Олексій Васильєв

Мова програмування Python

Київ 2020

Лекція 6.

Функції

- Оголошення і виклик функцій
- Іменовані аргументи функції
- Механізм передачі аргументів
- Аргументи за замовчуванням
- Довільна кількість аргументів
- Локальні/глобальні змінні
- Вкладені функції
- Лямбда-функції
- Функція як аргумент і результат
- Рекурсія
- Декоратори функцій
- Функції-генератори
- Анотації і документування

Оголошення/виклик

Функція: блок програмного коду, у якого є назва і котрий можна виконати, вказавши (в правильному контексті) назва функції (виклик функції)

Описання функції:

```
def назва(аргументи):
    # Описання функції
```

Функція може повертати значення, а може не повертати значення. Якщо функція повертає значення, використовуємо інструкцію return, після якої вказується значення, що повертається

Програма (Funcs.py)

```
# Функція для відображення букв з переданого
# аргументом тексту:
def show(txt):
    # Перетворення тексту в список і його сортування:
    symbs=sorted(list(txt))
    # Відображення вмісту списку:
    print(symbs)
# Виклик функції:
show("Python")
# Функція для розрахунку суми квадратів натуральних чисел:
def sqsum(n):
    # Створення списку з квадратів натуральних чисел:
    nums=[k*k for k in range(1,n+1)]
    # Результат функції:
    return sum(nums)
# Змінна з числовим значенням:
m=10
# Виклик функції для розрахунку суми квадратів чисел:
print("Сума квадратів чисел від 1 до",str(m)+":",sqsum(m))
```

```
['P', 'h', 'n', 'o', 't', 'y']
Сума квадратів чисел від 1 до 10: 385
```

Функції

```
# Функція без аргументів не повертає результат:
def next_day():
    txt=input("Який сьогодні день тижня? ")
    txt=txt.lower().strip()
    if txt=="понеділок":
       new_txt="вівторок"
    elif txt=="вівторок":
       new_txt="середа"
    elif txt=="середа":
       new_txt="четвер"
    elif txt=="четвер":
       new_txt="п'ятниця"
    elif txt=="п'ятниця":
       new_txt="субота"
    elif txt=="субота":
       new_txt="неділя"
    elif txt=="неділя":
       new_txt="понеділок"
    else:
       print("Немає такого дня тижня!")
       return
    print(f"Завтра - {new_txt}")
# Функція без аргументу повертає результат:
def get_name():
    name=input("Добрый день! Як Вас звати? ")
    if name.strip(" .,:;!?_")=="":
       name="Містер Ікс"
    return name
# Функція без аргументів не повертає результат:
def hello():
    name=get_name()
    print(f"Приємно познайомитись, {name}!")
    next_day()
# Виклик функції:
hello()
```

Програма (Funcs_2.py)

Змінні, котрі отримують свої значення в тілі функції - локальні (доступні тільки в тілі функції). Це ж правило стосується і аргументів функції. Крім локальних, в функціях можуть використовуватися і глобальні змінні

```
Добрый день! Як Вас звати? Прибулець Альф
Приємно познайомитись, Прибулець Альф!
Який сьогодні день тижня? Вівторок
Завтра - середа
```

```
Добрый день! Як Вас звати? _,! ...
Приємно познайомитись, Містер Ікс!
Який сьогодні день тижня? Вторніца
Немає такого дня тижня!
```

Функції - 2

```
# Імпорт функцій:
from random import *
# Функція для відображення списків,
# множин, текстів і словників:
def show(L,symb):
    for s in L:
        print(symb,s,sep="",end="")
    print(symb)
# Вихідні дані:
A=[1,2,3,4,5]           # Список
B={'A','B','C','D'}     # Множина
C="Python"              # Текст
D={"A":1,"B":2,"C":3} # Словник
# Виклик функції:
show(A,"|")
show(B,"/")
show(C,"*")
show(D,"#")
# Функція для створення списку чисел:
def get_nums(n,state):
    if type(n)!=int:
        return []
    if state:
        L=list(2*(k+1) for k in range(n))
    else:
        L=list(2*k+1 for k in range(n))
    return L
# Виклик функції:
print(get_nums(10,True))
```

