



CS324 - SKRIPTING JEZICI

Proceduralno programiranje -
promenljive, petlje, kontrola toka

Lekcija 03

PRIRUČNIK ZA STUDENTE

CS324 - SKRIPTING JEZICI

Lekcija 03

PROCEDURALNO PROGRAMIRANJE - PROMENLJIVE, PETLJE, KONTROLA TOKA

- ✓ Proceduralno programiranje - promenljive, petlje, kontrola toka
- ✓ Poglavlje 1: Vrste promenljivih
- ✓ Poglavlje 2: Imenici i tuple-ovi
- ✓ Poglavlje 3: Tipiziranje podataka u Python 3.x jeziku
- ✓ Poglavlje 4: Kontrola toka
- ✓ Poglavlje 5: Petlje
- ✓ Poglavlje 6: Pokazna vežba #3
- ✓ Poglavlje 7: Individualna vežba #3
- ✓ Poglavlje 8: Domaći zadatak #3
- ✓ 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

UVOD

U trećoj lekciji govoriće se o proceduralnom programiranju u Python-u, sa akcentom na vrste promenljivih, kontroli toka, i o petljama.

U trećoj lekciji govoriće se o proceduralnom programiranju u Python-u. Pre nego što se pređe na koncepte objektno-orijentisanog programiranja, veoma je bitno da se savlada proceduralno programiranje.

OOP možda jeste napredniji koncept, međutim neki zadaci, pogotovo u primeni Python-a u inženjerskim aplikacijama, proceduralno napisan kod bolje rešava.

Podsetimo se nekih od osobina proceduralnog programiranja:

Ključni koncept jesu procedure, odnosno funkcije, koje predstavljaju instrukcije koja su definisane nazivom.

Takođe, u radu sa funkcijama treba obratiti pažnju na *lokalne* i *globalne* promenljive. Lokalne važe samo u opsegu strukture podataka u kojoj su definisane, dok globalne promenljive se deklariraju izvan ostalih funkcija.

U ovoj lekciji, osnovni rad sa promenljivima biće predstavljen, kako sa numeričkim tako i sa alfa-numeričkim. Biće reči i o kontroli toka, i petljama i kako se predstavljaju u Python 3.x jeziku.

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

✓ Poglavlje 1

Vrste promenljivih

STRINGOVI U PYTHON 3.X JEZIKU

String tip podataka predstavlja niz slova, brojeva, simbola i razmaka.

String tip podataka predstavlja niz slova, brojeva, simbola i razmaka.

U Python 3.x programskom jeziku, stringovi mogu biti gotovo bilo koje dužine i mogu sadržati razmake (en. **space**), jer je i razmak validan karakter u Python jeziku. Stringovi mogu stajati između apostrofa ili između navodnika.

Brojevi kao stringovi

Brojevi (celi i razlomljeni) se takođe mogu definisati kao stringovi. Ukoliko je ceo, razlomljen ili kompleksni broj definisan pod apostrofima, čuva se kao string a ne kao broj.

Logičke vrednosti kao stringovi

Stringovi se mogu konvertovati u logičke promenljive korišćenjem `bool()` funkcije. Jedino prazan string `""` vraća **False**, a svi ostali vraćaju **True**.

String koji sadrži samo razmak, " ", nije prazan string i vratiće True, jer je razmak (blanko, space) validan karakter.

Indeksiranje stringova

Indeksiranjem stringova može se izvući konkretni karakter iz stringa u određenom redosledu.

U Python programskom jeziku, stringovi se indeksiraju korišćenjem srednjih zagrada `[]`.

U Python jeziku prvi indeks je 0 a poslednji je n-1.

Negativno indeksiranje stringova

Ukoliko u srednjim zagradama stavimo negativni broj, onda će se izvući karakter stringa od poslednjeg karaktera.

Opseg indeksiranja

Može se izvući sekvenca karaktera iz stringa. Ukoliko između srednjih zagrada stavimo prvi indeks, pa dvotačku, pa krajnji indeks, onda će se izvući karakteri unutar tog opsega.

UVOD U TIPOVE PODATAKA

Najčešće grupe tipova podataka jesu numerički, ne-numerički i Boolean tip

Podsetnik:

Tip podataka predstavlja klasifikaciju ili kategorizaciju podataka. Predstavlja tip vrednosti koji određuje koje operacije se mogu izvršiti nad tim podacima.

Tip podataka, tj. vrsta promenljive, obično se grupiše u 3 ili više grupa.

Najčešće grupe tipova podataka.

- Numerički tip podatak
- Ne-numerički tip podataka
- Boolean tip podataka (True/False)

U okviru svake grupe mogu postojati više tipova podataka, opet, u zavisnosti koji je programski jezik u pitanju.

Kod nekih programskih jezika (poput C jezika), pravi se razlika između tipa podataka *char* (jedan karakter) i *string* (više karaktera).

Svaki programski jezik ima svoje klasifikacije koje opisuju filozofiju tog programskog jezika.

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

*Koristeći funkciju **type()** koja vraća tip podataka uvek možemo proveriti o kojem je tipu podataka reč.*

▼ 1.1 Numeričke promenljive

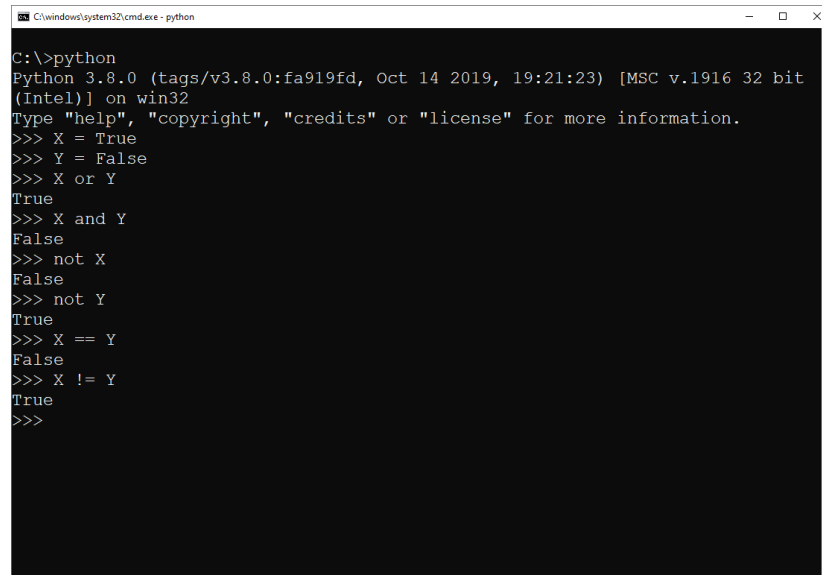
CELOBROJNI I RAZLOMLJENI TIPOVI PODATAKA

Python 3.x jezik podržava celobrojne i razlomljene numeričke tipove podataka.

