



CS324 - SKRIPTING JEZICI

Strukture podataka i funkcije

Lekcija 04

PRIRUČNIK ZA STUDENTE

CS324 - SKRIPTING JEZICI

Lekcija 04

STRUKTRURE PODATAKA I FUNKCIJE

- ✓ Struktrure podataka i funkcije
- ✓ Poglavlje 1: Strukture podataka
- ✓ Poglavlje 2: Funkcije
- ✓ Poglavlje 3: Lambda funkcije
- ✓ Poglavlje 4: Parametri i povratne vrednosti
- ✓ Poglavlje 5: Ugrađene funkcije
- ✓ Poglavlje 6: Pokazna vežba #4
- ✓ Poglavlje 7: Invidivualna vežba #4
- ✓ Poglavlje 8: Domaći zadatak #4
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

UVOD U ČETVRTU LEKCIJU

Uvod u lekciju #4

U četvrtoj lekciji nastavlja se sa proceduralnim programiranjem u Python 3.x jeziku.

Dok je prethodna lekcija opisala numeričke promenljive, stringove, imenike i druge tipove podataka, kao i kontrolu toka i petlje, u ovoj lekciji biće reči o strukturama podataka, funkcijama, njihovim parametrima i povratnim vrednostima, kao i nekim ugrađenim funkcijama u Python 3.x programskom jeziku.

U opštem slučaju, strukture podataka se mogu podeliti u postojeće (en. **built-in**) strukture, i u strukture koje sami korisnici definišu (en. **user-defined**).

Postojeće strukture podataka u Python 3.x jeziku jesu zapravo tipovi podataka koji sadrže više od jedne vrednosti, a koje su obrađivani u prethodnoj lekciji. To su liste, imenici, i Tuple-ovi. Sa druge strane, najčešće korišćene strukture podataka koje korisnik pravi jesu stekovi (en. **stack**), redovi (en. **queue**), stabla (en. **tree**), lančane liste (en. **linked lists**) i grafovi (en. **graphs**).

U ovoj lekciji detaljno će biti objašnjeni načini kreiranja korisničkih struktura podataka, a u sledećoj prelazi se na objektno-orijentisano programiranje u Pythonu 3x. jeziku.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 1

Strukture podataka

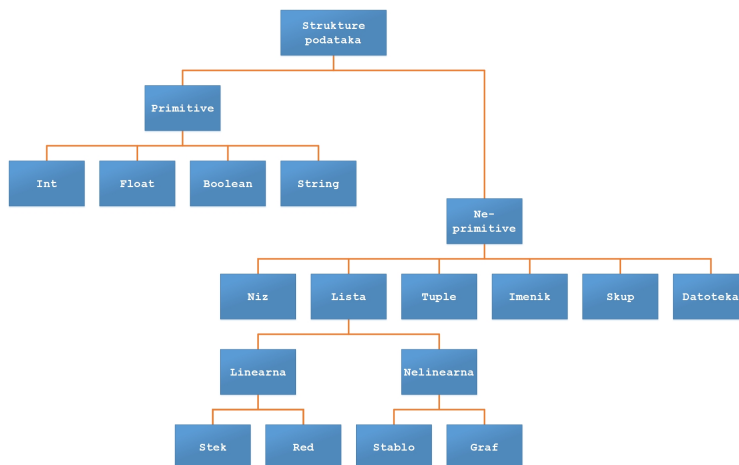
UVOD U STRUKTURE PODATAKA

Strukture podataka mogu u sebi sadržati iste ili različite tipove podataka.

Strukture podataka (en. **data structures**) jesu upravo to: strukture koje mogu sadržati podataka u sebi. Koriste se za smeštanje skupa povezanih podataka.

Strukture podataka se u opštem slučaju, pa tako i u Python jeziku dele na ugrađene skupove podataka, ali i na skupove podataka koje korisnik definiše. U nastavku biće reči o obe vrste struktura podataka.

Neke od struktura podataka su već obrađene, samo nisu formalno tako nazvani - a to su tipovi podataka, prvenstveno oni koji sadrže samo jednu vrednost. U opštem slučaju, na sledećoj slici data je opšta podela struktura podataka.



Slika 1.1.1 Podela struktura podataka u Python 3.x jeziku. [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ 1.1 Primitive u Python jeziku

PRIMIVITE U PYTHONU

Primitive jesu tipovi podataka koji svoja imena mapiraju na jedan objekat u memoriji.

Definicija

Primitivna struktura podataka (en. **primitive data structure**) jesu takve strukture podataka koje programski jezik ima ugrađenu podršku, i koje sadrže uglavnom jednu vrednost, u zavisnosti od programske jezika.

U Python 3.x jeziku, primitive jesu tipovi podataka koji svoja imena mapiraju na jedan objekat u memoriji. Primitive jesu takvi tipovi podataka koji programski jezik ne dozvoljava naknadno menjanje način funkcionisanja od strane programera.

Pitanje:

Zbog čega programski jezici ne dozvoljavaju modifikaciju primitiva?

Da li postoji programski jezik koji dozvoljava modifikaciju primitiva?

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Primitive u Python 3.x programskom jeziku su

- Logički tip podataka (en. **boolean**, **<bool>**)
- Celobrojni tip (en. **integer**, **<int>**)
- Razlomljeni tip podataka (en. **float**, **<float>**)
- Znakovni niz ili string (en. **string**, **<str>**)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

IMPLICITNA KONVERZIJA PRIMITIVA

U Python-u je moguće izvršiti konverziju tipa podataka nad primitivnim tipovima podataka.

U Pythonu je moguće izvršiti konverziju tipa podataka nad primitivnim tipovima podataka.

Postoje dva načina konverzije tipa podataka: implicitno i eksplicitno.