Програма (Funcs_3.py)

```
print(get_nums(8,False))
print(get_nums(12.5,True))
# Функція для створення множини
# випадкових букв:
def get_symbs(n):
    if n>10 or n<1:
        num=10
    else:
        num=n
    S=set()
    Nmin=ord("A")
    Nmax=ord("Z")
    while len(S)<num:
        S.add(chr(randint(Nmin,Nmax)))
    return S
# Ініціалізація генератора випадкових чисел:
seed(2019)
# Виклик функції:
print(get_symbs(7))
print(get_symbs(-5))
print(get_symbs(15))
```

```
|1|2|3|4|5|
/D/A/C/B/
*P*y*t*h*o*n*
#A#B#C#
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
[1, 3, 5, 7, 9, 11, 13, 15]
[]
{'U', 'Z', 'P', 'H', 'K', 'E', 'F'}
{'X', 'T', 'O', 'U', 'Y', 'C', 'Z', 'J', 'H', 'N'}
{'B', 'L', 'U', 'I', 'S', 'V', 'W', 'P', 'K', 'G'}
```

Назва функції

Програма (FuncName.py)

```
# Функції:
def alpha():
    print("Це Alpha!")
def bravo():
    print("Це Bravo!")
def hello():
    print("А це Hello!")
# Змінна з цілочисловим значенням:
num=123
# Виклик функцій і перевірка значення змінної:
print("Спочатку було так:")
alpha()
bravo()
hello()
print("num =",num)
# Зміна значень:
alpha,bravo=bravo,alpha
num=hello
hello=321
# Виклик "функцій" і перевірка значень "змінної":
print("А стало так:")
alpha()
bravo()
num()
print("hello =",hello)
```

```
Спочатку було так:
Це Alpha!
Це Bravo!
А це Hello!
num = 123
А стало так:
Це Bravo!
Це Alpha!
А це Hello!
hello = 321
```

Іменовані аргументи

Програма (FuncArgs.py)

```
# Функція з трьома аргументами:
def show(first,second,third):
    print(f"[1] Перший аргумент - {first}")
    print(f"[2] Другий аргумент - {second}")
    print(f"[3] Третій аргумент - {third}")
# Виклик функції:
show(1,2,3)
show(second="B",third="C",first="A")
show(1,third=3,second=2)
```

Позиційна передача аргументів

Передача аргументів за ключем

Спочатку вказуються позиційні аргументи, а потім - аргументи за ключем

```
[1] Перший аргумент - 1
[2] Другий аргумент - 2
[3] Третій аргумент - 3
[1] Перший аргумент - A
[2] Другий аргумент - B
[3] Третій аргумент - C
[1] Перший аргумент - 1
[2] Другий аргумент - 2
[3] Третій аргумент - 3
```

Передача аргументів

```
# Функції:
def shift(val):
    print("Функція shift()")
    print("Вихідне значення:",val)
    val=["A","B","C"]
    print("Кінцеве значення:",val)
def change(val):
    print("Функція change()")
    print("Вихідне значення:",val)
    if type(val)==list:
        for k in range(len(val)):
            val[k]+=1
    else:
        val+=1
    print("Кінцеве значення:",val)
# Змінна:
num=100
# Список:
L=[10,20,30]
# Виклик функцій:
print(f"Змінна num={num}")
change(num)
print(f"Змінна num={num}")
print(f"Список L={L}")
shift(L)
print(f"Список L={L}")
change(L)
print(f"Список L={L}")
```

Програма (Args.py)

Аргументи передаються за значенням - для аргументу автоматично створюється копія

```
Змінна num=100
Функція change()
Вихідне значення: 100
Кінцеве значення: 101
Змінна num=100
Список L=[10, 20, 30]
Функція shift()
Вихідне значення: [10, 20, 30]
Кінцеве значення: ['A', 'B', 'C']
Список L=[10, 20, 30]
Функція change()
Вихідне значення: [10, 20, 30]
Кінцеве значення: [11, 21, 31]
Список L=[11, 21, 31]
```

