ІНФОРМАТИКА ТА ПРОГРАМУВАННЯ

Тема 9. Підпрограми

Підпрограми

- **Підпрограма** – це логічно незалежна спеціальним чином оформлена частина програми для розв'язування певної задачі.
- До підпрограм можна багаторазово звертатися з інших частин програми. Таке звернення називають **викликом** підпрограми.
- Підпрограми обмінюються інформацією із зовнішнім світом за допомогою параметрів.
- У багатьох мовах програмування підпрограми поділяють на функції, які, як і математичні функції, повертають один результат, та процедури, які можуть повертати багато результатів або жодного.

9.1 ФУНКЦІЇ

Функції

- У Python усі підпрограми називають функціями.
- Якщо підпрограмі треба повернути декілька результатів, то вважається, що функція повертає кортеж, що складається з цих результатів.

Простий синтаксис функції

```
def f(x1, ..., xn):
    P
```

- де *f* – ім'я функції, x_1, …, x_n – параметри (аргументи) функції, *P* – інструкція.

```
def f(x1, ..., xn):
```

- називають **заголовком** функції, а *P* – **тілом** функції.
- x_1, …, x_n ще називають **формальними параметрами** (формальними аргументами).

- Виклик функції – це вираз

```
f(e1, ..., en)
```

- де *f* – ім'я функції, e_1, …, e_n – вирази, що є фактичними параметрами функції.

Простий синтаксис функції.2

- Для повернення результату у тілі функції повинен бути та виконуватись хоча б один оператор

```
return e
```

  - де e – вираз, який є результатом функції. Після виконання return функція завершує роботу.
- Функція може не повертати жодних результатів. Тоді return не вказують, а виклик функції

```
f(e1, ..., en)
```

є окремою інструкцією.

- Треба зазначити, що у останньому випадку Python вставляє у кінець тіла функції оператор return None.
- Змінні, які використовуються всередині функції та не є параметрами, називають локальними змінними.

Просте правило виклику функції

- Коли Python зустрічає виклик функції, він

1. Виділяє нові клітинки пам'яті для аргументів функції. Можна позначити ці нові змінні x_1', …, x_n'.
2. Виділяє нові клітинки пам'яті для локальних змінних функції під час її виконання, коли зустрічається нове ім'я локальної змінної. Можна позначити ці нові змінні y_1', …, y_m', де y_1, …, y_m - локальні змінні функції.
3. Виконує ланцюг присвоєнь

$x_1' = e_1$

…

$x_n' = e_n$

4. Виконує $P(x_1', \ldots, x_n', y_1', \ldots, y_m')$. Тобто, при виконанні замінює аргументи та локальні змінні новими змінними.
5. Повертає результат функції (за допомогою return *e*) та підставляє його у місце виклику.

Особливості параметрів-списків та параметрів-словників

- Правило виклику функції стверджує, що під параметри виділяються нові змінні, а передача параметрів у функцію здійснюється ланцюгом присвоєнь.
- З цього випливає, що будь-які зміни, що відбуваються з формальними параметрами у тілі функції не відображаються на фактичних параметрах.
- Це так. Майже так…
- Параметри, які є такими, що змінюються (mutable), в разі зміни їх у тілі функції, змінюють і відповідні фактичні параметри.
- З розглянутих нами типів це списки та словники.
- Вказана поведінка може бути бажаною або небажаною. Якщо така поведінка небажана, можна під час виклику функції передавати копію списку або словника за допомогою вирізки (наприклад, s[:] замість просто s).

Приклад

- Обчислення найменшого спільного кратного двох натуральних чисел з використанням функції обчислення найбільшого спільного дільника.

Функція format

- Вбудована функція format використовується для форматування рядків, зокрема, при виведенні.
- Для виклику функції треба вказати

s.format(z_1, …, z_k)

  - де s – рядок, що форматується, - z_1, …, z_k – вирази, які підставляються у рядок s.
- Рядок s повинен мати поля підстановки, які беруться у фігурні дужки ‘{‘ та ‘}’.
- Кожному z_i, як правило, відповідає одне поле.
- Найпростіший варіант – просто використання {} у тих місцях, де треба вставити вирази.
- Тоді аргументи z_i вибираються та підставляються замість полів підстановки у порядку слідування.
- Поля підстановки можуть також включати номери аргументів функції format або імена аргументів.
- У цьому випадку взаємний порядок слідування аргументів та полів підстановки може бути різним.

Функція format. Специфікація формату

- Окрім номерів та імен у полі підстановки можна також вказати специфікацію формату. Специфікація формату розташовується після двокрапки ‘:’ та може містити
  - вирівнювання,
  - порядок використання знаків,
  - ширину поля,
  - точність
  - тип.

Функція format. Вирівнювання

Вирівню-вання	Опис
'<'	Поле вирівнюється по лівому краю (значення за угодою для більшості об’єктів).
'>'	Поле вирівнюється по правому краю (значення за угодою для чисел).
'='	Після знаку та до числа вставляються нулі ‘0’.
'^'	Поле вирівнюється по центру.

- Перед символом вирівнювання може стояти символ для заповнення вільних позицій поля.

Функція format. Порядок використання знаків

Знак	Опис
'+'	вказує, що знак повинен ставитись перед від'ємними та додатними числами.
'-'	вказує, що знак повинен ставитись тільки перед від'ємними числами (за угодою).
пропуск ' '	вказує, що перед додатними числами повинен ставитись пропуск ' ', а перед від'ємними – знак мінус.

- Ширина поля – це мінімальна загальна кількість позицій, що відводиться для виведення.
- Точність – це кількість позицій під дробову частину для дійсних чисел.