Implicitna konverzija

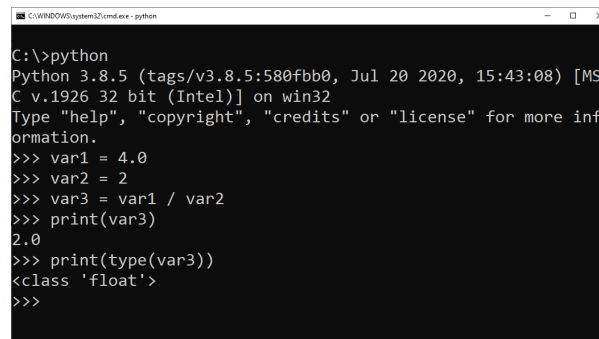
Implicitna konverzija zapravo automatski vrši konverziju prilikom interpretacije koda. Nije potrebno da se "ručno" unese u koji se tip podataka promenljiva želi konvertovati.

Primer:

Implicitna konverzija *int* u *float* deljenjem.

```
var1 = 4.0 #Float
var2 = 2    #int
var3 = var1 / var2
print(var3)
print(type(var3))
```

Rezultat deljenja razlomljenog tipa i podataka i celobrojnog tipa biće takođe razlomljeni (float) tip.



```
C:\>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MS
C v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more inf
ormation.
>>> var1 = 4.0
>>> var2 = 2
>>> var3 = var1 / var2
>>> print(var3)
2.0
>>> print(type(var3))
<class 'float'>
>>>
```

Slika 1.2.1 Implicitna konverzija deljenjem [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

EKSPPLICITNA KONVERZIJA PODATAKA

U Python-u je moguće izvršiti eksplicitnu konverziju tipa podataka, ali ne uvek.

EksPLICITNA konverzija

EksPLICITNA konverzija tipa podataka se vrši "ručno" tj. programer mora interpreteru da napiše naredbu konverzije tipa.

Interpreter ne može sam da shvati šta je programer hteo da postigne, pogotovo kada promenljive uključuju i stringove. Zbog toga programer mora biti obazriv oko konverzije.

Primer:

EksPLICITNA konkatencija stringa sa celobrojnim podatkom.

Sledeći kod javiće grešku:

```
var1 = "Odgovor na pitanja o suštini života je "  
var2 = 42  
var3 = var1 + var2  
print(var3)
```

Greška koja se javlja jeste greška tipa podataka, i javlja se jer ne možemo spojiti string i int u novi string.

Zbog toga je potrebno najpre eksplicitno izvršiti konverziju tipa:

```
var1 = "Odgovor na pitanja o suštini života je "  
var2 = 42  
var3 = var1 + str(var2)  
print(var3)
```

Funkcijom **str()** čiji je argument var2, vrednost promenljive var2 (koja je bila int) konvertujemo u string.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

✓ 1.2 Ne-primitive u Python jeziku

NE-PRIMITIVE U PYTHON-U

Nizovi u Python 3.x programskom jeziku predstavlja efikasan način da se smeštaju podaci istog tipa.

Definicija

Ne-Primitivna struktura podataka (en. **non-primitive data structure**) jesu takve strukture podataka koje sadrže više vrednosti. Zapravo, u ovakvim strukturama zadržan je skup podataka.

U opštem slučaju, ovakve strukture podataka dele se na:

- Nizove (en. **arrays**)
- Liste (en. **lists**)
- Datoteke (en. **files**)

Nizovi

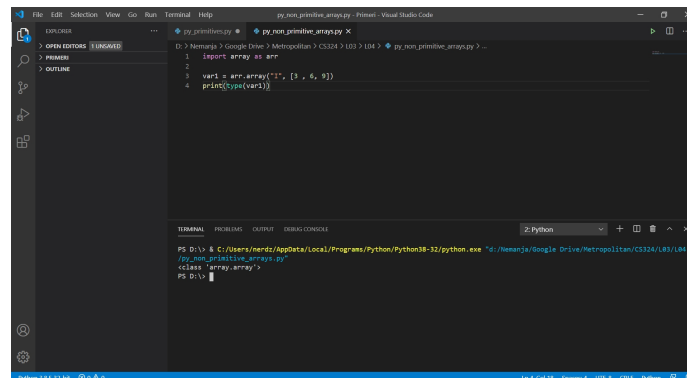
Nizovi u Python 3.x programskom jeziku predstavlja efikasan način da se smeštaju podaci istog tipa.

Svaki element niza je istog tipa podataka.

U Python 3.x programskom jeziku se nizovi kao tip podataka ne koristi često, već se umesto toga koristi ne-primitiva lista. Ukoliko želimo koristiti niz, mora se uvesti iz modula za rad sa nizovima.

```
import array as arr

var1 = arr.array("I", [3, 6, 9])
print(type(var1))
```



Slika 1.3.1 Uvoz modula array i imenovanje novog niza celobrojnih podataka. [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

LISTE I MANIPULACIJA LISTI

Dat je pregled najčešće korišćenih metod za rad sa listama i elementima lista

Liste u Python 3.x jeziku se koriste za smeštanje skupa heterogenih podataka (podataka koje ne moraju biti istog tipa). Lista su ugrađene u Python programski jezik.

Liste se mogu modifikovati, tj. može im se menjati sadržaj a da im se ne menja identitet tj. ime.

Liste se mogu najbolje prepoznati po srednjim zagradama, a elementi liste su razdvojeni zapetom.

Liste, naravno, mogu sadržati homogene podatke, i na taj način predstavljaju nizove.

Ono što nije do sada rečeno jeste da Python sadrži mnoštvo metoda za rad sa listama. U nastavku biće reči nekim najčešće korišćenim metodama za rad sa listama.