Celobrojni tip podataka

Celobrojni tip podataka (en. *Integer*) jeste upravo to - celi broj, koji može biti pozitivan, jednak nuli, ili negativan

```
int_a = 5
type(int_a)
int_b = -3
type(int_b)
z = 0
type(z)
```



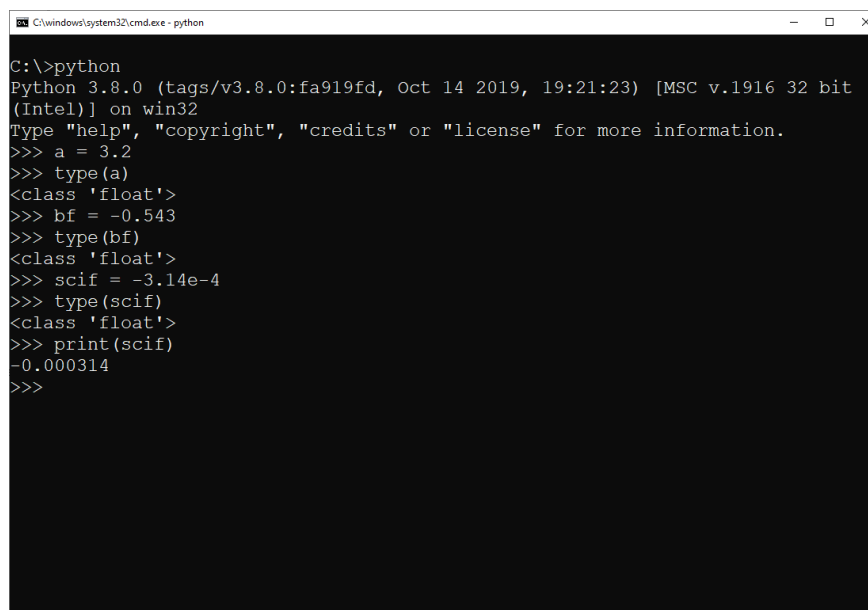
```
C:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> X = True
>>> Y = False
>>> X or Y
True
>>> X and Y
False
>>> not X
False
>>> not Y
True
>>> X == Y
False
>>> X != Y
True
>>>
```

Slika 1.1.1 Celobrojni tip podataka. [Izvor: Autor]

Razlomljeni tip podataka

Razlomljeni tip podataka (en. **floating point**) jesu brojevi sa decimalnom tačkom, koji takođe mogu biti pozitivni, jednaki nuli, i negativni.

Pored decimalne notacije, mogu se predstaviti i u *obliku sa eksponentom* (en. **scientific notation**), u kojoj eksponent može biti označen bilo malim bilo velikim slovom e.



```
C:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3.2
>>> type(a)
<class 'float'>
>>> bf = -0.543
>>> type(bf)
<class 'float'>
>>> scif = -3.14e-4
>>> type(scif)
<class 'float'>
>>> print(scif)
-0.000314
>>>
```

Slika 1.1.2 Razlomljeni tip podataka. [Izvor: Autor]

KOMPLEKSNI TIPOVI PODATAKA

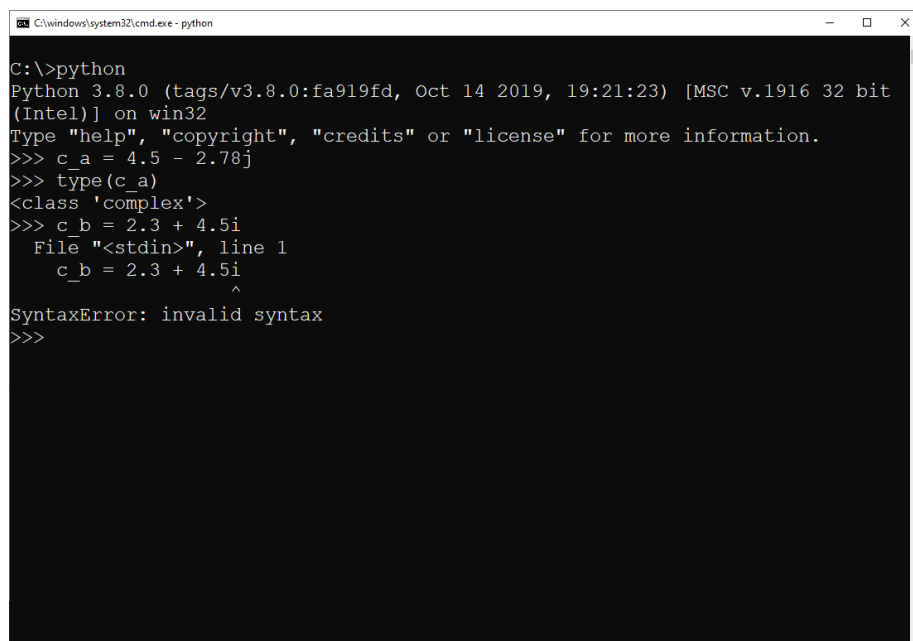
Python 3.x jezik podržava i kompleksne numeričke tipove podataka.

Kompleksni tip podataka

Kompleksni tip podataka (en. **complex number**) odnosi se na kompleksni broj koji ima realni i imaginarni deo.

Kompleksni brojevi se u Python 3.x jeziku pišu kao *realni_deo + imaginarni_deo * j*.

Koristi se *j* kao oznaka za imaginarnu jedinicu, dok ukoliko se koristi *i*, Python će vratiti grešku.



```
C:\windows\system32\cmd.exe - python
C:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> c_a = 4.5 - 2.78j
>>> type(c_a)
<class 'complex'>
>>> c_b = 2.3 + 4.5i
File "<stdin>", line 1
    c_b = 2.3 + 4.5i
                  ^
SyntaxError: invalid syntax
>>>
```

Slika 1.1.3 Kompleksni tip podataka. [Izvor: Autor]

▼ 1.2 Logičke promenljive - Boolean

TIP PODATAKA BOOLEAN

Svaki programski jezik, pa i Python, ima logički tip podataka Bool (Boolean)

Kao i kod gotovo svih ostalih programskih jezika, Python poseduje logički tip podataka koji može imati vrednost tačno (en. **true**) i netačno (en. **false**).

U Python 3.x jeziku, dodela Boolean vrednosti razlikuje veličinu slova (en. case sensitive), tako da je prvo slovo uvek veliko.

Primer:

Promenljivama **a** i **b** se pravilno dodeljuje logička vrednost, dok promenljivoj **c** se loše dodeljuje, što javlja grešku.

```
a = True
type(a)
b = False
type(b)
c = true
```

```
Microsoft Windows [Version 10.0.19041.450]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Nemanja Zdravkovic>py
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] ::
Anaconda, Inc. on win32
>>> a = True
>>> type(a)
<class 'bool'>
>>> b = False
>>> type(b)
<class 'bool'>
>>> c = true
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
>>>
```

Slika 1.2.1 Primer za Boolean tipa podataka u Python 3.x. [Izvor: Autor]

Celobrojni i razlomljeni podaci kao logičke vrednosti

Celi i razlomljeni brojevi se mogu konvertovati u logički tip podataka koristeći funkciju `bool()`. Izlaz funkcije jeste `True` ili `False`.

