Розділ 4

Функції

4.1 Виклик функції

У програмуванні, *функція* – це іменована послідовність інструкцій, що здійснює обчислення. Для визначення функції вказуються ім'я та послідовність інструкцій. Пізніше, функцію можна «викликати» за назвою. Ми вже зустрічались з прикладом *виклику функції*:

```
>>> type(32)
<class 'int'>
```

Ім'я цієї функції – type. Вираз в дужках називається *аргументом* функції, це значення чи змінна, що передається у функцію як вхідні дані. Результатом функції **type** є тип аргументу. 2

Зазвичай кажуть, що функція «отримує» аргумент і «повертає» результат, який називається *повернене значення*.

4.2 Вбудовані функції

У Python передбачена низка важливих вбудованих функцій, якими можна користуватися без визначення функції. Розробники Python написали набір функцій для виконання поширених завдань і внесли їх у Python для подальшого використання користувачами.

Функції **max** і **min** знаходять найбільше і, відповідно, найменше значення у списку:

```
>>> max('Привіт, світе')
'і'
>>> min('Привіт, світе')
' '
>>>
```

Функція **max** показує «найбільший символ» у рядку (ним виявилася літера «і»), а функція **min** – найменший символ (у цьому випадку – пробіл).

Ще одна поширена вбудована функція – функція **len**, яка визначає, скільки елементів міститься в її аргументі. Якщо аргументом **len** є рядок, вона повертає кількість символів у рядку.

```
>>> len('Привіт, світе')
13
>>>
```

Вбудовані функції не обмежуються опрацюванням рядків. Вони можуть функціонувати з будь-яким набором значень, розглянемо це в наших наступних розділах.

Імена вбудованих функцій необхідно сприймати як зарезервовані слова (наприклад, не можна надавати змінній ім'я «max»).

4.3 Функції перетворення типів

У Python також є вбудовані функції, які перетворюють значення з одного типу в інший. Функція **int** отримує будь-яке значення і конвертує його в ціле число, якщо це можливо, а інакше повідомляє про помилку:

```
>>> int('32')
32
>>> int('Привіт')
ValueError: invalid literal for int() with base 10: 'Привіт'
```

int може перетворювати значення з рухомою крапкою в цілі, однак дробова частина не округлюється, а просто відкидається:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

float перетворює цілі числа та рядки у числа з рухомою крапкою:

```
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

А от **str** перетворює свій аргумент на рядок:

```
>>> str(32)
'32'
>>> str(3.14159)
'3.14159'
```

4.4 Математичні функції

У Python є математичний модуль (math), який надає більшість відомих математичних функцій. Щоб скористатися цим модулем, його спершу необхідно імпортувати:

```
>>> import math
```

Ця інструкція створює *об'єкт модуля* math. Якщо задати print об'єкт модуля, то можна отримати коротку інформацію про нього:

```
>>> print(math)
<module 'math' (built-in)>
```

Об'єкт модуля містить функції та змінні, визначені в модулі. Щоб отримати доступ до однієї з функцій, слід вказати ім'я модуля та ім'я функції, розділивши їх крапкою. Такий формат називається *записом через крапку.*

```
>>> ratio = signal_power / noise_power
>>> decibels = 10 * math.log10(ratio)

>>> radians = 0.7
>>> height = math.sin(radians)
```

У першому прикладі обчислюється логарифм за основою 10 від співвідношення сигнал / шум. Модуль math також надає функцію **log**, яка обчислює логарифми з основою e.

У другому прикладі відбувається пошук синуса в **радіанах**. Назва змінної підказує, що **sin** та інші тригонометричні функції (**cos**, **tan** тощо) отримують аргументи в радіанах. Щоб перетворити градуси в радіани, слід поділити на 360 і помножити на 2*π*:

```
>>> degrees = 45
>>> radians = degrees / 360.0 * 2 * math.pi
>>> math.sin(radians)
0.7071067811865476
```

Вираз math.pi отримує змінну pi з математичного модуля. Значення цієї змінної є наближеним до значення числа *π*, з точністю до 15 цифр.

Якщо ви розумієтесь на тригонометрії, то можете перевірити попередній результат, порівнявши його з квадратним коренем з двох, поділеним на два:

```
>>> math.sqrt(2) / 2.0
0.7071067811865476
```

4.5 Випадкові числа

За однакових вхідних даних більшість комп'ютерних програм щоразу генерують однакові вихідні дані, через це їх називають *детермінованими*. Загалом, детермінізм досить непогане явище, оскільки ми очікуємо, що одні й ті самі обчислення дадуть однаковий результат. Проте, під час роботи з деякими застосунками, ми хочемо, щоб комп'ютер був непередбачуваним. Звичайно ж, це стосується ігор, але таких прикладів можна знайти безліч.

Не так вже і просто зробити програму повністю недетермінованою, та є способи зробити так, аби вона, принаймні, здавалася такою. Один з них – скористатися *алгоритмами*, які генерують *псевдовипадкові* числа. Псевдовипадкові числа насправді не є випадковими, адже вони генеруються детермінованим обчисленням, однак по вигляду їх майже неможливо відрізнити.

Модуль **random** надає функції, що генерують псевдовипадкові числа (далі – «випадкові»).

Функція **random** повертає випадкове число з плаваючою крапкою від 0.0 до 1.0 (0.0 включно, 1.0 не включно). Щоразу після виклику random, ви отримуватимете наступне число в довгій послідовності. Спробуйте запустити цей цикл, щоб переглянути приклад:

```
import random

for i in range(10):
    x = random.random()
    print(x)
```

Ця програма видасть ось такий перелік з 10 випадкових чисел від 0.0 до 1.0, не включаючи 1.0.

```
0.11132867921152356
0.5950949227890241
0.04820265884996877
0.841003109276478
0.997914947094958
0.04842330803368111
0.7416295948208405
0.510535245390327
0.27447040171978143
0.028511805472785867
```

Вправа 1: Запустіть програму у вашій системі та перегляньте отримані числа. Виконайте це кілька разів і прогляньте отриманий результат.

З випадковими числами працюють й інші функції. Наприклад, функції randint передають параметри low і high, і вона повертає ціле число з-поміж них (включно з обома значеннями).

```
>>> random.randint(5, 10)
5
>>> random.randint(5, 10)
9
```

Щоб випадковим чином обрати елемент з послідовності, скористайтеся choice:

```
>>> t = [1, 2, 3]
>>> random.choice(t)
2
>>> random.choice(t)
3
```

Модуль random також передбачає функції генерації випадкових величин з неперервних розподілів, зокрема, Гауссового, експоненціального, гамма та кількох інших.

4.6 Додаємо нові функції

Поки що ми працювали лише з наявними в Python функціями, проте також можна додавати і нові. *Визначення функції* встановлює ім'я нової функції та послідовність інструкцій, котрі виконуватимуться під час її виклику. Після того, як функцію визначено, нею можна неодноразово користуватись у своїй програмі. Розгляньте приклад:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')
```

def – ключове слово, яке вказує на те, що це визначення функції. Ім'я функції – **print_lyrics**. Щодо імен функцій діють ті самі правила, що й для змінних: дозволено використовувати літери, цифри та деякі розділові знаки, однак не можна, щоб першим символом була цифра. Заборонено використовувати ключові слова для назви функції, також слід уникати однакових імен змінних та функцій.

Порожні дужки після імені означають, що функція не приймає аргументів. Пізніше розглянемо функції, які прийматимуть аргументи як вхідні дані.

Перший рядок визначення функції називається *заголовком*, а решта – *тілом*. Заголовок слід закінчувати двокрапкою, а тіло має бути із відступом. За умовчанням, відступ завжди становить чотири пробіли. У тілі може бути довільна кількість інструкцій.

Якщо ввести визначення функції в інтерактивному режимі, інтерпретатор виведе багатокрапку (...), вказуючи, що визначення не завершено:

```
>>> def print_lyrics():
...     print("I'm a lumberjack, and I'm okay.")
...     print('I sleep all night and I work all day.')
...
```

Щоб завершити функцію, потрібно ввести порожній рядок (у скрипті це необов'язково).

Визначення функції створює змінну з тим самим ім'ям.

```
>>> print(print_lyrics)
<function print_lyrics at 0xb7e99e9c>
>>> print(type(print_lyrics))
<class 'function'>
```

Значення print_lyrics – це *функціональний об'єкт*, який має тип «function».