Функція format. Деякі значення типу для цілих чисел

Тип	Опис
'b'	у двійковій системі числення.
'c'	перетворює ціле у символ Unicode з відповідним кодом.
'd'	у десятковій системі числення (значення за угодою).
'o'	у вісімковій системі числення.
'x'	у системі числення за основою 16 (для цифр 10-15 використовуються маленькі латинські літери).
'X'	у системі числення за основою 16 (для цифр 10-15 використовуються великі латинські літери).
'n'	Те ж саме , що 'd', окрім того, що використовує відповідні локалізовані установки для символів-розділювачів у числах.

Функція format. Деякі значення типу для дійсних чисел

Тип	Опис
'e'	Представлення з плаваючою крапкою. Точність за угодою – 6 знаків.
'f'	Представлення з фіксованою крапкою. Точність за угодою – 6 знаків.
'g'	Загальний формат. Якщо порядок числа є порівнюваним із заданою точністю, число представляється з фіксованою крапкою, інакше – з плаваючою крапкою (формат за угодою). Точність за угодою – 6 знаків.
'%'	Процентне представлення. Множить число на 100, зображуе у форматі з фіксованою крапкою та додає символ '%'.

Рядки документації

- Рядки документації у функціях дозволяють легко отримувати підказку по функції просто у інтерпретаторі Python.
- Рядок документації функції – це рядок, який обмежений трьома апострофами ‘’’ ‘’’ або трьома подвійними лапками “”” “”” і, таким чином, включає декілька фізичних рядків. Рядок документації повинен йти відразу після заголовку функції та мати такий же відступ, як і тіло функції.
- По суті, рядок документації є багаторядковим коментарем.
- Щоб побачити рядок документації функції, треба у інтерпретаторі набрати

f.__doc__

або

help(f)

  - де f – ім’я функції.

Рядки документації.2

- Рекомендовано перший фізичний рядок рядка документації починати з великої літери та закінчувати крапкою.
- Другий фізичний рядок залишати порожнім, а з третього, - давати докладний опис функції, якщо потрібно.

- Наприклад, заголовок функції разом з рядками документації

```
def prepare_string (s):
    '''Готує рядок s до перевірки на симетричність.

    Видаляє з s усі символи-розділювачі та
    переводить рядок до нижнього регістру.
    '''
```

Приклад

- Дано рядок, у якому міститься речення. Перевірити, чи є цей рядок паліндромом (без урахування пропусків, верхнього або нижнього регістру та розділових знаків).

Повернення функцією декількох результатів

- Як вже відзначалося, функція може повертати декілька результатів.
- У цьому випадку у return вказують декілька виразів через кому, що еквівалентно вказанню кортежу.
- Під час виклику функції у лівій частині присвоєння також вказують декілька змінних через кому. Ці змінні набувають значень результатів функції.

Приклад

- Визначення символу, який входить у рядок найбільшу кількість разів, а також кількості його входжень

Значення параметрів за угодою

- Окремі параметри функцій можна визначити за угодою.

```
param = value
```

  - де param – ім’я параметру, value – значення за угодою.
- Для таких параметрів можна у виклику не вказувати відповідні фактичні параметри.
- Тоді Python використає їх значення за угодою.
- Якщо ж значення фактичних параметрів вказати, то саме вони будуть передані у функцію.
- Параметри, значення яких визначається за угодою, повинні розташовуватись після параметрів, які не визначаються за угодою.

Позиційні та ключові параметри

- Параметри функцій, які ми розглядали раніше, ще називають **позиційними параметрами**, тому що співставлення між фактичними та формальними параметрами здійснюється за позицією у списку параметрів.
- У багатьох випадках зручно робити таке співставлення не за позицією, а за іменем параметра. Такі параметри називають **ключовими**. При виклику для ключових параметрів пишуть $x_i = e_i$, де x_i – ім'я формального параметру, а e_i – вираз для фактичного параметру.
- Порядок слідування ключових параметрів у виклику функції може бути довільним.

Приклад

- Порахувати кількість компонент вектору, які належать відрізку [a, b].
  - Використати функції для введення вектору та обчислення кількості компонент.
  - За угодою вектор складається з 10 компонент, а відрізок - [0, 1] (версія 1)

Змінна кількість параметрів

- Функції можуть також мати змінну кількість параметрів. Причому, як позиційних, так і ключових.
- У заголовку функції для змінної кількості позиційних параметрів пишуть *args, а для змінної кількості ключових параметрів - **kwargs.
- Це означає, що args є кортежем, а kwargs, - словником.
- Кількість переданих параметрів у цьому випадку нескладно обчислити за допомогою функції len: len(args), len(kwargs).
- Також у циклі по всіх елементах кортежу args або словника kwargs можемо перебрати всі аргументи та виконати над ними потрібні дії.
- Функція, яка приймає довільну кількість позиційних та ключових параметрів, виглядає так:

```
def f(*args, **kwargs):
    P
```

Приклад

- Обчислення середнього значення та медіани дійсного вектора з n компонент.
  - Середнє значення обчислюється як сума компонент розділена на їх кількість.
  - Якщо кількість компонент непарна, то медіана – це компонента відсортованого вектора, яка має середній індекс.
  - Якщо ж кількість компонент парна, то медіана – це середнє значення двох сусідніх середніх компонент відсортованого вектора.

