Objektno programiranje
⟨ Funkcije, iznimke ⟩

Tihomir Žilić, Mario Essert, Vladimir Milić

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

Zagreb, 2020./2021.

Sadržaj

Funkcije

- Funkcije se mogu svrstati u 3 kategorije: ugrađene (engl. *built-in function*), funkcije u Pythonovim programskim modulima/paketima/bibliotekama i korisničke funkcije (engl. *user-defined function*).
- Korisničke funkcije u Pythonu se po pozivu ne razlikuju od ugrađenih funkcija, a po ustroju odgovaraju funkcijama od kojih su napravljeni svi Pythonovi moduli/paketi/biblioteke.
- Korisničkim funkcijama poželjno je davati različita imena od ugrađenih funkcija i tom prilikom poštivati sva pravila koja vrijede za imena varijabli.

Definicija funkcije

Definicija:

```
def ime(par1,par2,...):
    '''opis funkcije - pozivom help(ime)'''
    tijelo funkcije, tj. naredbe
    return objekt # objekt bilo kojeg tipa
```

- `par1, par2, ...` se zovu formalni parametri ili samo **parametri**, koji se koriste kod poziva funkcija za pridruživanje stvarnim vrijednostima koji se navode kao **argumenti**. U najjednostavnijem slučaju, funkcija nema nikakvih parametara, što znači da kod poziva funkcija ne uzima nikakve argumente.
- Parametri su lokalne varijable funkcije, te svaki poziv funkcije povezuje te varijable s odgovarajućim stvarnim vrijednostima koje pozivatelj navodi kao argumente.

Poziv funkcije

`ime_funkcije(argumenti)`

- `ime_funkcije`: identifikator tj. ime koje pokazuje na objekt funkcije,
- `( )`: zagrade tj. operacija poziva funkcije,
- `argumenti`: objekti (tj. adrese objekata) koji se pri pozivu funkcije povezuju s parametrima funkcije.
- Niz naredbi koji nije prazan, poznatiji kao tijelo funkcije (engl. *function body*), ne izvršava se kada i naredba `def`. Ono se izvršava kasnije, svaki put kada se funkcija poziva.

Primjer:

```
def obrni(x):
    '''Ova funkcija obrce elemente ulaznog niza'''
    L=x[::-1]   # tijelo funkcije
    return L    # objekt L je niz tj.lista, string i sl.

>>> help(obrni)
Help on function obrni in module __main__:

obrni(x)
    Ova funkcija obrce elemente ulaznog niza
>>> obrni("Dobar dan")
    'nad raboD'
>>> obrni([1,3,5,78,"ob"])
    ['ob', 78, 5, 3, 1]
>>> obrni((["L",1+4j],(23,"A"),"ob"))
    ('ob', (23, 'A'), ['L', (1+4j)])
```

Parametri funkcije

- **Pozicijski** `(a, b, c, ...)`: kada je važno njihovo mjesto u redoslijedu.
- **Slijedni** `(*x)`: kada nije važan njihov broj.
- **Imenovani** `(**r)`: kada se želi imati pretpostavljene vrijednosti.
- Generički `(pozicijski, slijedni, imenovani)`: redoslijed parametara bitan.

Napomena: pozicijski moraju biti ispred slijednih i imenovanih parametara.

Pozicijski parametri

Primjer:

```
def plus(a,b):
    rezultat=a+b
    return rezultat

>>> plus(10,7)
    17
>>> plus([1,2,3],["5",6])
    [1, 2, 3, '5', 6]
>>> plus("Ovo je string: ","plus")
    'Ovo je string: plus'
>>> plus("plus","Ovo je string: ")
    'plusOvo je string: '
>>> plus((1,2,3),("5",6))
    (1, 2, 3, '5', 6)
```

A ako je unutar definicije: `rezultat=a-b` ?