Синтаксис виклику нової функції такий самий, як і для вбудованих:

```
>>> print_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Після завершення процесу визначення функції, нею можна скористатись всередині іншої функції. Наприклад, для повторення попереднього приспіву можна написати функцію repeat_lyrics:

```
def repeat_lyrics():
    print_lyrics()
    print_lyrics()
```

А потім викликати repeat_lyrics:

```
>>> repeat_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

Але, насправді, у цієї пісні трішки інші слова...

4.7 Визначення та використання

Якщо поєднати усі частини коду з попереднього пункту, то ціла програма буде виглядати так:

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

def repeat_lyrics():
```

```
    print_lyrics()
    print_lyrics()

repeat_lyrics()
# Код: http://www.py4e.com/code3/lyrics.py
```

У цій програмі використано два визначення функцій: print_lyrics та repeat_lyrics. Визначення функцій виконуються так само, як і решта інструкцій, проте в результаті утворюються функціональні об'єкти. Інструкції всередині функції не виконуються, доки її не викличуть, отже, визначення функції не генерує вихідні дані.

Як і належить, спочатку необхідно створити функцію, перш ніж можна буде її виконати. Іншими словами, перед першим викликом потрібно виконати визначення функції.

Вправа 2: Перемістіть останній рядок цієї програми на початок, щоб виклик функції з'явився перед визначеннями. Запустіть програму та зверніть увагу на помилку, яку ви отримаєте.

Вправа 3: Перемістіть виклик функції назад униз і розташуйте визначення print_lyrics після визначення repeat_lyrics. Що сталось після запуску цієї програми?

4.8 Потік виконання

Для того, щоб переконатися, що функція була визначена до її першого застосування, необхідно вивчити порядок виконання інструкцій, який називається *потоком виконання*.

Виконання завжди починається з першої інструкції програми. Інструкції виконуються по одній, зверху донизу.

Визначення функцій не змінюють хід виконання програми, однак запам'ятайте, інструкції всередині функції не виконуються аж до моменту виклику функції.

Виклик функції – це ніби крюк у потоці виконання. Замість того, щоб перейти до наступної інструкції, потік перескакує до тіла функції, виконує всі інструкції, а потім повертається назад, щоб продовжити з місця, де зупинився.

Звучить доволі просто, якщо не згадувати, що одна функція може викликати іншу. Під час виконання однієї функції, програмі може знадобитися виконати інструкції в іншій функції. А під час виконання цієї нової функції, програмі може знову знадобитися виконати ще іншу функцію!
На щастя, Python чудово відслідковує усі етапи роботи, тому щоразу, коли функція завершується, програма продовжує роботу з того місця, на якому зупинилася у функції, що її викликала. Коли програма доходить до кінця, вона завершується.

Ну то і яка мораль цього всього? Не завжди слід читати програму зверху донизу. Іноді, більше користі буде, якщо слідкуватимете саме за потоком виконання.

4.9 Параметри та аргументи

Для роботи деяких вбудованих функцій необхідні аргументи. Наприклад, під час виклику math.sin, аргументом слід вказати число. Деякі функції використовують більше одного аргументу: наприклад, math.pow – два, основу і показник степеня.
Усередині функції аргументи присвоюються змінним, які називаються параметрами. Нижче наведено приклад функції користувача, яка отримує аргумент:

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
```

Ця функція присвоює аргумент параметру з іменем bruce. Під час виклику функція двічі виводить значення параметра (незалежно від того, який він).

Ця функція працює із будь-яким значенню, яке можна вивести.

```
>>> print_twice('Спам')
Спам
Спам
>>> print_twice(17)
17
17
>>> import math
>>> print_twice(math.pi)
3.141592653589793
3.141592653589793
```

До функцій, визначених користувачем, застосовуються ті самі правила побудови, що й до вбудованих функцій, тому для print_twice можна застосовувати будь-який вираз як аргумент:

```
>>> print_twice('Спам '*4)
Спам Спам Спам Спам
Спам Спам Спам Спам
>>> print_twice(math.cos(math.pi))
-1.0
-1.0
```

Аргумент обчислюється перед викликом функції, тому в прикладах вирази 'Спам '*4 і math.cos(math.pi) обчислюються лише один раз.