Повний синтаксис та повне правило виклику функції

- Повний синтаксис функції

```
def f(x_1, ..., x_n, d_1 = a_1, ..., d_k = a_k, *args, **kwargs):
    P
```

де

- *f* – ім’я функції,
- $x_1, \ldots, x_n$ – позиційні параметри функції,
- $d_1, \ldots, d_k$ – параметри зі значеннями за угодою,
- $a_1, \ldots, a_k$ – значення за угодою,
- args – змінна кількість позиційних параметрів,
- kwargs – змінна кількість ключових параметрів,
- *P* – інструкція.

Виклик функції

$f(g_1, ..., g_N)$

- де f – ім'я функції,
- $g_1, \ldots, g_N$ – позначення фактичних параметрів.
- При цьому,
  - g_i – це або вираз e_i
  - або присвоєння виду $x_i = e_i$ (або $d_i = e_i$ або $z_i = e_i$, де z_i – ім'я ключового параметру, що не входить у іменовані параметри x_i, d_i).

Повне правило виклику функції

- Повне правило виклику функції
- Коли Python зустрічає виклик функції, він

1. Виконує співставлення фактичних та формальних параметрів наступним чином:
   1) Співставляє наявні позиційні фактичні параметри з x_1, …, x_n, d_1, …, d_k по позиціях.
   2) Співставляє наявні ключові фактичні параметри з формальними параметрами x_1, …, x_n, d_1, …, d_k, які залишились без співставлених після п.1.1, по іменах.
   3) Співставляє позиційні фактичні параметри, що залишились після п.1.1, з кортежем args.
   4) Співставляє ключові фактичні параметри, що залишились після п. 1.2, з словником kwargs.
2. Виділяє нові клітинки пам'яті для аргументів функції. Можна позначити ці нові змінні x_1', …, x_n', d_1', …, d_k', args', kwargs'.
3. Виділяє нові клітинки пам'яті для локальних змінних функції під час її виконання, коли зустрічається нове ім'я локальної змінної. Можна позначити ці нові змінні y_1', …, y_m', де y_1, …, y_m - локальні змінні функції.

Повне правило виклику функції.3

4. Виконує ланцюг присвоєнь

```
...
x_i = e_j                          #для співставлених на кроці 1.1 параметрів
...
d_s = e_r                          #для співставлених на кроці 1.1 параметрів
...
args = (e_i1,... , e_it)           #для співставлених на кроці 1.3 параметрів
kwargs = dict(g_j1,..., g_ju)      #для співставлених на кроці 1.4 параметрів
...
d_p = a_p                #для неспівставлених параметрів, які задані за угодою
...
```

5. Виконує *P*. При виконанні замінює аргументи та локальні змінні новими змінними.
6. Повертає результат(и) функції (за допомогою return *e*) та підставляє його(їх) у місце виклику.

lambda-функції

- lambda-функції або lambda-вирази призначені для опису коротких неіменованих функцій.
- Синтаксис lambda-функції:
- `lambda x: e(x)`
  - де *x* – параметр, *e(x)* – вираз, що залежить від *x*.
- Наприклад:
- `lambda x: x**2`
  - lambda-функція може залежати від декількох аргументів
- `lambda x, y: x+y`
- Правило виконання lambda-функції.

1. Python повертає функцію, що обчислює значення виразу *e(x)* при заданому *x*.

- Як правило, lambda-функції використовують у виразах разом з функціями map(), filter().

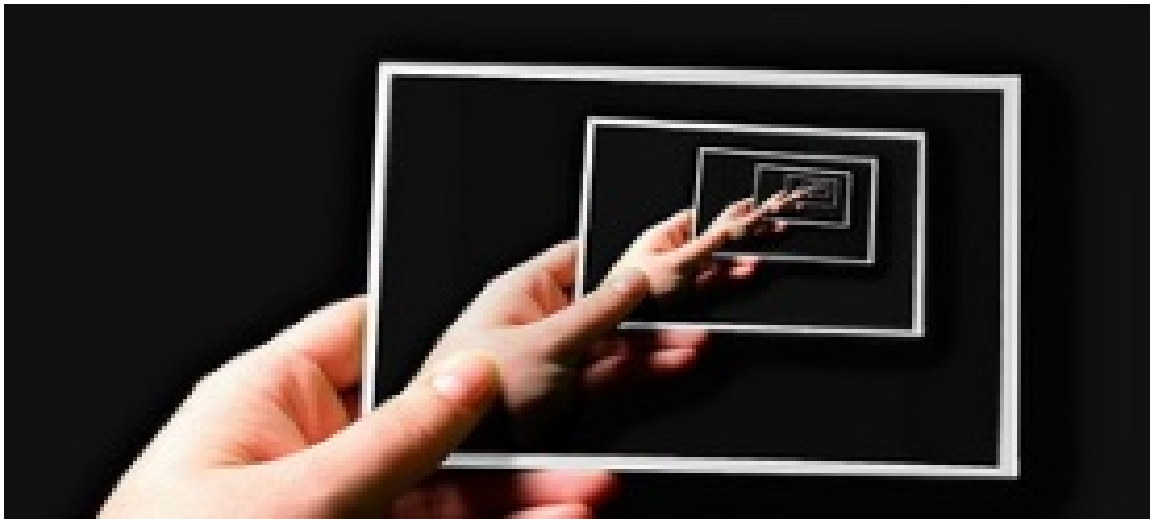
Приклад

- Порахувати кількість компонент вектору, які належать відрізку [a, b].
  - Використати функції для введення вектору та обчислення кількості компонент, а також lambda-функцію, що перевіряє належність числа відрізку.
  - За угодою вектор складається з 10 компонент, а відрізок - [0, 1] (версія 2)

