Про автора Head First Python

Під час прогулянки Пол зупиняється, аби обговорити правильну вимову слова «tuple» (тобто «кортеж») зі своєю багатостраждальною дружиною.

Звичайна реакція Дейдри. ☺

Пол Беррі живе і працює в Карлоу — маленькому містечку з населенням близько 35 тисяч осіб, розташованому за 80 км на південний захід від Дубліна.

Пол має ступінь бакалавра наук у галузі інформаційних систем і ступінь магістра в галузі обчислень. Він також закінчив аспірантуру й отримав свідоцтво на право викладання і навчання.

Пол працює в Технологічному інституті Карлоу з 1995 року. Перш ніж почати викладацьку діяльність, він десять років присвятив IT-індустрії, працював в Ірландії і Канаді, причому більша частина його роботи була пов'язана з медичними закладами. Пол має дружину Дейдру, у них троє дітей (двоє зараз вчаться в коледжі).

Мова програмування *Python* (і пов'язані з нею технології) стала невід'ємною частиною бакалаврського лекційного курсу Пола починаючи з 2007 року.

Він є автором (або співавтором) ще чотирьох книжок: дві з них про *Python*, а ще дві — про мову програмування *Perl*, а також цілої низки статей, опублікованих у *«Linux Journal Magazine»*.

Пол виріс у Белфасті (Північна Ірландія), і це багато в чому поясню деякі особливості його поглядів і кумедний акцент.

Ви можете зв'язатися з Полом Беррі у Твіттері (*@barrypj*); також він має власну домашню сторінку за адресою *http://paulbarry.itcarlow.ie.*

Зміст (коротко)

Зміст (докладно)

1 Підвалини

Швидке занурення

Починаємо програмувати на Python якомога швидше. У цьому розділі ми ознайомимося з основами програмування на Python і зробимо це в характерному для *Head First* стилі: із місця в кар'єр. Через кілька сторінок ви запустите свою першу програму. До кінця розділу ви зможете не лише запускати типові програми, але й розуміти їхній код (і це ще не все!) Водночас ви познайомитеся з деякими особливостями мови **Python**.

2 Списки даних

Робота з упорядкованими даними

Усі програми обробляють дані, і програми на Python — не виняток. Насправді, досить роззирнутися — *дані скрізь*. Адже програмування — це, переважно, робота з даними: *отримання* даних, *обробка* даних, *інтерпретація* даних. І щоби працювати з даними ефективніше, потрібен якийсь контейнер, куди їх можна *складати*. Python надає зручні структури даних *широкого застосування*: **списки, словники, кортежі** і **множини**. У цьому розділі ми побіжно розглянемо всю цю четвірку, а тоді заглибимося у вивчення *списків* (інші три структури будуть докладніше розглянуті в наступному розділі). Ми вже торкалися цієї теми раніше, оскільки все, із чим нам доводиться стикатися при програмуванні на Python, так чи інакше стосується роботи з даними.

3 Структуровані дані

Робота зі структурованими даними

Списки в Python дуже зручні, але вони не є панацеєю. Коли є *дійсно* структуровані дані (для зберігання яких список виявляється не кращим вибором), то порятунок треба шукати у вбудованих **словниках** Python. Словники «із коробки» дозволяють зберігати й обробляти дані, що їх можна представити у вигляді пар *ключ / значення*. У цьому розділі ми докладно розглянемо словники, а також **множини** і **кортежі**. Разом зі списками (із ними ми знайомилися в Розділі 2) словники, множини і кортежі є вбудованими інструментами для роботи з даними, що дозволяють створити потужну комбінацію даних мовою Python.

4 Повторне використання

Функції та модулі

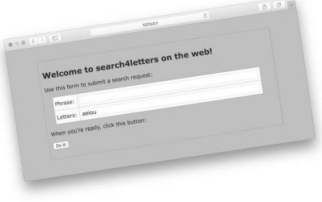
Повторне використання коду — ключ до створення стабільних систем. У випадку з Python все повторне використання починається і закінчується **функціями**. Візьміть кілька рядків коду, дайте їм ім'я — й у вас уже готова **функція** (що її можна використовувати повторно). Візьміть колекцію функцій, запакуйте їх у файл — й у вас з'явиться готовий **модуль** (його теж можна використовувати повторно). Це правда, коли кажуть: *ділитися приємно*, тож наприкінці цього розділу ви вже зможете створювати код для **багаторазового** та **спільного** використання, завдяки усвідомленню того, як працюють функції та модулі Python.