Celi, razlomljeni i kompleksni brojevi koji su jednaki nuli vraćaju `False`, dok ostale vrednosti (i pozitivne i negativne) vraćaju `True`.

```
zero_int = 0
bool(zero_int)
neg_fpoint = -10.23
bool(neg_fpoint)
pos_cmplx = 3 + 6j
bool(pos_cmplx)
```

```
C:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> zero_int = 0
>>> bool(zero_int)
False
>>> neg_fpoint = -10.23
>>> bool(neg_fpoint)
True
>>> pos_cmplx = 3 + 6j
>>> bool(pos_cmplx)
True
>>>
```

Slika 1.2.2 `bool()` funkcija koja vraća logičke vrednosti. [Izvor: Autor]

ARITMETIKA NAD BOOLEAN TIPOM PODATAKA

U Pythonu je moguće raditi osnovne logičke operacije (logička aritmetika) nad podacima tipa Boolean.

Nad Boolean tipom podataka se mogu vršiti logičke operacije sabiranja, množenja i negacije.

Operatori u Python jeziku nad Boolean tipom podataka

- Logičko ili - *or*
- Logičko i - *and*
- Logičko ne - *not*
- Jednako - *==*
- Nije jednako - *!=*

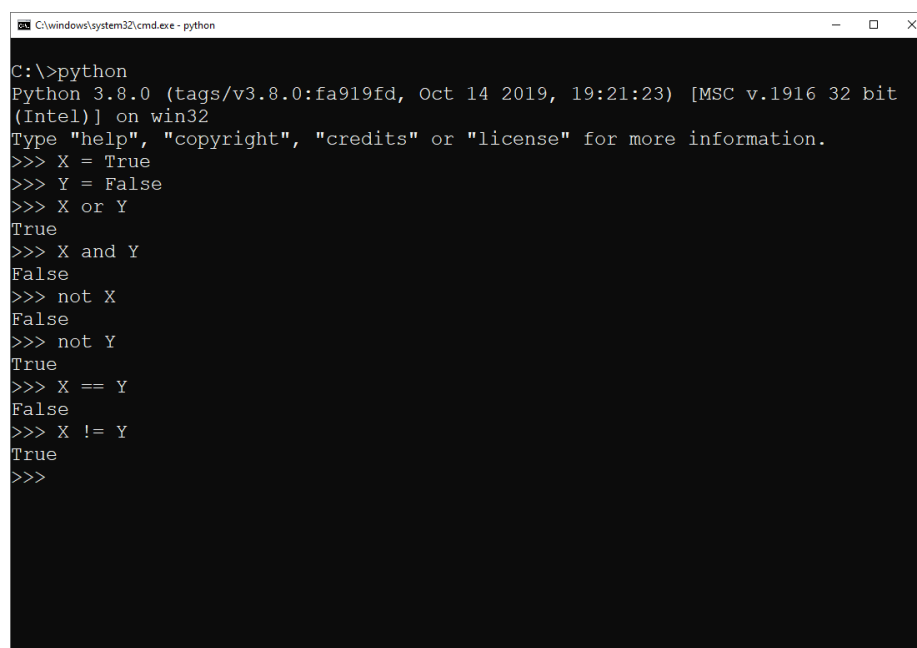
Kroz kratke primere dodelićemo logičke vrednosti promenljivima X i Y i izvršiti logičke operacije nad njima.

```
X = True
Y = False

X or Y
X and Y

not X
not Y

X == Y
X != Y
```



```
C:\windows\system32\cmd.exe - python
C:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> X = True
>>> Y = False
>>> X or Y
True
>>> X and Y
False
>>> not X
False
>>> not Y
True
>>> X == Y
False
>>> X != Y
True
>>>
```



Slika 1.2.3 Operacije nad Boolean tipovima podataka. [Izvor: Autor]

1.3 Stringovi i liste

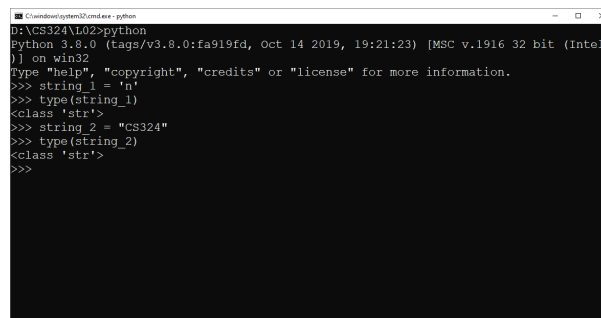
PRIMERI SA STRINGOVIMA U PYTHON 3.X JEZIKU

Primeri u radu sa stringovima u Python jeziku

Primer #1 - Definisanje stringova koji sadrže jedan i više karaktera

```
string_1 = 'n'
type(string_1)

string_2 = "CS324"
type(string_2)
```

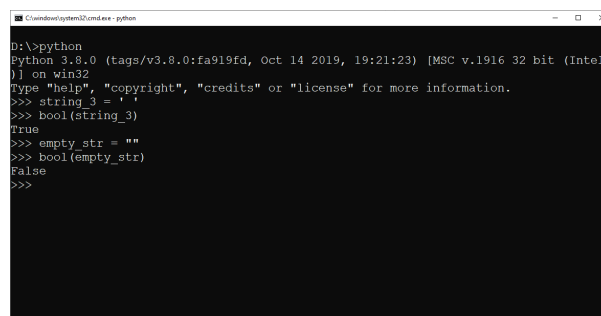


Slika 1.3.1 Deklarisanje stringova. [Izvor: Autor]

Primer #2 - Stringovi kao logičke vrednosti i kao brojevi

```
string_3 = ' '
bool(string_3)

empty_str = ""
bool(empty_str)
```

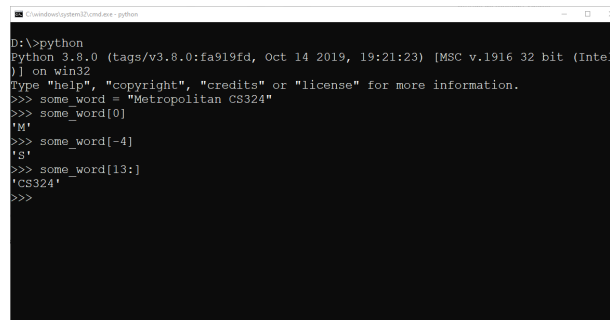


Slika 1.3.2 Stringovi i logičke vrednosti. [Izvor: Autor]

Primer #3 - Pozitivno I negativno indeksiranje stringova i opsezi indeksiranja

```
some_word = "Metropolitan CS324"

some_word[0]
some_word[-4]
some_word[13:]
```



```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> some_word = "Metropolitan CS324"
>>> some_word[0]
'M'
>>> some_word[-4]
'S'
>>> some_word[13:]
'CS324'
>>>
```

Slika 1.3.3 Indeksiranje stringova. [Izvor: Autor]

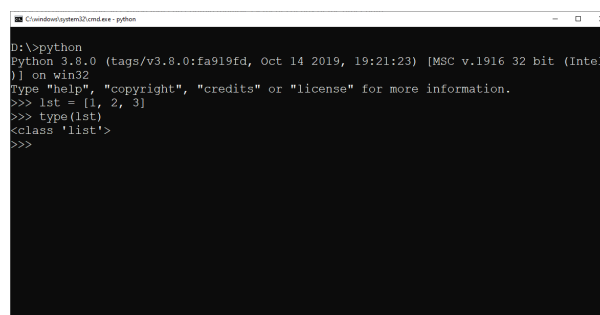
LISTE U PYTHON 3.X JEZIKU

Lista može sadržati više elemenata od kojih svaki element može pripadati različitom tipu podataka.