9.2 РЕКУРСІЯ

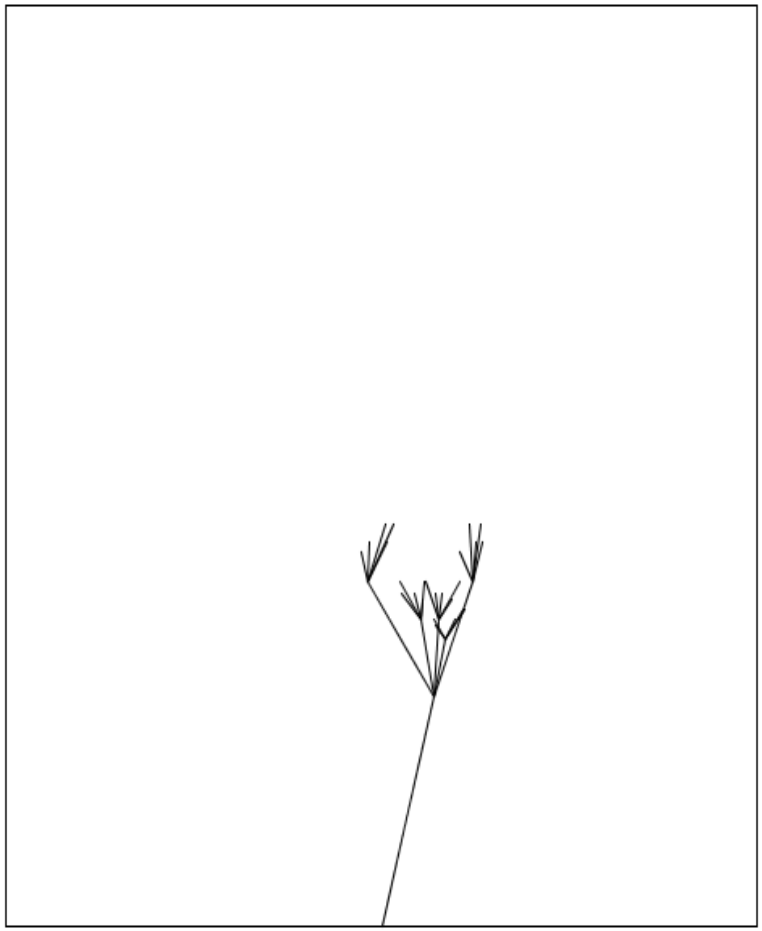
Рекурсія

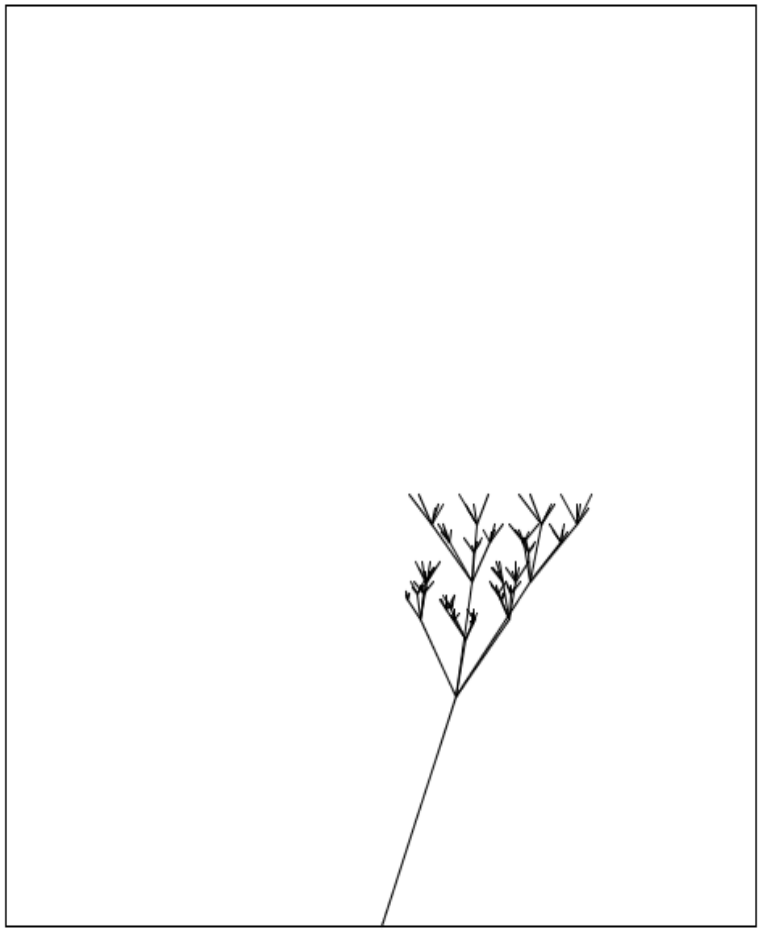
- **Рекурсивною** називається підпрограма, яка прямо або непрямо викликає сама себе.
- Приклади рекурсії ми бачимо в оточуючому житті: рекурсивні зображення, структура рослин та кристалів, рекурсивні розповіді або вірші.