Також, як аргумент можна використовувати змінну:

```
>>> michael = 'Eric, the half a bee.'
>>> print_twice(michael)
Eric, the half a bee.
```

```
Eric, the half a bee.
```

Назва змінної, яка подається як аргумент (michael), не має нічого спільного з назвою параметра (bruce). Байдуже, як це значення називалося раніше (у програмі, що його викликала); тут, у print_twice, ми всіх називаємо bruce.

4.10 Продуктивні функції та порожні функції

Деякі з наших функцій, наприклад, математичні, дають певний результат; щось креативнішого не вдалось придумати, тому називаємо їх *продуктивні функції* (анг. *fruitful*). Інші функції, такі як print_twice, виконують дію, але не повертають значення. Вони називаються *порожніми функціями (анг. void)*.

Майже завжди, коли ви викликаєте продуктивну функцію, потрібно щось зробити з її результатом; наприклад, можна присвоїти його змінній чи використати як частину виразу:

```
x = math.cos(radians)
golden = (math.sqrt(5) + 1) / 2
```

Якщо викликати функцію в інтерактивному режимі, Python виводить результат на екран:

```
>>> math.sqrt(5)
2.23606797749979
```

Але якщо викликати продуктивну функцію і не зберегти результат функції в змінній у скрипті, повернене значення зникне, його як вітром здує!

```
math.sqrt(5)
```

Цей скрипт обчислює квадратний корінь з 5, однак, через те, що результат не зберігається у змінній і не виводиться на екран, користі від нього небагато.

Void-функції можуть виводити щось на екран чи робити щось ще, проте у них немає поверненого значення. Якщо спробувати присвоїти результат змінній, отримаєте спеціальне значення – None.

```
>>> result = print_twice('Дзинь')
Дзинь
Дзинь
>>> print(result)
None
```

Значення None – далеко не те саме, що і рядок «None». Це особливе значення, яке має власний тип:

```
>>> print(type(None))
<class 'NoneType'>
```

Щоб повернути результат з функції, потрібно скористатися інструкцією return у цій функції. Як приклад, можна створити дуже просту функцію addtwo, яка додає два числа і повертає результат.

```
def addtwo(a, b):
    added = a + b
    return added

x = addtwo(3, 5)
print(x)

# Код: http://www.py4e.com/code3/addtwo.py
```

Після виконання цього скрипта, інструкція print виведе «8», тому що функція addtwo була викликана з аргументами 3 і 5. Усередині функції параметрами a і b були 3 і 5 відповідно. Функція обчислила суму двох чисел і помістила її у локальну змінну функції added. Потім, за допомогою інструкції return, обчислене значення було передано назад у код, що його викликав, як результат функції, який було присвоєно змінній x і виведено на екран.

4.11 Для чого потрібні функції?

Можливо, не зовсім зрозуміло, навіщо взагалі розділяти програму на функції. Ось кілька причин:

- Створення нової функції дозволяє назвати групу інструкцій, це спрощує прочитання, розуміння та налагодження вашої програми.
- Функції можуть зробити програму меншою, вилучивши код, що повторюється. Пізніше, якщо потрібно буде вносити певні зміни, достатньо зробити це лише в одному місці.
- Розподіл великої програми на функції дозволяє налагоджувати їх по черзі, а потім збирати частини програми в одне ціле.
- Добре розроблені функції зазвичай використовуються і приносять користь великій кількості програм. Написавши та налагодивши одну функцію, можна буде використовувати її повторно.

У цій книзі ми часто будемо користуватися визначенням функції для пояснення певних понять. Одним з важливих моментів у створенні та використанні функцій є те, що вони повинні належним чином відображати її головну ідею, наприклад, «знайти найменше значення у списку значень». Пізніше розглянемо код, який визначає найменше значення у списку, і подамо його як функцію min, яка отримує список значень як аргумент і повертає найменше значення у списку.

4.12 Налагодження програми

Якщо для написання скриптів ви користуєтеся текстовим редактором, у вас можуть виникнути проблеми з пробілами і табуляцією. Найкращий спосіб запобігти цим проблемам – користуватися виключно пробілами (без табуляції). Більшість текстових редакторів, орієнтованих на Python, роблять це за замовчуванням, однак не всі.

Ми зазвичай не бачимо пробіли та табуляцію, це значно ускладнює їх налагодження, тож радимо спробувати знайти редактор, який сам контролюватиме усі відступи.