List tip podataka jeste struktura podataka u Python programskom jeziku koja može sadržati više elemenata bilo kojeg drugog tipa podataka.

Lista je definisana srednjim zagradama [], a elementi liste se razdvajaju zarezima.

```
lst = [1, 2, 3]
type(lst)
```



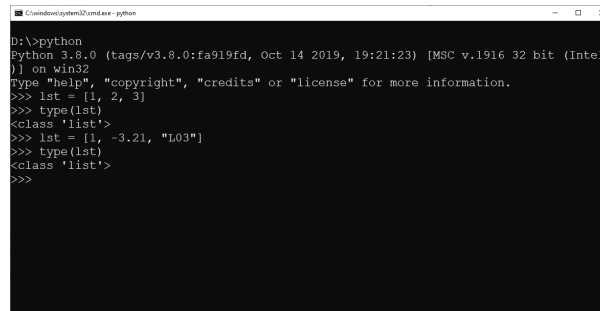
```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> lst = [1, 2, 3]
>>> type(lst)
<class 'list'>
>>>
```

Slika 1.3.4 Pravljenje liste u Python jeziku. [Izvor: Autor]

Indeksiranje lista

Kao i kod stringova, pojedinačni elementi liste se mogu pristupiti indeksiranjem putem srednjih zagrada.

```
lst = [1, -3.21, "L03"]
type(lst)
lst[2]
type(lst[1])
```



```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> lst = [1, 2, 3]
>>> type(lst)
<class 'list'>
>>> lst = [1, -3.21, "L03"]
>>> type(lst)
<class 'list'>
>>>
```

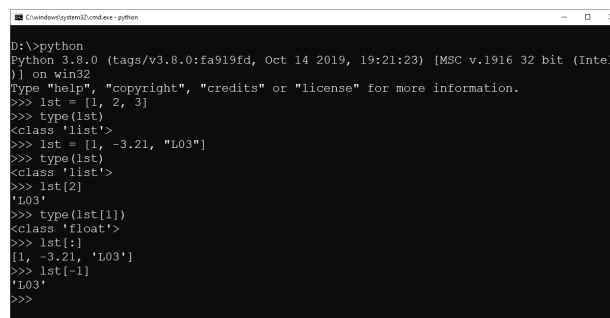
Slika 1.3.5 Indeksiranje liste. [Izvor: Autor]

Opsezi indeksiranja lista

Moguće je indeksirati više elemenata liste korišćenjem dvotačke.

Ukoliko treba indeksirati sve elemente liste, u srednje zagrade se stavlja samo dvotačka, odnosno za listu `lst`, svi elementi prikazaće se ukoliko se napiše `lst[:]`.

Isto važi i za negativno indeksiranje kao kod stringova, ali i opseg.



```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> lst = [1, 2, 3]
>>> type(lst)
<class 'list'>
>>> lst = [1, -3.21, "L03"]
>>> type(lst)
<class 'list'>
>>> lst[2]
'L03'
>>> type(lst[1])
<class 'float'>
>>> lst[:]
[1, -3.21, 'L03']
>>> lst[-1]
'L03'
>>>
```

Slika 1.3.6 Indeksiranje lista u određenom opsegu i negativno indeksiranje lista. [Izvor: Autor]

▼ Poglavlje 2

Imenici i tuple-ovi

TUPLE PODACI U PYTHON 3.X JEZIKU

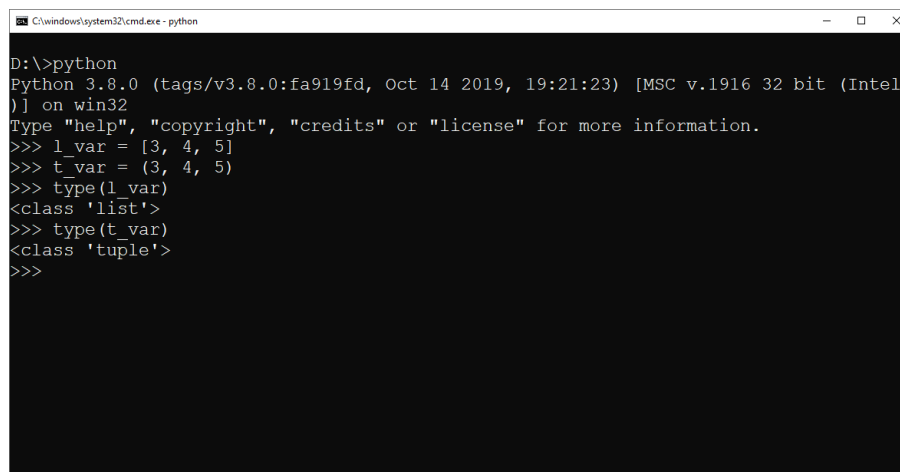
Listu sa nepromenljivim elementima predstavlja tuple.

tuple tip podataka predstavlja listu koja je nepromenljiva. Za razliku od obične liste čiji se elementi mogu modifikovati, elementi u tuple tipu podataka se samo mogu pristupiti, ali ne i modifikovati. Tuple podaci se definišu u Python jeziku tako što su elementi tuple-a u malim zagradama, a pojedinačni elementi su razdvojeni zapetama.

Primer: Poređenje liste i tuple tipa podataka

Napomena. funkcija `type()` vraća tip podataka argumenta.

```
l_var = [3, 4, 5]
t_var = (3, 4, 5)
type(l_var)
type(t_var)
```

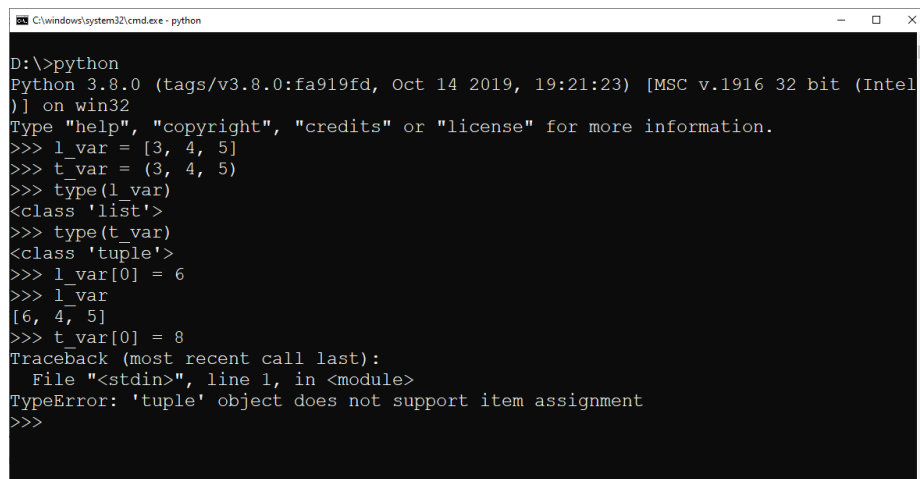


```
C:\windows\system32\cmd.exe - python
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> l_var = [3, 4, 5]
>>> t_var = (3, 4, 5)
>>> type(l_var)
<class 'list'>
>>> type(t_var)
<class 'tuple'>
>>>
```

Slika 2.1 Definisanje elemenata liste i tuple-a. [Izvor: Autor]

Ukoliko želimo da promenimo jedan element tuple-a, to neće biti moguće, dok za listu neće vratiti nikakvu grešku.

```
l_var[0] = 8
l_var
t_var[0] = 8
```



```

D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel
)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> l_var = [3, 4, 5]
>>> t_var = (3, 4, 5)
>>> type(l_var)
<class 'list'>
>>> type(t_var)
<class 'tuple'>
>>> l_var[0] = 6
>>> l_var
[6, 4, 5]
>>> t_var[0] = 8
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>

```

Slika 2.2 Promena elemenata u listi i neuspeli pokušaj u tuple-u. [Izvor: Autor]

Pitanje:

Kako definisati tuple koji ima samo jedan element?

