трьох невдалих спроб програма має вивести `Ви зробили 3 невдалі спроби` і завершити роботу.
- Перед завершенням роботи програми вона має вивести `Бувай!`

Приклад виводу::

```
Будь ласка, введіть пароль: >>>secret
Неправильно
Будь ласка, введіть пароль: >>>Secret
Неправильно
Будь ласка, введіть пароль: >>>SeCrEt
Правильно
Бувай!
```

Можливі розв'язання:

```
# solution for chapter control flow,  task 1, variant A
# Розв'язання для розділу «Потік керування», завдання 1, варіант А.
for a in range(3):
    текст = input("Будь ласка, введіть пароль: >>>")
```

(continues on next page)

(continued from previous page)

```
    if текст == "SeCrEt":
        print("Правильно")
        break
    else:
        print("Неправильно")
else:
    print("Ви зробили 3 невдалі спроби")
print("Бувай!")
```

```
# solution for chapter control flow,  task 1, variant B
# Розв'язання для розділу «Потік керування», завдання 1, варіант B
for _ in range(3):
    if input("Будь ласка, введіть пароль: >>>") == "SeCrEt":
        print("Правильно")
        break
    print("Неправильно")
else:
    print("Ви зробили 3 невдалі спроби")
print("Бувай!")
```

```
# solution for chapter control flow,  task 1, variant C
# Розв'язання для розділу «Потік керування», завдання 1, варіант C

спроба = 1
max_спроб = 3
дійсний_пароль = "SeCrEt"
while True:
    print(f"Це спроба {спроба} із {max_спроб}")
    текст = input("Будь ласка, введіть пароль: >>>")
    if текст == дійсний_пароль:
        print("Правильно")
        break
    else:
        print("Неправильно")
    спроба += 1
    if спроба > 3:
        print("Ви зробили 3 невдалі спроби")
        break
print("Бувай!")
```

```
# solution for chapter control flow,  task 1, variant D
# Розв'язання для розділу «Потік керування», завдання 1, варіант  D
спроба = 0
while спроба < 3:
    спроба += 1
    if input("Будь ласка, введіть пароль: >>>") == "SeCrEt":
        print("Правильно")
        break
    print("Неправильно")
else:
    print("Ви зробили 3 невдалі спроби")
```

(continues on next page)

(continued from previous page)

```
print("Бувай!")
```

```
# solution for chapter control flow, task 1, variant E
# Розв'язання для розділу «Потік керування», завдання 1, варіант E
спроба = 1
while input("Будь ласка, введіть пароль: >>>") != "SeCrEt":
    print("Неправильно")
    спроба += 1
    if спроба > 3:
        print("Ви зробили 3 невдалі спроби")
        break
else:
    print("Правильно")
print("Бувай!")
```

Завдання 2

Наведена нижче програма не працює так, як задумано. Що *має* робити програма:

- Програма має дозволити користувачу ввести пароль.
- Якщо користувач вводить правильний пароль(`secret`) програма має вивести `Правильно` і завершитися.
- Якщо користувач вводить неправильний пароль, програма має запитати знову.
- Якщо користувач 3 рази введе неправильний пароль, програма має вивести `Ви зробили 3 невдалі спроби` і завершитися.»

Твої завдання після аналізу програми:

- З'ясуй, чому ця програма не працює так, як задумано.
- Запропонуй, як змінити програму, щоб вона працювала відповідно до задуму.

```
# problem  control flow,  task 2,
# Задача з розділу "Потік керування", завдання 2
пароль = "secret"
max_спроб = 3
спроба = 1
while спроба < max_спроб:
    print(f"Спроба {спроба} of {max_спроб}")
    здогадка = input("введіть пароль: >>>")
    if здогадка == пароль:
        print("Правильно")
        break
    спроба += 1
else:
    print("Ви зробили 3 невдалі спроби")
print("Бувай!")
```

Розв'язання коду:

- Проблема у наступносу рядку: "while спроба < max_спроб:"
- Виправлення: `while спроба <= max_спроб:`

Завдання 3

Будь ласка, виправте наведену нижче програму, щоб вона виводила тільки одну відповідь.

```
# problem control_flow task 3
# Задача з розділу "Потік керування", завдання 3
дохід = input("Будь ласка, введіть свій щомісячний (чистий) дохід в євро")
дохід = int(дохід)
if дохід < 1000:
    print("Ти небагато заробляєш...")
if дохід < 2000:
    print("Могло б бути краще...")
elif дохід < 3000:
    print("Добре")
if дохід < 4000:
    print("Дуже добре")
if дохід < 5000:
    print("Чудово")
else:
    print("Насправді??")
```

Розв'язання коду:

```
дохід = input("Будь ласка, введіть свій щомісячний (чистий) дохід в євро")
дохід = int(дохід)
if дохід < 1000:
    print("Ти небагато заробляєш...")
elif дохід < 2000:
    print("Могло б бути краще...")
elif дохід < 3000:
    print("Добре")
elif дохід < 4000:
    print("Дуже добре")
elif дохід < 5000:
    print("Чудово")
else:
    print("Насправді??")
```

Завдання 4

Вправи для циклу "for loop" та функції "range"

- Ознайомтеся з документацією Python щодо функції `range`: https://docs.python.org/3/library/stdtypes.html#range Для розуміння параметрів `start`, `stop` and `step`.

