Розділ 1

Чому варто опанувати програмування?

Написання програм (або програмування) – дуже творча та загалом корисна справа. Існує безліч причин для програмування, як от заробіток та розв'язок складних задач, пов'язаних з аналізом даних, чи то щоб допомогти комусь, або ж просто задля розваги. Ця книга побудована на припущенні, що кожному варто вміти програмувати, а вже після опанування нових навичок, ви зрозумієте, де саме хочете їх застосувати.

Наше повсякденне життя переповнене різними комп'ютерами, від ноутбуків до звичайних мобільних телефонів, яких ми вважаємо нашими «особистими помічниками», що можуть виконати купу різних справ за нашим дорученням. Власне, обладнання сучасних комп'ютерів так і створено аби постійно запитувати нас: «Що робимо далі?».

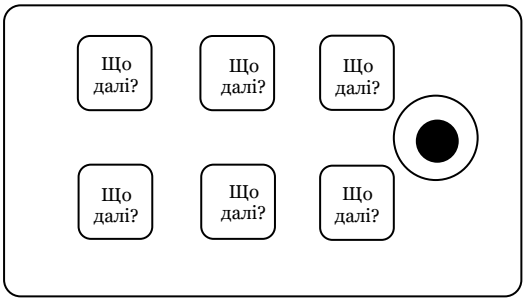

Рисунок 1.1. Кишеньковий комп'ютер

До пристрою програмісти додають операційну систему та ряд застосунків, і в результаті ми отримуємо кишеньковий комп'ютер, який здатний допомогти нам з різними завданнями.

Наші комп'ютери швидкі та мають величезні обсяги пам'яті, вони б нам дуже допомогли, якби ми тільки знали мову, якою можна пояснити, що ми хочемо, аби вони «зробили далі». Якби ми знали цю мову, ми могли б доручити їм виконувати однотипні завдання за нас. Досить цікаво, що комп'ютери можуть робити найкраще те, що люди зазвичай вважають надокучливим та нудним.

От, наприклад, прогляньте перші три абзаци цього розділу і скажіть, яке слово вживається найчастіше та скільки разів воно зустрічається. Попри те, що прочитати та зрозуміти слова всього за кілька секунд вам не склало жодних зусиль, рахувати їх – просто нестерпно, адже людський розум не призначений для виконання таких завдань. Для комп'ютера все навпаки: прочитати та зрозуміти сенс тексту йому важко, а от порахувати кількість слів і визначити найуживаніше з них – легко:

```
python words.py
Enter file:words.txt
що 6
```

Наш «помічник з аналізу персональної інформації» швидко підказав нам, що у перших трьох абзацах цієї глави слово «що» зустрічається шість разів.

Власне через те, що комп'ютери вправні в тому, що погано вдається людям, варто навчитися розмовляти «комп'ютерною мовою». Щойно ви опануєте цю нову мову, ви зможете доручити рутинні завдання своєму напарнику (комп'ютеру) та приділити більше часу і зосередитись на справах, до яких ви маєте унікальний хист. У цій взаємодії, ви – творчість, інтуїція та винахідливість.

1.1 Творчість та мотивація

Хоч ця книга і не призначена для фахівців, професійне програмування може бути вельми хорошою роботою як у фінансовому, так і в особистому плані. Створення корисних, ефективних і продуманих програм для людей – надзвичайно творча справа. Ваш комп'ютер або смартфон зазвичай містять велику кількість різноманітних програм від великої кількості різноманітних груп програмістів, кожна з яких бореться за вашу увагу та зацікавленість. Вони намагаються зробити все можливе, аби задовольнити ваші потреби та надати чудовий досвід користувача в процесі роботи. У деяких випадках, програмісти безпосередньо отримують винагороду за вибір їхнього програмного забезпечення.

Якщо ми розглядаємо програми як результат творчої роботи груп спеціалістів, то, можливо, наступний рисунок нашого смартфона вам буде більш зрозумілим:

Рисунок 1.2. Звернення програмістів

Наразі наша основна мотивація полягає не в заробітку і не в задоволенні кінцевого користувача, а в тому, щоб стати більш продуктивними в роботі з даними та інформацією, з якими ми стикаємося в житті. Спочатку ви будете одночасно і програмістом, і кінцевим користувачем ваших програм. Згодом, набувши більше навичок і відчувши, що програмування стає більш творчим процесом, ви зможете замислитися над розробкою програм для інших.