Модуль

5 Створення вебзастосунку

Повернення в реальний світ

На цьому етапі ви вже достатньо знаєте про Python, аби вважатися небезпечними. Ви вже засвоїли перші чотири розділи цієї книжки, тож тепер здатні продуктивно використовувати Python у багатьох галузях (хоча ще багато чого вам належить дізнатися). Замість того щоби звернутися до цих питань, у цьому та наступних розділах ми зосередимося на розробці вебзастосунків, що дозволить нам оцінити переваги мови Python. Водночас ви трохи більше дізнаєтесь про Python.



6 Зберігання та обробка даних

Де зберігаються дані

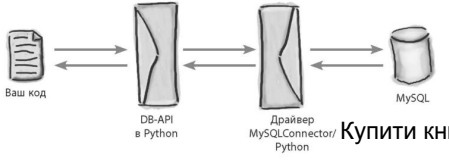
Рано чи пізно виникає необхідність забезпечити надійне зберігання даних. І коли справа доходить до **зберігання даних**, Python вам допоможе. У цьому розділі ви дізнаєтеся про зберігання й отримання даних із *текстових файлів*, що як технології зберігання можуть здатися надто простими, однак використовуються у багатьох проблемних галузях. Окрім збереження та отримання даних із файлів, ви засвоїте ще деякі премудрості, коли йтиметься про обробку даних. «Серйозний матеріал» (зберігання інформації в базах даних) припасений нами для наступного розділу, однак із файлами теж доведеться повозитися.

Form Data	Remote_addr	User_agent	Results
ImmutableMultiDict([('phrase', 'hitch-hiker'), ('letters', 'aeiou')])	127.0.0.1	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526 .106 Safari/537.36	{'e', 'i'}

7 Застосування бази даних

Застосовуємо DB-API у Python

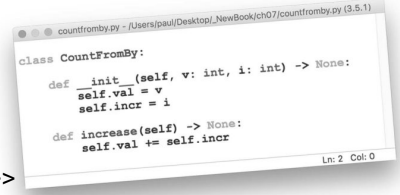
Зберігати інформацію в реляційній базі даних дуже зручно. У цьому розділі ви дізнаєтеся, як організувати взаємодію з популярною базою даних (БД) **MySQL**, використовуючи універсальний прикладний програмний інтерфейс **DB-API**. Інтерфейс DB-API (входить до складу стандартної бібліотеки Python) дозволяє створювати код, що не залежить від конкретної бази даних… за умови, якщо база даних розуміє SQL. Хоча ми будемо використовувати MySQL, ніщо не завадить вам використовувати код DB-API з вашою улюбленою реляційною базою даних, хай би якою вона була. Давайте подивимося, як користуватися реляційною базою даних у Python. У цьому розділі не так багато нового з погляду вивчення Python, але застосування Python для спілкування з БД — це **дуже важливо**, тому варто докласти зусиль.



8 Трохи про класи

Абстракція поведінки і стану

Класи дозволяють зв'язати поведінку коду і його стан. У цьому розділі ми поки що залишимо у спокої вебзастосунок і почнемо вчитися створювати **класи** Python. Це вміння знадобиться вам при створенні диспетчера контексту. Класи є настільки корисною річчю, що вам у будь-якому разі варто ознайомитися з ними ближче, тому ми присвятили їм окремий розділ. Ми не будемо розглядати класи у всіх подробицях, а торкнемося лише тих аспектів, що стануть у пригоді при створенні диспетчера контексту, на який очікує наш вебзастосунок.


countfromby.py - /Users/paul/Desktop/_NewBook/ch07/countfromby.py (3.5.1)

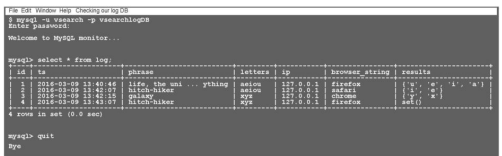
```
class CountFromBy:

    def __init__(self, v: int, i: int) -> None:
        self.val = v
        self.incr = i

    def increase(self) -> None:
        self.val += self.incr
```

Ln: 2 Col: 0

9 Протокол управління контекстом

Підключення до інструкції «with» мови Python

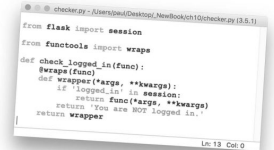
Настав час застосувати усе вивчене на практиці. У Розділі 7 ми обговорили використання **реляційних баз даних** у Python, а в Розділі 8 ознайомилися з використанням **класів** у коді Python. У цьому розділі ми об'єднаємо обидві методики і створимо **диспетчер контексту**, що дозволить розширити інструкцію **with** для роботи з системами реляційних баз даних. У цьому розділі ви підключитеся до інструкції **with**, створивши новий клас, що відповідає вимогам **протоколу управління контекстом** у мові Python.