Значення за замовчуванням

```
# Функція зі значеннями аргументів за замовчуванням:
def show(first,second="Bravo",third="Charlie"):
    print(f"[1] - {first}")
    print(f"[2] - {second}")
    print(f"[3] - {third}")
    print("-"*13)
# Виклик функції:
show("Alpha")
show("A","B","C")
show(10,20)
show(100,third=300)
show(third="третій",first="перший")
```

Програма (ArgsVal.py)

```
[1] - Alpha
[2] - Bravo
[3] - Charlie
-------------
[1] - A
[2] - B
[3] - C
-------------
[1] - 10
[2] - 20
[3] - Charlie
-------------
[1] - 100
[2] - Bravo
[3] - 300
-------------
[1] - перший
[2] - Bravo
[3] - третій
-------------
```

Спочатку вказуються звичайні аргументи, а потім - аргументи зі значенням за замовчуванням

Аргументи можна передавати за позицією або/і за іменем, причому деякі аргументи можуть бути відсутніми. Команда виклику функції має бути такою, що за нею можна однозначно визначити, яке значення отримує кожен аргумент

Довільна кількість аргументів

```
# Функції з довільною кількістю аргументів:
def sqr_sum(*n):
    s=0
    for a in n:
        s+=a*a
    return s
def get_sum(*n):
    s=0
    for a in n:
        if type(a)==int:
            s+=a
    return s
def get_text(*t):
    return " ".join(t)
# Виклик функцій:
print("Сума квадратів:",sqr_sum(1,3,5))
print("Сума квадратів:",sqr_sum(2,4,6,8,10))
print("Сума чисел:",get_sum(2,"A",4,"B",6))
print("Сума чисел:",get_sum(1,[2,3],4))
print("Сума чисел:",get_sum())
print("Текст:",get_text("Всім","привіт"))
print("Текст:",get_text("A","B","C","D"))
```

Програма (AnyArgs.py)

```
Сума квадратів: 35
Сума квадратів: 220
Сума чисел: 12
Сума чисел: 5
Сума чисел: 0
Текст: Всім привіт
Текст: A B C D
```

Довільна кількість аргументів 2

Коли ми викликаємо функцію, котра може отримувати довільну кількість аргументів, технічно аргументи у функцію передаються у вигляді кортежу. В цьому нескладно переконатися за допомогою функції

```
def show(*val):
    print("Тип:",type(val))
    print("Аргумент:",val)
```

Крім аргументу з "зірочкою", у функції можуть бути й інші аргументи. Також деякі аргументи можуть мати значення за замовчуванням

```
# Функція с різними аргументами:
def my_sum(n,*a,txt="Сума чисел"):
    s=0
    for m in range(len(a)):
        s+=a[m]**n
    print(txt+":",s)
# Виклик функції:
my_sum(1,100,200,300)
my_sum(2,10,20,30,txt="Сума квадратів")
```

Програма (AnyArgs_2.py)

```
Сума чисел: 600
Сума квадратів: 1400
```

Локальні/глобальні змінні

```
# В функції використовуються глобальні
# і локальні змінні:
def myfunction():
    # Глобальні змінні:
    global A,B
    # Присвоєння значень змінним:
    A="Альфа"
    B="Браво"
    D="Дельта"
    # Перевірка значень:
    print("В функції: A =",A)
    print("В функції: B =",B)
    print("В функції: C =",C)
    print("В функції: D =",D)
# Глобальні змінні:
A="Alpha"
C="Charlie"
D="Delta"
# Перевірка значень змінних:
print("До виклику функції: A =",A)
print("До виклику функції: C =",C)
print("До виклику функції: D =",D)
# Виклик функції:
myfunction()
# Перевірка значень змінних:
print("Після виклику функції: A =",A)
print("Після виклику функції: B =",B)
print("Після виклику функції: C =",C)
print("Після виклику функції: D =",D)
```