Спробуйте розв'язати це завдання без використання комп'ютера: який буде результат виконання наступної послідовності чисел:

```
print(list(range(5)))
```

Розв'язання:

```
[0,1,2,3,4]
```

Завдання 5

Напишіть рядок коду на Python (використовуючи `range`) щоб отримати наступний результат: `[1,2,3,4,5]`

Розв'язання:

```
# solution control flow, task 5
# Розв'язання задачі з розділу "Потік керування", завдання 5
print(list(range(1,6)))
```

Завдання 6

Напишіть рядок коду на Python (використовуючи `range`) щоб отримати наступний результат: `[10,20,30,40,50]`

Розв'язання:

```
# solution control flow task 6
# Розв'язання задачі з розділу "Потік керування", завдання 6
print(list(range(10,51,10)))

де число 10-це крок
```

Завдання 7

Напишіть рядок коду на Python (використовуючи `range`) щоб отримати наступний результат: `[50,40,30,20,10,0]`

Розв'язання:

```
# Розв'язання задачі з розділу "Потік керування", завдання 7
print(list(range(50,-1,-10)))
```

Завдання 8

Напишіть рядки коду на Python (використовуючи `range`) який виводить усі числа від 1 до 10. Кожне число має бути надруковане в окремому рядку.

Розв'язання:

```
# Розв'язання задачі з розділу "Потік керування", завдання 8
for x in range(1,11):
    print(x)
```

Завдання 9

Напишіть рядок коду на Python (використовуючи `range`), який виводить усі числа від 1 до 10 в одному рядку. Числа мають бути розділені комами (після останнього числа може стояти кома).

Розв'язання:

```
# Розв'язання задачі з розділу "Потік керування", завдання 9
for x in range(1,11):
    print(x, end=",")
```

Завдання 10

Напишіть програму на Python (використовуючи вбудовану функцію `range`), яка перемножує кожне число від 2 до 5 з кожним іншим числом у цьому діапазоні. Програма повинна виводити окремий рядок для кожного обчислення, як у цьому (скороченому) прикладі: :

```
2 x 2 =  4
2 x 3 =  6
2 x 4 =  8
2 x 5 = 10
3 x 2 =  6
3 x 3 =  9
...
```

Розв'язання

```
# Розв'язання задачі з розділу "Потік керування", завдання 10
for a in range(1,6):
    for b in range(1,6):
        print(f"{a} x {b} = {a*b:>2}")
```

Завдання 11

6.8. Скілька рядків буде виведено цією Python програмою?

```
# problem control flow task 11
# Розв'язання задачі з розділу "Потік керування", завдання 11
for x in "abcde":
    for y in "wxyz":
        print(x,y)
```

Розв'язання:

20 рядків

Завдання 12

Скільки рядків виведе ця програма на Python?

```
# problem control flow task 12
# Розв'язання задачі з розділу "Потік керування", завдання 12
for a in ("abc","def","ghi"):
    for b in (100,200,300,400):
        for c in "yz":
            print(a,b,c)
```

Розв'язання:

24 рядка

Завдання 13

Чому ця програма не буде працювати?

```
# problem control flow task 13
# Проблема з розділу "Потік керування", завдання 13
```

(continues on next page)

(continued from previous page)

```
for a in range(-10,11):
    for b in range(-10,11):
        print(f"{a} + {b} = {a+b}")
        print(f"{a} - {b} = {a-b}")
        print(f"{a} x {b} = {a*b}")
        print(f"{a} / {b} = {a/b}")
```

Розв'язання:

Внаслідок помилки ділення на нуль у шостому рядку коду. Рядок 6 : `print(f"{a} / {b} = {a/b}")` намагається поділити a на b. Але змінна b береться з `range(-10, 11)`,який містить 0.

Завдання 14

Будь ласка, прочитайте в книзі "Byte of Python", розділ "Потік керування" про оператори `break` і `continue`(обидва оператори можуть використовуватися всередині блоку `while` або `for`). Також перевірте, чи ви розумієте офіційну документацію Python щодо цих команд: https://docs.python.org/3/tutorial/controlflow.html#break-and-continue-statements

Якщо ви вважаєте, що зрозуміли використання `breakicontinue`, проаналізуйте цю програму (програма працює правильно):

```
# problem control flow task 14 A
# Задача з розділу "Потік керування", завдання 14 а
while True:
    print("Яка відповідь на це ПИТАННЯ?")
    print("Введіть 'допомога' щоб переглянути довідковий текст.")
    print("Введіть 'вийти' , щоб вийти з цієї гри.")
    команда = input(">>>")
    if команда == "допомога":
        print("Дивіться 'Путівник Галактикою для автостопників'")
        print("автор - Дуглас Адамс")
    elif команда == "вийти":
        break
    elif команда == "42":
        print("Вітаю, ти знаєш ТУ САМУ ВІДПОВІДЬ.")
        print("Але яке ж було запитання насправді.... ? ")
        break

print("Бувай-бувай")
```

- Спробуйте зрозуміти, що робить програма (запусти її та перевір).
- Спробуйте зрозуміти цю програму, розглядаючи не лише код, а й блок-схему (flowchart) цієї програми:

Дізнайтеся більше про блок-схеми (англ."flowcharts") у Вікіпедії: https://en.wikipedia.org/wiki/Flowchart

Ви можете створювати блок-схеми за допомогою ручки й паперу або за допомогою комп'ютерних програм (наприклад, MS Word чи LibreOffice Draw). Блок-схему, наведену вище, створено за допомогою інструмента для побудови діаграм "Mermaid". Див. https://mermaid.js.org для ознайомлення з документацією та використання онлайн-редактора.