Рекурсивні зображення

Рекурсивні зображення.2





Рекурсивні зображення.3

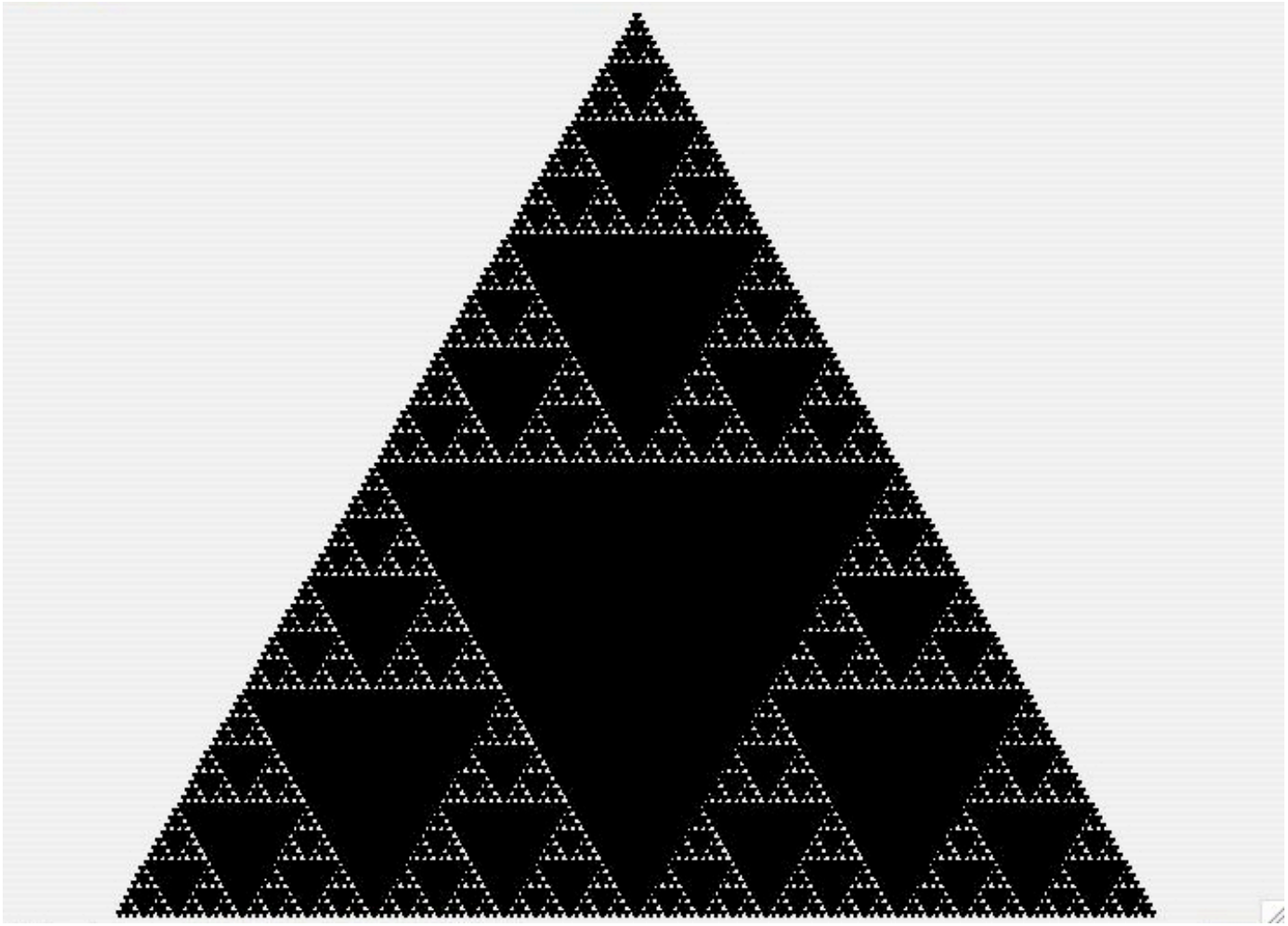


Схема примітивної рекурсії

- Розглянемо, звідки з'являються рекурсивні підпрограми.
- Скажемо, що функція *f* натурального аргументу *n* **задана схемою примітивної рекурсії**, якщо її визначення має вигляд

$$\begin{cases} f(0) = c, \\ f(n+1) = \phi(n, f(n)) \end{cases} \qquad (9.1)$$

  - де ϕ - деяка задана функція.

Схема примітивної рекурсії.2

- Схему примітивної рекурсії можна застосовувати і для визначення функцій багатьох змінних.
- При цьому рекурсія ведеться по одній з змінних, яка належить до натурального типу:

$$\begin{cases} f(0, x_1, \ldots, x_n) = h(x_1, \ldots, x_n), \\ f(n+1, x_1, \ldots, x_n) = \phi(n, x_1, \ldots, x_n, f(n, x_1, \ldots, x_n)) \end{cases} \quad (9.2)$$

  - де ϕ і h - задані функції.

Приклади функцій, заданих за схемою примітивної рекурсії

- $n!$

$$\begin{cases} f(0) = 1, \\ f(n+1) = (n+1) \cdot f(n) \end{cases}$$

- x^n

$$\begin{cases} f(0, x) = 1, \\ f(n+1, x) = x \cdot f(n, x) \end{cases}$$

Обчислення функції, заданої за схемою примітивної рекурсії

- Функцію, задану схемою примітивної рекурсії (9.1), можна обчислити за допомогою підпрограми, яка побудована за наступною схемою:

```
def f(n):
    if n == 0:
        y = c
    else:
        y = ϕ(n-1,f(n-1))
    return y
```