Primer:

Postoje sve liste, jedna sa brojevima, a druga karakterima

```
list_num = [1, 2, 3, 6, 9, 43, 90, 27, 333]
list_char = ['s', 'k', 'r', 'i', 'p', 't', 'i', 'n', 'g']
```


Dodavanje elementa

Element se može dodati (po **default**-u) na kraj liste metodom **.append()**

```
list_num.append(324)
```

Element se može dodati na željenu poziciju metodom **.insert()**, gde se navodi pozicija, i sama vrednost.

```
list_num.insert(0, 749)
```

Oduzimanje elementa

Element se može oduzeti metodom **.remove()**, gde unosimo vrednost element kojeg želimo da uklonimo. Ukoliko lista ima vi[e istih elemenata, ukloniće prvi na koji naiđe.

```
list_char.remove('i')
```

Element sa određenim indeksom se može ukloniti korišćenjem metode **.pop()**, gde se navodi indeks (pozitivni ili negativni) elementa kojeg uklanjamo.

```
list_char.pop(-2)
```

METODE ZA RAD SA LISTAMA

Dat je pregled najčešće korišćenih metod za rad sa listama

Sortiranje listi

Python podržava sortiranje listi metodama **.sort()** (za sortiranje po rastućem redosledu) i **.reverse()** (za sortiranje po opadajućem redosledu)

```
list_num.sort()
list_num.reverse()
```

Spajanje elemenata liste u string

Pojedinačni elementi liste mogu se spojiti u jedan string korišćenjem **''.join()** metode pri pozivu stringa.

```
predmet_naziv = ''.join(list_char)
print(predmet_naziv)
```

Dodavanje više elemenata u listu

Metodom **.append()** može se dodati element po element u listu, dok metodom **.extend()** mogu se dodati više elemenata odjednom.

```
list_num.extend([4, 5])
```

Pitanje:

Kako dodati više elemenata listi korišćenjem `.append()`?

Šta bi se desilo ako za dati primer ukucamo `list_num.append([4, 5])`?

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

STEKOVI I REDOVI

Stekovi i redovi se implementiraju kroz liste

Stek (en. **stack**) sadrži objekte koji se ubacuju i izbacuju po sistemu *Last-in-First-Out* ili LIFO.

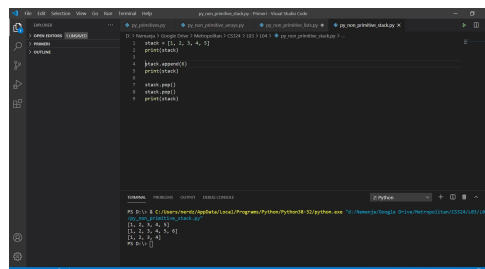
U računarstvu stekovi se koriste za računanje naredbi i sintaksno parsiranje, za algoritme i rutine raspoređivanja.

Stekovi se mogu implementirati u Python 3.x jeziku preko listi. Kada se element doda u stek, to se naziva operacija **push**, a kada se ukloni element, to je **pop** operacija.

Primer: Dodavanje i uklanjanje elemenata u steku

```
stack = [1,2,3,4,5]
stack.append(6)
print(stack)
```

```
stack.pop()
stack.pop()
print(stack)
```



Slika 1.3.2 Dodavanje i oduzimanje elementa steka. [Izvor: Autor]

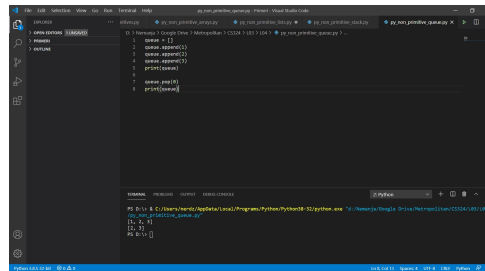
Red (en. **queue**) sadrži objekte koji se ubacuju i izbacuju po sistemu *First-In-First-Out* ili FIFO.

Red se ne može efikasno implementirati u Python 3.x jeziku kao stek, jer push/pop naredbe stavljaju elemente na kraj liste, a pri svakom ulazu/izlazu, potrebno je promeniti indekse elemenata.

Međutim, moguće je implementirati kroz liste tako što bi uz metodu `.pop()` uvek bio argument 0.

Primer: Dodavanje i uklanjanje elemenata u redu

```
queue = []  
queue.append(1)  
queue.append(2)  
queue.append(3)  
print(queue)  
queue.pop(0)  
print(queue)
```



Slika 1.3.3 Dodavanje i oduzimanje elementa reda. [Izvor: Autor]

PRIMER ZA STEKOVE I REDOVE

Implementacije steka je nešto jednostavnije nego redova, ali i redovi se lako implementiraju pomoću listi u Python jeziku.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

GRAFOVI

Graf se sastoji od čvorova i veza između čvorova. Može biti neusmeren ili usmeren.

Graf (en. **graph**) u matematici i računarskoj tehnici se sastoji od čvorova (en. **nodes**) koji mogu biti međusobno povezani.

Ukoliko postoji smer kako je koji čvor povezan sa drugim, onda je to usmeren graf.

Mnogi problemi u matematici i računarskoj tehnici, ali i šire, se mogu predstaviti kroz grafove. Nauka matematike koja se bavi proučavanjem grafova jeste teorija grafova (en. **graph theory**).

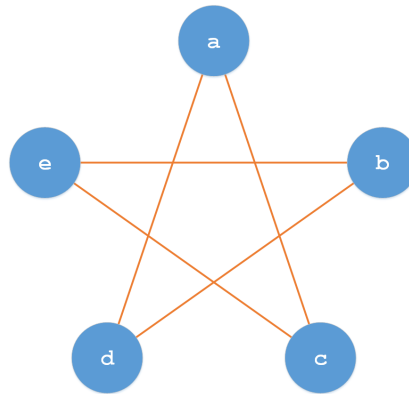
Čvor opisuje iz god se grafa dolazi u taj čvor, kao i ka kom (ili kojim) čvorovima se može doći iz tog čvora.

Grafovi u Pythonu

Graf ne postoji kao ugrađen tip, već se može realizovati kao posebna struktura, kao klasa, ili preko lista ili imenika.

Primer: Graf preko imenika

Realizovati graf sa slike preko imenika, tako da ključ bude ime čvora (string), a vrednost bude lista stringova sa povezanim čvorovima.