Задача з розділу "Потік керування", завдання 14 B

- Проаналізуйте наведену нижче програму на Python.
- Створіть для неї блок-схему (за допомогою будь-якого інструмента на ваш вибір).

```
# problem control flow task 14 B # Задача з розділу "Потік керування", завдання 14 B
# password creation/створення пароля

while True:
    print("Введіть 'допомога', щоб відобразити довідковий текст.")
    команда = input("Будь ласка, введіть новий пароль >>>")
    if команда == "допомога":
        print("Пароль повинен мати: ")
        print("Щонайменше одну цифру (0-9) ")
        print("Щонайменше одну літеру в нижньому регістрі (а-я)")
        continue
    #перевірити пароль

    for char in "абвггдеєжзиіїйклмнопрстуфхцчшщьюя":
        if char in команда:
            break
    else:
        print("Не знайдено жодної літери в нижньому регістрі (а-я). Будь ласка,␣
↪спробуйте ще раз.")
        continue

    for цифра in "0123456789":
        if цифра in команда:
            break
    else:
        print("Не знайдено жодної цифри (0-9). Будь ласка, спробуйте ще раз.")
        continue
    print("Вітаємо! Ваш пароль успішно прийнято.")
    break
print("Бувай!")
```

можливе розв'язання

3.39.4 Завдання до розділу *Функція* (англ.*Function*):

 англійська: *Function*

Завдання 1

Напишіть *функцію* на Python з назвою `привітання`. Функція має імітувати працівника готелю. Отримавши *параметр* `година_доби`(0 - 24), функція повинна *повертати* привітання:

година з	година до	привітання
0	6	Добраніч
6	11	Доброго ранку
11	14	Доброго дня
14	18	Доброго пообіду
18	22	Доброго вечора
22	24	Добраніч

Додайте код, щоб перевірити роботу функції **привітання**: Виведіть на екран кожну годину від 1 до 24 та відповідне привітання для цієї години (по одному рядку на кожну годину).

можливі розв'язання

```
# solution chapter function task 1, variant A
# Розв'язання з розділу "Функція", завдання 1, варіант А
def привітання (година_доби):
    if 6 <= година_доби <11:
        return "Доброго ранку"
    elif 11 <= година_доби < 14:
        return "Доброго дня"
    elif 14 <= година_доби < 18:
        return "Доброго пообіду"
    elif 18 <= година_доби < 22:
        return "Доброго вечора"
    elif (22 <= година_доби) or (година_доби <6):
        return "Добраніч"
# Тест
for h in range(1,25):
    print(f"година: {h:>2} привітання: {привітання(h)}")
```

```
# solution chapter function task 1, variant B
# Розв'язання з розділу "Функція", завдання 1, варіант В
def привітання(година_доби):

    # словник : ключ : година_доби значення: привітання (dictionary: key: hour_of_day␣
↪value: greeting)
    розклад = {6:"Добраніч",
               11:"Доброго ранку",
               14:"Доброго дня",
               18:"Доброго пообіду",
               22:"Доброго вечора",
               24.1:"Добраніч",# особливий випадок для обробки числа 24
               }
    for ключ in розклад:
        if година_доби < ключ:
            return розклад[ключ] #повертає значення словника

# тест
for h in range(1,25):
    print(f"година: {h:>2} привітання: {привітання(h)}")
```

Завдання 2

- Напишіть функцію з назвою `удосконалене_привітання`.
- Функція повинна відтворювати привітання співробітника готелю, подібно до завдання 1.
- Функція повинна мати два *параметри*:
  - `година_доби` (число від 1 до 24)
  - `стать` ("чоловіча" чи "жіноча")
- Функція повинна *повертати* привітання, що залежить від `години доби` (див. таблицю в завданні 1) та `статі`:
  - додайте "Пане", якщо `стать` — чоловіча
  - додайте "Пані", якщо `стать` - жіноча

Приклади:

```
Доброго ранку, Пане
Добрий день, Пані
```

Крім того, як і в попередньому завданні, додайте код для тестування функції для кожної години доби (0–24) та для обох статей ("чоловічої" і "жіночої").

можливі розв'язання:

```
# solution chapter function task 2, variant A
# Розв'язання з розділу "Функція", завдання 2, варіант A
def удосконалене_привітання(година_доби,стать):
    суфікс = ""
    if стать == "чоловіча":
        суфікс = ", Пан"
    elif стать == "жіноча":
        суфікс = ", Пані"
    if 6 <= година_доби <11:
        return "Доброго ранку" + суфікс
    elif 11 <= година_доби < 14:
        return "Доброго дня" + суфікс
    elif 14 <= година_доби < 18:
        return "Доброго пообіду" + суфікс
    elif 18 <= година_доби < 22:
        return "Доброго вечора" + суфікс
    elif (22 <=година_доби) or (година_доби <6):
        return "Добраніч" + суфікс
# Тест
for h in range(1,25):
    for g in ("чоловіча", "жіноча"):
        print(f"година: {h:>2} стать: {g:<6} привітання: {удосконалене_привітання(h,g)}
↪")
```

```
# solution chapter function task 2 , variant B
# Розв'язання з розділу "Функція", завдання 2, варіант B
def удосконалене_привітання(година_доби,стать):
```

(continues on next page)