10 Декоратори функцій

Обгортання функцій

Що стосується розширення вашого коду, то протокол управління контекстом із Розділу 9 — не єдина можливість. Python також дозволяє застосовувати **декоратори** функцій — технологію, за допомогою якої можна додавати код до існуючих функцій, не змінюючи наявний код. Якщо ви побачите у цьому схожість із чорною магією, не переймайтеся: нічого подібного. Однак чимало програмістів, які працюють із Python, вважають створення декораторів одним із найскладніших процесів, тому користуються цією можливістю рідше, ніж мали б. У цьому розділі ми продемонструємо, що створення і використання декораторів — це не дуже складно, незважаючи на їхню «просунутість».



11 Обробка винятків

Що робити, коли щось іде не так

Хай би яким гарним був ваш код, іноді все одно щось іде не так. Ви успішно виконали всі приклади із книжки та, ймовірно, переконалися, що наявний код працює. Однак чи означає це, що його можна вважати надійним? Найвірогідніше, ні. Створювати код і сподіватися, що нічого поганого ніколи не станеться, у кращому разі наївно. А в гіршому — небезпечно, оскільки щось непередбачене часом усе ж відбувається (і буде відбуватися). При створенні коду краще бути обережним, ніж довірливим. І обережність потрібна, щоби ваш код робив саме те, що вам потрібно, і чітко реагував, якщо щось піде не так.


```
...
Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
    |    +-- FloatingPointError
    |    +-- OverflowError
    |    +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
         ...
```


11¾ Обробка очікування

Трохи про багатопотоковість

Іноді для виконання коду може знадобитися чимало часу. Інколи це стає проблемою, інколи — ні. Якщо код працює «за кулісами» і йому потрібно 30 секунд, аби зробити якісь свої справи, очікування не перетворюється на проблему. Однак якщо користувач протягом 30 секунд очікує відповіді від застосунку, це стає аж надто відчутним. Розв'язання проблеми залежить від того, які саме дії ви намагаєтеся виконати (і хто саме очікує). У цьому короткому розділі ми обговоримо деякі варіанти, а потім розглянемо одне з рішень наступної проблеми: *як діяти, якщо щось вимагає занадто багато часу*?

12 Просунуті ітерації

Божевільні цикли

Можна тільки дивуватися, скільки часу наші програми проводять у циклах! А втім, у цьому немає нічого дивного, бо більшість програм призначена для швидкого виконання однакових дій безліч разів поспіль. Говорячи про оптимізацію циклів, можна виділити два підходи: 1) поліпшення синтаксису (щоби спростити визначення циклів) і 2) поліпшення способу виконання (щоби прискорити роботу циклів). За часів Python 2 (тобто досить давно) творці мови додали єдину мовну можливість, що реалізує обидва підходи і називається доволі дивно — **comprehension** , або **генераторний вираз**.

Додатки

A: Інсталяція. *Інсталяція Python*

Передусім давайте встановимо Python на вашому комп'ютері. Хай би якою ОС ви користувалися — Windows, Mac OS X або Linux,— Python для вас доступний. Послідовність установки на кожну із цих платформ залежить від організації процесу в кожній з операційних систем (ми знаємо, що це… трохи шокує, чи не так?), але розробники Python добре попрацювали, створивши майстер установки для кожної з популярних систем. У цьому додатку ви дізнаєтеся, як установити Python на свій комп'ютер.

B: PythonAnywhere. *Розгортання вашого вебзастосунку*

Наприкінці Розділу 5 ми зазначили, що розгортання вебзастосунку у хмарі забере лише 10 хвилин. Настав час виконати обіцянку. У цьому додатку ми збираємося провести вас через весь процес розгортання вебзастосунку в PythonAnywhere за 10 хвилин від самого початку до повністю розгорнутого застосунку. PythonAnywhere — це служба, популярна у спільноті кодерів на Python, тому що вона працює в цілковитій відповідності з очікуваннями, має відмінну підтримку Python (а також Flask) і — що найважливіше — дозволяє почати з безкоштовного розміщення вебзастосунку.