1.2 Архітектура апаратного забезпечення комп'ютера

Перш ніж почати вивчати мову програмування, потрібно розібратися з будовою комп'ютера. Всередині вашого комп'ютера чи телефона ви можете побачити таке:

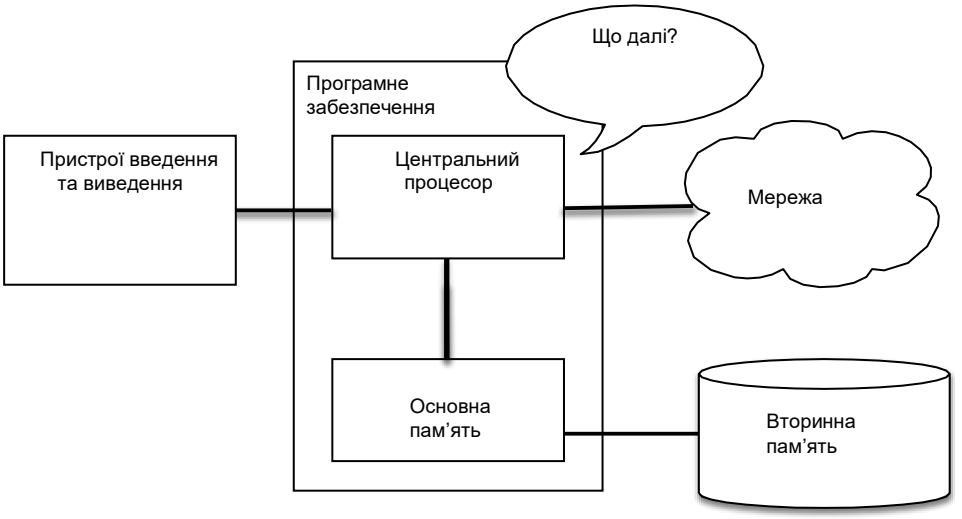

Рисунок 1.3. Архітектура апаратного забезпечення

Нижче наведені визначення до кожного елементу:

- *Центральний процесор* (*ЦП*) – це компонент комп'ютера, який створений для того, щоб бути одержимим питанням «Що далі?». Якщо тактова частота комп'ютера становить 3,0 гігагерца, це означає, що ЦП запитуватиме «Що далі?» три мільярди разів на секунду. Аби не відставати від процесора, вам доведеться натренуватися дуже швидко говорити.
- *Основна пам'ять* призначена для зберігання інформації для негайного використання центральним процесором. Вона працює майже так само швидко, як і ЦП. Проте інформація, що зберігається в основній пам'яті, зникає, щойно комп'ютер вимикається.
- *Вторинна пам'ять* теж використовується для зберігання інформації, але вона значно повільніша за основну. Перевага вторинної пам'яті полягає в можливості зберігати інформацію навіть за відсутності живлення комп'ютера. Це наші дискові накопичувачі та флешпам'ять (зазвичай використовується в USB-накопичувачах і портативних музичних програвачах).
- *Пристрої введення та виведення* – це просто наші екран, клавіатура, комп'ютерна миша, мікрофон, динамік, сенсорна панель тощо. Це всі засоби, за допомогою яких ми взаємодіємо з комп'ютером.
- Нині, більшість комп'ютерів також мають *мережеве підключення* для отримання інформації через мережу. Мережу можна уявити як дуже повільне місце для зберігання та отримання даних, яке не завжди буває доступним. Тож у певному сенсі мережа є повільнішою і часом ненадійною формою *вторинної пам'яті*.

Попри те, що більшість подробиць щодо роботи цих компонентів краще залишити виробникам комп'ютерів, доцільно володіти певною термінологією, завдяки якій ми зможемо обговорювати ці елементи під час написання наших програм.

Ваше завдання, як програміста, полягає у використанні та впорядкуванні кожного з цих ресурсів для розв'язання задач та аналізу даних, які ви отримуєте в процесі. Ви будете «розмовляти» переважно з центральним процесором, повідомляючи йому наступні дії. Іноді йому потрібно буде вказати скористатися оперативною пам'яттю, вторинною пам'яттю, мережею або пристроями введення/виведення.

Рисунок 1.4. Де знаходитесь ви?

Ви маєте бути людиною, яка відповідає на запитання процесора «Що далі?». Але було б дуже незручно зменшити вас до 5 мм на зріст і помістити в комп'ютер лише для того, щоб ви могли посилати команди три мільярди разів на секунду. Тому замість цього потрібно заздалегідь записати свої інструкції. Ми називаємо ці збережені інструкції *програмою*, а процес запису цих інструкцій і перетворення їх на правильні – *програмуванням*.

1.3 Знайомство з програмуванням

У наступних розділах цієї книги ми спробуємо перетворити вас на людину, яка володіє мистецтвом програмування. Зрештою, ви станете *програмістом* – можливо, не професійним, але принаймні матимете навички, необхідні для розуміння задач аналізу даних / інформації та розробки програми для їх розв'язання.

Таким чином, щоб стати програмістом, потрібно опанувати дві навички:

- По-перше, ви повинні вивчити мову програмування (Python) – її лексику та граматику. Варто вміти грамотно писати слова цією мовою і вміти будувати правильно сформовані «речення».
- По-друге, потрібно «розповісти історію». Під час написання історії ви поєднуєте слова та речення, щоб донести ідею до читача. Складання історії – це майстерність і мистецтво, а майстерність у написанні історій покращується, коли ви пишете і маєте певний зворотній зв'язок. У програмуванні наша програма – це «історія», а проблема, яку ви намагаєтесь вирішити, – це «ідея».

Після вивчення однієї мови програмування, такої як Python, вам буде набагато легше вивчити другу мову програмування, як от JavaScript чи C++. Інша мова програмування відрізняється лексикою та граматикою, але навички розв'язання задач будуть однаковими всюди.