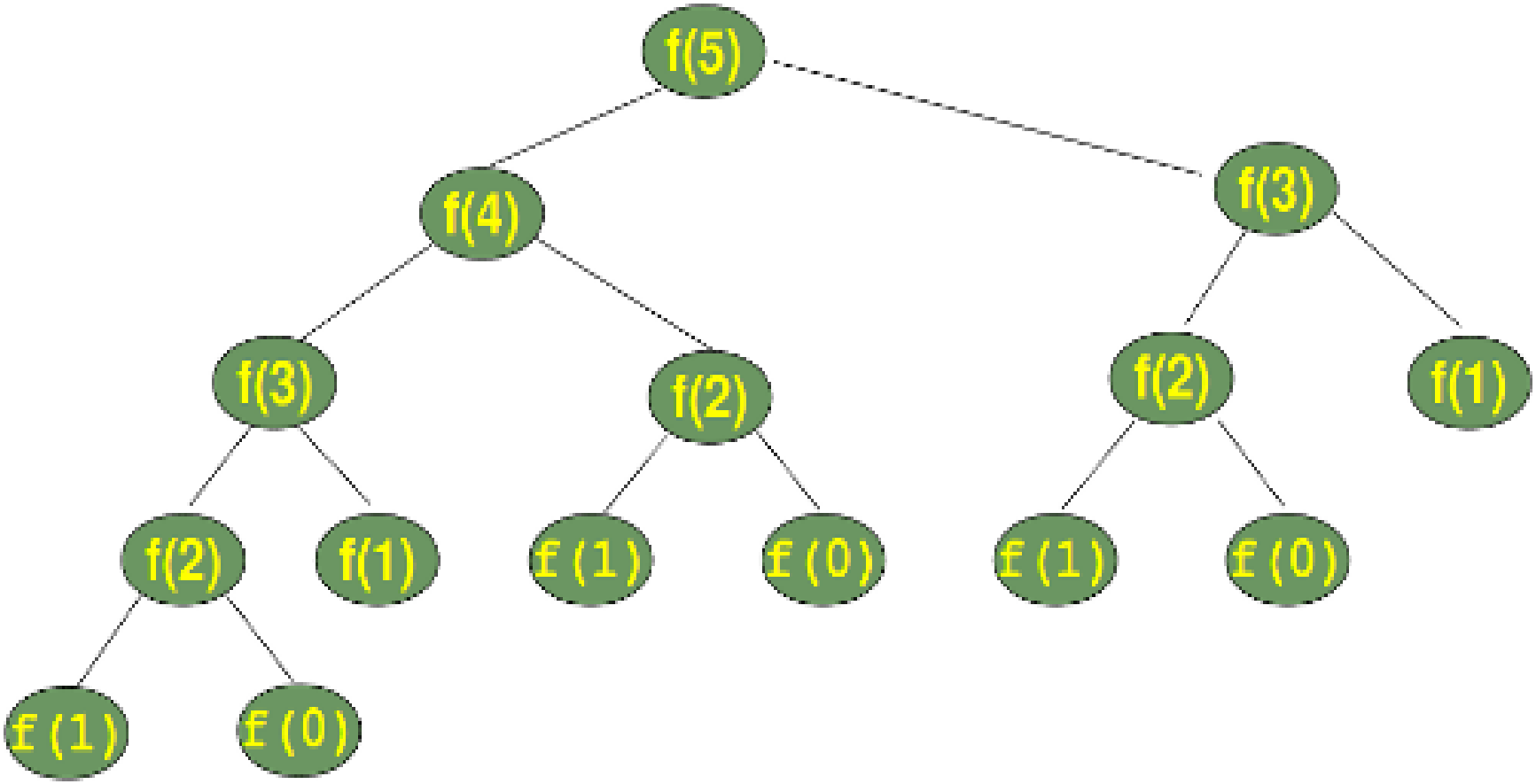
Приклади

- Обчислити *n*! при заданому значенні *n*.

- Обчислити n-те число Фібоначчі за рекурсивним визначенням

$$\begin{cases} f(0) = 1, \quad f(1) = 1 \\ f(n+2) = f(n+1) + f(n) \end{cases}$$

Обчислення функції fib при n = 5



Рекурсивне визначення в ітеративній формі

- Скажемо, що функція Φ(*n*, *x*, *y*) **задана рекурсивним визначенням в ітеративній формі**, якщо

$$\begin{cases} \Phi(0, x, y) = y, \\ \Phi(n+1, x, y) = \Phi(n, g(x), h(x, y)) \end{cases} \qquad (9.2)$$

  - де *g*, *h* – відомі функції.
- Теорема 9.1
- Нехай функція Φ(*n*, *x*, *y*) задана співвідношеннями (9.2). Тоді справджується трійка

```
#x == a and y == b
for i in range(n):
    y = h(x,y)
    x = g(x)
#y == Φ(n,a,b)
```

- Тобто, значення Φ(*n*, *a*, *b*) можна обчислити, не використовуючи рекурсію

Зв'язок між примітивно-рекурсивними та ітеративними функціями

- Теорема 9.2. (про зв'язок між примітивно-рекурсивними та ітеративними функціями)
- Нехай функція *f* задана за схемою примітивної рекурсії (9.1). Тоді