Програма (Vars.py)

- Змінні, створені в функції - **локальні** і доступні тільки в функції
- **Глобальні** змінні створюються поза тілом функції
- Якщо в тілі функції змінній присвоюється значення, то така змінна інтерпретується як локальна, навіть якщо є глобальна змінна з таким самим іменем
- Якщо значення змінної тільки зчитується, то використовується глобальна змінна
- Якщо ми бажаємо використовувати в тілі функції глобальну змінну (в тому числі і присвоювати їй значення), її необхідно оголосити з ключовим словом `global`

```
До виклику функції: A = Alpha
До виклику функції: C = Charlie
До виклику функції: D = Delta
В функції: A = Альфа
В функції: B = Браво
В функції: C = Charlie
В функції: D = Дельта
Після виклику функції: A = Альфа
Після виклику функції: B = Браво
Після виклику функції: C = Charlie
Після виклику функції: D = Delta
```

Вкладені функції

Програма (InnerFuncs.py)

```
# Функція з вкладеною функцією:
def mysum(*a):
    # Список:
    txt=["чисел","квадратів","кубів"]
    # Вкладена функція:
    def calc(n):
        s=0
        for m in range(len(a)):
            s+=a[m]**n
        return s
    # Виклик вкладеної функції:
    for k in range(len(txt)):
        print("Сума",txt[k]+":",calc(k+1))
# Виклик функції:
mysum(1,3,5,7)
```

```
Сума чисел: 16
Сума квадратів: 84
Сума кубів: 496
```

- Функцію можна описувати в іншій функції: це вкладена або внутрішня функція
- Вкладена функція доступна всередині тої функції, в якій вона описана
- Вкладена функція має доступ до змінних у зовнішній функції
- Зовнішня функція доступу до локальних змінних вкладеної функції не має

Лямбда-функції

Лямбда-вираз:
вираз, який визначає функцію (без імені)

Синтаксис:
`lambda аргументи: результат`

```
Непарні числа:
1 3 5 7 9 11 13 15 17 19
Степені двійки:
1 2 4 8 16 32 64 128 256 512
Квадрати чисел:
1 4 9 16 25 36 49 64 81 100
Виклик F(3,5): 15
Виклик f(3,5): 8
Виклик calc(3,5): 15
```

Програма (Lambda.py)

```
num=10
# Функція на основі лямбда-виразу:
L=lambda n: 2*n+1
# Перевірка результату:
print("Непарні числа:")
for k in range(num):
    print(L(k),end=" ")
# Нове значення:
L=lambda n: 2**n
# Перевірка результату:
print("\nСтепені двійки:")
for k in range(num):
    print(L(k),end=" ")
# Прямий виклик лямбда-функції:
print("\nКвадрати чисел:")
for k in range(num):
    print((lambda x: x*x)(k+1),end=" ")
# Звичайна функція:
def calc(x,y):
    return x+y
# Використання функції в лямбда-виразі:
F=lambda x,y: calc(x,y)
# Змінній присвоюється ім'я функції:
f=calc
# Імені функції присвоюється лямбда-вираз:
calc=lambda x,y: x*y
# Перевірка результату:
print("\nВиклик F(3,5):",F(3,5))
print("Виклик f(3,5):",f(3,5))
print("Виклик calc(3,5):",calc(3,5))
```