Slijedni parametri

Primjer:

```
def srednja(*A):
    rezultat = sum(A)/len(A)
    return rezultat


>>> srednja(3,3,5,5,4,4,2,5,5)
    4.0

>>> ocjene=[3,3,5,5,5]
>>> srednja(*ocjene)  #otpakiravanje
    4.2
```

Imenovani parametri - pretpostavljene vrijednosti

Primjer:

```
def pretpostavljene(a,gr="glagol",sem="agens"):
    return a,gr,sem

>>> pretpostavljene(2,sem=[3,3])
    (2, 'glagol', [3, 3])
```

Imenovani parametri

Primjer:

```
def kljuc(**K):
    rezultat=K.keys()
    return rezultat
```

```
>>> kljuc(ime=2,pr=3)
dict_keys(['ime', 'pr'])

>>> rjecnik={"A":1,"B":5,"2.7":[1,7]}
>>> kljuc(**rjecnik) #otpakiravanje
dict_keys(['A', 'B', '2.7'])
```

Generički parametri

Redoslijed parametara **bitan**: (pozicijski, slijedni, imenovani)

Primjer:

```
def genericki(a,b,*R,**X):
    print("pozicijski: ",a,b)
    print("slijedni: ",R)
    print("imenovani: ",X)

>>> genericki(2,4,6,7,8,9,to=10,ono=11)
pozicijski:  2 4
slijedni:  (6, 7, 8, 9)
imenovani:  {'to': 10, 'ono': 11}


>>> genericki(2,4,6,7,8,9,to=10,a=11)
TypeError: genericki() got multiple
           values for argument 'a'
```

Promjenjive pretpostavljene vrijednosti

Primjer:

```
def f(x,y=[]):
    y.append(x)
    return y

>>> f(23)
[23]
>>> f(42)
[23, 42]
>>> f(3,[4])
[4, 3]
>>> f(5)
[23, 42, 5]
```

Funkcija je objekt

- Funkcija kao **varijabla**:

```
def jutro(ime):
    print("Dobro jutro: %s !" % ime)
>>> jutro("Ivana")
Dobro jutro: Ivana !
>>> x=jutro  # identifikator
>>> x("Marko")
Dobro jutro: Marko !
```

- Funkcija kao **argument**:

```
def F_kao_arg(func,arg):
    func(arg)
>>> F_kao_arg(x,"Matea")
Dobro jutro: Matea !
```

- Funkcija vraća **funkciju**:

```
def F_vraca_F(ar):
    def nova_F(x):
        return "{} je trostruki {}".format(ar,x)
    return nova_F
>>> N=F_vraca_F("Ivo")
>>> N("prvak")
'Ivo je trostruki prvak'
```

Funkcijski prostor imena

- Lokalni prostor imena,
  - lokalne varijable unutar tijela funkcije i nisu dohvatljive izvana.
- Ne-lokalni prostor imena (*keyword* **nonlocal**),
  - lokalne varijable unutar ugnježdene funkcije dohvatljive iz tijela vanjske funkcije.
- Globalni prostor imena (*keyword* **global**),
  - globalne varijable dohvatljive i izvan funkcije.

Lokalni prostor imena

Primjer:

```
def mnozenje(x):
    return 7*x

>>> mnozenje(3)
21
>>> x
NameError: name 'x' is not defined
```

Ne-lokalni prostor imena (ugnježđene funkcije)

Primjer 1:

```
def mnozenje3():
    poruka = "Vani"
    def zbroj():
        poruka = "Unutra"
        print(poruka)
    zbroj()
    print(poruka)

>>> mnozenje3()
Unutra
Vani
```

Primjer 2:

```
def mnozenje3():
    poruka = "Vani"
    def zbroj():
        nonlocal poruka
        poruka = "Unutra"
        print(poruka)
    zbroj()
    print(poruka)

>>> mnozenje3()
Unutra
Unutra
```

Globalni prostor imena

Primjer 1:

```
def mnozenje1(x):
    return a*x
>>> a=5
>>> mnozenje1(7)
35
>>> x
NameError:
name 'x' is not defined
```

Primjer 1 (loše), bolje je:

```
def mnozenje4(a,x):
    return a*x
>>> b=5
>>> mnozenje4(b,7)
35
>>> mnozenje4(3,7)
21
>>> a
NameError:name 'a' is not defined
>>> x
NameError:name 'x' is not defined
```