Важливо також не забувати зберігати програму перед її запуском. У деяких середовищах розробки це робиться автоматично, проте в деяких – ні. У такому випадку, програма, яку ви бачите у текстовому редакторі, буде відрізнятися від програми, яку ви запускаєте.

Налагодження може зайняти у вас купу часу, якщо будете продовжувати наступати на ті самі граблі та запускати неправильну програму знову і знову!

Переконайтеся, що код, який ви бачите, відповідає коду, який ви запускаєте. Якщо сумніваєтеся, додайте щось на кшталт print("привітик") на початку програми і запустіть її ще раз. Якщо не побачите «привітик», то ви запускаєте не ту програму!

4.13 Словник

algorithm (алгоритм) Загальний спосіб вирішення певної категорії завдань.

argument (аргумент) Значення, яке надається функції під час її виклику. Це значення присвоюється відповідному параметру у функції.

body (тіло) Послідовність інструкцій всередині визначення функції.

composition (побудова) Використання виразу як частини більшого виразу або інструкції як частини більшої інструкції.

deterministic (детермінований) Стосується програми, яка виконує одну й ту саму дію кожного запуску за одних і тих самих вхідних даних.

dot notation (крапковий запис) Синтаксис для виклику функції в іншому модулі через зазначення імені модуля, за яким слідує крапка та ім'я функції.

flow of execution (потік виконання) Порядок, згідно з яким виконуються інструкції під час запуску програми.

fruitful function (Fruitful-функція) Функція, яка повертає значення.

function (функція) Іменована послідовність інструкцій, яка виконує певну дію. Функції можуть отримувати або не мати аргументів, а також можуть видавати результат, або ні.

function call (виклик функції) Інструкція, яка виконує функцію. Складається з імені функції, за яким вказується список аргументів.

function definition (визначення функції) Інструкція, яка створює нову функцію, задаючи її ім'я, параметри та інструкції, які та буде виконувати.

function object (функціональний об’єкт) Значення, створене визначенням функції. Ім'я функції – це змінна, яка посилається на функціональний об'єкт.

header (заголовок) Перший рядок визначення функції.

import statement (інструкція import) Інструкція, яка зчитує файл модуля і створює об'єкт модуля.

module object (об’єкт модуля) Значення, створене інструкцією import, яке надає доступ до даних і коду, визначених у модулі.

parameter (параметр) Ім'я, яке застосовується всередині функції для відображення значення, переданого як аргумент.

pseudorandom (псевдовипадковий) Стосується послідовності чисел, які здаються випадковими, хоча генеруються детермінованою програмою.

return value (повернене значення) Результат роботи функції. Якщо виклик функції використовується як вираз, поверненим значенням є значення виразу.
void function (void-функція) Функція, яка не повертає значення.

4.14 Вправи

Вправа 4: Вкажіть призначення ключового слова «def» в Python.

a) Сленгове слово, яким користуються програмісти. Означає: «цей код справді крутецький»
b) Вказує на початок функції
c) Вказує на те, що наступну частину коду з відступом слід зберегти для подальшого використання
d) Обидва b та c – правильні
e) Немає правильної відповіді

Вправа 5: Вкажіть, що саме виведе подана програма Python.

```
def fred():
    print("Zap")

def jane():
    print("ABC")

jane()
fred()
jane()
```

a) Zap ABC jane fred jane
b) Zap ABC Zap
c) ABC Zap jane
d) ABC Zap ABC
e) Zap Zap Zap

Вправа 6: Перепишіть програму для розрахунку оплати праці з урахуванням півторагодинної ставки за понаднормову роботу і створіть функцію computepay, яка приймає два параметри (години і ставку).

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

Вправа 7: Перепишіть програму оцінювання з попереднього розділу, застосувавши функцію computegrade, яка приймає оцінку як параметр і повертає її у вигляді рядка.

Score	Grade
>= 0.9	A
>= 0.8	B
>= 0.7	C
>= 0.6	D
< 0.6	F

```
Enter score: 0.95
A

Enter score: perfect
Bad score

Enter score: 10.0
Bad score

Enter score: 0.75
C

Enter score: 0.5
F
```

Запустіть програму кілька разів, як показано вище, щоб протестувати різні значення вхідних даних.