Функція як аргумент і результат

Програма (ArgRes.py)

```
x       1    2    3    4    5
A(x)    1    4    9   16   25
B(x)    1    8   27   64  125
C(x)    5    9   13   17   21
F(x->x*x)(5):  625
F(x->2*x+1)(5): 23
```

```
# Аргумент функції - функція (і два числа):
def display(f,a,b):
    for k in range(a,b+1):
        print("{0:4}".format(f(k)),end=" ")
    print()
# Результат функції - функція:
def mypow(n):
    return lambda x: x**n
# Аргументи функції - функції. Результат - функція:
def apply(f,h):
    def calc(x):
        return f(h(x))
    return calc
# Визначення функцій:
A=mypow(2)
B=mypow(3)
C=apply(lambda x: 2*x+1,lambda x: 2*x)
# Перевірка результату:
print("x   ",end="")
display(lambda x: x,1,5)
print("A(x)",end="")
display(A,1,5)
print("B(x)",end="")
display(B,1,5)
print("C(x)",end="")
display(C,1,5)
# Визначення функції:
F=lambda f: lambda x: f(f(x))
# Перевірка результату:
print("F(x->x*x)(5): ",F(lambda x: x*x)(5))
print("F(x->2*x+1)(5):",F(lambda x: 2*x+1)(5))
```

Рекурсія

Рекурсія — спосіб описання функції, коли функція викликає сама себе

```
Сума чисел:
0 1 3 6 10 15 21 28 36 45 55 66
Числа Фібоначчі:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
Інверсія тексту:
|n|o|h|t|y|P| |o|l|l|e|H|
Інверсія списку:
|5|4|3|2|1|
```

Програма (Recursion.py)

```
# Функція для розрахунку суми чисел:
def mysum(n):
    if n==0:
        return 0;
    else:
        return n+mysum(n-1)
# Функція для розрахунку чисел Фібоначчі:
def fib(n):
    if n==1 or n==2:
        return 1
    else:
        return fib(n-1)+fib(n-2)
# Функція для інверсного відображення
# тексту/списку:
def show(txt):
    if len(txt)==0:
        print("|")
    else:
        print("|",txt[-1],end="",sep="")
        show(txt[:-1])
# Виклик функції:
print("Сума чисел:")
for k in range(12):
    print(mysum(k),end=" ")
print("\nЧисла Фібоначчі:")
for k in range(15):
    print(fib(k+1),end=" ")
print("\nІнверсія тексту:")
show("Hello Python")
print("Інверсія списку:")
show([1,2,3,4,5])
```

Декоратори

```
# Функції для використання в декораторах:
def A(h):
    return lambda x: h(x)*h(7-x)
def B(h):
    return lambda x,y: h(x,y)+h(y,x)
def C(h):
    return lambda x: h(x,10-x)
# Функції с декоратором:
@A
def f(x):
    return 2*x-1
@B
def F(x,y):
    return (8-x)*(y+1)
@C
def H(x,y):
    return x*y
# Перевірка результату:
print("f(3) =",f(3))
print("F(5,7) =",F(5,7))
print("H(6) =",H(6))
```

Програма (Decorators.py)

Декоратор функції: інструкція виду `@A` перед описанням деякої функції `h()`, причому `A` - ім'я функції, у якої аргумент - функція і результат - теж функція. В результаті функція `h()` переозначається відповідно до правила `h=A(h)`

```
f(3) = 35
F(5,7) = 30
H(6) = 24
```

Генератори

Програма (Generators.py)

```
# Функції-генератори:
def names():
    yield "Дядя Федір"
    yield "Пес Шарік"
    yield "Кіт Матроскін"
def colors():
    L=["Червоний","Жовтий","Зелений","Синій"]
    for clr in L:
        yield clr
def myrange(n):
    for k in range(n):
        yield 2*k+1
# Використання функцій-генераторів:
print("Вони з Простоквашино:")
for name in names():
    print(name)
print(list(names()))
R=colors()
print("Кольори:")
for r in R:
    print(r,end=" ")
print("\nЩе одна спроба...")
for r in R:
    print(r,end=" ")
print("Нічого нема? Це нормально.")
print("Непарні числа:")
print(list(myrange(10)))
print(tuple(myrange(10)))
N=myrange(8)
A=list(N)
print("A =",A)
B=list(N)
print("B =",B)
for num in myrange(8):
    print(num,end=" ")
print()
```

```
Вони з Простоквашино:
Дядя Федір
Пес Шарік
Кіт Матроскін
['Дядя Федір', 'Пес Шарік', 'Кіт Матроскін']
Кольори:
Червоний Жовтий Зелений Синій
Ще одна спроба...
Нічого нема? Це нормально.
Непарні числа:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
(1, 3, 5, 7, 9, 11, 13, 15, 17, 19)
A = [1, 3, 5, 7, 9, 11, 13, 15]
B = []
1 3 5 7 9 11 13 15
```

