Wizualizacja danych
semestr letni 2024

Dr Anna Muranova
UWM w Olsztynie

Wykład 5

Funkcje w Python

Funkcje definiuje się używając słowa `def`. Po nim następuje nazwa identyfikująca funkcji, następnie para nawiasów, które mogą zawierać kilka nazw zmiennych (parametrów, argumentów), a na końcu dwukropek. Poniżej zaczyna się blok wyrażeń, które są częścią tej funkcji.

```
def nazwa_funkcji(parametr_1, parametr_2, ):
    polecenie_1
    polecenie_2
    ... i td.

    return wartosc
```

Najprostsza funkcja

```
def przywitanie():
    # Blok należący do funkcji.
    print ("Pozdrowienia z mojej funckji!")
# Koniec funkcji.

przywitanie()   # Wywołanie funkcji.
przywitanie()   # Ponowne wywołanie funkcji.
```

Funkcja z argumentami

```
def przywitanie_imienne(imie, zyczenia):
    print ("Witaj" + imie + ". Zycze Tobie " + zyczenia)

przywitanie_imienne("Jacek", "zdrowia") # Wywołanie funkcji.
```

Domyślny argument:

```
def przywitanie_imienne(imie, zyczenia = "zdrowia" ):
    print ("Witaj, " + imie + ". Zycze Tobie " + zyczenia)

przywitanie_imienne("Jacek") # Wywołanie funkcji z domyslnym argumentem
przywitanie_imienne("Jacek", "szczęścia") # Wywołanie funkcji.
```

Funkcja, która zwraca wartość

```
def przywitanie():
    return "Witaj"

przywitanie() # Wywołanie funkcji.
print(przywitanie() )
a = przywitanie()
print(a)
```

Funkcja z argumentami:

```
def przywitanie_imienne(imie, zyczenia = "zdrowia" ):
    return ("Witaj, " + imie + ". Zycze Tobie " + zyczenia)

print(przywitanie_imienne("Jacek"))
print(przywitanie_imienne("Jacek", "szczęścia"))
```

Funkcja zwraca **None**

```
def przywitanie():
    print ("Witaj")

przywitanie() # Wywołanie funkcji.
print(przywitanie() )
a = przywitanie()
print(a)
```

Funkcja z argumentami:

```
def przywitanie_imienne(imie, zyczenia = "zdrowia" ):
    print ("Witaj, " + imie + ". Zycze Tobie " + zyczenia)

print(przywitanie_imienne("Jacek"))
print(przywitanie_imienne("Jacek", "szczęścia"))
```

Funkcje: dwie funkcje z rożną ilością argumentów nie są możliwe

```
def area(l, b):
  return l * b

def area(r):
  import math
  return math.pi * r ** 2


area(3, 4)
area(7)
```

Funkcja: dla różnych typów argumentów

```
def aFunction(input):
    if type(input) == str:
        print('you gave string as input')
    if type(input) == dict:
        print('you gave a dict as input')

aFunction('test')
aFunction({'one':1, 'two':2})
aFunction(5)
```

Funkcje: domyślny argument

```
def sumsub(a, b, c=0, d=0):
    return a - b + c - d

print(sumsub(12, 4))
print(sumsub(3, 4, 5, 7))

#print(sum(12, 4))
#print(sum(3, 4, 5, 7))

print(sum([12, 4]))
print(sum((3, 4, 5, 7)))
```

Funkcje: *args

- ```
  def sumsub(a, b=0, c=0, d=0):
      return a - b + c - d

  print(sumsub(12, 4))
  print(sumsub(3, 4, 5, 7))
  ```
- ```
  def sumsub1(*values):
      s = 0
      for i in values:
          s+=i
      return s

  print(sumsub1(12, 4))
  print(sumsub1(3, 4, 5, 7))
  ```
- ```
  def sumsub2(*values):
      return sum(values)

  print(sumsub2(12, 4))
  print(sumsub2(3, 4, 5, 7))
  ```
  ```

Funkcje: *args

- ```
  def srednia(*values):
      return (sum(values)) / (len(values))

  print(srednia(2, 3, 4, 6))
  print(srednia(45))
  ```

- ```
  def srednia(coeff, *values):
      return (sum(values*coeff)) / (len(values))

  print(srednia(2, 3, 4, 6))
  print(srednia(2,4))
  ```

Funkcje: **kwargs

- ```
  def f(**kwargs):
  print(kwargs)

  f()
  f(pl="Polish", en="English")
  ```
- ```
  def f(**kwargs):
  print(kwargs)

  f()
  f(**{'pl':"Polish", 'en':"English"})
  ```

Funkcje: rzymski liczby do arabskich

```
def romanian_to_arabic(number):
    romanian_numerals = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D
    n = 0
    # 4 (IV) and 9 (IX), 40 (XL), 90 (XC), 400 (CD) and 900 (CM)
    number = number.replace('IV', 'IIII')
    number = number.replace('IX', 'VIIII')
    number = number.replace('XL', 'XXXX')
    number = number.replace('XC', 'LXXXX')
    number = number.replace('CD', 'CCCC')
    number = number.replace('CM', 'DCCCC')
    #for i in number:
        #n += romanian_numerals[i]
    #return n
    return sum(romanian_numerals[i] for i in number)

print(romanian_to_arabic('MCDLXIV'))
```

Funkcje: arabski do rzymskich, bez słownika