IMENICI U PYTHON 3.X JEZIKU

Imenik, za razliku od liste, sastoji se od para podataka: ključa i vrednosti.

Imenik (en. **dictionary**) je tip podataka koji se sastoji od para podataka, i to po šablonu *ključ: vrednost*.

Imenicima u Python jeziku su organizovani koristeći par ključa i vrednosti, i tako im se i pristupa.

Lokacija para ključ-vrednost u imeniku nije bitna.

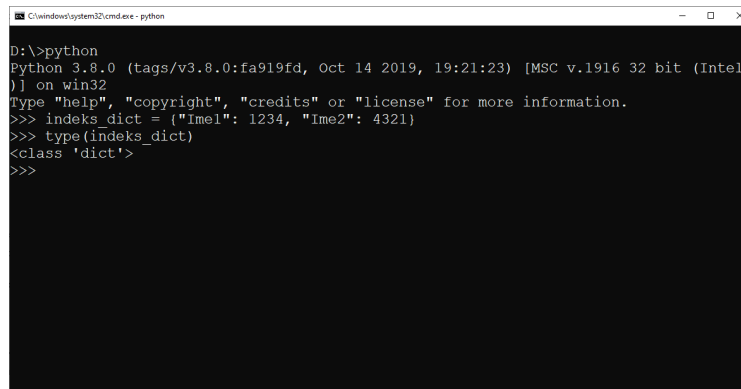
Imenici se u Python jeziku definišu u velikim zagradama {}. Zapere razdvajaju ove parove, a svaki par je povezan dvotačkom.

Primer - Definisanje imenika

```

indeks_dict = {"Ime1": 1234, "Ime2": 4321}
type(indeks_dict)

```



```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel
)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> indeks_dict = {"Ime1": 1234, "Ime2": 4321}
>>> type(indeks_dict)
<class 'dict'>
>>>
```

Slika 2.3 Definisanje imenika. [Izvor: Autor]

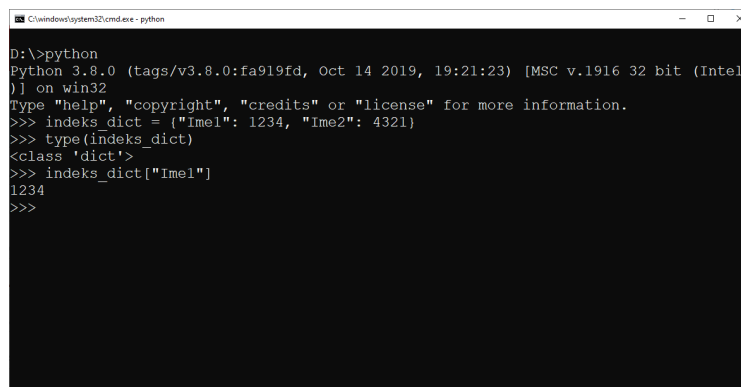
Pristupanje vrednostima

Vrednostima unutar imenika se pristupa sledećom sintaksom

```
dict_name[key] = value
```

Primer - pristupanje vrednosti imenika

```
indeks_dict["Ime1"]
```



```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel
)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> indeks_dict = {"Ime1": 1234, "Ime2": 4321}
>>> type(indeks_dict)
<class 'dict'>
>>> indeks_dict["Ime1"]
1234
>>>
```

Slika 2.4 Pristupanje vrednosti imenika. [Izvor: Autor]

Pitanje:

Kako se imenici mogu pretvoriti u liste?

DODAVANJE I ODUZIMANJE ELEMENTA U IMENIKU

U Python jeziku se lako dodaju novi elementi imenika, a korišćenjem .pop() metode.

Dodavanje elemenata imeniku

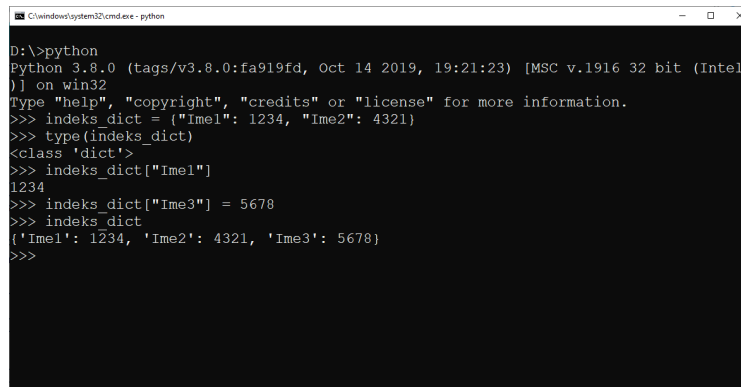
Dodavanje novog elementa postojećem imeniku može se uraditi upotrebom sledeće sintakse:

```
dict_name[new_key] = new_value
```

Primer: Dodavanje elementa imeniku

Na primeru **indeks_dict** dodaće se još jedan element

```
indeks_dict["Ime3"] = 5678
```



```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> indeks_dict = {"Ime1": 1234, "Ime2": 4321}
>>> type(indeks_dict)
<class 'dict'>
>>> indeks_dict["Ime1"]
1234
>>> indeks_dict["Ime3"] = 5678
>>> indeks_dict
{'Ime1': 1234, 'Ime2': 4321, 'Ime3': 5678}
>>>
```

Slika 2.5 Dodavanje elementa imeniku [Izvor: Autor]

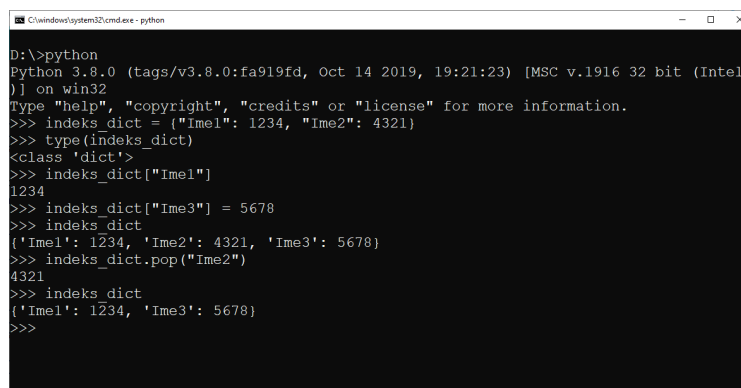
Oduzimanje elementa imenika

Elementi imenika se mogu oduzeti pozivanjem `.pop()` metode. Vrednost ključa koja se prosleđuje metodi `.pop()` određuje koji će se par ključ-vrednost izbaciti.