Slika 1.3.4 Graf sa 5 čvorova. [Izvor: Autor]

```
graph = { "a" : ["c", "d"],  
          "b" : ["d", "e"],  
          "c" : ["a", "e"],  
          "d" : ["a", "b"],  
          "e" : ["b", "c"]  
        }
```

STABLA

Stabla je najbolje realizovati kroz korisničke strukture, ili kroz klasu

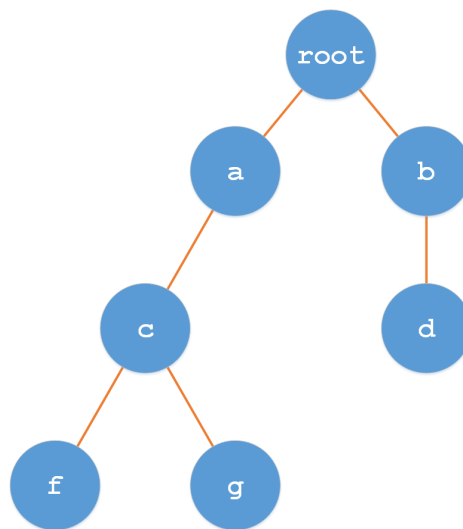
Stablo (en. **tree**) u matematici i računarskoj tehnici predstavlja strukturu podataka koja se takođe sastoji od čvorova, ali su hijerarhijski povezani.

Čvor korena (en. **root node**) je takav čvor da od njega kreću veze ka ostalim čvorovima.

Svi ostali čvorovi imaju roditeljski čvor (en. **parent node**) a mogu imati i decu čvorove (en. **child node**).

Binarno stablo

Binarno stablo je poseban tip stabla, kod kojeg svaki čvor ima veze ka dva čvora u nižem stepenu hijerarhije.



Slika 1.3.5 Stablo. [Izvor: Autor]

Slika 1.3.6 Binarno stablo. [Izvor: Autor]

Stabla u Python 3.x jeziku

Stablo se takođe može opisati kroz liste ili imenike, ali prava funkcionalnost stabla se može dobiti definisanjem nove strukture podataka, ili, još bolje, kroz klasu stabla.

O klasama biće reči u lekciji #5, kada se obrađuju paradigme objektno-orjentisanog programiranja - OOP.

PRIMER ZA GRAFOVE I STABLA

Sledi video o grafovima i stablima

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 2

Funkcije

FUNKCIJE U PYTHON 3.X JEZIKU

Funkcije se mogu podeliti na ugrađene i korisničke, u zavisnosti da li su "gotove za korišćenje" ili ih korisnik sam treba definisati.

Funkcija (en. **function**) u opštem slučaju predstavlja deo koda koji se može ponovo iskoristiti.

Enkapsuliranjem dela koda, odnosno niz naredbi u funkciju, taj niz naredbi se može ponovo ponoviti samo navođenjem tj. pozivanjem funkcije, a ne ponovnim pisanjem koda.

Funkcije u Python 3.x jeziku

Do sada je već bilo reči o funkcijama, a da prethodno nije eksplicitno rečeno da se o njima radi. To je prvenstveno zbog toga što su se koristile ugrađene (en. **built-in**) funkcije.

Svaka komanda koja sadrži zagrade na kraju, na primer **print()**, predstavlja funkciju.

To znači da je funkcija definisana svojim imenom, zagradama, a promenljive koje se ubacuju u zagrade jesu argumenti te funkcije.

Gotovo svaki programski jezik dozvoljava kreiranje vlastitih, tj. korisničkih (en. **user-defined**) funkcija koje se mogu naknadno pozvati u programu.

Definisanje funkcija u Python 3.x jeziku

Korisničke funkcije se mogu definisati korišćenjem ključne reči **def**, nakon čega se navodi ime funkcije, parametri, ukoliko postoje, i dvotačka da bi označila da deo koda koji sledi pripada funkciji. Funkcija takođe može imati jednu ili više povratnih vrednosti.

Sintaksa je sledeća:

```
def fun_name(par1, par2, ...):  
    function_code  
    return ...
```

O parametrima funkcije i povratnim vrednostima biće detaljno reči u nastavku lekcije.

ŽIVOTNI CIKLUS PROMENLJIVIH UNUTAR FUNKCIJA

Funkcija ne pamti vrednost promenljive iz ranijeg poziva.

Opseg (en. **scope**) promenljive jeste deo koda gde se ta promenljiva može prepoznati,

Parametri i promenljive definisani unutar funkcije nisu vidljivi izvan funkcije. Zbog toga oni imaju lokalni opseg.

Životni ciklus promenljive (en. **the lifetime of a variable**) jeste vreme za koje ta promenljiva postoji u memoriji. Životni ciklus promenljive unutar funkcije je onoliko koliko se potrebno da se ta funkcija izvrši.

Kada se nakon poziva funkcije izvrši povratak u glavni program, lokalne promenljive se brišu.

Funkcija ne pamti vrednost promenljive iz ranijeg poziva.

Primer

Napisati funkciju koja ima lokalnu promenljivu var1 i koja ispisuje vrednost promenljive.

Nakon toga, u glavnom programu takođe napraviti promenljivu var1.

Pozvati funkciju, pa zatim ispitati vrednost promenljive var1.

```
def local_var():
    var1 = "CS324"
    print("Vrednost promenljive var1 unutar funkcije:", var1)

var1 = "Skripting jezici"
local_var()
print("Vrednost promenljive var1 izvan funkcije:", var1)
```

PRIMER KORISNIČKE FUNKCIJE BEZ PARAMETARA U PYTHON 3.X JEZIKU

Primer definisanje i pozivanja jednostavne funkcije u Python 3.x jeziku

Primer: Ispisivanje šifre i imena predmeta

Definisati funkciju sifra_predmeta() koja u sebi ima tri promenljive:

smer (string) - prvi deo šifre predmeta (CS/IT/AD i sl.)

kod (int) - drugi deo šifre predmeta, trocifreni kod

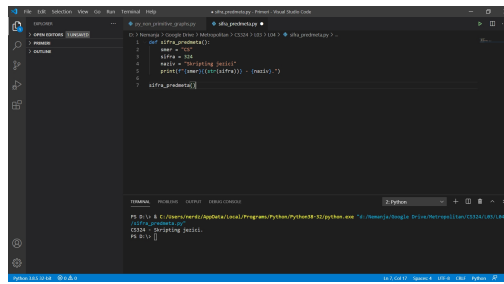
naziv (string) - Naziv predmeta na srpskom jeziku.