Ви досить швидко вивчите «словниковий запас» та «речення» мови Python. Знадобиться більше часу саме для того, аби ви змогли написати цілісну програму для розв'язання абсолютно нової задачі. Програмування вивчається так само як і письмо. Починаємо з читання та пояснення програм, потім пишемо прості програми, а з часом переходимо до дедалі складніших. У якийсь момент вас «осяє розуміння», і ви самостійно почнете бачити закономірності, стане зрозуміліше, як сприймати задачу та писати програму, яка розв'яже її. Щойно досягаєте цього моменту, програмування перетворюється на дуже приємний і творчий процес.

Ми почнемо з вивчення лексики та структури програм мовою Python. Наберіться терпіння, адже нескладні приклади нагадають вам, як це було, коли ви вперше взяли до рук книгу і почали читати.

1.4 Слова та речення

На відміну від людської мови, словниковий запас Python насправді досить невеликий. Цей «словник» називають «зарезервованими словами». Коли Python бачить ці слова, вони мають для нього певне значення. Пізніше, коли ви будете писати програми, ви будете вигадувати власні слова, які матимуть своє значення і називатимуться *змінними*. Ви зможете використати усю свою фантазію у виборі імен для змінних, але не зможете скористатися жодним із зарезервованих слів Python для цього.

Під час муштрування собаки, ми користуємося певними словами, такими як «сидіти», «лежати» та «принеси». Проте, коли ви розмовляєте з собакою і не використовуєте жодного із цих зарезервованих слів, вона просто допитливо дивиться на вас, поки ви знову не вимовите одне з них. Наприклад, якщо ви скажете: «От якби люди частіше ходили гуляти, аби покращити здоров'я», більшість собак, швидше за все, почують: «Бла-бла-бла, *гуляти*, бла-бла-бла». Річ у тім, що слово «гуляти» у собачій мові є зарезервованим. Чимало людей вважають, що у мові котів і людей немає зарезервованих слів[1].

До зарезервованих слів у мові, якою люди спілкуються з Python, належать:

```
and       del       global    not       with
as        elif      if        or        yield
assert    else      import    pass
break     except    in        raise
class     finally   is        return
continue  for       lambda    try
def       from      nonlocal  while
```

От і все. І на відміну від собаки, Python вже повністю натренований. Коли ви скажете «try», Python буде виконувати цю команду щоразу, без помилок.

Ми згодом вивчимо ці зарезервовані слова і те, як вони використовуються, а поки що зосередимося на еквіваленті слова «голос» в Python (мовою спілкування людини з собакою). Що особливо приємно, ми можемо навіть вказати Python, що саме сказати, взявши повідомлення в лапки:

[1] http://xkcd.com/231/

```
print('Привіт  світе!')
```

І ми навіть написали наше перше синтаксично правильне речення мовою Python. Речення починається з функції *print*, за якою йде рядок тексту будь-якого змісту, взятий в одинарні лапки. Рядки у виразах print потрібно брати в лапки. Одинарні та подвійні лапки виконують однакову функцію; більшість людей використовують одинарні, за винятком випадків, коли в рядку з'являються одинарні лапки (яка також є апострофом).

1.5 Спілкування з Python

Тепер, після того, як ми засвоїли слово і просте речення, нам потрібно дізнатися, як почати розмову за допомогою Python, щоб перевірити наші набуті мовні навички.

Перш ніж розпочати спілкування, необхідно встановити програмне забезпечення Python на комп'ютер і навчитися його запускати. Ця тема занадто велика, аби вмістити усі деталі в одному розділі, тому раджу переглянути www.py4e.com, де є детальні інструкції та знімки екрана щодо налаштування та запуску Python на системах Macintosh та Windows. На певному етапі ви відкриєте термінал або вікно командного рядка і введете *python,* після чого інтерпретатор Python почне працювати в інтерактивному режимі, це виглядатиме приблизно так:

```
Python 3.5.1 (v3.5.1:37a07cee5969, Dec. 6  2015,  01:54:25)
[MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Команда >>> – це спосіб, яким інтерпретатор Python запитує: «Що робимо далі?». Python готовий поспілкуватися з вами. Все, що вам потрібно – це вміти розмовляти мовою Python.

Припустимо, що ви не знаєте навіть найпростіших слів або речень мови Python. Можливо, вам захочеться скористатися стандартною фразою, яку говорять астронавти, коли потрапляють на далеку планету і намагаються заговорити з її мешканцями:

```
>>>I come in peace, please take me to your leader File
"<stdin>", line 1
  I come in peace, please take me to your leader
       ^
SyntaxError: invalid syntax
>>>
```

Здається, знайомство не складається. Потрібно щось швидко вигадати, інакше для вас все закінчиться списом біля горла і вечерею інопланетян, де головною стравою станете ви.

На щастя, у вас знайшовся примірник цієї книги, який ви взяли з собою в подорож, тож ви гортаєте до цієї сторінки і пробуєте ще раз:

```
>>> print('Привіт світе!')
Привіт світе!
```

Виглядає набагато краще, тож ви намагаєтесь продовжити спілкування:

```
>>> print('Ви, мабуть, той легендарний Бог, що спустився з небес')
Ви, мабуть, той легендарний Бог, що спустився з небес
>>> print('Ми дуже довго чекали на Вас')
Ми дуже довго чекали на Вас
>>> print('У легендах оспівується, що Вас краще смакувати з
гірчицею')
У легендах оспівується, що Вас краще смакувати з гірчицею
>>> print 'Ми влаштуємо банкет ввечері, якщо ти не
скажеш
File "<stdin>", line 1
    print 'Ми влаштуємо банкет ввечері, якщо ти не скажеш
                                                        ^
SyntaxError: Missing parentheses in call to 'print'
>>>
```

Деякий час розмова йшла досить добре, а потім ви припустилися крихітної помилки, і в результаті Python знову почав нагострювати списи.

На цьому етапі ви також повинні усвідомити, що попри те, що Python є напрочуд складною та потужною мовою і дуже вибагливою до синтаксису, який ви використовуєте під час спілкування, ця мова все-таки *не є* розумною. Ви насправді розмовляєте самі з собою, лиш використовуєте потрібні вирази.

У певному сенсі, коли ви використовуєте програму, створену кимось іншим, розмова відбувається між вами та цими програмістами, а Python виступає в ролі посередника. Python – це спосіб творців програм висловити своє бачення того, як має відбуватися ця розмова. Всього через кілька розділів ви станете одним з тих програмістів, які використовують Python для спілкування з користувачами вашої програми.

Перш ніж ми завершимо нашу першу розмову з інтерпретатором Python, вам, мабуть, варто знати, як правильно прощатися при спілкуванні з мешканцями Планети Python:

```
>>> good-bye
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'good' is not defined
>>> if you don't mind, I need to leave
File "<stdin>", line 1
  if you don't mind, I need to leave
          ^
SyntaxError: invalid syntax
>>> quit()
```

Ви можете зауважити, що ця помилка не схожа на перші дві. Вона відрізняється тим, що *if* є зарезервованим словом, і Python сприйняв його саме в цьому значенні та вирішив, що ми намагаємося щось сказати, проте неправильно побудували речення.

Правильно попрощатися з Python можна за допомогою команди *quit()*, яку потрібно ввести після >>> запиту. Ймовірно, вам знадобилося б чимало часу, аби здогадатися про це, тому досить корисно мати під рукою цю книгу.

1.6 Термінологія: Інтерпретатор та компілятор

Python – це *високорівнева* мова, призначена бути відносно зрозумілою людині у читанні та письмі й комп'ютеру у читанні та обробці. Серед інших таких мов виділяють Java, C++, PHP, Ruby, Basic, Perl, JavaScript тощо. Обладнання всередині центрального процесора (ЦП) не розуміє жодну з них.

Проте, процесор розуміє мову, яку ми називаємо *машинною*. Машинна мова дуже проста, і, відверто кажучи, писати її досить виснажливо, оскільки вона складається лише з нулів та одиниць:

```
00101000111010010010101010000001111
11100110000011101010010101101101
...
```

Хоч, на перший погляд, так не здається, проте синтаксис машинної мови ще складніший і набагато заплутаніший, ніж у мові Python. Саме тому дуже мало програмістів пишуть машинною мовою. Натомість ми створюємо різноманітні транслятори, які дозволяють писати високорівневими мовами, такими як Python або JavaScript, які перекладають програми на машинну мову для їх подальшого виконання процесором.

Оскільки машинна мова повністю інтегрована в апаратне забезпечення комп'ютера, її не можна перенести на інші види обладнання, тобто вона не *портативна*. А от програми, написані високорівневими мовами, можна переносити між різними комп'ютерами, використовуючи інший інтерпретатор, або шляхом перекомпіляції коду задля створення програми машинною мовою для іншого комп'ютера.

Транслятори мов програмування поділяються на: (1) інтерпретатори та (2) компілятори.

Інтерпретатор зчитує вихідний код програми, написаний програмістом, здійснює синтаксичний аналіз та на ходу інтерпретує інструкції. Python – інтерпретатор, і коли ми запускаємо його в інтерактивному режимі, ми можемо набрати рядок мовою Python (речення), а він негайно обробить його і буде готовий до введення наступного рядка.

Деякі рядки вказують Python, що ви хочете, аби він запам'ятав певні значення для подальшого використання. Потрібно вибрати ім'я для цього значення, щоб запам'ятати його і мати можливість пізніше використати його для отримання цього ж значення. Для позначення міток, які ми використовуємо для посилання на ці збережені дані, застосовується термін *змінна*.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
```

```
>>>print(y)
42
>>>
```

У цьому прикладі ми просимо Python запам'ятати значення 6 і використати мітку *x*, щоб пізніше отримати це значення. Перевіряємо, чи Python дійсно запам'ятав значення за допомогою *print*. Потім просимо Python отримати *x*, помножити його на 7 і зберегти нове значення як *y*. Після цього просимо Python вивести значення, яке позначає *y*.