Primer: Oduzimanje elementa imeniku

Na primeru **indeks_dict** oduzeće se jedan element

```
indeks_dict.pop("Ime2")
```



```
D:\>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> indeks_dict = {"Ime1": 1234, "Ime2": 4321}
>>> type(indeks_dict)
<class 'dict'>
>>> indeks_dict["Ime1"]
1234
>>> indeks_dict["Ime3"] = 5678
>>> indeks_dict
{'Ime1': 1234, 'Ime2': 4321, 'Ime3': 5678}
>>> indeks_dict.pop("Ime2")
4321
>>> indeks_dict
{'Ime1': 1234, 'Ime3': 5678}
>>>
```

Slika 2.6 Oduzimanje elementa imeniku. [Izvor: Autor]

Pitanje:

U čemu je razlika između funkcije i metode?

▼ Poglavlje 3

Tipiziranje podataka u Python 3.x jeziku

PODELA TIPIZIRANJA

Postoje jako i slabo tipizirani programski jezici, kao i statički i dinamički tipizirani. Svaki programski jezik spada u jednu od četiri kombinacije.

Pri procesu izvršenja programa (bilo pri kompajliranju ili interpretiranju), vrši se provera tipa podataka (en. type checking).

Ukoliko se proces provere tipa podataka izvršava u vremenu kompajliranja, onda se vrši statička provera.

Ukoliko se proces provere tipa podataka izvršava u vremenu interpretiranja, onda se vrši dinamička provera.

Provera tipa podataka vrši se u cilju da program bude *tipski* siguran (en. *type-safe*), što umanjuje mogućnost za greškama.

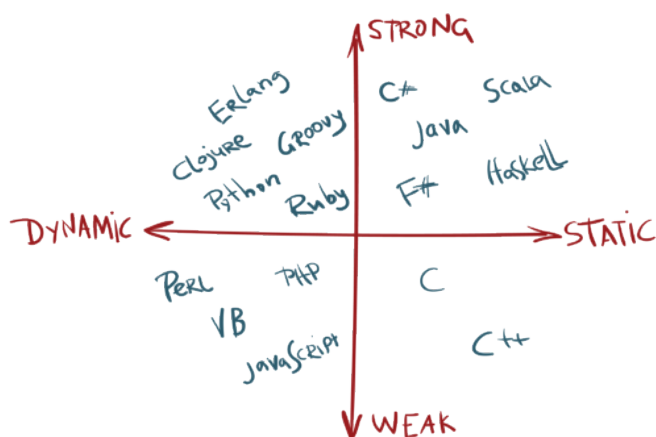
Kada program nije tipski siguran, javiće se greška u tipiziranju, i u zavisnosti koji se programski jezik (i koji kompajler ili interpreter) koristi, greška će obično zaustaviti izvršenje programa.

Postoje dve podele tipiziranja podataka.

Jedna podela jeste na *jako tipizirane* (en. *strong typed*) i *slabo tipizirane* (en. *weak typed*) programske jezike.

Druga podela jeste na *statičko tipizirane* (en. *statically typed*) i *dinamičko tipizirane* (en. *dynamically typed*) programske jezike.

Po ovoj podeli, svaki programski jezik spada u jednu od četiri moguće kombinacije.



Slika 3.1 Podela programskih jezika po načinu tipiziranja podataka. [Izvor: <https://itnext.io>]

U nastavku biće reči o svakoj od ovih kategorija tipiziranja.

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

JAKO I SLABO TIPIZIRANJE

Kod jako tipiziranih programskih jezika, vrednost promenljive mora da se poklopi sa tipom podataka koji se očekuje.

Jako tipiziran programski jezik

Kod programskog jezika koji spada u kategoriju jako tipiziranog jezika promenljive su vezane za konkretan tip podataka.

Prilikom kompajliranja ili interpretacije, ukoliko se vrednost promenljive ne poklopi sa tipom podataka koja se očekuje u izrazu.

Primer: Dodavanje numeričke vrednosti ne-numeričkoj promenljivoj.

```
temp = "Hello World!"
temp = temp + 10
```

Ukoliko pokušamo ovako nešto u programskom jeziku koji je jako tipiziran, doći će do greške.

Slika 3.2 Greška tipiziranja u Python 3.x jeziku. [Izvor: Autor]

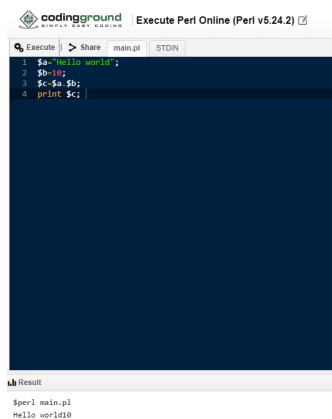
Programski jezici koji imaju jako tipiziranje jesu Java, Python, C#, Ruby, Haskell, sok slabo tipiziranje imaju C, C++, Perl, PHP, JavaScript.

Slabo tipiziran programski jezik

Kod programskoj jezika koji spada u kategoriju slabo tipiziranoj jezika, promenljive nisu vezane za konkretan tip podataka. Promenljive i dalje imaju tip podataka, ali su ograničenja manja u odnosu na jako tipizirane jezike. Sledeći primer daje sintaksu u [Perl](#) jeziku.

Primer: Dodavanje numeričke vrednosti ne-numeričkoj promenljivoj.

```
$a="Hello world";  
$b=10;  
$c=$a.$b;  
print $c;
```



Slika 3.3 Ne dolazi do greške tipiziranja u Perl jeziku. [Izvor: Autor]

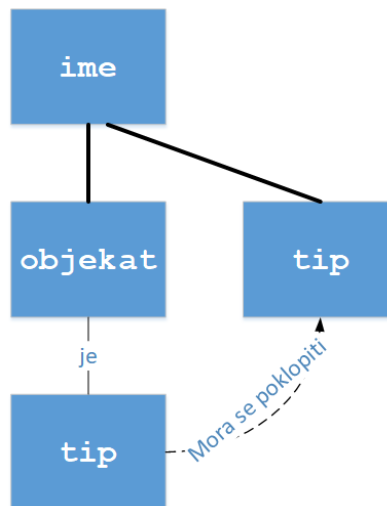
STATIČKO TIPIZIRANJE

Kod statičko tipiziranih programskih jezika, promenljivoj se ne može nakdnano dodeliti drugi tip podataka.

Statičko tipiziran programski jezik

Kod programskog jezika koji spada u kategoriju statičko tipiziranog jezika, tip promenljive je poznat u vremenu kompajliranja (en. **compile-time**) umesto u vremenu izvršenja (en. **run-time**).