(continued from previous page)

```
    суфікс = {"чоловіча": "Пане",
              "жіноча":"Пані",
              }

    #   словник : ключ : година_доби значення: привітання (англ.dictionary: key: hour_of_
↪day value: greeting)
    розклад = {6:"Добраніч",
               11:"Доброго ранку",
               14:"Доброго дня",
               18:"Доброго пообіду",
               22:"Доброго вечора",
               24.1:"Добраніч", # особливий випадок для обробки числа 24
              }
    for ключ in розклад:
        if  година_доби < ключ:
            return розклад[ключ] + ", " + суфікс[стать]
# тест
for h in range(1,25):
    for g in ("чоловіча", "жіноча"):
        print(f"година: {h:>2} стать: {g:<6} привітання: {удосконалене_привітання(h,g)}
↪")
```

Завдання 3

Використайте приклад із попереднього завдання (завдання 2), але внесіть такі зміни:

- Перейменуйте функцію на `комплексне_привітання`
- Додайте додатковий *параметр* з назвою `дитина`
- Змініть *повернене значення* функції так, щоб коли значення параметра `дитина` дорівнює `True`, функція повертала "молодий чоловік" замість "Пане" і "молода леді" замість "Пані"

Додайте код для перевірки функції для всіх комбінацій `година_доби` (0-24), `стать` ("чоловіча", "жіноча") та `дитина` (True, False)

можливі розв'язання:

```
# solution chapter function task 3, variant A
# Розв'язання з розділу "Функція", завдання 3, варіант А

def комплексне_привітання(година_доби, стать, дитина):
    суфікс = ""
    if стать == "чоловіча":
        суфікс = ", Пане"
        if дитина: #  якщо дитина == True:
            суфікс = ", молодий чоловік"
    elif стать == "жіноча":
        суфікс = ", Пані"
        if дитина:
            суфікс = ", молода леді"
    if 6 <= година_доби <11:
        return "Доброго ранку" + суфікс
    elif 11 <= година_доби < 14:
```

(continues on next page)

(continued from previous page)

```
        return "Доброго дня" + суфікс
    elif 14 <= година_доби < 18:
        return "Доброго пообіду" + суфікс
    elif 18 <= година_доби < 22:
        return "Доброго вечора" + суфікс
    elif (22 <= година_доби) or (година_доби <6):
        return "Добраніч" +суфікс
# тест
for h in range(1,25):
    for g in ("чоловіча", "жіноча"):
        for c in (True, False):
            print(f"година: {h:>2} стать: {g:<6} дитина: {str(c):<5} "
                  f"привітання: {комплексне_привітання(h,g,c)}")
```

```
# solution chapter function task 3, variant B
# Розв'язання з розділу "Функція", завдання 3, варіант B
def комплексне_привітання(година_доби, стать, дитина):

    # словник : ключ : стать значення:(привітання_дорослих, привітання_дітей)
    # англ.(dictionary: key: gender value: (greeting_adult, greeting_child))
    суфікс = {"чоловіча": (" Пане","молодий чоловік"),
              "жіноча":("Пані","молода леді"),
              }
    #  словник : ключ : година_доби значення:привітання
    #  dictionary: key: hour_of_day value: greeting
    розклад = {6:"Добраніч",
               11:"Доброго ранку",
               14:"Доброго дня",
               18:"Доброго пообіду",
               22:"Доброго вечора",
               24.1:"Добраніч", # особливий випадок для обробки числа 24
              }
    for ключ in розклад:
        if година_доби < ключ:
            return розклад[ключ] + ", " + суфікс[стать][дитина]
            # True має значення 1, а False має значення  0
            # Тому "дитина" можна використовувати як індекс для першого/другого елемента
# тест
for h in range(1,25):
    for g in ("чоловіча", "жіноча"):
        for c in (True, False):
            print(f"година: {h:>2} стать: {g:<6} дитина: {str(c):<5} "
                  f"привітання: {комплексне_привітання(h,g,c)}")
```

Завдання 4

Це дуже просте завдання:

- Напишіть функцію з назвою `той_хто_вітає`:
- Функція не повинна мати жодних *параметрів*
- Функція повинна завжди *повертати рядок*: "Доброго ранку"

Додайте код, щоб вивести результат *виклику функції* в `той_хто_вітає_1`

Розв'язання

```
# solution chapter function, task4
# розв'язання з розділу "Функція", завдання 4

def той_хто_вітає_1():
    return "Добрий ранок!"

# виклик функції (виклик той_хто_вітає_1 без аргументів)
print("----- виклик той_хто_вітає_1 -------")
print(той_хто_вітає_1())
```

Завдання 5

- Створіть функцію з назвою `той_хто_вітає_2`:
- Функція повинна мати один *параметр* з назвою `прикметник`
- Типове значення параметра `прикметник` має бути `"гарний"`
- Функція повинна *повертати* рядок, що складається з `прикметника` та `" Ранок"`
- Протестуйте функцію, викликаючи її з різними аргументами (а також без аргументів). Завжди виводьте повернуте значення під час виклику функції.