Funkcija **sifra_predmeta()** ispisuje pun naziv predmeta. Na primer:

CS324 - Skripting jezici.

```
def sifra_predmeta():
    smer = "CS"
    sifra = 324
    naziv = "Skripting jezici"
    print(f"{smer}{{(str(sifra))}} - {naziv}.")

sifra_predmeta()
```



Slika 2.1 Funkcija `sifra_predmeta()`. [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 3

Lambda funkcije

DEFINICIJA LAMBDA FUNKCIJE

Lambda funkcija može imati bilo koji broj parametra, ali se mora izvršiti u jednom redu koda.

Korisničke definisane funkcije u opštem slučaju mogu sadržati proizvoljni broj linije koda.

Ukoliko se želi definisati funkcija koja je jednostavna i koja se može izvršiti u jednoj liniji, preporuka je koristiti tzv. lambda funkciju. (en. **lambda function**).

Lambda funkcije poznate su kao i anonimne funkcije. (en. **anonymous function**) jer nemaju ime, ali u Python 3.x jeziku im se može i dodeliti ime.

Definisanje lambda funkcije

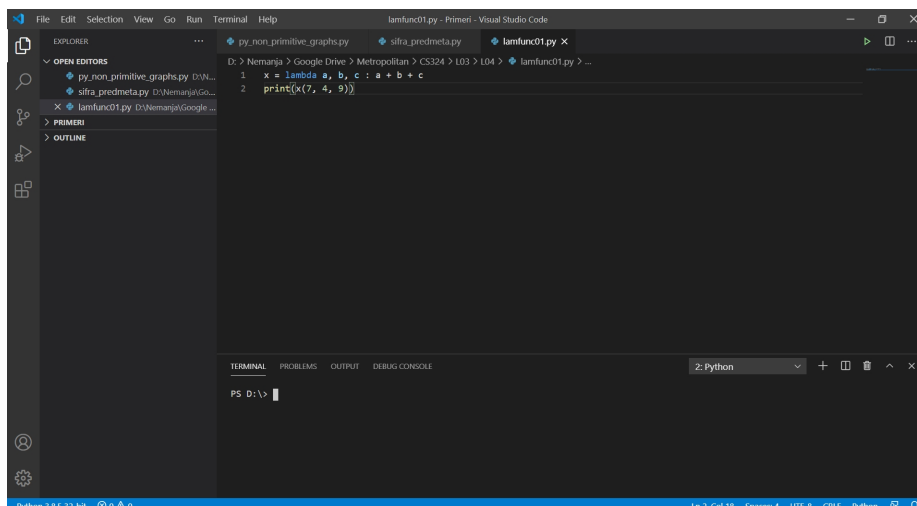
Standardne korisničke funkcije se definišu pomoću ključne reči **def**, a lambda funkcija se definiše rečju **lambda**.

Lambda funkcija može imati bilo koji broj parametra, ali se mora izvršiti u jednom redu koda.

Primer: Sabiranje tri broja pomoću lambda funkcije

Napisati lambda funkciju za sabiranje tri broja koja se unose kao parametri.

```
x = lambda a, b, c : a + b + c
print(x(7, 4, 9))
```



Slika 3.1 Primer jednostavne lambda funkcije. [Izvor: Autor]

GDE KORISTITI LAMBDA FUNKCIJE?

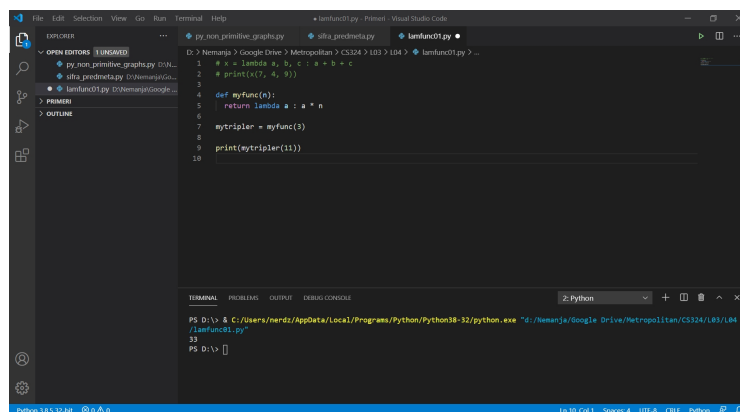
Lambda funkcije jesu brze funkcije i najbolje ih je koristiti u telu drugih korisnički-definisanih funkcija.

Lambda funkcije jesu brze funkcije i najbolje ih je koristiti u telu drugih korisnički-definisanih funkcija.

Primer:

Napisati funkciju koja će vraćati trostruku vrednost unetog broja.

```
def myfunc(n):  
    return lambda a : a * n  
  
mytripler = myfunc(3)  
  
print(mytripler(11))
```



Slika 3.2 Lambda funkcija unutar funkcije. [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 4

Parametri i povratne vrednosti

PARAMETRI I PROSLEĐIVANJE ARGUMENATA

Informacija koje se prosleđuje funkciji predstavlja argument funkcije.

Do sada je bilo reči o pozivanju funkcije samim pozivom funkcije. Međutim, funkcija može imati ulazni parametar, odnosno argument koji se prosleđuju funkciji.

Informacija koje se prosleđuje funkciji predstavlja argument funkcije.

Pri definisanju funkcije, u opštem slučaju, ne postoji granica o broju parametara koji se prosleđuje funkciji. Svi ulazni parametri se razdvajaju zapetom.

```
def puno_ime(ime, prezime):  
    print(ime + " " + prezime)  
  
puno_ime("Petar", "Petrovic")
```

Postavlja se pitanje da li je pravilno reći argument ili parametar.

Parametar predstavlja promenljivu koja se upisuje unutar zagrada prilikom definicije funkcije.

Argument predstavlja vrednost koja se šalje funkciji kada se ta funkcija poziva.

Funkcija može imati i prozvoljan broj argumenta, tako što se navodi ključni znk * pre imena parametra

```
def brojevi(*broj):  
    ...
```

Argumenti funkcije se mogu prosleđivati i u preko sledeće sintakse:

```
key = value
```

Na ovaj način redosled argumenata nije bitan.

```
def puno_ime(ime, prezime):  
    print(ime + " " + prezime)  
  
puno_ime(prezime = "Vukasinovic", ime = "Milic")
```

Funkcija se može definisati i sa **default**-nim argumentom, prilikom same definicije funkcije.

```
def ispit(sifra = "CS324"):  
    print("Slušate predmet " + sifra)
```

<autorski video, ime >

VRAĆANJE JEDNOG ARGUMENTA

Funkcija takođe može vratiti informaciju nazad u glavni program.