C: ТОП-10 тем, які ми не висвітлили.
Завжди є чому повчитися

Ми й не прагнули розглянути всі питання та аспекти. Наша мета — познайомити вас із Python, аби ви якомога швидше могли почати працювати із ним. Ми могли б охопити більше тем, але не зробили цього. У цьому додатку ми обговоримо топ-10 тем, які ми обов'язково розглянули би докладніше, якби мали ще приблизно 600 вільних сторінок. Не всі із цих 10 пунктів будуть для вас цікавими або актуальними, але ви можете швидко проминути їх у пошуках більш важливого для вас матеріалу або відповіді на наболіле питання. Усі технології програмування, наведені у цьому додатку, готові до використання в Python і його інтерпретаторі.

D: ТОП-10 проєктів, що ми їх не розглянули.
Ще більше інструментів, бібліотек і модулів

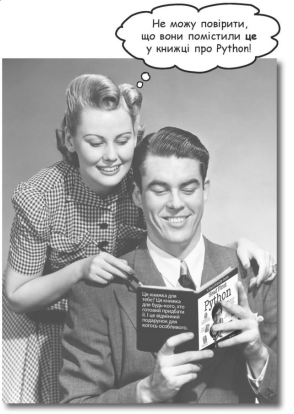
Ми знаємо, про що ви подумали, прочитавши заголовок цього додатка. А чому би їм не назвати попередній додаток: «Топ-20 тем, що не були висвітлені»? І навіщо нам ще 10 проєктів? Річ у тім, що в попередньому додатку ми обмежили обговорення інструментами, що поставляються разом із Python (ті самі «батарейки в комплекті»). У цьому додатку ми рушимо далі й обговоримо низку технологій, що стали доступними саме тому, що існує Python. Тут перелічено багато корисного, але — як і у випадку з попереднім додатком,— якщо ви побіжно переглянете ті кілька сторінок, це неодмінно піде вам на користь.

E: Долучайтеся! *Спільнота Python*

Python — це набагато більше, ніж відмінна мова програмування. Це чудова компанія. Спільнота Python гостинна, різноманітна, відкрита, дружня, щедра і готова ділитися. Навіть дивно, що ніхто ще не надрукував таку вітальну листівку! Якщо серйозно, то Python — не стільки мова, скільки особлива наука програмування. Навколо Python сформувалася ціла екосистема: корисні книжки, блоги, вебсайти, конференції, зустрічі, групи користувачів і неперервні особистості. У цьому додатку ми познайомимося зі спільнотою Python і подивимося, що вона може запропонувати. Не пишіть код на самоті: **долучайтеся до спільноти!**

Як користуватися цією книжкою

Вступ



У цьому розділі ми відповімо на актуальне питання: «ЧОМУ вони помістили це в книжку про Python?»

Для кого ця книжка?

Якщо ви можете ствердно відповісти на всі ці питання:

1. Чи ви вже володієте якоюсь іншою мовою програмування?
2. Чи хочете ви мати покрокове ноу-хау із програмування мовою *Python*, додати його до свого переліку інструментів і змусити робити нові речі?
3. Чи надаєте ви перевагу активній розмові за вечерею перед сухою та нудною технічною лекцією?

Це НЕ довідник, і ми припускаємо, що ви вже програмували раніше.

Тоді ця книжка для вас.

Кому, ймовірно, краще триматися якнайдалі від цієї книжки?

Якщо ви можете відповісти **«*так*»** на будь-яке із цих питань:

1. Можливо, ви вже знаєте більшість того, що необхідно для програмування за допомогою *Python*?
2. Ви досвідчений програміст і шукаєте довідник із *Python*, що охоплює всі подробиці і наводить найдрібніші деталі?
3. Ви радше дозволите видерти ваші нігті 15 мавпам, аніж спробуєте вивчити щось нове? І заразом вважаєте, що книжка про *Python* має охоплювати геть усе, і якщо читачеві стає нудно аж до сліз, то тим краще?

У такому разі ця книжка ***не*** для вас.

[Примітка від маркетингового відділу: ця книжка призначена для всіх, хто має кредитну картку... ми також приймаємо чеки.]

Ми знаємо, про що ви подумали

«Як ця книжка із програмування мовою Python може бути серйозною?»

«Що це за ілюстрації?»

«Чи можу я насправді оволодіти матеріалом, що подається таким чином?»

І ми знаємо, чим переймається ваш мозок

Ваш мозок вважає ЦЕ важливим.

Ваш мозок жадає новизни. Він завжди шукає, сканує, чекає на щось незвичне. Він побудований так, і саме це допомагає вам залишатися живими.