Розв'язання

```
# solution function task 5
# Розв'язання з розділу "Функція", завдання 5

def той_хто_вітає_2 (прикметник="гарний"):
    return прикметник + " Ранок"


# виклик функції!l (виклик той_хто_вітає_2 з різними аргументами)
print("---- виклик той_хто_вітає_2 -----")
print(той_хто_вітає_2("добрий"))
print(той_хто_вітає_2("чудовий"))
print(той_хто_вітає_2())
print(той_хто_вітає_2(" "))
```

Завдання 6

- Створіть функцію з назвою `той_хто_вітає_3`:
- Функція повинна мати 2 параметра: `прикметник` and `година_доби` (обидва є рядками)
- Обидва параметри повинні мати значення аргументів за замовчуванням (англ. "default values") (наприклад,`"доброго"` та `"Ранку"`)
- Функція повинна повертати один рядок, що складається зі значення `прикметник`,пробілу та значення `година_доби`
- Протестуйте функцію, викликаючи її з різними аргументами (а також без аргументів) для обох параметрів, і завжди виводьте *повернуте значення* (англ.*return value*) кожного виклику функції.

```
# Задача з розділу "Функція", завдання 6
def той_хто_вітає_3 (прикметник="гарний", година_доби="Ранок"):
    return прикметник + " " + година_доби
# виклик функції (виклик той_хто_вітає_3)
print("----  виклик той_хто_вітає_3 -----")
print(той_хто_вітає_3())
print(той_хто_вітає_3("сонячний"))
print(той_хто_вітає_3("сонячний", "Вечір"))
print(той_хто_вітає_3(година_доби= "Вечір"))
```

Завдання 7

- Створіть функцію з назвою `той_хто_вітає_4`:
- Функція повинна мати один параметр з назвою `година_доби`
- Значення аргументів за замовчуванням (англ. "default values") `година_доби` повино бути `"Ранок"`
- Функція повинна приймати БУДЬ-ЯКУ кількість додаткових аргументів (включно з нулем).
- Функція повинна повертати рядок, що складається з усіх додаткових аргументів (розділених комами), пробілу та значення `година_доби` (усі параметри є рядками).
- Протестуйте функцію, викликаючи її кілька разів, щоразу з різною кількістю аргументів (у тому числі без аргументів). Виводьте повернуте значення кожного виклику функції. Наприклад: `print(той_хто_вітає_4("Вечір", "чудовий", "м'який", "прекрасний"))`

Можливі розв'язання:

```
# Задача з розділу "Функція", завдання 7 варіант А
# Функція, яка приймає будь-яку кількість параметрів і повертає їх усі.
def той_хто_вітає_4(година_доби="Ранок", *args):
    текст = ""
    for a in args:
        текст += a + ","
    if len(args) > 0:
        текст = текст[:-1]  # приберіть останню кому
        текст += " "
    текст += година_доби
    return текст

print("------ виклик той_хто_вітає_4 ----")
print(той_хто_вітає_4())
print(той_хто_вітає_4("Ніч"))
print(той_хто_вітає_4("вечір", "Тихий", "чудовий", "героїчний", "романтичний"))
print(той_хто_вітає_4("ніч", "Гарна"))
print(той_хто_вітає_4("день", "Сонячний", "теплий", "емоційний"))
```

```
# Задача з розділу "Функція", завдання 7 варіант В
# Функція, яка приймає будь-яку кількість параметрів і повертає їх усі.
def той_хто_вітає_4(година_доби="Ранок", *args):
    текст = ",".join(args)
    if len(текст) > 0:
        текст += " "
```

(continues on next page)

(continued from previous page)

```
    текст += година_доби
    return текст

print("------  виклик той_хто_вітає_4 ----")
print(той_хто_вітає_4())
print(той_хто_вітає_4("Ніч"))
print(той_хто_вітає_4("вечір", "Тихий", "чудовий", "героїчний", "романтичний"))
print(той_хто_вітає_4("ніч", "Гарна"))
print(той_хто_вітає_4("день", "Сонячний", "теплий", "емоційний"))
```

Завдання 8

- Створіть функцію з назвою `той_хто_вітає_ 5`:
- Функція повинна мати один параметр з назвою `година_доби`
- Значення аргументів за замовчуванням (англ. “default values”) `година_доби` повино бути `"Ранок"`
- Функція повинна приймати БУДЬ-ЯКУ кількість додаткових аргументів ключових слів (англ.“additional keyword arguments”), наприклад: `той_хто_вітає_5("Ранок",повітря="чудове", погода="сонячна")`
- Функція повинна повертати багаторядковий рядок (див. нижче), який містить усі аргументи в такій формі:
  - для *виклику функції*: `той_хто_вітає_5("Ранок", повітря="чудове", погода="сонячна")`
  - Повернене значення має бути таким: `"Який ранок!\nПовітря чудове.\nПогода сонячна."`
- Протестуйте функцію, викликаючи її кілька разів, щоразу з різною кількістю аргументів (у тому числі без аргументів). Виводьте повернуте значення кожного виклику функції.

