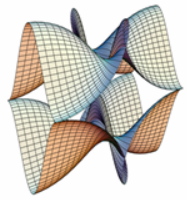

Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAZVOJ WEB-APLIKACIJA

Osnove Pythona

3. listopada 2025.

Sastavio: Zvonimir Bujanović

UVOD

Python

Python je programski jezik opće namjene.

Iznimno je popularan zbog svoje jednostavnosti, univerzalnosti primjene, te brojnih proširenja koja su većinom otvorenog koda.

Brojnim paketima moguće je jako proširiti funkcionalnost, čime Python omogućava:

- brzi razvoj i prototipiranje aplikacija
- izvođenje kompleksnih proračuna i simulacija
- analizu podataka i primjenu u strojnom učenju
- vizualizaciju rezultata
- web-programiranje

Trenutno aktualna verzija je Python 3.13.

U ovom kolegiju ćemo koristiti Pythonovu biblioteku Flask.

- Potrebno je samo minimalno poznavanje Pythona.

Pokretanje programa u Pythonu

Python i pripadne biblioteke se obično koriste iz virtualne okoline.

- Na jednom računalu možemo postaviti više virtualnih okolina, svaku sa svojom verzijom Pythona i svojom kolekcijom biblioteka.
- Koristit ćemo alat mamba za upravljanje virtualnim okolinama.
- Za instalaciju radne okoline za Flask, slijedite postupak naveden na web-stranici kolegija.

Programe u Pythonu spremamo u datoteke s ekstenzijom `py`.

- Aktiviramo virtualnu okolinu u konzoli.
- Pokrenemo program sa `python program.py`.

Uočite:

- Za razliku od C/C++, Python je interpretirani jezik.
- To znači da programe nije potrebno kompajlirati prije izvođenja.
- No izvođenje će zbog toga biti sporije, jer se programski kod "u letu" prilikom izvođenja prevodi liniju po liniju u strojni jezik.

OSNOVE PROGRAMIRANJA U PYTHONU

Tipovi podataka

Python je slabo tipiziran jezik. Nije potrebno deklarirati varijable; pojedine varijable mogu mijenjati svoj tip “u letu”:

```
1 a = [1, 2, 3]; print(type(a));
2 a = 'pero'; print(type(a));
3 a = (1, 2); print(type(a));
```

```
<class 'list'>
<class 'str'>
<class 'tuple'>
```

Osnovni tipovi podataka

Standardni tipovi podataka: `bool`, `int`, `float`, `str`, `list`, `tuple`, `dict`.

- `True`, `False` – logičke vrijednosti (`bool`)
- `7` – cijeli broj (`int`)
- `0.314` ili `3.14e-1` – realni broj (`float`, 64-bitni)
- `-3+0.5j` – kompleksni broj
- `'iter'` ili `"iter"` – string (tip `str`, 1D polje znakova)
- `'x'` ili `"x"` – također string (tip `str`)
- `[1, 2, 3]` – lista (tip `list`)
- `(1, 2)` – n-torka (tip `tuple`)
- `{'ime': 'Pero', 'ocjena': 5}` – rječnik (tip `dict`)

- `print(type(x))` – ispis tipa varijable `x`
- `del(x)` – brisanje varijable `x`.

Rad sa stringovima (1)

```
1 s = "Hello world";
2 print( len(s) );
3
4 s2 = s.replace( "world", "test" );
5 print( s2 );
```

```
11
Hello test
```

```
1 print( s[:5] );
2 print( s[6:] );
3 print( s[::2] );
```

```
Hello
world
Hlowrd
```

```
1 s3 = "Hello" + "world";
2 s3[3] = "x"; # Ovo NE radi, ne možemo mijenjati znakove u stringu!
```

Rad sa stringovima (2)

Iznimno korisni su tzv. f-stringovi.

```
1 x = 2; y = "abc";
2 print( f"Vrijednost od x je {x}, a od y je {y}." );
```

```
Vrijednost od x je 2, a od y je abc.
```

Rad s listama (1)

```
1 l = [1, 2, 3, 4];
2 print( l[1:3] );
```

```
[2, 3]
```

```
1 l2 = [1, 'a', 1.0, 1-1j];
2 l3 = [1, [2, [3, [4, [5]]]]];
```

```
1 s2 = list( s2 );
2 print( s2 );
```

```
['H', 'e', 'l', 'l', 'o', ' ', 't', 'e', 's', 't']
```