Попри те, що ми вводимо ці команди по одному рядку, Python розглядає їх як впорядковану послідовність команд, в якій наступні команди можуть отримувати дані, створені в попередніх. Пишемо наш перший простий блок з чотирьох речень, розташованих у логічному та змістовному порядку.

Наведений вище приклад зображує сутність *інтерпретатора,* а саме ведення такої інтерактивної розмови. Своєю чергою, *компілятору* потрібно надати всю програму у файлі, після чого він відтворює процес перетворення високорівневого вихідного коду на машинну мову, а потім поміщає її у файл для подальшого виконання.

У системі Windows ці програми машинною мовою найчастіше мають суфікс «.exe» або «.dll», що означає "executable" і "dynamic link library" відповідно. У Linux та Macintosh немає суфікса, який би позначав виконуваний файл.

Якщо спробувати відкрити виконуваний файл у текстовому редакторі, він буде виглядати безглуздо і нечитабельно:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

Читати чи писати машинною мовою непросто, тож тішить існування *інтерпретаторів* та *компіляторів*, які дозволяють нам писати високорівневими мовами, такими як Python чи C.

Тепер, на цьому етапі нашої розмови про компілятори та інтерпретатори, вам напевно цікаво дізнатися більше про інтерпретатор Python. Якою мовою він написаний? Чи ця мова скомпільована? Що саме станеться, якщо ввести «python»?

Інтерпретатор Python написаний високорівневою мовою «C». Ви можете ознайомитися з вихідним кодом інтерпретатора Python за посиланням www.python.org. Отже, Python – це програма, яка компілюється у машинний код. Після встановлення Python на ваш комп'ютер, ви скопіювали машинний код перекладеної програми Python у вашу систему. У Windows виконуваний машинний код програми Python, найімовірніше, знаходиться у файлі з іменем на зразок:

```
C:\Python35\python.exe
```

Це більше, ніж вам дійсно потрібно, аби стати програмістом Python, але не буде зайвим знати відповіді на ці незначні надокучливі питання ще на початку.

1.7 Написання програми

Введення команд в інтерпретатор Python – чудовий спосіб поекспериментувати з його можливостями, проте для вирішення більш складних завдань він не підходить.

Щоб написати програму, ми використовуємо текстовий редактор для запису інструкцій Python у файл, який називається *скрипт*. За замовчуванням, скрипти Python мають назви, що закінчуються на .py.

Аби виконати скрипт, інтерпретатору Python необхідно вказати ім'я файлу. У вікні командного рядка можна ввести python hello.py наступним чином:

```
$ cat hello.py
print('Привіт світе!')
$ python hello.py
Привіт світе!
```

«$» – підказка операційної системи, а «cat hello.py» показує, що у файлі «hello.py» є однорядкова програма мовою Python.

Замість того, щоб вводити рядки коду Python в інтерактивному режимі, ми викликаємо інтерпретатор Python і просимо його прочитати вихідний код з файлу «hello.py».

Можна помітити, що в кінці програми Python у файлі не потрібно писати команду *quit()*. Коли Python читає ваш вихідний код з файлу, він знає, що має зупинитися, коли дійде до кінця файлу.

1.8 Що таке програма?

Простими словами, *програма* – це послідовність команд мовою Python, яка створена для виконання певних дій. Навіть наш простий скрипт *hello.py* є програмою. Хоч вона і не дуже корисна, проте підпадає під визначення і є однорядковою програмою, написаною мовою Python.

Можливо, простіше зрозуміти значення програми, якщо подумати про задачу, для розв'язку якої вона може бути створена.

Уявімо, що ви проводите дослідження у сфері соціальних обчислень у Фейсбуці та вас цікавить слово, яке найчастіше вживається у ряді дописів. Ви можете видрукувати ланцюжок дописів у Фейсбуці та переглянути текст у пошуках найпоширенішого слова, але це займе багато часу і підвищить ймовірність похибок. Натомість краще написати програму в Python, яка швидко і точно впорається з цим завданням, а ви зможете провести вихідні, займаючись власними справами.

Наприклад, перегляньте наведений нижче текст про клоуна та автомобіль. З'ясуйте, яке слово є найбільш уживаним і скільки разів воно зустрічається.

Побіг клоун за машиною, клоун її майже наздогнав, машина
в'їхала в намет, клоун був поруч, і намет впав на клоуна і машину

А тепер уявіть, що вам потрібно проглянути мільйони рядків тексту, щоб виконати це завдання. Чесно кажучи, буде швидше вивчити Python і написати програму для підрахунку слів, аніж шукати їх вручну.

Маю гарну новину, я вже придумав просту програму для пошуку найпоширенішого слова в текстовому файлі. Я написав її, протестував, а тепер ділюся нею, щоб ви могли заощадити власний час.

```
name = input('Enter file:')
handle = open(name, 'r')
counts = dict()

for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in list(counts.items()):
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)

# Code: http://www.py4e.com/code3/words.py
```

Вам навіть не потрібно володіти Python, щоб користуватися цією програмою. Необхідно прочитати розділ 10 цієї книги, щоб повністю зрозуміти неймовірні методи Python, які були використані при створенні цієї програми. Ви – кінцевий користувач, який просто працює з програмою та захоплюється її продуманістю і тим, як вона заощадила вам стільки зусиль. Достатньо ввести код у файл *words.py* і запустити його, або ж завантажити вихідний код з сайту http://www.py4e.com/code3/ і зробити те ж саме.

