Funkcje

- Funkcje w Pythonie tworzone są z wykorzystaniem słowa kluczowego def.
- Funkcje mogą przyjmować tzw. parametry pozycyjne oraz parametry ze słowem kluczowym.

Ogólna sygnatura funkcji ma postać:

```
def fun(a,*b,c=3,**d):
  '''łańcuch dokumentacyjny'''
  pass
```

Parametry pozycyjne

- sama nazwa (bez żadnych dodatkowych znaków) oznacza parametr pozycyjny, który musi być do funkcji przekazany przy jej wywołaniu:

```
def fun(a,b):
    print(a,b)

fun(1,2)
fun(a=1,b=2)
fun(b=1,a=2)
```

- poprzedzenie nazwy * oznacza, że tworzymy funkcję ze zmienną liczbą parametrów pozycyjnych, tzn. mogą ale nie muszą się one pojawić przy wywołaniu funkcji, wewnątrz funkcji są one przechowywane w krotce:

```
def fun(a,*b):
    print(a,b)

fun(1)                      #1 ()
fun(1,2)                    #1 (2,)
fun(1,2,'a',[1,2,3])        #1 (2, 'a', [1, 2, 3])
fun(a=1,b=2)                #błąd
fun(b=1,a=2)                #błąd
```

Parametry z wartością domyślną

- oprócz parametrów pozycyjnych funkcje mogą mieć parametry ze słowem kluczowym, inaczej mówiąc z wartością domyślną:

```
def fun(a,b=4):
    print(a,b)

fun(1)              #1 4
fun(1,2)            #1 2
fun(a=1,b=2)        #1 2
fun(b=1,a=2)        #2 1
```

- parametrów ze słowem kluczowym też może być zmienna liczba, wewnątrz funkcji są one przechowywane w słowniku:

```
def fun(**d):
    print(d)

fun()                   #{}
fun(a=1,b=2,c=3)        #{'a': 1, 'b': 2, 'c': 3}
```

Funkcje

Funkcja może przyjmować zarówno parametry pozycyjne, jak i ze słowem kluczowym, ale pozycyjne muszą zawsze poprzedzać parametry ze słowem kluczowym:

```
def fun(a,b,c=4):
    print(a,b,c)

fun(1,2)                #1 2 4
fun(1,2,3)              #1 2 3
fun(a=2,b=3,c=5)        #2 3 5
fun(c=5,a=1,b=2)        #1 2 5
```

ALE:

```
def fun(a,*b,c=4):
    print(a,b,c)

fun(1,2,3)              #1 (2, 3) 4
fun(1,2,3,c=5)          #1 (2, 3) 5
fun(c=5,1,2,3)          #błąd
```

Parametry specjalne

```
def fun(a,*,c=4):
    print(a,c)

fun(1,2)          #błąd
fun(1,c=2)        #1 2
fun(a=1,c=2)      #1 2
fun(c=1,a=2)      #2 1
```

od 3.8

```
def fun(a,/,b,*,c=4):
    print(a,b,c)

fun(1,2)              #1 2 4
fun(1,2,c=3)          #1 2 3
fun(1,b=2,c=3)        #1 2 3
fun(1,c=2,b=3)        #1 3 2
fun(a=1,b=2,c=3)      #błąd
```

Podsumowując

```
def fun(a,/,*b,c=4,**d):
    print(a,b,c,d)

fun(1,2)                          #1 (2,) 4 { }
fun(1,2,c=3)                      #1 (2,) 3 { }
fun(1,b=2,c=3)                    #1 ( ) 3 {'b': 2}
fun(1,c=2,b=3)                    #1 ( ) 2 {'b': 3}
fun(1,2,3,c=2,b=3)                #1 (2, 3) 2 {'b': 3}
fun(1,2,3,c=2,b=3,d=4,e=5)        #1 (2, 3) 2 {'b': 3, 'd': 4, 'e': 5}
```