Нині ви малоймовірно станете перекусом для тигра. Але ваш мозок усе ще пильнує. Ви просто не помічаєте цього.

Отже, що робить ваш мозок зі всіма повсякденними, звичайними, нормальними речами, із якими ви стикаєтеся? Він робить усе можливе, аби вони не заважали йому запамʼятовувати дійсно важливу інформацію. Він не переймається запамʼятовуванням нудних речей; вони ніколи не встигають просочитися крізь фільтр «Це, вочевидь, не важливо».

Як саме ваш мозок дізнається, що є важливим? Припустімо, ви вийшли прогулятися і раптово перед вами зʼявляється тигр. Що відбувається у вашій голові?

Нейрони напружуються. Емоції зашкалюють. *Активізуються хімічні процеси*. Ось як ваш мозок працює...

Це дуже важливо! Не забувайте цього!

А зараз уявіть, що ви вдома чи в бібліотеці. Це безпечне затишне місце, де немає жодного тигра. Ви щось вивчаєте. Готуєтесь до іспиту. Або намагаєтесь засвоїти якусь важку технічну тему, опанувати яку, на думку вашого керівника, можна впродовж тижня, максимум за десять днів.

Однак є проблема. Ваш мозок намагається зробити вам велику послугу. Він намагається переконатися, що ця, вочевидь, не важлива інформація не споживає дефіцитні ресурси. Ресурси, що їх краще витратити на запамʼятовування дійсно важливих речей. На кшталт тигрів. Або небезпеки раптової пожежі. Або настанови ніколи більше не кататися на сноуборді у шортах.

Не існує простого способу сказати своєму мозкові: «Агов, мозку, дуже дякую, але, хай якою нудною була б ця книжка, і як би мало емоцій вона зараз викликала, я дійсно хочу, аби ти затямив усі ці речі».

Ми вважаємо читача «Head First» учнем

Як ми щось *дізнаємося*? Спочатку потрібно це «щось» *зрозуміти*, а потім *не забути*. Заштовхати в голову якнайбільше фактів недостатньо. Згідно з даними новітніх досліджень у галузі когнітивістики, нейробіології та психології навчання, для *засвоєння* матеріалу потрібно щось більше, ніж простий текст на сторінці. Ми знаємо, як змусити ваш мозок працювати.

Основні принципи серії «Head First»:

Робити все наочним. Графіка запам'ятовується значно краще за звичайний текст і значно підвищує ефективність сприйняття інформації (до 89 % підвищується пригадування та відтворення вивченого). До того ж, матеріал стає зрозумілішим. Якщо **розташувати відповідний текст на малюнках або поряд з ними**, а не під ними або на сусідній сторінці, учні отримають удвічі більше шансів зрозуміти зміст.

Використовувати розмовний та персоналізований стиль викладу. Нещодавні дослідження показали, що за особистісного розмовного стилю викладу матеріалу замість формальних лекцій поліпшення результатів на підсумковому тестуванні складало до 40 %. Розповідайте історію замість того, щоби читати лекцію. Не ставтеся до себе занадто серйозно. Що ймовірніше приверне вашу увагу: цікава бесіда за вечерею чи лекція?

Змушувати учня мислити глибше. Поки ви не почнете ворушити звивинами, у вашій голові нічого не відбуватиметься. Читач повинен бути зацікавлений у результаті; він повинен натхненно вирішувати завдання, формулювати висновки і генерувати нові знання. А для цього необхідні вправи і каверзні питання, у пошуку відповідей на які задіяні обидві півкулі мозку й різноманітні почуття.

Привертати увагу читача — і фіксувати її. Ситуація, знайома кожному: «Я дуже хочу вивчити це, але засинаю на першій сторінці». Мозок звертає увагу на цікаве, дивне, приваблює, несподіване. Вивчення складної технічної теми не має бути нудним. Ваш мозок осягне це набагато швидше, якщо це буде цікаво викладено.

Залучати емоції. Відомо, що наша здатність запам'ятовувати значною мірою залежить від емоційного співпереживання. Ми запам'ятовуємо те, що нам не байдуже. Ми запам'ятовуємо, коли щось відчуваємо. Ні, ми не маємо на увазі хвилюючі історії про хлопчика та його собаку. Мова йде про такі емоції, як здивування, цікавість, інтерес і відчуття власної крутизни при розв'язанні задачі, яку оточення вважає надто складною, або коли ви зрозуміли, що розбираєтеся в темі краще за всезнайку Боба з технічного відділу.