Розв’язання:

```
# Задача з розділу "Функція", завдання 8
def той_хто_вітає_5(година_доби="Ранок", **kwargs):
    текст = "Який " + година_доби + "!\n"  # \n створює новий рядок
    for ключ, значення in kwargs.items():
        текст += ключ + " - " + значення + ".\n"
    return текст

print("------- виклик той_хто_вітає_5 ---------")
print(той_хто_вітає_5())
print(той_хто_вітає_5(Повітря="чудове", Настрій="радісний", Майбутне ="світле"))
print(той_хто_вітає_5("День", Температура="морозна", Вітер="сильний"))
```

Завдання 9

- Створіть функцію з назвою `той_хто_вітає_6`:
- Функція повинна мати один параметр з назвою `година_доби`
- Значення аргументів за замовчуванням (англ. “default values”) `година_доби` повино бути `"Ранок"`
- Функція повинна приймати БУДЬ-ЯКУ кількість додаткових аргументів (їхні значення — рядки), наприклад: `той_хто_вітає_6("День","добрий", повітря="чудове", погода="сонячна")`
- Функція повинна повертати багаторядковий рядок (див. нижче), який містить усі аргументи та всі аргументи ключових слів в такій формі:

- для *виклику функції*: `той_хто_вітає_6("Ранок", "добрий", "тихий", повітря="чудове", погода="сонячна")`
- Повернене значення має бути таким: `"Який добрий, тихий ранок!\nПовітря чудовеl.\nПогода сонячна"`
- Протестуйте функцію, викликаючи її кілька разів, щоразу з різною кількістю аргументів та іменованих аргументів. Виводьте повернуте значення кожного виклику функції.

```
# Задача з розділу "Функція", завдання 9
def той_хто_вітає_6(година_доби="Ранок", *args, **kwargs):
    текст = "Який "
    #ітератувати по *args
    for a in args:
        текст += a + ", "
    if len(args) > 0:
        текст = текст[:-2] + " "
    текст +=година_доби + "!\n"
    # ітератувати over **kwargs
    for ключ,значення in kwargs.items():
        текст += ключ + " - " + значення + ".\n"
    return текст

print("------ виклик той_хто_вітає_6 ----")
print(той_хто_вітає_6())
print(той_хто_вітає_6("Вечір"))
print (той_хто_вітає_6("Ранок", "сонячний", "теплий", "емоційний",
                  повітря="чудове", настрій="радісний", майбутне="якраве"))
print(той_хто_вітає_6(повітря="ароматне"))
```

3.39.5 Завдання до розділу об'єктно-орієнтоване програмування :

 англійська: *Tasks for chapter object oriented programming*

Завдання 1:

- Напишіть клас з назвою `Гра`
- Клас повинен мати *змінну класу* з назвою `гравець`. *Значенням* цієї змінної має бути "Bugs Bunny".
- Клас повинен мати змінну класу з назвою `рекорд`. *Значенням* цієї змінної має бути `1000`.
- Клас повинен мати змінну класу з назвою `кредит`. *Значенням * цієї змінної має бути 2.
- Напишіть код на Python, який виведе всі змінні класу `Гра` та їхні значення.

Розв'язання:

```
# solution oop task 1 variant A
# Розв'язання з розділу "ООП" Завдання 1 варіант А
class Гра:
    гравець = "Bugs Bunny"
    рекорд = 1000
    кредит = 2
```

(continues on next page)

(continued from previous page)

```
print("Гра.гравець", Гра.гравець)
print("Гра.рекорд", Гра.рекорд)
print("Гра.кредит", Гра.кредит)
```

```
# solution oop task 1 variant B
# Розв'язання з розділу "ООП" Завдання 1 варіант B
class Гра:
    гравець = "Bugs Bunny"
    рекорд = 1000
    кредит = 2

for ключ, значення in Гра.__dict__.items():
    if ключ[:2] != "__":
        print(ключ, значення)
```

- Напишіть клас з назвою `Іграшка`
- Клас повинен мати метод `__init__`.
- Напишіть клас так, щоб кожен *екземпляр* цього класу мав такі *атрибути* (також звані *змінними об'єкта*):
  - `ціна`зі значенням `10`
  - `коляр` зі значенням `"зелений"`
- Створіть змінну з назвою `teddy`.
- Значенням змінної `teddy` повинен бути *екземпляр* класу `Іграшка`
- Напишіть код, який встановлює атрибут `висота`об'єкта `teddy` у значення `7`
- Напишіть код, який виводить імена та значення всіх атрибутів об'єкта `teddy`

Можливі розв'язання:

```
# solution oop task 2 variant A
# Розв'язання з розділу "ООП" Завдання 2 варіант A
class Іграшка:
    def __init__(self):
        self.ціна = 10
        self.коляр = "зелений"

teddy = Іграшка()
teddy.висота = 7
print("ціна", teddy.ціна)
print("коляр", teddy.коляр)
print("висота", teddy.висота)
```

```
# solution oop task 2 variant B
# Розв'язанняа з розділу "ООП" Завдання 2 варіант B
class Іграшка:
    def __init__(self):
        self.ціна = 10
        self.коляр = "зелений"
```

(continues on next page)

(continued from previous page)

```
teddy = Іграшка()
teddy.висота = 7
for ключ, значення in teddy.__dict__.items():
    print(ключ, значення)
```

Завдання 3

- Створіть клас з назвою `Walker`.
- Додайте до цього класу *метод* з назвою `Walk`.
- *Значення* що повертається цим методом, повинно бути рядком `"I'm walking"`
- Створіть клас з назвою `Swimmer`.
- Додайте до цього класу *метод* з назвою `Swim`.
- *Значення* що повертається цим методом, повинно бути рядком `"I'm swimming"`
- Створіть клас з назвою `Flyer`.
- Додайте до цього класу *метод* з назвою `Fly`.
- *Значення* що повертається цим методом, повинно бути рядком `"I'm flying"`
- Створіть клас з назвою `Diver`.
- Додайте до цього класу *метод* з назвою `Dive`.
- *Значення* що повертається цим методом, повинно бути рядком `"I'm diving"`
- Створіть класи з назвами та *методами* відповідно до таблиці нижче. Використайте наслідування від наявних класів для нових класів. Усередині кожного класу напишіть лише команду `pass` , *НЕ* пишіть методи або атрибути.
- Створіть змінну з назвою `tux`. Значення цієї змінної має бути екземпляром класу `Penguin`.
- Напишіть код Python, який виводить результат виклику `tux.dive()`