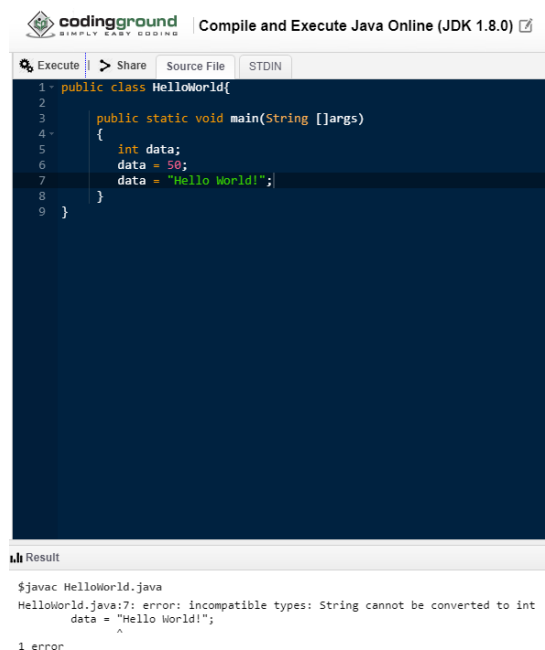
Ovo znači da kada se deklarise tip podataka promenljivoj, ne može se dodeliti drugom tipu podataka. Ukoliko se to desi, u vremenu kompajliranja desiće se ponovo greška u tipiziranju.



Slika 3.4 Ograničenja tipiziranja. [Izvor: Autor]

Primer: Promena tipa podataka u Javi

```
int data;
data = 50;
data = "Hello World!";
```



Slika 3.5 Greška u tipiziranju. [Izvor: Autor]

DINAMIČKO TIPIZIRANJE

Kod dinamičko tipiziranih programskih jezika, promenljivoj se može naknadno dodeliti drugi tip podataka.

Dinamičko tipiziran programski jezik

Kod programskog jezika koji spada u kategoriju dinamičko tipiziranog jezika, tip promenljive proverava u vremenu izvršenja programa.

Ovo efektivno znači da su promenljive vezane za tip podataka tek kada se vrši izvršenje programa, i moguće je vezati iste promenljive za različite tipove podataka.



Slika 3.6 Nepostojanje ograničenja tipiziranja. [Izvor: Autor]

Provera tipiziranja u dinamičko tipiziranom jeziku rezultuje u manje optimalnom kodu u odnosu na programske jezike sa statičkom proverom tipiziranja.

Posledica jeste mogućnost pojave grešaka tipiziranja u vremenu izvršenja, odnosno svakog tupa kada se program izvrši.

Primer: Promena tipa podataka u Python 3.x jeziku

```
data = 10  
data = "Hello World!"
```

```
C:\>python  
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bi  
t (Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> data = 10  
>>> data = "Hello World!"  
>>>
```

Slika 3.7 Nema greške u tipiziranju. [Izvor: Autor]

KADA KORISTITI STATIČKO, A KADA DINAMIČKO TIPIZIRANJE?

Nekada je povoljnije koristiti programske jezike koji imaju statičko tipiziranje, ali nekad prevladavaju povoljnosti dinamičkog tipiziranja.

Nekada je povoljnije koristiti programske jezike koji imaju statičko tipiziranje, ali nekad prevladavaju povoljnosti dinamičkog tipiziranja.

Prednosti statičkog tipiziranja

- Velika grupa grešaka koja se javlja može se primetiti (i otkloniti) u ranoj fazi razvoja programa.
- Kompajlirani kod se izvršava brže jer kompajler tačno zna koji se tipovi podatka koriste, i na taj način može napisati optimizovan mašinski kod.

Prednosti dinamičkog tipiziranja

- Prilikom izvršenja programa, svaki objekat ima svoj tag odnosno referencu o tipu (en. **type reference**). Ova informacija (en. **run-time type information**, RTTI) se može iskoristiti za implementaciju dodatnih funkcija.
- Nepostojanje koraka kompajliranja omogućava da se kod može izmeniti a da se ne čeka svaki put na kompajliranje. Ovo čini ciklus debugovanja kraćim.

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

IME PROMENLJIVE I OBJEKTI U MEMORIJI U PYTHON JEZIKU

Treba znati razliku između imena promenljive, objekta koji sadrži vrednost, i njihovu vezu.

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

▼ Poglavlje 4

Kontrola toka

ŠTA PREDSTAVLJA KONTROLA TOKA U PROGRAMIRANJU?

Kontrola toka je eksplicitna u imperativnom programskom jeziku kao što je Python 3.x.

Kontrola toka (en. **control flow**) predstavlja redosled po kojim pojedinačne naredbe, izrazi, pozivi funkcija i instrukcije se pozivaju prilikom izvršenja programa pisanog u imperativnom programskom jeziku.

Podsetnik:

Eksplicitna kontrola toka jeste ono što razlikuje imperativni programski jezik od deklarativnog programskog jezika.

U imperativnom programskom jeziku, naredba za kontrolu toka daje opciju o biranju dve ili više putanje.

Mehanizmi kontrole toga niskog nivoa

Prekid (en. **interrupt**) i signal predstavljaju mehanizme kontrole toga niskog nivoa koje mogu promeniti tok izvršenja programa na sličan način kao i pod-rutina (en. **subroutine**).

Za razliku od in-line naredbe za kontrolu toga, mehanizmi niskog nivoa obično se dešavaju po nekom eksternom pozivu ili događaju, koji je obično asinhron.

Na nivou mašinskog i asemblerskog jezika, instrukcije za kontrolu toga rade tako što se menja programski brojač. Centralna procesorska jedinica (en. central processing unit, CPU) razume ove uslovne i безусловne instrukcije koje se nazivaju instrukcije skoka (en. **jump**).

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

OSNOVE KONTROLA TOKA U PYTHON 3.X JEZIKU

U bilo kom programskom jeziku, pa i u Python 3.x, biće potrebno da kontrolišete tok programa.

U bilo kom programskom jeziku, pa i u Python 3.x, biće potrebno da kontrolišete tok programa.

Kao što je rečeno, kontrola toka se javlja kada je potrebno da se napravi neka odluka - ukoliko je ispunjen neki uslov, izvršiće se jedan skup koda, a ako nije, izvršiće se drugi.

U Python 3.x jeziku, postoje više uslovnih naredbi (en. **conditional statements**), koje služe za kontrolu toka. To su:

- **if**
- **elif**
- **else**

Najjednostavnija naredba je, naravno **if**, nešto složeniji skup naredbi jeste **if-else**, dok je **if-elif-else** najsloženiji od sva tri.

U nastavku lekcije biće reči i o ugnježdenim uslovnim naredbama (en. **nested conditional statements**)

Naredba if

Najjednostavnija uslovna naredba koja proverava uslov, i vraća Boolean vrednost kao rezultat. Ukoliko je vrednost True, onda će se deo koda neposredno ispod uslova, tj. nakon ključnog simbola dvotačke **:** izvršiti.

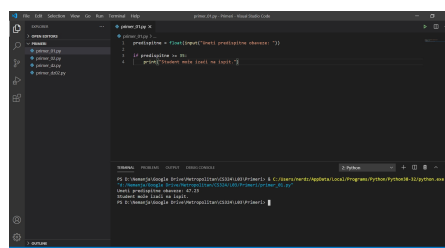
Generalna sintaksa izgleda:

```
if (uslov):  
    Deo koda ako uslov vrati True
```

Primer - Provera uslova izlaska na ispit

Program proverava da li je unet broj predispitnih obaveza veći od 35. Ukoliko jeste, ispisuje da student može izaći na ispit.

```
predispitne = float(input("Uneti predispitne obaveze: "))  
  
if predispitne >= 35:  
    print("Student može izaći na ispit.")
```



Slika 4.1 Primer uslovne komande if. [Izvor: Autor]

NAREDBE IF I IF-ELSE U PYTHON 3.X JEZIKU

Kod Python 3.x jezika ključni znak dvotačka razdvaja uslov od naredbi koje treba da se izvrše posle provere uslova.