Rad s listama (2)

```
L = [];
L.append( "A" );
L.append( "d" );
L.append( "d" );
print( L );
```

```
['A', 'd', 'd']
```

```
L[1] = 'A';
print( L );
```

```
['A', 'A', 'd']
```

```
L.remove( "A" );
print( L );
```

```
['A', 'd']
```

Rad sa n-torkama (tuples)

n-torke su kao liste, ali im se elementi ne smiju mijenjati/dodavati.

```
1 point = (10, 20);
2 point[0] = 20;
```

```
TypeError: 'tuple' object does not support item assignment
```

Rad s rječnicima (dict)

Ključevi su obično int-ovi ili stringovi, vrijednosti mogu biti bilo što.

```
student = { "ime": "Pero", "ocjena": 5, 3: "nesto" };
print("Student " + student["ime"] + " ima ocjenu " + str(student["ocjena"]));

# Lakši ispis pomoću f-stringa:
print( f"Student {student["ime"]} ima ocjenu {student["ocjena"]}" );
```

```
Student Pero ima ocjenu 5
Student Pero ima ocjenu 5
```

Kontrola toka

```
izjava1 = False;
izjava2 = False;

if izjava1:
    print( "izjava1 je True" );
elif izjava2:
    print( "izjava2 je True" );
else:
    print( "izjava1 i izjava2 su False" );
```

```
izjava1 i izjava2 su False
```

```
if not izjava1:
    if not izjava2:
        print( "izjava1 i izjava2 su False" )
```

```
izjava1 i izjava2 su False
```

Petlje (1)

```
1 for x in range(4): # range počinje od 0
2     print(x)
```

```
0
1
2
3
```

```
1 for broj in [ "jen", "dva", "tri" ]:
2     print(word)
```

```
jen
dva
tri
```

```
1 params = { "prvi": 3, "drugi": "osam", 5: "blabla"}
2 for key, value in params.items():
3     print( f"{key} -> {value}" );
```

```
prvi -> 3
drugi -> osam
5 -> blabla
```

Petlje (2)

Python je **jako** osjetljiv na razmake.
Tijelo if-a, petlji, funkcija, klasa itd. mora biti odmaknuto jednim tab-om ili 4 razmaka.

```
1 i = 0;
2 while( i < 2 ):
3     print( i );
4     i = i + 1;
5
6 print( "gotovo" );
```

```
0
1
gotovo
```

Funkcije

```
def zero():
    return 0;

print( zero() );
```

```
0
```

Sljedeća funkcija vraća n-torku (tuple).

```
def powers(x):
    """
    Potencije od x.
    """
    # Može i return (x ** 2, x ** 3, x ** 4);
    return x ** 2, x ** 3, x ** 4;

# Može i (x2, x3, x4) = powers( 3 ) ili [x2, x3, x4] = powers( 3 );
x2, x3, x4 = powers( 3 );
print( x3 );
```

```
27
```

Klase (1)

```
1  class Point:
2      """
3      Jednostavna klasa koja služi za rad s točkama u Euklidskoj ravnini.
4      """
5      def __init__( self, x, y ):
6          # Konstruktor. Umjesto "this" je u Pythonu običaj koristiti "self".
7          self.x = x;
8          self.y = y;
9
10     def translate( self, dx, dy ):
11         # Članska funkcija.
12         self.x += dx;
13         self.y += dy;
14
15     def __str__( self ):
16         # Konverzija varijable tipa Point u string.
17         return( f"Point at [{self.x:5.3f}, {self.y:5.3f}]" );
18
19 p1 = Point( 1, 1 );
20 p1.translate( 2, 3.5 );
21 print( p1 ); # Point at [3.000, 4.500]
```

Klase (2)

Nasljeđivanje.

```
1  # Paket za apstraktne klase (ABC) i metode.
2  from abc import ABC, abstractmethod;
3
4  # Bazna klasa za apstraktne klase je uvijek ABC.
5  class Lik( ABC ):
6      @abstractmethod
7      def opseg(self):
8          pass # Ne treba implementacija apstraktne metode!
9
10 # Bazna klasa za Trokut je Lik.
11 class Trokut( Lik ):
12     def __init__( self, a, b, c ):
13         self.a = a; self.b = b; self.c = c;
14
15     def opseg(self):
16         return self.a + self.b + self.c;
17
18 t = Trokut( 3, 4, 5 );
19 print( t.opseg() );
```

`@abstractmethod` je tzv. dekorator (više o tome kasnije).