назва класу	walk()	swim()	fly()	dive()
Falcon	Yes	No	Yes	No
Penguin	Yes	Yes	No	Yes
Duck	Yes	Yes	Yes	Yes
Eurasian_swift	No	No	Yes	No

possible solution

```
# solution oop task 3 variant A
# Розв'язання з розділу "ООП" Завдання 3 варіант А


# parent classesб(укр.:"базовий клас (або батьківський) ")
class Walker:
   def walk(self):
      return "i'm walking"
```

(continues on next page)

(continued from previous page)

```
class Swimmer:
    def swim(self):
        return "i'm swimming"

class Flyer:
    def fly(self):
        return "i'm flying"

class Diver:
    def dive(self):
        return "i'm diving"

# child classes (укр.:"похідний клас (або дочірній)")
class Falcon(Walker, Flyer):
    pass

class Penguin(Walker, Swimmer, Diver):
    pass

class Duck(Walker, Swimmer, Flyer, Diver):
    pass

class Eurasian_swift(Flyer):
    pass

tux = Penguin()    # створіть екземпляр класу
print(tux.dive())
```

Завдання 4

Наведено такий код:

```
# chapter oop, question task 4 and 5
#  розв'язання з розділу "ООП" Завдання 4 та 5

class Bird:
    def __init__(self, name):
        self.name = name
        self.can_fly = True
        self.can_walk = True
        self.can_swim = False
        self.can_dive = False

    def __str__(self):
        """Ця функція викликається під час виведення екземпляра класу."""
        return f"i am a {self.__class__.__name__}"
```

- Напишіть код Python для створення змінної з назвою `tux`.
- Значенням цієї змінної повинен бути *екземпляр класу* `Bird`.
- **Атрибут** `name` цього екземпляра класу повинен мати значення `"Duck"`.
- Напишіть новий рядок коду Python, який встановлює *атрибут* `can_swim` об'єкта `tux` у значенні

`True`.

Можливі розв'язанняп

```
# solution chapter oop, task 4
# розв'язання з розділу "ООП" Завдання 4
tux = Bird(name="Duck")
tux.can_swim = True
```

Завдання 5

- Наведено клас `Bird` із Завдання 4.
- Напишіть код Python для класу з назвою `Penguin`.
- Цей клас повинен бути дочірнім класом `Bird`.
- Цей клас повинен успадковувати всі методи класу `Bird`, включно з методом `__init__` класу `Bird`.
- Змініть метод **init** класу `Penguin` так, щоб *атрибут* `continent` завжди мав значення `"Antarctic"`
- Змініть метод **init** класу `Penguin` так, щоб *атрибути* `can_swim` та `can_dive` були встановлені в `True`, а *атрибут* `can_fly-` у `False`

Можливі розв'язання

```
# solution chapter oop Task 5 variant A
# Розв'язання з розділу "ООП" Завдання 5 варіант A
class Penguin(Bird):
    def __init__(self, name):
        #У разі використання назви батьківського класу потрібно додати
        Bird.__init__(self, name)
        self.continent = "Antarctic"
        self.can_swim = True
        self.can_dive = True
        self.can_fly = False
```

```
# solution chapter oop Task 5 variant B
# Розв'язання з розділу "ООП" Завдання 5 варіант B
class Penguin(Bird):
    def __init__(self, name):
        # використовуючи super() для звернення до батьківського класу
        super().__init__(name) # self не потрібен
        self.continent = "Antarctic"
        self.can_swim = True
        self.can_dive = True
        self.can_fly = False
```

3.39.6 Завдання до розділу введення-виведення Tasks:

 англійська:**Input and Output**

Завдання 1

- Напишіть програму на Python, яка записує рядок `"Привіт,Світ!"` у файл з назвою *файлу* `hello.txt`.
- Виведіть текст `Файл було записано на диск`

Можливі розв'язання:

```
# solution chapter IO task 1 variant A
# Розв'язання з розділу "введення-виведення" Завдання 1 варіант A
text = "Привіт,Світ!"
myfile = open("hello.txt", "w") # режим запису
myfile.write(text)
myfile.close()
print("Файл було записано на диск")
```

```
# solution chapter IO task 1 variant B
text = "Hello World!"
with open("hello.txt", "w") as myfile:
    myfile.write(text)
# закривається автоматично!
print("Файл було записано на диск")
```

Завдання 2

- Якщо ви не виконали завдання 1: створіть (за допомогою текстового редактора) новий текстовий файл, який містить один рядок тексту. Останнім символом цього рядка тексту повинен бути *знак оклику* (`!`). Збережіть цей текстовий файл під назвою `hello.txt`. Якщо ви правильно виконали завдання 1, цей файл уже існує.
- Напишіть програму на Python, яка змінює наявний текстовий файл з назвою `hello.txt` так, щоб:
  - У кінець наявного тексту було додано два порожні рядки
  - Після цих двох порожніх рядків було додано ще один рядок тексту (в новому рядку): `Як справи?`
  - Текст повинен закінчуватися *перенесенням рядка*
- Напишіть код Python, який виводить на екран слова `Текстовий рядок додано`