$f(n) = \Phi(n, 0, c)$,

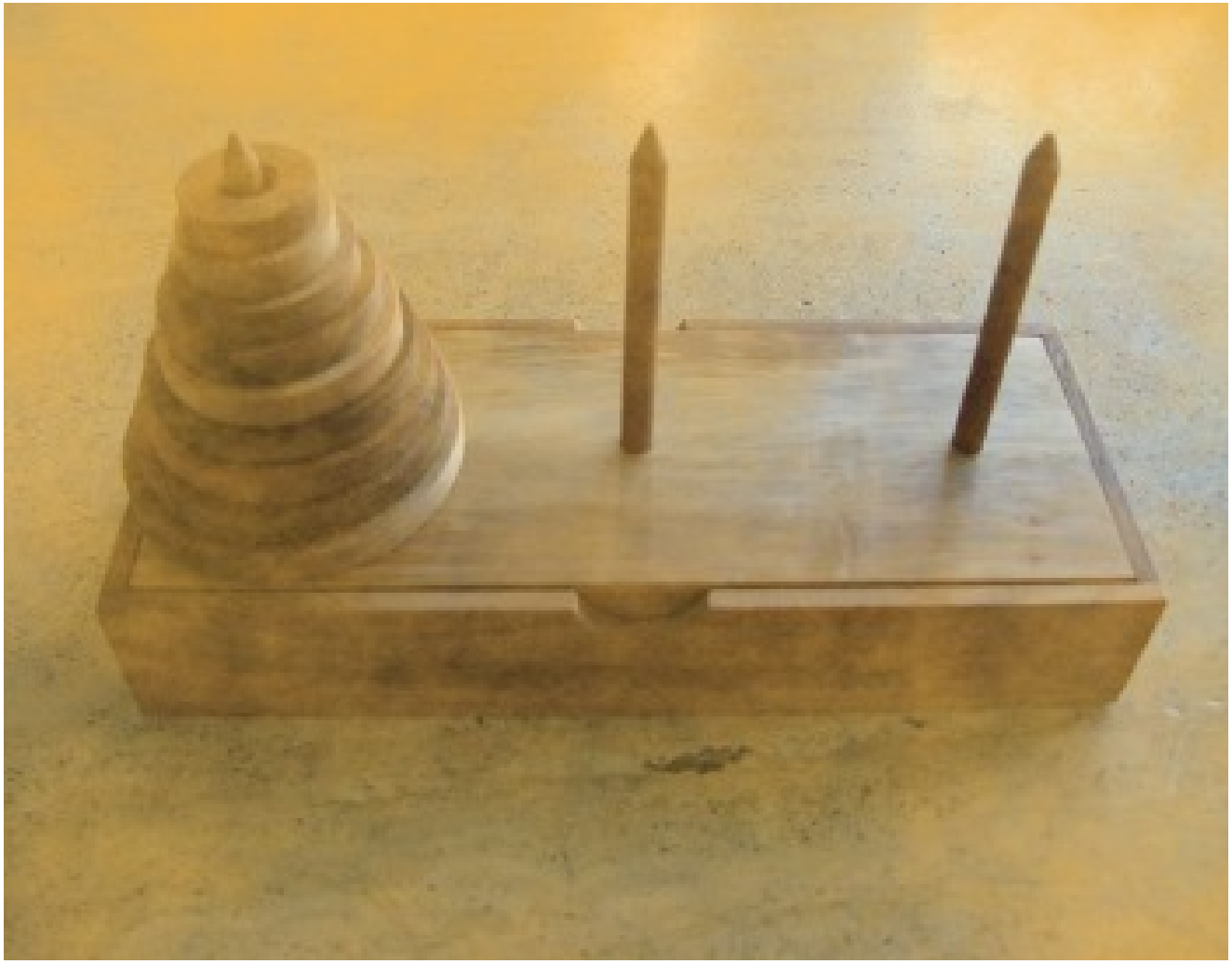
- де функція Φ задана співвідношеннями

$$\begin{cases} \Phi(0, x, y) = y, \\ \Phi(n+1, x, y) = \Phi(n, x+1, \phi(x, y)) \end{cases}$$

Приклад

- «Ханойські вежі». Дошка має три стержні.
- На першому нанизані *n* дисків спадного вгору діаметра.
- Потрібно, перекладаючи диски по одному, розташувати їх в колишньому порядку на другому стержні, використовуючи третій стержень для тимчасового зберігання дисків.
- При цьому більший диск ніколи не повинен розташовуватися над меншим.

Ханойські вежі



Глобальні змінні

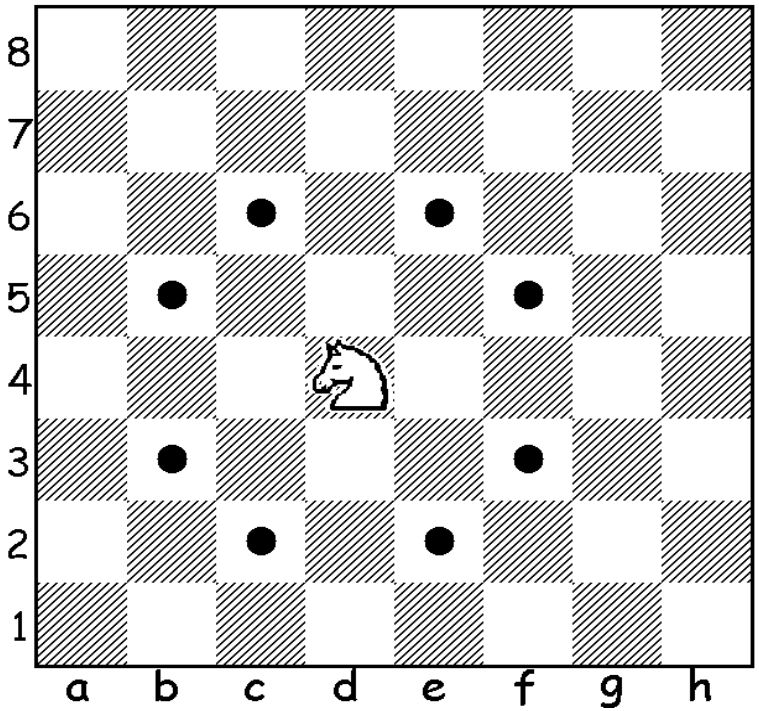
- За угодою, всі змінні, які фігурують у функціях є локальними, тобто прихованими у тілі функції.
- Якщо потрібно бачити значення якихось змінних з тіла функції ззовні цієї функції, або, навпаки, бачити та/або змінювати у функції зовнішні змінні, використовують глобальні змінні.
- Глобальні змінні позначають у тілі функції наступним чином:

```
global x, y, z
```

- Глобальні змінні є альтернативою параметрам в якості засобу обміну інформацією між функціями та зовнішнім світом.
- Їх можна застосовувати, коли декілька функцій обробляють одні і ті ж дані.
- Але використовувати глобальні змінні слід обережно.
- Бажано також обмежувати їх використання, тому що є небезпека неконтрольованої зміни цих змінних у будь-якій функції.
- Зазвичай кращим способом обміну інформацією для функцій є використання параметрів.