Метапізнання: мислення про мислення

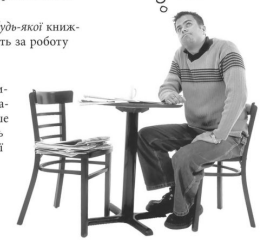
Якщо ви дійсно хочете вчитися, і хочете дізнатися швидше і глибше, зверніть увагу на те, як ви звертаєте увагу. Подумайте, як ви мислите. Дізнайтеся, як ви вчитеся.

Більшість із нас не вивчали теорію метапізнання або навчання під час навчання. Ми *маємо* вчитися, але нас рідко цьому *навчають*.

Проте, оскільки ви тримаєте в руках цю книжку, то, імовірно, дійсно хочете вивчати мову Python. І ви, мабуть, не хочете витрачати на це забагато часу.

Аби отримати максимум користі від цієї книжки, або *будь-якої* книжки чи досвіду навчання, візьміть на себе відповідальність за роботу власного мозку. Вашого мозку у *цьому* контенті.

Хитрість полягає в тому, аби змусити ваш мозок сприймати новий матеріал, що ви вивчаєте, як дійсно важливий. Як те, що має вирішальне значення для вашого благополуччя. Як те, що не менш важливе, ніж тигр. Інакше вам доведеться постійно боротися, і ваш мозок зробить усе можливе, аби ухилитися від запам'ятовування нової інформації.



Отже, як переконати ваш мозок сприймати мову Java так само, як і зголоднілого тигра?

Існує спосіб повільний і нудний, а є швидкий та ефективний. Перший ґрунтується на безглуздому затверджуванні. Усім відомо, що навіть найнуднішу інформацію можна запам'ятати, якщо повторювати її знов і знов. За достатньої кількості повторень ваш мозок розмірковує: «*Здається*, ніби це не є суттєвим для нього, але якщо те саме повторюється *знов*, і *знов*, і *знов*... гаразд, хай буде так».

Ефективніший спосіб ґрунтується на ***стимулюванні активності мозку***, переважно на поєднанні різних *видів* мозкової діяльності. Доведено, що всі фактори, перелічені на попередній сторінці, допомагають вашому мозку працювати на вас. Наприклад, дослідження засвідчують, що розміщення слів *усередині* малюнків (а не в підписах чи основному тексті) змушує мозок аналізувати зв'язок між текстом і графікою, а це призводить до активізації значно більшої кількості нейронів. Більше нейронів — вища ймовірність того, що інформація буде визнана важливою і вартою того, аби її запам'ятати.

Розмовний стиль теж важливий: зазвичай люди виявляють більше уваги, коли беруть участь у розмові, оскільки їм доводиться стежити за перебігом бесіди і висловлювати власну думку. Причому мозок абсолютно *не обходить* те, що ви «розмовляєте» лише з книжкою! З іншого боку, якщо текст сухий і формальний, то мозок відчуває те саме, що ви відчуваєте під час нудної лекції в ролі пасивного слухача. Його хилить у сон.

Проте малюнки і розмовний стиль — це лише початок.

Ось що зробили МИ:

Ми використовували ***малюнки***, бо мозок краще пристосований до сприйняття візуальних ефектів, аніж тексту. Із точки зору мозку найпростіший малюнок вартий 1024 слів. А коли текст комбінується із графікою, ми вбудовуємо текст безпосередньо в малюнки, бо мозок при цьому працює ефективніше.

Ми задіяли ***дублювання***: повторюємо те саме кілька разів, застосовуючи *різні* засоби передачі інформації, звертаємося до різних почуттів — і все це для підвищення ймовірності того, що матеріал буде закодований в кількох ділянках вашого мозку.

Ми використовуємо концепції і малюнки у дещо ***несподіваний*** спосіб, бо мозок краще сприймає нову інформацію, і використовуємо фотографії, оскільки вони зазвичай мають ***емоційний*** *зміст*, бо мозок звертає увагу на біохімію емоцій. Те, що змушує нас *відчувати*, краще запам'ятовується, навіть якщо це невеличкий ***жарт***, ***здивування*** або ***інтерес***.

Ми використовуємо персоніфікований ***розмовний стиль***, бо мозок краще сприймає інформацію, коли ви берете участь у розмові, а не пасивно слухаєте лекцію. Це відбувається і під час *читання*.

Книжка містить понад 80 ***вправ***, бо мозок краще запам'ятовує, коли ви щось ***робите***, а не просто читаєте. Ми постаралися зробити їх непростими, але цікавими, бо такому підходу надає перевагу більшість *читачів*.