- **Функції-генератори дозволяють створювати особливі ітеровані об'єкти, котрі можна потім використовувати в операторах циклу або створювати на їх основі списки і інші послідовності**
- **В функції-генераторі замість ключового слова** `return` **використовують інструкцію** `yield`

Анотації

- Функція реалізується у вигляді спеціального об'єкту класу `function`
- У функції є деякі властивості, обумовлені тим, що по суті це об'єкт

- В описанні функції відразу після рядка з назвою функції можна розмістити текст - *текст документування*. При виконанні коду функції він ігнорується. Його можна "прочитати", вказавши після імені функції (без круглих дужок) через крапку поле `__doc__`

- Можна створювати спеціальні "пояснення" для аргументів функції і результату функції - *анотації*

- Аргумент з анотацією:

```
аргумент: анотація = значення
```

- Анотація для результату:

```
def функція(аргументи)->анотація:
    # Код функції
```

- Поле `__annotations__`: словник, ключі — назви аргументів, значення елементів — анотації. Анотація до результату запам'ятовується за ключем `"return"`

Анотації - 2

Програма (Annotations.py)

```
# Функції з документуванням:
def show(txt):
    "Це функція show() з одним аргументом."
    print("Єдиний аргумент:",txt)
def display(a,b):
    "Це функція display() з двома аргументами."
    print("[1] Перший аргумент:",a)
    print("[2] Другий аргумент:",b)
# Функція без документування:
def hello():
    print("Всім привіт!")
# Виклик функцій і перевірка документування:
print(show.__doc__)
show("A")
print(display.__doc__)
display("B","C")
# Змінна посилається на функцію:
f=show
# Виклик функцій і перевірка документування:
print(f.__doc__)
f("D")
# Новий текст документування для функції:
display.__doc__="Новий текст для display()"
# Перевірка результату:
print(display.__doc__)
display("E","F")
# Створюється документування для функції:
hello.__doc__="Функція hello()"
# Перевірка результату:
print(hello.__doc__)
hello()
```

Програма (Annotations_2.py)

```
Функція show()
{'txt': 'Текст', 'return': 'Результату нема'}
txt - Текст
return - Результату нема
```

```
# Функція з анотаціями:
def show(txt:"Текст"="Функція show()")->"Результату нема":
    print(txt)
# Виклик функції:
show()
# Словник анотацій:
print(show.__annotations__)
# Анотації:
for k in show.__annotations__:
    print(k,"-",show.__annotations__[k])
```

```
Це функція show() з одним аргументом.
Єдиний аргумент: A
Це функція display() з двома
аргументами.
[1] Перший аргумент: B
[2] Другий аргумент: C
Це функція show() з одним аргументом.
Єдиний аргумент: D
Новий текст для display()
[1] Перший аргумент: E
[2] Другий аргумент: F
Функція hello()
Всім привіт!
```

Завдання - 1

Напишіть програму з функцією, аргументом якій передається числовий список, а результатом є ще один список, в якій включені тільки непарні числа зі списку-аргументу

Завдання - 2

Напишіть програму, в якій описується функція з довільною кількістю числових аргументів, а результатом повертається список з трьох елементів: середнє значення аргументів, максимальне значення серед аргументів і мінімальне значення серед аргументів

Домашнє завдання

[1] Напишіть програму, в якій створюється функція з одним текстовим аргументом і довільною кількістю цілочислових аргументів. Результатом є текст, сформований з букв першого текстового аргументу. Цілочислові аргументи визначають індекси букв, котрі потрібно включити в текст-результат

[2] Напишіть програму, в якій використовується функція-генератор, яка створює ітерований обїект зі степенями двійки. Кількість елементів визначається аргументом функції-генератора