Це чудовий приклад того, як Python і мова Python виступають посередником між вами (кінцевим користувачем) і мною (програмістом). Python – це спосіб обміну корисними командами (тобто програмами) спільною мовою, яку може використовувати кожен, хто встановить його на своєму комп'ютері. Тож ніхто з нас не розмовляє *з* Python, натомість ми спілкуємося один з одним *через* Python.

1.9 Структурні елементи програм

У наступних розділах ми дізнаємося більше про словниковий запас, структуру речень, абзаців та історій у Python. Довідаємось про широкі можливості мови Python

і про те, як компонувати їх разом, щоб створити необхідні програми.

Існує кілька низькорівневих концептуальних шаблонів, які використовуються для створення програм. Ці шаблони існують не лише для програм мовою Python, вони є частиною кожної мови програмування, від машинної до високорівневої.

input Збір даних із «зовнішнього світу». Це може бути зчитування даних з файлу або навіть з певного датчика, наприклад, мікрофона чи GPS. У наших початкових програмах вхідні дані надходитимуть від користувача, який вводить їх на клавіатурі.

output Результати роботи програми виводяться на екран або зберігаються у файлі, чи, скажімо, записуються на пристрій, наприклад, на динамік, щоб відтворювати музику або озвучувати текст.

sequential execution Послідовне виконання команд у тому порядку, в якому вони вказані у скрипті.

conditional execution Перевірка певних умов, а потім виконання або пропуск послідовності команд.

repeated execution Багаторазове виконання конкретного набору команд, як правило, з певними змінами.

reuse Разовий запис набору інструкцій, присвоєння назви, і подальше їх використання у програмі за необхідності.

Звучить занадто просто, аби бути правдою, і саме тому це не так. Це все одно, що сказати, що ходити – це просто «послідовно переставляти ноги». «Мистецтво» написання програми полягає в поєднанні цих базових складників, аби створити щось справді корисне для її користувачів.

Наведена вище програма підрахунку слів безпосередньо використовує всі ці шаблони, окрім одного.

1.10 Що може піти не так?

Як видно з наших перших розмов з Python, під час написання коду, потрібно висловлюватися чітко і ясно. Найменше незначне відхилення чи помилка – і Python вже не працює з вашою програмою.

Програмісти-початківці часто вважають Python злим і жорстоким через те, що він такий суворий щодо помилок. Сприймають все це дуже особисто, от саме їх Python недолюблює! Через це він відхиляє їхні ідеально написані програми, аби просто познущатися.

```
>>> primt 'Hello world!'
File "<stdin>", line 1
  primt 'Hello world!'
        ^
SyntaxError: invalid syntax
>>> primt ('Hello world')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'primt' is not defined
```

```
>>> I hate you Python!
File "<stdin>", line 1
 I hate you Python!
       ^
SyntaxError: invalid syntax
>>> if you come out of there, I would teach you a lesson
File "<stdin>", line 1
  if you come out of there, I would teach you a lesson
           ^
SyntaxError: invalid syntax
>>>
```

Суперечки з Python навряд чи принесуть користь. Адже це всього лише інструмент. У нього немає емоцій, він готовий допомогти вам у будь-який момент. Хоч його повідомлення про помилки здаються різкими, насправді вони лиш просять допомогти йому. Python переглянув введені вами дані і просто не може зрозуміти, що саме ви ввели.

Python більше схожий на собаку, який безумовно любить вас, розуміє кілька ключових слів, дивиться на вас з милим виразом обличчя (>>>) і чекає, поки Ви скажете щось, що він розуміє. Коли Python видає «SyntaxError: invalid syntax», це щось на кшталт того, як він виляє хвостом і каже: «Здається, ви щось сказали, але я не розумію, що саме, проте, будь ласка, продовжуйте говорити зі мною (>>>)».

Щоразу, як ваші програми ставатимуть дедалі складнішими, ви стикатиметеся з трьома загальними типами помилок:

Syntax errors Це вид помилок, з якими ви зіткнетесь в першу чергу, проте їх дуже легко виправити. Syntax error вказує на порушення «граматики» Python. Python робить все можливе, аби вказати на рядок і символ, де була виявлена помилка. Складність полягає в тому, що іноді він неправильно *визначає* місце помилки, і вона знаходиться десь раніше в програмі. Тому рядок і символ, які Python вказує у syntax error, можуть бути лише початком вашого пошуку.

Logic errors Це помилка, спричинена неправильним порядком команд чи зв'язком між ними. Наведемо приклад: «Випийте води, покладіть пляшку в рюкзак, дійдіть до бібліотеки, а потім закрийте пляшку».