Ми ***поєднали кілька стилів навчання***, бо одні читачі надають перевагу покроковим описам, у той час як інші прагнуть спочатку уявити «загальну картину», а третім вистачає фрагментів коду. Проте незалежно від ваших особистих уподобань завжди корисно мати справу із кількома варіантами подання того самого матеріалу.

Ми постаралися задіяти ***обидві півкулі вашого мозку***, бо що більше ділянок мозку задіяно, то вища ймовірність засвоєння матеріалу і зосередженості на навчанні. Поки одна половина мозку працює, інша часто має можливість відпочити; це підвищує ефективність навчання протягом тривалого часу.

А ще книжка містить ***історії*** та вправи, що відтворюють ***інші точки зору***, оскільки мозок якнайше засвоює інформацію, коли йому доводиться оцінювати і робити припущення.

У книжці деінде зустрічаються ***завдання*** і ***питання***, на які не завжди можна дати просту відповідь, бо мозок швидше вчиться і запам'ятовує, коли йому доводиться щось *робити*. Судіть самі, неможливо накачати *м'язи*, *спостерігаючи* за тим, як займаються інші. Однак ми подбали про те, щоби зусилля читачів були спрямовані у правильному напрямку. ***Вам не доведеться ламати голову*** над незрозумілими прикладами або розбиратися у складному, перенасиченому технічним жаргоном або занадто лаконічному тексті.

В історіях, прикладах, картинках «оживають» ***люди***. Адже і ви людина. І ваш мозок більше уваги приділяє *людям*, а не *речам*.

Виріжте це та прилаштуйте на холодильнику.

Ось що ВИ можете зробити, аби змусити свій мозок підкорятися

Отже, ми свою справу зробили. Решта за вами. Ці поради стануть відправною точкою; прислухайтеся до свого мозку й визначте, що вам підходить, а що не підходить. Спробуйте нове.

❶ Не поспішайте. Що більше ви зрозумієте, то менше доведеться запам'ятовувати.
Просто *читати* недостатньо. Коли книжка ставить вам запитання, не переходьте одразу до відповіді. Уявіть, що хтось дійсно ставить вам запитання. Що глибше ваш мозок буде думати, то швидше ви зрозумієте і запам'ятаєте матеріал.

❷ Виконуйте вправи. Робіть власні нотатки.
Ми включили вправи до книги, але виконувати їх за вас не збираємося. І не просто *розглядайте* вправи. **Беріть олівець** і пишіть. Фізична активність *під час* навчання підвищує його ефективність.

❸ Обов'язково читайте рубрику «Жодного безглуздого питання».
Це означає: читайте все. Врізки — це **частина основного матеріалу**! Не пропускайте їх.

❹ Не читайте інші книжки після цієї перед сном. Або нехай ця буде останньою складною річчю на сьогодні.
Частина навчання (особливо перенесення інформації в довгостроковий пам'ять) відбувається *після* того, як ви відкладаєте книжку. Ваш мозок не одразу засвоює інформацію. Якщо під час обробки надійде нова інформація, частина того, що ви дізналися раніше, може бути втрачена.

❺ Промовляйте прочитане. Уголос.
Мова активізує інші ділянки мозку. Якщо ви намагаєтеся щось зрозуміти або краще запам'ятати, промовте вголос. А ще краще спробуйте пояснити це комусь іншому. Ви будете швидше [illegible], відкриєте для себе щось нове.

❻ Пийте воду. І якомога більше.
Мозок найкраще працює в умовах високої вологості. Дегідратація (яка може статися ще до того, як ви відчуєте спрагу) знижує когнітивні здатності.

❼ Прислухайтеся до свого мозку.
Слідкуйте за тим, коли ваш мозок починає втомлюватися. Якщо ви починаєте поверхнево сприймати матеріал або забуваєте щойно прочитане — настав час зробити перерву. Якщо ви цього не зробите, може настати мить, коли ви не зможете пришвидшитись у навчанні, намагаючись засвоїти більше, а навпаки — лише зашкодите процесу.

❽ Нехай це стане реальністю!
Ваш мозок повинен знати, що матеріал книжки дійсно *важливий*. Переймайтеся перебігом наших історій. Вигадуйте власні підписи до фотографій. Покепкуйте над невдалим жартом *усе одно* краще за відсутність будь-яких почуттів.