Приклад

- «Тур коня». Знайти шлях шахової фігури «кінь» з поля шахової дошки (*x*,*m*) на поле (*y*,*k*), де *x*, *y* - вертикалі (позначаються літерами від a до h), *m*, *k* - горизонталі (позначаються цифрами від 1 до 8).



9.3 ФУНКЦІОНАЛЬНИЙ ТИП ДАНИХ

Використання функцій в якості змінних та параметрів

- Функції можна не тільки викликати напряму, але й передавати у підпрограми в якості параметрів, використовувати у виразах тощо.
- Для цього існують змінні функціонального типу.
- Ми вже зустрічалися з таким використанням функцій, коли розглядали функцію *map*(), у яку передається функція (ім'я функції), яка потім застосовується до послідовності.
- Ми можемо змінній функціонального типу присвоїти значення деякої функції наступним чином: f = sin, після чого виклик функції f(x) буде рівнозначний виклику sin(x).
- Тобто, існує функціональний тип даних. Як і для інших типів, для функціонального типу визначено носій, операції відношення та інструкції.

Функціональний тип даних

- -носій
- Нехай множини $M_1, \ldots, M_n$ є носіями типів $t_1, \ldots, t_n$, до яких можуть належати аргументи. $L_1, \ldots, L_k$ – носії типів результату.
- Тоді носієм функціонального типу буде множина відображень.

$M_t = \{M_1 \times \ldots \times M_n \rightarrow L_1 \times \ldots \times L_k\}$

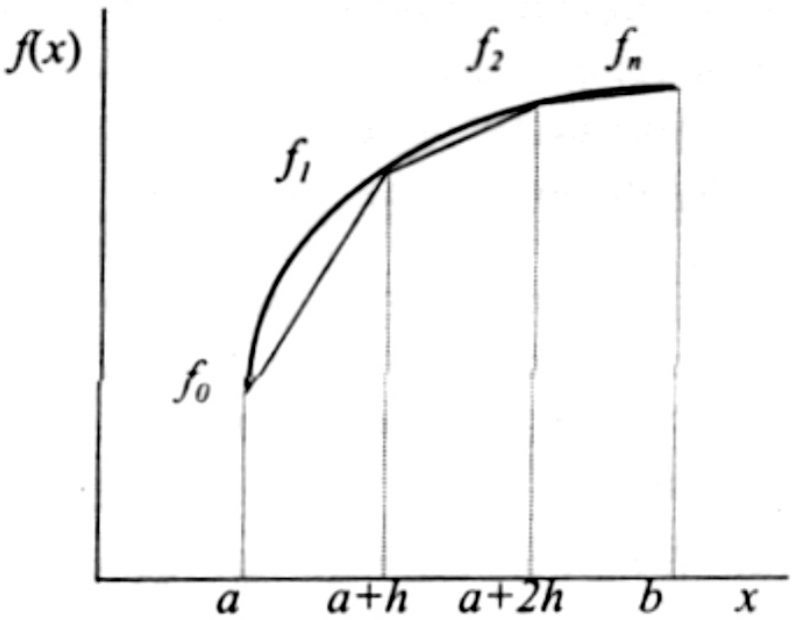
- -операції
- Єдиною операцією для функціонального типу є виклик функції
- $f(e_1, \ldots, e_n)$
- -відношення
- Для функціонального типу визначені відношення == та !=.
- інструкції
- Для функціонального типу визначено присвоєння та виведення

```
f = f2
print(f)
```

Приклад

- Обчислити визначений інтеграл функції f(x) на відрізку [a, b] методом трапецій

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f(x_i)+f(x_{i-1})}{2}h, \quad x_i = a+ih, \quad h=\frac{b-a}{n}, \quad n=2^k, \quad k=1,2,...$$



Резюме

- Ми розглянули:
  1. Означення підпрограми.
  2. Простий синтаксис та просте правило виклику функції
  3. Передачу параметрів різних типів у підпрограми
  4. Функцію format
  5. Оформлення документації функції
  6. Повернення функцією декількох результатів
  7. Значення параметрів за угодою
  8. Позиційні та ключові параметри
  9. Змінну кількість параметрів функції
  10. Повний синтаксис та повне правило виклику функції
  11. lambda-функції
  12. Рекурсію та рекурсивні підпрограми
  13. Глобальні змінні
  14. Функціональний тип даних

Де прочитати

1. A Byte of Python (Russian) Версия 2.01 Swaroop C H (Translated by Vladimir Smolyar), http://wombat.org.ua/AByteOfPython/AByteofPythonRussian-2.01.pdf
2. Бублик В.В., Личман В.В., Обвінцев О.В.. Інформатика та програмування. Електронний конспект лекцій, 2003 р.,
3. Марк Лутц, Изучаем Python, 4-е издание, 2010, Символ-Плюс
4. Python 3.4.3 documentation
5. Марк Саммерфилд, Программирование на Python 3. Подробное руководство. - Символ-Плюс, 2009.
6. http://www.python-course.eu/python3_functions.php