```
def arabic_to_romanian(n):
    num = [1, 4, 5, 9, 10, 40, 50, 90,
           100, 400, 500, 900, 1000]
    sym = ["I", "IV", "V", "IX", "X", "XL",
           "L", "XC", "C", "CD", "D", "CM", "M"]
    i = 12
    s = ''
    while n:
        div = n // num[i]
        n %= num[i]

        while div:
            s += sym[i]
            div -= 1
        i -= 1
    return s


print(arabic_to_romanian(1464))
```

Funkcje: arabski do rzymskich, ze słownikiem

```
#ze slownikiem
def arabic_to_romanian(n):
    romanian_numerals = {1000:'M', 900: 'CM', 500: 'D', 400: 'CD',
100: 'C', 90:'XC',50: 'L', 40:'XL', 10:'X', 9:'IX',
5: 'V' ,4:'IV', 1: 'I'}
    s = ''
    for i in romanian_numerals:
        div = n // i
        n %= i
        while div:
            s += romanian_numerals[i]
            div -= 1
    return s
```

Funkcje rekurencyjne

Funkcje rekurencyjne są to funkcje, których charakterystyką jest to, iż przed zakończeniem bieżącego wywołania funkcji następuje kolejne wywołanie tej funkcji (czyli w ciele danej funkcji następuje wywołanie tejże funkcji).

Ich zawsze można zastąpić pętlą.

Silnia

```
def silnia(n):
    if n:
        return silnia(n-1)*n
    else:
        return 1

print(silnia(5))
print(silnia(0))
```

```
def silnia(n):
    s = 1
    for i in range(1,n+1):
        s*=i
    return s

print(silnia(5))
print(silnia(0))
```

Algorytm Euklidesa

```
def NWD(a,b):
    while b:
        a,b = b, a % b
    return a

print(NWD(255, 30))
```

```
def NWD(a,b):
    while b:
        return NWD (b,a % b)
    return a
```

Ciąg Fibonacciego

```
def Fib(n):
    if n < 2:
        return n
    else:
        return Fib(n-1) + Fib(n-2)

print(Fib(12))
```

```
def Fib1(n):
    if (n == 0):
        return 0
    i0 = 0
    i1 = 1
    for j in range(1,n):
        i0, i1 = i1, i0+i1
    return i1

print(Fib1(12))
```

Dlaczego rekurencja w ciągu Fibonacciego jest złym rozwiązaniem

```
count = 0
def Fib(n):
    global count
    count += 1
    if n < 2:
        return n
    else:
        return Fib(n-1) + Fib(n-2)

print(Fib(12))
print(count)

#print(Fib(30))
#print(count) #2692537
```

Dlaczego rekurencja w ciągu Fibonacciego jest złym rozwiązaniem

```
count = 0
def Fib1(n):
    global count
    if (n == 0):
        return 0
    i0 = 0
    i1 = 1
    for j in range(1,n):
        i0, i1 = i1, i0+i1
        count += 1
    return i1

print(Fib1(30))
print(count) #29
```

Funkcje: import math

Link do dokumentacji https://docs.python.org/3/library/math.html

- ```
  import math

  a=0
  b=math.sin(2*math.pi)
  print(b)
  print(math.isclose(a,b, rel_tol=1e-09, abs_tol=1e-09))
  ```
- ```
  import math

  print(math.e)
  print(math.exp(1))
  print(math.ceil(math.e))
  print(math.floor(math.e))
  print(math.factorial(10))
  ```

Funkcje: import math

Link do dokumentacji https://docs.python.org/3/library/math.html

- ```
  import math

  print(math.gcd())
  print(math.gcd(20,16))
  print(math.gcd(20,15,35,25))
  ```
- ```
  import math

  #print(math.prod())
  #print(math.prod(0,2))
  print(math.prod([0,2]))
  print(math.prod([12,3,21]))
  print(math.prod([12,3,21], start=2))
  ```

Funkcje: potęga

```
import math

print(math.pow(2,100))
print(pow(2,100))
print(2**100)

#print(math.pow(-2,0.5))
print(pow((-2),0.5))
print((-2)**0.5)

print(math.exp(1e-5) - 1)  # gives result accurate to 11 places
print(math.expm1(1e-5) )
```

Oprócz tego `math.exp(x)` dokładniej niż `math.e ** x` oraz `pow(math.e, x)`.

Funkcje: reszta

```
import math

print(math.remainder(8,3))
print(math.fmod(8,3))
print(8%3)

print(math.remainder(8.5,3))
print(math.fmod(8.5,3))
print(8.5%3)

print(math.remainder(-1e-100,1e100))
print(math.fmod(-1e-100,1e100))
print(-1e-100 % 1e100)
```

Funkcje: reszta

```
import math

print(math.remainder(8,3))
print(math.fmod(8,3))
print(8%3)

print(math.remainder(8.5,3))
print(math.fmod(8.5,3))
print(8.5%3)

print(math.remainder(-1e-100,1e100))
print(math.fmod(-1e-100,1e100))
print(-1e-100 % 1e100)
```