Pored primanja informacija, funkcija može i vratiti informaciju nazad u glavni program.

Naredba `return` se koristi da se poziv funkcije završi i da se rezultat (vrednost naredbe koja sledi posle ključne reči `return`) vrati u program, odnosno onom ko je funkciju i pozvao

Kompletna sintaksa definicija funkcije je sledeća:

```
def func():  
    func_body  
    .  
    .  
    .  
    return expression
```

Primer:

Napisati funkciju koja vraća `True` ukoliko je uneti broj paran, a `False` ukoliko je uneti broj neparan.

```
def prover_a_parnosti(broj):  
    if broj % 2 == 0:  
        return True  
    else:  
        return False
```

<autorski video, ime>>

VRAĆANJE VIŠE ARGUMENATA

U Python 3.x jeziku, moguće je da funkcija vrati više argumenata

U Python 3.x programskom jeziku moguće je da **return** ključna reč vrati više od jednog argumenta.

Prilikom definisanje funkcije, bitno je kako odvojiti više argumenata koji se vraćaju, jer se mogu vratiti u više ne-primitivnih tipova podataka.

Vraćanje više argumenata kao tuple

Najjednostavniji načina vraćanja više podataka jeste da se argumenti odvoje zarezom. Na taj način svi argumenti se smeštaju u **tuple**.

```
def funkcija01():
    smer = "CS"
    sifra = 324
    naziv = "Skripting Jezici"
    return smer, sifra, naziv

lst = funkcija01()
print(type(lst))
print(lst)

print(lst[0]+str(lst[1]), " - ", lst[2])
```

Vraćanje više argumenata kao lista

Najlakši način vraćanja više podataka jeste da se ti podaci smeste u listu, pogotovo zbog osobine liste da može sadržati elemente različitog tipa, a i da se vrednosti mogu modifikovati.

```
def funkcija02():
    smer = "CS"
    sifra = 324
    naziv = "Skripting Jezici"
    return [smer, sifra, naziv]

lst2 = funkcija02()
print(type(lst2))
print(lst2)

print(lst2[0]+str(lst2[1]), " - ", lst2[2])
```

Vraćanje više argumenata kao imenik

Ukoliko imamo samo dva izlazna argumenta, možemo ih smestiti u imenik.

Vraćanje više argumenata kao objekat

Izlazni argumenti se mogu vratiti i kao objekat. O ovome više reči više u lekciji iz objektno-orjentisanog programiranja u Python 3.x jeziku.

▼ Poglavlje 5

Ugrađene funkcije

UGRAĐENE FUNKCIJE U PYTHON 3.X JEZIKU

Ugrađene funkcije jesu uvek dostupne jer su deo interpretera.

Ugrađene funkcije (en. **built-in functions**) jesu funkcije koje su deo Python interpretera i one su uvek dostupne.

Mnoge funkcije su već bile korišćene, i neke su i intuitivne, ali neke nisu tako očigledne. Sledi tabela ugrađenih funkcija u Python 3.x jeziku.

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Slika 5.1 Tabela ugrađenih funkcija. Izvor: [<https://docs.python.org/3/library/functions.html>]

Ugrađene funkcije se mogu podeliti na funkcije ulaza i izlaza (en. **input and output functions**), matematičke funkcije (en. **math functions**), funkcije za tipove podataka (en. **type functions**), i ostale funkcije koje nisu specifično kategorisane.

Spisak ugrađenih funkcija se može naći u dokumentaciji trenutne verzije Python programskog jezika:

<https://docs.python.org/3/library/functions.html>

Pored samog spiska, dati su i primer i korišćenja svake od funkcija.

STANDARDNI ULAZ/IZLAZ

Za ulazno/izlazne operacije najpre se koriste funkcije `input()` i `print()`.

Za unos i štampu podataka (sa i na konzolu, respektivno) u Python 3.x programskom jeziku koriste se standardne funkcije ulaza i izlaza.

Standardni ulaz - `input()`

Ukoliko programer želi da unese podatak putem tastaturu kroz konzolu, treba pri pisanju programa koristiti funkciju `input()`.

Sintaksa `input()` funkcije je sledeća:

```
input([prompt])
```

gde je `prompt` string koji se ubacuje.

Pitanje:

U Python 2.x i 3. drugačije je definisana standardna unkcija `input()`. U čemu je razlika?

U svakom primeru do sada gde je bilo potrebno uneti neku vrednost sa tastature, korišćena je funkcija `input()`.

Standardni izlaz - `print()`

Ukoliko programer želi da ispiše podatak na ekran kroz konzolu, treba pri pisanju programa koristiti funkciju `print()`.

Sintaksa `print()` funkcije je sledeća:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

gde su `objects` argumenti koji se štampaju, `sep` je separator između vrednosti argumenata. a `end` se štampa na kraju svih vrednosti, i po default-u to je novi red.

Pitanje:

Da li je moguće promeniti default-ne vrednosti `sep` i `end` argumenta?

O formatiranju standardnog izlaza bilo je reči u prethodnoj lekciji.

MATEMATIČKE FUNKCIJE

Python interpreter ima ugrađene matematičke funkcije za neka osnovna izračunavanja bez potreba za pozivanjem dodatnog modula.

Matematičke ugrađene funkcije pomažu pri osnovnim matematičkim izračunavanjima.

Funkcija apsolutne vrednosti: `abs()`

Ova funkcija vraća apsolutnu vrednost broja, koji može biti ceo broj, razlomljen broj, ili kompleksni broj.

Konverzija u binarni broj - `bin()`

Ova funkcija konvertuje ceo broj u binarni string, sa prefiksom "0b". Ovaj prefiks se može izostaviti dodatnim formatiranjem.

Konverzija u heksadecimalni broj - `hex()`

Ova funkcija konvertuje ceo broj u heksadecimalni string sa prefiksom "0x". Karakter "x" može se formatirati da bude ispisan velikim ili malim slovom, ili skroz da bude izostavljen.