Primjer 2:

```
def mnozenje2(x):
    global z
    z=x
    return a*z
>>> a=3
>>> mnozenje2(7)
21
>>> x
NameError:name 'x' is not defined
>>> z
7
```

Anonimna funkcija - lambda funkcija

```
lambda parametri:  izrazi
```

Primjer:

```
a=lambda B,C: B*C+5
>>> a(9,8)
77
```

Isto kao:

```
def a(B,C):
    return B*C+5
>>> a(9,8)
77
```

Inline definiranje i pozivanje:

```
>>> (lambda B, C: B*C+5)(9, 8)
77
```

Napomena: **lambda** je ključna riječ.

Rekurzivna funkcija - funkcija poziva samu sebe

Primjer:

```
k_fac = 1
xk = 1
def my_exp2(x, N):
    global k_fac, xk
    if N == 0:
        return 1
    else:
        ek = my_exp2(x, N-1)
        k_fac = k_fac*N
        xk = xk*x
        s = ek+xk/k_fac
        return s
```

Dekoratori - funkcija koja mijenja originalnu funkciju

```
@dekorator
def original(): ...
```

ili

```
def original(): ...
original = dekorator(original)
```

```
# DEFINIRANJE DEKORATORA
def dekorator(original):
    def wrapper():
        izmijenjena = ... promjena originala...
        return izmijenjena
    return wrapper                  # vraca izmijenjenu funkciju

# PRIMJENA DEKORATORA:
#1. ILI
@dekorator                          # ispred funkcije originala
def original():
    return ...nesto...
#2. ILI
def original():
    return ...nesto...
original=dekorator(original)        # nakon funkcije originala
```

Dekoratori - primjer

```
""" dekorator dekor koji mijenja originalnu funkciju reciHej()
tako sto se stavi ili @dekor ispred ili reciHej=dekor(reciHej)
iza definicije funkcije reciHej"""

#DEKORATOR
def dekor(func):  # ulazi originalna funkcija
    def wrapper(k):
        #print(k)   #ispis argumenta funkcije func pri njenom pozivu
        originalna=func(k)
        izmijenjena=originalna*5
        return  izmijenjena  #izmijenjena ulazna funkcija
    return wrapper  # vraca novu funkciju

#IZMIJENJENA
@dekor     # ispred funkcije koju modificira
def reciHej(x):     #originalna za ulaz "Hej!" vraca: "Hej!Hej!"
    return str(x)*2
#reciHej=dekor(reciHej)  # ili iza funkcije koju modificira

#sada pozivom reciHej('Hej!') u biti pozivamo wrapper('Hej!')
print("Izmjena: ",reciHej('Hej!'))
[Out] Izmjena:  Hej!Hej!Hej!Hej!Hej!Hej!Hej!Hej!Hej!Hej!
```

[0] [Out] označava ispis na zaslon interpretera

Dokumentiranje funkcije: imefunkcije.__doc__

```
def obrni(x):
    '''
    Ova funkcija obrce elemenate ulaznog niza

    Ulazni parametar je niz (lista, n-terac, string)
    Izlazni objekt je takodjer niz
    '''
    L=x[::-1]
    return L

>>> print(obrni.__doc__)
    Ova funkcija obrce elemenate ulaznog niza

    Ulazni parametar je niz (lista, n-terac, string)
    Izlazni objekt je takodjer niz
>>> help(obrni)
Help on function obrni in module __main__:
obrni(x)
    Ova funkcija obrce elemenate ulaznog niza

    Ulazni parametar je niz (lista, n-terac, string)
    Izlazni objekt je takodjer niz
```

Pogreške (*engl.* errors) pri programiranju

- **Sintaktičke pogreške** (netočna sintaksa)

```
>>> while True print(2+2)
File "<pyshell>", line 1
    while True print(2+2)
                     ^
SyntaxError: invalid syntax
```

- **Pogreške u radu** (neispravan rad s objektima)

```
>>> 3+2/0
Traceback (most recent call last):
    File "<pyshell>", line 1, in <module>
ZeroDivisionError: division by zero
```

- **Logičke pogreške** (neispravan algoritam)

```
>>> x,y=3,4
>>> prosjek=x+y/2
>>> print(prosjek)
5.0
```

Iznimke

Iznimka: događaj koji se podiže nakon pojave **sintaktičke pogreške** kao i **pogreške u radu**.