Можливе розв'язання:

англійський варіант:

```
# solution chapter IO task 2 variant A
text = "\n\n\nhow do you do?\n"
with open("hello.txt", "a") as myfile:
    myfile.write(text)
print("Textline added")
```

український варіант:

```
# solution chapter IO task 2 variant A
# розв'язання з розділу "введення-виведення" Завдання 2 варіант A
текст = "\n\n\nЯк справи??\n"
with open("hello.txt", "a") as мій_файл:
```

(continues on next page)

(continued from previous page)

```
    мій_файл.write(текст)
print("Текстовий рядок додано")
```

3.39.7 Завдання 3

- Напишіть програму, яка відкриває наявний текстовий файл з назвою `hello.txt`.
- Програма повинна визначити, скільки рядків тексту (скільки перенесень рядка) міститься в цьому текстовому файлі.
- Програма повинна вивести рядок: `знайдено рядків:` і вивести кількість текстових рядків.

Можливе розв'язання:

```
# solution chapter IO task 3 variant A
# Розв'язання з розділу "введення-виведення" Завдання 3 варіант А
with open("hello.txt") as myfile:
    lines = myfile.readlines()
print("знайдено рядків:", len(lines))
```

3.39.8 Завдання до розділу Винятки:

 англійська:**Exceptions**

Завдання 1

Наведено таку програму на Python:

```
# question chapter exception task 1
# запитання, розділ «Винятки», завдання 1
print("будь ласка, введіть рік вашого народження у форматі YYYY")
рік_текст = input(">>>")
рік = int(рік_текст)
print("у 2050 році вам буде ", 2050-рік, "років")
```

- Змініть програму так, щоб вона НЕ завершувалася помилкою `ValueError` , коли користувач вводить некоректні дані (наприклад, коли користувач вводить літери замість числа). Натомість програма повинна знову запитувати введення доти, доки введене значення не буде числом.

Можливе розв'язання

```
# solution chapter exception task 2 variant A
# Розв'язання з розділу «Винятки», завдання 2 варіант А
while True:
    print("будь ласка, введіть рік вашого народження у форматі YYYY")
    рік_текст = input(">>>")
    try:
        рік = int(рік_текст)
    except ValueError:
        print("введено не число, будь ласка, спробуйте ще раз")
        continue
    # введення було правильним
    break
print("у 2050 році вам буде ", 2050-рік, "років")
```

```
# solution chapter exception task 2 variant B
# розв'язання з розділу «Винятки», завдання 2 варіант B
while True:
    print("будь ласка, введіть рік вашого народження у форматі YYYY")
    рік_текст = input(">>>")
    if рік_текст.isdigit():
        рік = int(рік_текст)
        break
    print("введено не число, будь ласка, спробуйте ще раз")
print("у 2050 році вам буде ", 2050-рік, "років")
```

Завдання 3

Наведено таку програму на Python:

```
# запитання, розділ «Винятки», завдання 3 ( на англійській)
chessboard = [
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
              ["white","black","white","black","white","black","white","black"],
              ["black","white","black","white","black","white","black","white"],
             ]

def get_color(row, column):
    row = int(row)
    column = int(column)
    return chessboard[row][column]

while True:
    r = input("enter row number: (0-7) >>>")
    c = input("enter column number (0-7) >>>")
    result = get_color(r,c)
    print(f"The color the field (row {r} column {c}) is: {result}")
```

3.40 Запитання, розділ «Винятки», завдання 3 (на українській)

```
# запитання, розділ «Винятки», завдання 3 ( на англійській)
шахова_дошка = [
              ["біла","чорна","біла","чорна","біла","чорна","біла","чорна"],
              ["чорна","біла","чорна","біла","чорна","біла","чорна","біла"],
              ["біла","чорна","біла","чорна","біла","чорна","біла","чорна"],
              ["чорна","біла","чорна","біла","чорна","біла","чорна","біла"],
              ["біла","чорна","біла","чорна","біла","чорна","біла","чорна"],
              ["чорна","біла","чорна","біла","чорна","біла","чорна","біла"],
              ["біла","чорна","біла","чорна","біла","чорна","біла","чорна"],
              ["чорна","біла","чорна","біла","чорна","біла","чорна","біла"],
             ]
```

(continues on next page)

(continued from previous page)

```
def отримати_колір(рядок, стовпець):
    рядок = int(рядок)
    стовпець = int(стовпець)
    return шахова_дошка[рядок][стовпець]

while True:
    r = input("введіть номер рядка (0-7) >>>")
    c = input("введіть номер стовпця (0-7) >>>")
    результат = отримати_колір(r,c)
    print(f"Колір поля (рядок {r} стовпець {c}) є: {результат}")
```

- Змініть код функції `отримати_колір` так, щоб:
- функція повертала рядок `"введено не числа"` коли `рядок` чи `стовпець` (або обидва) не є цілими числами;
- функція повертала рядок `"некоректний індекс"`, коли `рядок` чи `стовпець` (або обидва) менші за 0 або більші за 7.

Можливе розв'язання:

```
# solution chapter exception task 4
# запитання, розділ «Винятки», завдання 4
def отримати_колір(рядок,стовпець):
    try:
        r=int(рядок)
        c=int(стовпець)
    except ValueError:
        return "введено не числа"
    if (c < 0) or (c>7) or (r<0) or (r>7):
        return "некоректний індекс"
    return шахова_дошка[r][c]
```