Funkcje

Gwiazdki mogą też być użyte przy wywołaniu funkcji do wypakowania wartości z obiektu:

```
def fun(a=1,b=2):
    print(a,b)
```

```
fun()                       #1 2
fun(1)                      #1 2
fun(b=1)                    #1 1
fun(a=3,b=4)                #3 4
fun(b=3,a=4)                #4 3
fun(*(4,))                  #4 2
fun(*(4,5))                 #4 5
fun(**{'a':3})              #3 2
fun(**{'a':3, 'b':4})       #3 4
```

Przekazywanie wartości do funkcji

Obiekty niemodyfikowalne przekazywane są do funkcji przez wartość, a modyfikowalne przez referencję:

```
def fun(n,li,s,sl):
    n=3
    li[1]=3
    s='3'
    sl[1]=3

zn=1
zli=[1,2,3,4]
zs='1'
zsl={1:1, 2:2, 3:3, 4:4}

fun(zn,zli,zs,zsl)

print(zn)      #1
print(zli)     #[1, 3, 3, 4]
print(zs)      #1
print(zsl)     #{1: 3, 2: 2, 3: 3, 4: 4}
```

UWAGA

```
def fun(el,p=[]):
    p.append(el)
    print(p)

fun(1)          #[1]
fun(2)          #[1,2]
fun(3)          #[1,2,3]
```

```
def fun(k,v,s={}):
    s[k]=v
    print(s)

fun(1,2)        #{1: 2}
fun(2,3)        #{1: 2, 2: 3}
fun(3,4)        #{1: 2, 2: 3, 3: 4}
```

Funkcje

- zdefiniowanie dwóch funkcji o takiej samej nazwie nie powoduje przeciążenia funkcji, późniejsza definicja przesłania wcześniejszą
- każda funkcja jest jednak przeciążona z definicji (jeśli nie stosujemy typowania), bo nie określamy typu parametru
- kontroli typu parametru wewnątrz funkcji można dokonać wywołując funkcję wbudowaną isinstance, której pierwszym parametrem jest obiekt a drugim typ lub krotka z typami (tu z kolei przydatny może być moduł types), np:

```
isinstance(1,int)                  #True
isinstance(1.,(int,float))         #True
isinstance('1',(int,float))        #False
```

Zwracanie wartości

Wartości z funkcji zwracane są instrukcją return, może się po niej znaleźć więcej niż jeden obiekt, co nie oznacza, że w Pythonie można z funkcji zwracać kilka obiektów, zwracana jest krotka opakowująca zwracane obiekty:

```
import random
def fun(n):
    res=[a for _ in range(n) if (a:=random.randrange(100))%2]
    return res, min(res), max(res)

np,wmin,wmax=fun(20)
```

Funkcje

Funkcje w Pythonie zawsze coś zwracają, jeśli nie robią tego jawnie zwracana jest wartość None:

```
def fun():
    pass

a=fun()
print(a)      #None
```

Widoczność zmiennych

```
x=7
def fun(p):
    y=x+p
    return y

print(fun(3))
```

Widoczność zmiennych

```
x=7
def fun(p):
    y=x+p
    return y

print(fun(3))      #10
```

Widoczność zmiennych

```
x=7
def fun(p):
    y=x+p
    return y

print(fun(3))      #10
```

```
x=7
def fun(p):
    y=x+p
    x=5
    return y

print(fun(3))
```

Widoczność zmiennych

```
x=7
def fun(p):
    y=x+p
    return y

print(fun(3))      #10
```

```
x=7
def fun(p):
    y=x+p
    x=5
    return y

print(fun(3))      #BŁĄD
```

CHYBA ŻE

(chociaż jest to niebezpieczne i staramy się tak NIE robić)

```
x=7
def fun(p):
    global x
    y=x+p
    x=5
    return y

print(fun(3))      #10

print(x)           #5
```

Funkcje

Do zadania przyda się funkcja wbudowana eval, która oblicza i zwraca wartość wyrażenia przekazanego jako parametr:

```
eval('2+7')          #9
```

```
x=4
eval('2+x+7')        #13
```