Semantic errors Це повідомлення означає, що помилка знаходиться саме в роботі програми. Тобто, програма написана абсолютно правильно, але вона видає не те що ви від неї *очікували*. Це може бути схоже на ситуацію, коли ви описуєте другу дорогу до ресторану і говорите: «Як доїдеш до перехрестя з заправкою, поверни ліворуч і рухайся ще кілометр, рестораном буде червона будівля зліва від тебе». Ваш друг дуже запізнюється і дзвонить вам, аби сказати, що він знаходиться на фермі, де замість ресторану він бачить лише сарай. Ви запитуєте: «На заправці ти повернув ліворуч чи праворуч?», а він відповідає: «Я чітко слідував твоїм вказівкам, у мене вони записані, там сказано повернути ліворуч і проїхати кілометр до заправки». Все, що ви можете йому відповісти, це лише: «Мені дуже шкода, хоч мої поради і були правильно сформульовані синтаксично, та, на жаль, містили непомітну семантичну помилку».

Знову ж таки, усі три типи помилок є свідченням того, що Python просто намагається зробити саме те, що ви попросили.

1.11 Налагодження програми

Коли Python видає помилку чи результат, який відрізняється від того, що ви очікували, починається полювання на причину помилки. Налагодження – це процес пошуку причини помилки у вашому коді. Під час налагодження програми, а особливо якщо ви працюєте над складним багом, варто спробувати зробити це:

перевірити Перегляньте свій код, прочитайте його про себе і перевірте, чи він відповідає тому, що ви хотіли донести.

переробити Поекспериментуйте, внесіть зміни та запустіть різні версії. Часто, змінивши порядок розташування елементів у програмі, проблема стає очевидною, але іноді доводиться витратити деякий час, щоб побудувати риштування.

переміркувати Не спішіть, подумайте трішки! Який це тип помилки? Яку інформацію ви можете отримати з повідомлення про помилку або з результатів роботи програми? Що могло спричинити виникнення проблеми? Що ви змінювали востаннє перед тим, як з'явилася проблема?

повернутись Іноді найкраще відступити, скасовуючи нещодавні зміни до тих пір, доки не повернетесь до моменту, коли програма працює, де ви все розумієте. Після цього можна розпочати все заново.

Програмісти-початківці іноді зациклюються на чомусь одному і забувають про інші способи. Щоб знайти складний баг потрібно все ретельно перевіряти, переробляти, перемірковувати, а іноді навіть повернутися назад. Якщо ви застрягли на одному з цих етапів, спробуйте інші. Кожен з них має свій власний алгоритм невдач.

Наприклад, перевірка коду може допомогти, якщо проблема полягає в друкарській помилці, але навряд буде ефективною, якщо йдеться про помилку в змісті. Якщо ви не розумієте, що робить ваша програма, ви можете переглянути її хоч 100 разів, а так і не побачите помилку, адже вона у вас в голові.

Переробити та внести зміни особливо допомагає, якщо ви проведете невеличке просте випробування вашої програми. Однак, якщо робити це бездумно, не перечитавши код, можна потрапити в ситуацію, яку я називаю «блукання в нетрях програми», тобто процес внесення випадкових змін доти, поки програма не буде працювати належним чином. Зрозуміло, що таке блукання може зайняти багато часу.

Вам потрібен час, щоб подумати. Налагодження – це як експериментальна наука. Необхідно мати принаймні одну гіпотезу, в чому полягає проблема. Якщо їх дві чи більше, спробуйте зрозуміти, як можна виключити одну з них.

Також, може допомогти відпочинок. Так само як і розмова. Якщо ви поясните проблему комусь іншому (або навіть собі), то іноді знайдете відповідь ще до того, як закінчите ставити питання.

Але навіть найкращі методи налагодження не спрацюють, якщо помилок занадто багато, або якщо код, який ви намагаєтеся виправити, занадто великий і складний. Іноді найкращим варіантом є повернення назад, спрощення програми, доки ви не повернетесь до моменту, коли все працює.

Програмісти-початківці часто не хочуть повертатися назад, адже їм не хочеться видаляти рядок коду (навіть якщо він неправильний). Щоб стало легше, скопіюйте свою програму в інший файл, перш ніж починати її видаляти. Таким чином ви зможете поступово додавати частини коду назад до програми.

1.12 Шлях навчання

Не лякайтеся, якщо з першого разу буде здаватися, що поняття не дуже вдало поєднуються між собою. Протягом перших кількох років, коли ви вчилися говорити, ви могли видавати лише милі булькатючі звуки, і це нормально. І нічого страшного, якщо вам знадобилося пів року, щоб перейти від простої лексики до найпростіших речень, ще 5-6 років, аби перейти від речень до абзаців, і ще кілька років, щоб самостійно написати цікаву повноцінну коротку розповідь.