❾ Пишіть багато коду!
Оволодіти мовою Python можна лише в один спосіб: **писати багато коду на Python.** Саме цим вам і належить займатися. І це те, що ви маєте робити впродовж читання цієї книжки. Програмування — це навичка, і єдиний спосіб розвинути цю навичку — практика. Ми запропонуємо вам чимало практичних завдань: у кожному розділі є вправи, до яких треба знайти рішення. Не пропускайте їх: істотна частина навчання відбувається саме тоді, коли ви виконуєте вправи. Ми наводимо розв'язання до кожного завдання — **не бійтеся зазирнути туди**, якщо застряли! (Дуже легко застряти через якусь дрібницю.) Але все ж таки спробуйте розв'язати завдання самостійно і лише тоді звертайтеся до відповіді. І, звісно, перевірте його на практиці, перш ніж продовжите читати.

Прочитай мене (1)

Це навчальний практикум, посібник, а не довідник. Ми свідомо прибрали все, що може зашкодити оволодінню матеріалом. Приступаючи до навчання вперше, читайте все від самого початку, оскільки ми вже врахували все, що ви могли бачити і знати раніше.

Ця книжка створена для того, щоби ви якнайшвидше оволоділи матеріалом.

Ми навчимо вас лише тому, що необхідно знати. Тому ви не знайдете тут ані довгих переліків технічних подробиць, ані таблиць операторів Python, ані правил послідовності операторів. Ми не охопили *гетьусе*, але наполегливо попрацювали, щоби охопити основний матеріал якнайкраще, і тепер ви зможете *швидко* «завантажити» Python у свій мозок і втримати його там. Єдине припущення, яке ми робимо,— це те, що ви вже знаєте принципи кодування якоюсь іншою мовою програмування.

Ця книжка орієнтована на Python 3.

У цій книжці ми використовуємо версію 3 мови програмування Python. Далі ми розглянемо, як отримати і встановити Python 3 (див. Додаток А). У цій книжці **не** використовується Python 2.

Ми одразу починаємо роботу з Python.

Починаючи з Розділу 1 ми навчатимемо вас корисним речам і надалі будемо на них спиратися. Немає сенсу ходити навколо, тому що ми хочемо, аби ви одразу почали продуктивно застосовувати Python.

Практичні завдання НЕ є факультативними — ви повинні виконувати їх.

Вправи та практичні завдання не є додатковими, це частина основного матеріалу книжки. Деякі з них сприяють запам'ятовуванню матеріалу, деякі важливі для розуміння, а деякі допоможуть вам застосовувати те, про що ви дізналися. ***Не оминайте вправи!***

Надмірність у книжці є навмисною і важливою.

Важливою відмінною рисою книжки є наше щире прагнення до того, щоби ви *дійсно* засвоїли матеріал. І ми хочемо, щоби ви, прочитавши її, пам'ятали те, про що дізналися. Більшість довідників не ставлять за мету збереження та утримання інформації, але ця книжка призначена для навчання. І ви побачите, що деякі поняття зустрічаються набагато більше одного разу.

Приклади максимально спрощені.

Наші читачі зазначають, що їх розчаровує, коли доводиться продиратися крізь 200 рядків коду, щоби відшукати два рядки, потрібні для розуміння. Більшість прикладів у цій книжці надані в мінімально можливому контексті, тож та частина, що ви намагаєтеся вивчити, буде зрозумілою і простою. Не очікуйте, що всі приклади виявляться надійними або навіть повними — вони написані спеціально для навчання і не завжди є цілком функціональними (хоча ми намагалися максимально це забезпечити).

Прочитай мене (2)

Так, ще трохи...

Це друге видання, і воно НЕ таке саме, як перше.

Це оновлення першого видання Head First Python, що побачило світ наприкінці 2010 року. Незважаючи на те що автор той самий, тепер він став старшим і (сподіваюся) мудрішим, тому було вирішено цілковито переробити матеріал першого видання. Отже, тут усе нове: послідовність викладу відрізняється, зміст оновлено, приклади покращені, історії або зникли зовсім, або змінилися. Ми зберегли обкладинку з незначними виправленнями, бо вирішили не дуже розхитувати човен. Минуло чимало років... і ми сподіваємося, що вам сподобається те, що ми зробили.

Де код?

Ми розмістили приклади коду на вебсайтах, аби ви могли копіювати і вставляти їх за необхідності (хоча ми рекомендуємо вводити код вручну під час освоєння матеріалу). Ви можете знайти потрібний код за такими посиланнями:

http://bit.ly/head-first-python-2e
http://python.itcarlow.ie