Najmanji broj - `min()`

Ova funkcija vraća najmanji element u strukturi ili manji od dva argumenta.

Konverzija u oktalni broj - `oct()`

Ova funkcija konvertuje ceo broj u oktalni string sa prefiksom "0o". Kao i kod drugih konverzija, moguće je izostaviti ovaj prefiks formatiranjem stringa.

Stepenovanje - `pow()`

Ova funkcija vraća stepen argumenta **base** na **exp**. Moguće je ubaciti i moduo kao opcioni argument.

Zaokruživanje - `round()`

Ova funkcija vraća broj zaokružena na željeni broj decimala.

Sumiranje - `sum()`

Ova funkcija vraća sumiranu vrednost elemenata ne-primitive.

Napomena:

*Za veći izbor funkcija, treba uvesti modul **math**. O math modulu, kao i o drugim modulima, biće reči u posebnoj lekciji.*

▼ Poglavlje 6

Pokazna vežba #4

UVOD U POKAZNU VEŽBU #4

Pokazna vežba #4 obrađuje primere iz različitih struktura podataka, kao i rada sa funkcijama.

Pokazna vežba #4 obrađuje različite strukture podataka, kako primitive, tako i ne-primitive.

Najpre je dat zadatak spajanje dva imenika u jedan, a posle je dato ručno sortiranje liste, sa korišćenjem pomoćne liste, i bez.

Nastavak predstavlja zadatak koji od sortirane liste menja mesta najmanjem i najvećem elementu liste.

Nakon toga, biće obrađen primer koji opisuje operacije rad za funkcijama.

Dat je zadatak u kom treba napisati funkciju koja vrši transformaciju vrednosti otpornika vezanih u trougao, u otpornike vezanih u zvezdu.

Procenjeno trajanje pokazne vežbe iznosi 45 minuta.

SPAJANJE DVA IMENIKA U JEDAN, RUČNO SORTIRANJE LISTE

Iako postoji `.sort()` metoda za sortiranje, potrebno je znati i ručno sortirati elemente liste.

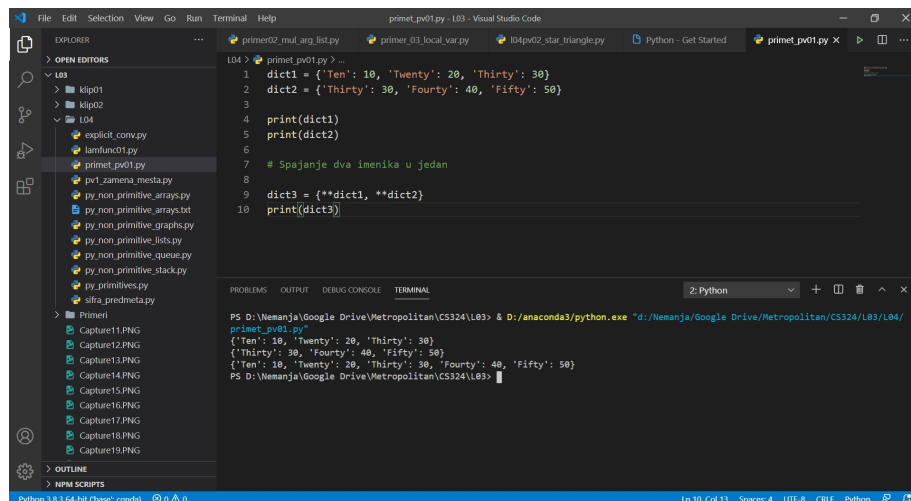
Zadatak #1 (10 minuta)

Napisati program koji će spojiti dva imenika u jedan.

```
dict1 = {'Ten': 10, 'Twenty': 20, 'Thirty': 30}
dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}

dict3 = {**dict1, **dict2}
print(dict3)
```

Komanda `**` u kontekstu pozivu funkcija odnosi se za raspakovanje imenika u drugi imenik. Na taj način ostaće *ključ: vrednost*.



Slika 6.1 Spajanje dva imenika u jedan. [Izvor: Autor]

Zadatak #2 (15 minuta)

Napisati program koji će sortirati niz (listu sa numeričkim vrednostima) bez korišćenja `.sort()` metode.

Rešenje #1 - pravljenje novog niza

```

data_list = [-5, -23, 5, 0, 23, -6, 28, 67]
new_list = []

while data_list:
    minimum = data_list[0]
    for x in data_list:
        if x < minimum:
            minimum = x
    new_list.append(minimum)
    data_list.remove(minimum)
print(new_list)
  
```

Rešenje #2 - direktna zamena mesta elemenata u nizu

```

data_list = [-5, -23, 5, 0, 23, -6, 28, 67]
n = len(data_list)
for i in range(n):
    for j in range(i + 1, n):
        if(data_list[i] > data_list[j]):
            temp = data_list[i]
            data_list[i] = data_list[j]
            data_list[j] = temp
print(data_list)
  
```

ZAMENA VREDNOSTI NAJMANJEG I NAJVEĆEG ELEMENTA NIZA

Često pitanje na intervjuima jeste zamena vrednosti bez korišćenja pomoćne promenljive.

Zadatak #3 (10 minuta)

Napisati program koji će sortirati niz od 10 elemenata. Zatim napisati funkciju koja kao ulazni parametar uzima niz, i menja mesta prvom i poslednjem članu, bez korišćenja dodatne pomoćne promenljive.

```
import random
lst_ran = []
for i in range(10):
    lst_ran.append(random.randint(0, 1000))
print(lst_ran)

lst_ran.sort()
print(lst_ran)

def swap_lst(lst):
    lst[0] = lst[0] + lst[-1]
    lst[-1] = lst[0] - lst[-1]
    lst[0] = lst[0] - lst[-1]
    return lst

print(swap_lst(lst_ran))
```

FUNKCIJA SA VIŠE ARGUMENATA I VIŠE POVRATNIH VREDNOSTI

Zadatak koji ima tri ulazna parametra, a na izlazu daje listu od tri vrednosti.

Zadatak #4 (10 minuta)

Zadatak #4 opisuje pisanje funkciju sa više argumenata i više povratnih vrednosti, kao elemente liste.