Ми хочемо, аби ви вивчили Python якнайшвидше, тому в наступних розділах розглядаємо всі теми одночасно. Та все це схоже на вивчення нової мови, що потребує часу для засвоєння та розуміння, перш ніж цей процес стане для вас природним. Через це може виникати певна плутанина, коли ми повторюємо деякі теми, намагаючись допомогти вам побачити загальну картину, в той час, як описуємо дрібні деталі, що її доповнюють. Попри те, що книга написана послідовно, і якщо ви проходите курс, він буде розвиватися лінійно, не соромтеся бути вкрай непослідовними у вашому підході до матеріалу. Гортайте вперед чи назад, знайдіть свій ненапружений темп. Переглядаючи більш складний матеріал без повного розуміння деталей, ви зможете краще осягнути «для чого?» програмування. Переглядаючи попередні розділи і навіть повторно виконуючи пройдені вправи, ви зрозумієте, що насправді вивчили багато нового, навіть якщо матеріал, на який ви зараз дивитеся, здається дещо неприступним.

Зазвичай, під час вивчення своєї першої мови програмування буває кілька «О, це воно!» моментів, коли ви можете відірватися від довбання каменю молотком і зубилом, відступити на крок і побачити, що дійсно створюєте прекрасну скульптуру.

Якщо щось здається особливо важким, зазвичай немає сенсу не спати всю ніч і витріщатися на нього. Зробіть перерву, подрімайте, перекусіть, поясніть комусь (можна навіть своєму собаці), що саме вас турбує, а потім поверніться до справи зі свіжим поглядом на ситуацію. Запевняю, після вивчення принципів програмування, описаних у книзі, ви озирнетеся назад і побачите, що все це було дуже просто і зрозуміло, і вам лиш знадобилося трохи часу, щоб все це засвоїти.

1.13 Словник

баг Помилка в роботі програми.
центральний процесор Серце комп'ютера. Саме він запускає програмне забезпечення, яке ми пишемо; його також називають «ЦП» або «процесором».
компілювати Одномоментний переклад програми з високорівневої мови на мову низького рівня для підготовки до подальшого виконання.

високорівнева мова Мова програмування, як-от Python, розроблена для зручності сприйняття та запису людиною.
інтерактивний режим Спосіб використання інтерпретатора Python шляхом введення команд і виразів у полі запиту.
інтерпретувати Виконання програми високорівневою мовою, перекладаючи її по одному рядку за раз.
низькорівнева мова Мова програмування, яка розроблена для полегшення виконання комп'ютером; також називається «машинним кодом» або «мовою асемблера».
машинний код Мова найнижчого рівня програмного забезпечення, тобто мова, яка безпосередньо реалізується процесором (ЦП).
основна (оперативна) пам'ять Зберігає програми та дані. При вимкненні живлення оперативна пам'ять втрачає інформацію.
синтаксичний аналіз Дослідження програми та аналіз її синтаксичної структури.
портативність Властивість програми, яка може бути запущена на декількох типах комп'ютерів.
функція print Команда, яка змушує інтерпретатор Python вивести значення на екран.
розв'язок задачі Процес формулювання задачі, пошуку її розв'язку та оформлення результату.
програма Набір інструкцій, який задає обчислення.
підказка Коли програма виводить на екран повідомлення і робить паузу для введення користувачем певних даних у програму.
вторинна пам'ять Зберігає програми та дані і не втрачає інформацію навіть після вимкнення живлення. Зазвичай повільніша за оперативну пам'ять. Прикладами є дискові накопичувачі та флешпам'ять на USB-флешках.
семантика Значення програми.
semantic error Помилка в програмі, яка змушує її робити щось інше, ніж було задумано програмістом.
вихідний код Програма, написана високорівневою мовою.

1.14 Вправи

Вправа 1: Яку функцію виконує вторинна пам'ять у комп'ютері?

a) Виконувати всі обчислення та алгоритм роботи програми
b) Завантажувати вебсторінки в інтернеті
c) Зберігати інформацію протягом тривалого часу, незалежно від живлення
d) Приймати вхідні дані від користувача

Вправа 2: Надайте визначення терміну *програма*.

Вправа 3: Назвіть різницю між компілятором та інтерпретатором?

Вправа 4: Що з наведеного нижче містить «машинний код»?

a) Інтерпретатор Python
b) Клавіатура
c) Вихідний файл Python
d) Текстовий редактор

Вправа 5: Знайдіть помилку у наведеному нижче коді:

```
>>> primt 'Привіт світе!'
File "<stdin>", line 1
  primt 'Привіт світе!'
                ^
SyntaxError: invalid syntax
>>>
```

Вправа 6: Де в комп'ютері зберігається змінна типу «x» після завершення наступного рядка Python?

```
x = 123
```

a) Центральний процесор
b) Оперативна пам’ять
c) Вторинна пам’ять
d) Пристрої введення
e) Пристрої виведення

Вправа 7: Вкажіть, що буде виведено цією програмою:

```
x = 43
x = x - 1
print(x)
```

a) 43
b) 42
c) x + 1
d) Error тому що x = x + 1 неможливо математично

Вправа 8: Поясніть кожне з наступних понять на прикладі людських здібностей: (1) Центральний процесор, (2) Оперативна пам’ять, (3) Вторинна пам’ять, (4) Пристрій введення, і (5) Пристрій виведення. Наприклад, «Який існує відповідник Центрального процесора для людини»?

Вправа 9: Як виправити «Syntax Error»?