Klase (3)

Enkapsulacija:

- imena protected članova počinju sa _ (ovo je samo konvencija).
- imena private članova počinju sa __.

```
1  class Osoba:
2      def __init__( self, ime ):
3          self.__ime = ime; # private član
4
5      def get_ime( self ):
6          return self.__ime;
7
8  class Student( Osoba ):
9      def __init__( self, ime, JMBAG ):
10         super().__init__( ime ); # poziv baznog konstruktora!
11         self._JMBAG = JMBAG;     # protected član
12
13 pero = Student( 'pero', '1234567890' );
14
15 print( pero.get_ime() ); # pero
16 print( pero.__ime );     # greška: pristup privatnom članu!
17 print( pero._JMBAG );    # 1234567890, OK iako pristupamo protected članu!
```

Klase (4)

Statički članovi i metode.

```
1  class Student:
2      broj_studenata = 0; # Statički član klase, inicijalizacija.
3
4      def __init__( self, JMBAG ):
5          self.JMBAG = JMBAG; # Član koji pripada ovoj instanci.
6          Student.broj_studenata += 1;
7
8      # Funkcija koja pripada instanci.
9      def vrati_JMBAG( self ):
10         return self.JMBAG;
11
12     # Statička funkcija - koristimo dekorator. Nema self!
13     @staticmethod
14     def koliko_studenata():
15         return Student.broj_studenata;
16
17 pero = Student( '1234567890' ); ana = Student( '9876543210' );
18
19 print( pero.vrati_JMBAG() );          # 1234567890
20 print( Student.broj_studenata );      # 2
21 print( Student.koliko_studenata() ); # 2
22 print( pero.broj_studenata );         # 2
```

Moduli (1)

Dio koda možemo spremiti u datoteku i kasnije koristiti kao modul. Neka je ovo spremljeno u datoteku `mojmodul.py`:

```
"""
mojmodul.py
Primjer modula. Sadrži varijablu x, funkciju f te klasu K.
"""
x = 0;

def f():
    return x + 1;

class K:
    def __init__( self ):
        self.clan = x + 2;

    def get_clan( self ):
        return self.clan;
```

Moduli (2)

Korištenje modula s prethodnog slajda:

```
1 import mojmodul;
2
3 print( mojmodul.x );
4 print( mojmodul.f() );
5 k = mojmodul.K(); print( k.get_clan() );
```

```
0
1
2
```

Umjesto `modmodul`, možemo koristiti alternativno ime:

```
1 import mojmodul as mm;
2
3 print( mm.x );
4 print( mm.f() );
5 k = mm.K(); print( k.get_clan() );
```

Moduli (3)

Možemo i sve iz modula koristiti direktno, bez ikakvog prefiksa (ovo nije dobra praksa):

```
1 from mojmodul import *;
2
3 print( x );
4 print( f() );
5 k = K(); print( k.get_clan() );
```

Moduli (4)

Kod aplikacija u Flasku ćemo imati ovakvu situaciju:

- Glavna datoteka u korijenskom direktoriju je `app.py`.
- U poddirektoriju `util` imamo modul `nesto.py` sa funkcijom `f`.
- U poddirektoriju `blabla` imamo modul `drugo.py` u kojem želimo koristiti funkciju `f`.

To će funkcionirati ako u `blabla/drugo.py` napišemo

```
1 from util.nesto import f;
2 f();
```

ili

```
1 from util import nesto;
2 nesto.f();
```

ili

```
1 import util;
2 util.nesto.f();
```

Dekoratori

Dekoratorom osiguravamo da se neki kod izvrši prije ili poslije neke funkcije.

```
1  # Definicija dekoratora
2  def dekorator( func ):
3      def wrapper():
4          print( "Ovo se ispiše prije!" );
5          func();
6          print( "Ovo se ispiše poslije!" );
7
8      return wrapper;
9
10 # Sljedeći kod je zapravo pokrata za "pozdrav=dekorator(pozdrav)"
11 @dekorator
12 def pozdrav():
13     print( "Bok!" );
14
15 pozdrav();
```

```
Ovo se ispiše prije!
Bok!
Ovo se ispiše poslije!
```