Napisati funkciju koja računa transformaciju ulaznih parametara, i kao izlaz daje listu tri elemenata. Transformacija se računa na sledeći način:

$$R_1 = \frac{R_b R_c}{R_a + R_b + R_c}, R_2 = \frac{R_a R_c}{R_a + R_b + R_c}, R_3 = \frac{R_a R_b}{R_a + R_b + R_c}$$

Zanimljivost:

Ova transformacija se često javlja u elektrotehnici/elektronici.

Više detalja na: https://en.wikipedia.org/wiki/Y-Δ_transform

```
def triange_to_star(Ra, Rb, Rc):  
    R1 = (Rb * Rc)/(Ra + Rb + Rc)  
    R2 = (Ra * Rc)/(Ra + Rb + Rc)  
    R3 = (Ra * Rb)/(Ra + Rb + Rc)  
    return [R1, R2, R3]  
  
star = triange_to_star(100, 14000, 10000)  
print(star)
```

▼ Poglavlje 7

Invidivualna vežba #4

UVOD U INDIVIDUALNU VEŽBU #4

Individualna vežba #4 odnosi se na rad sa funkcijama.

Individualna vežba #4 daje dva primera u kojima se radi sa funkcijama.

Procenjeno trajanje individualnih vežbi iznosi 90 minuta.

INDIVIDUALNA VEŽBA #4

Napisati jednostavne programe u Python programskoj jeziku koristeći instalirana okruženja.

Zadatak #1 (60 minuta)

Napisati program za unos imena, prezimena, godina rođenja, broja indeksa.

Zatim, napraviti imenik koji će kao ključ imati šifru predmeta, a vrednost ocena iz tog predmeta. Ubacite samo predmete koje ste položili.

Izračunati prosečnu ocenu tokom studiranja.

a) Napraviti novu promenljivu koja je string i koja je oblika:

"Ime Prezime, **broj_indeksa**, rođen/a **godina_rođenja** ima trenutno prosečnu osenu **prosecna_ocena**" i štampati tu promenljivu.

b) Uraditi isto, ali bez posebne string promenljive.

Zadatak #2 (30 minuta)

Napisati dve funkcije za transformaciju ulaznih vrednosti u izlazne parametre.

Prva funkcija na osnovu **Ra**, **Rb** i **Rc** računa **R1**, **R2**, i **R3**.

Druga funkcija na osnovu **R1**, **R2** i **R3** računa **Ra**, **Rb**, i **Rc**.

U glavnom programu uneti "1" za prvu transformaciju, a "2" za drugu transformaciju.

Ukoliko se nešto treće unese, program ispisuje grešku.

Transformacije se računaju na sledeći način:

$$R_1 = \frac{R_b R_c}{R_a + R_b + R_c}, R_2 = \frac{R_a R_c}{R_a + R_b + R_c}, R_3 = \frac{R_a R_b}{R_a + R_b + R_c}$$
$$R_a = \frac{R_1 R_2 + R_2 R_3 + R_3 R_1}{R_1}, R_b = \frac{R_1 R_2 + R_2 R_3 + R_3 R_1}{R_2}, R_c = \frac{R_1 R_2 + R_2 R_3 + R_3 R_1}{R_3}$$

▼ Poglavlje 8

Domaći zadatak #4

DOMAĆI ZADATAK

Domaći zadatak #4 se okvirno radi 2.5h

Zadatak #1

Napisati funkciju sa dva ulazna parametra: *ime* i *br_indeks*

Vrednost *br_indeks* (četvorocifrena promenljiva) određuje broj elemenata u nizu koji se kreira unutar funkcije.

Elementi niza jesu sledeći:

- Ukoliko je broj slova u promenljivoj *ime* paran, generisati cele brojeve od 0 do *br_indeks* sa uniformnom raspodelom
- Ukoliko je broj slova u promenljivoj *ime* neparan, generisati razlomljene brojeve u opsegu od negativne vrednosti prve dve cifre, to pozitivne vrednosti druge dve cifre u *br_indeks*.

Koristiti *random* biblioteku za generisanje brojeva.

Sortirati niz korišćenjem for petlje i privremene promenljive.

Kao povratnu vrednost vratiti sortirani niz.

U glavnom programu uneti promenljive *ime* i *br_indeks*, pozvati funkciju sa tim promenljivima i odštampati sortirani niz.

Zadatak #2

Napisati rekursivnu funkciju za računanje Fibonačijevog niza do elementa *n*.

Napisati iterativnu funkciju za računanje Fibonačijevog niza do elementa *n*.

Opisati sličnosti i razlike u ova dva pristupa.

Predaja domaćeg zadatka:

Tradicionalni studenti:

Domaći zadatak treba dostaviti najkasnije nedelju dana nakon predavanja za 100% poena. Nakon toga poeni se umanjuju za 50%.

Online studenti:

Domaći zadatak treba dostaviti najkasnije 10 dana pred polaganja ispita. **Domaći zadaci se brane!**

Svi studenti domaći zadatak poslati dr Nemanji Zdravkoviću:
nemanja.zdravkovic@metropolitan.ac.rs

▼ Poglavlje 9

Zaključak

ZAKLJUČAK

Zaključak lekcije #4

Rezime:

U ovoj lekciji bilo je reči o strukturama podataka, ugrađenim, ali i kako se mogu praviti i korisničke strukture, koje se uglavnom mogu kategorisati u nekoliko grupa.

Nakon toga, bilo je reči o funkcijama, opsegu promenljivih, ulaznim i izlaznim parametrima, kao i o anonimnim i brzim lambda funkcijama.

Sve oblasti praćene su adekvatnim primerima.

Literatura:

1. David Beazley, Brian Jones, Python Cookbook: Recipes for Mastering Python 3, 3rd edition, O'Reilly Press, 2013.
2. Mark Lutz, Learning Python, 5th Edition, O'Reilly Press, 2013.
3. Andrew Bird, Lau Cher Han, et. al, The Python Workshop, Packt Publishing, 2019.
4. Al Sweigart, Automate the boring stuff with Python, 2nd Edition, No Starch Press, 2020.