Dijagnostička poruka sadrži:

- *traceback* - početak poruke,
- liniju koda i njen broj
- tip pogreške i njen opis

npr.

```
>>> "2"+2
Traceback (most recent call last):
    File "<pyshell>", line 1, in <module>
TypeError: must be str, not int
```

Tipovi iznimki

Sve iznimke nastaju iz osnovne klase **BaseException**. Unutar nje postoji podjela na:

- SystemExit
- KeyboardInterrupt
- GeneratorExit
- Exception
  - ArithmeticError
    - FloatingPointError
    - OverflowError
    - ZeroDivisionError
  - NameError
  - TypeError
  - ValueError
  - SystemError
  - SyntaxError
  - ...

Rad s iznimkama, dohvaćanje i obrada

`try-except-else` blok naredbe.

```
try:
    Naredbe
    raise Iznimka3("tekst iznimke")
    ........................
except Iznimka1:
    Ako se pojavi Iznimka1, izvrsi ovaj blok.
except (Iznimka2,Iznimka3) as detalj:
    Ako se pojavi Iznimka2 ili Iznimka3,
    izvrsi ovaj blok.
    ........................
else:
Ako se ne pojavi iznimka, izvrsi ovaj blok.
```

Rad s iznimkama, dohvaćanje i obrada

- "Ručno" podizanje iznimke može se naredbom `raise`
- Varijabla `detalj` nakon ključne riječi **`as`** sadrži tekst podignute iznimke.
- Blok `else` se izvršava ako u `try` bloku nije podignuta iznimka.
- Ignoriranje podignutih iznimki ostvaruje se ključnom riječju **`pass`** unutar `except` bloka.
- Obuhvaćanje svih iznimki ako se koristi samo `except:` tj. bez iznimki i argumenata.
- Blok `else` nije nužan.
- Grupiranje više iznimki unutar jednog bloka `except` ostvaruje se koristeći n-terac

Primjer try-except-else

```
try:
    x = 0.5
if x < 0.5:
    raise ValueError("premala vrijednost")
except (NameError,ValueError,TypeError) as detalj:
    print("Greska u vrijednosti, imenu ili tipu:",detalj)
else:
    print(x)

>>> Greska u vrijednosti, imenu ili tipu: premala vrijednost
>>> 0.5
>>> Greska u vrijednosti, imenu ili tipu: '<' not
supported between instances of 'str' and 'float'
>>>  File "try_except_else.py", line 2
     x = "str
         ^
SyntaxError: EOL while scanning string literal
```

U primjeru `x` ima redom vrijednosti **`0.4`**, **`0.5`**, **`"str"`**, **`"str`**

Rad s iznimkama, dohvaćanje i obrada

`try-except-finally` blok naredbe.

```
try:
    Naredbe
    ......................
except Iznimka1:
    Ako se pojavi Iznimka1, izvrsi ovaj blok.
    ......................
finally:
    Uvijek izvrsi ovaj blok.
```

`finally` se izvšava bez obzira je li iznimka podignuta ili nije u `try` bloku.
Napomena: `finally` i `else` ne mogu zajedno.

Primjer try-except-finally

```
try:
    x = float(input("Unesi broj: "))
    inverse = 1.0 / x
except ValueError:
    print("Tvoj broj treba biti int ili float")
except ZeroDivisionError:
    print("INF")
finally:
    print("Iznimka se ili pojavila ili nije.")
>>> Unesi broj: 0
INF
Iznimka se ili pojavila ili nije.
>>> Unesi broj: 13
Iznimka se ili pojavila ili nije.
>>> Unesi broj: "abs"
Tvoj broj treba biti int ili float
Iznimka se ili pojavila ili nije.
```