Specifičnosti za Python 3.x jezik

Može se primetiti da posle naredbe uslova stoji ključni znak dvotačka :

U Python 3.x jeziku dvotačka se koristi da bi razdvojila uslov od naredbi koje treba da se izvrše posle provere uslova.

Naredbe if-else

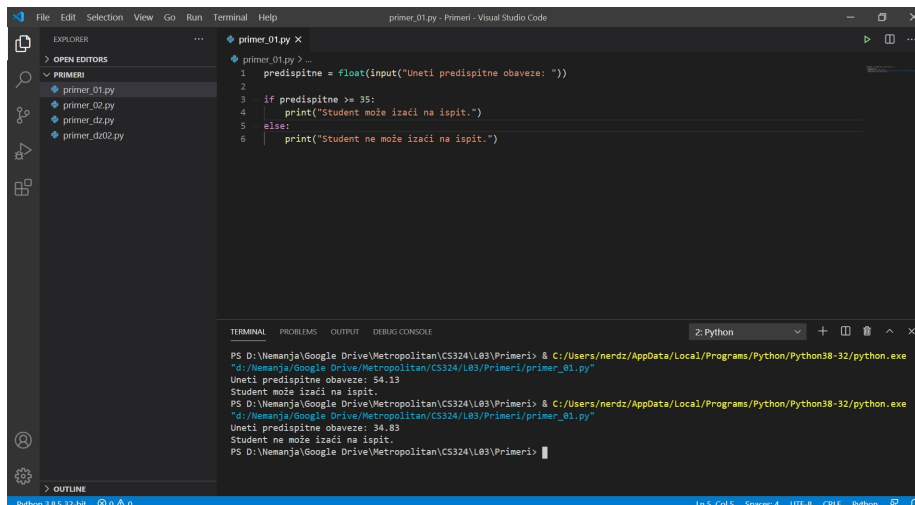
Ukoliko postoji potreba da se izvrši deo koda i kada nije ispunjen uslov (tačnije, jedan deo koda se izvršava kada uslov vrati True, a drugi deo koda se izvršava kad ulov vrati False), onda se koristi kombinacija if i else naredbi. Sintaksa je sledeća:

```
if (uslov):  
    Deo koda ako uslov vrati True  
else:  
    Deo koda ako uslov vrati False
```

Primer - Provera uslova izlaska na ispit (nastavak)

U nastavku prilikom unošenja poena za predispitne obaveze izlaz štampa i kada student može, i kada ne može da izađe na ispit.

```
predispitne = float(input("Uneti predispitne obaveze: "))  
  
if predispitne >= 35:  
    print("Student može izaći na ispit.")  
else:  
    print("Student ne može izaći na ispit.")
```



The screenshot shows the Visual Studio Code interface. The Explorer pane on the left shows a file named 'primer_01.py'. The main editor area displays the following Python code:

```
1 predispitne = float(input("Uneti predispitne obaveze: "))  
2  
3 if predispitne >= 35:  
4     print("Student može izaći na ispit.")  
5 else:  
6     print("Student ne može izaći na ispit.")
```

The Terminal pane at the bottom shows the execution of the script. It displays the prompt 'Uneti predispitne obaveze: 54.13' followed by the output 'Student može izaći na ispit.'. Below that, it shows the prompt 'Uneti predispitne obaveze: 34.83' followed by the output 'Student ne može izaći na ispit.'. The status bar at the bottom indicates the file is 'primer_01.py' and the Python version is 'Python 3.8.5 32-bit'.

Slika 4.2 Primer uslovne komande if-else. [Izvor: Autor]

NAREDBE IF-ELIF-ELSE

Python podržava višestruku proveru uslova u if-elif-else skupu naredbi.

If-elif-else

Python podržava višestruku proveru uslova u if-elif-else skupu naredbi.

Najpre se proverava prvi uslov. Ukoliko on vrati False, proverava se uslov u elif (elif je of else-if). Ukoliko i taj uslov vrati False, onda se ide na (čistu) else granu koja izvršava liniju koda. Sintaksa je sledeća

```
if (uslov1):
    Deo koda ako vrati True uslov1
elif (uslov2):
    Deo koda ako vrati False na uslov1, a True na uslov2
else:
    Deo koda ako vrati False na uslov1 i False na uslov2
```

Primer: Provera poena na predispitnim obavezama i projektu

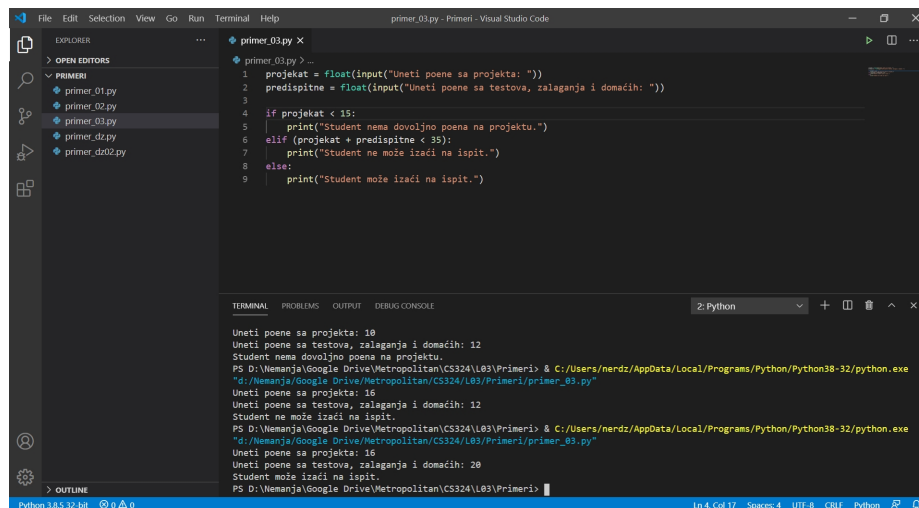
Uneti poene sa predispitnih obaveza i projekta.

Ukoliko student nema 15 poena na projektu, izlaz javlja da nema dovoljno poena na projektu.

Ukoliko ima, proverava se da li zbor poena na projektu i ostalim predispitnim obavezama iznosi barem 35. Ukoliko iznosi, izlaz javlja da student može izaći na ispit, a ukoliko nema, izlaz javlja da student ne može izaći na ispit.

```
projekat = float(input("Uneti poene sa projekta: "))
predispitne = float(input("Uneti poene sa testova, zalaganja i domaćih: "))

if projekat < 15:
    print("Student nema dovoljno poena na projektu.")
elif (projekat + predispitne < 35):
    print("Student ne može izaći na ispit.")
else:
    print("Student može izaći na ispit.")
```



Slika 4.3 Primer if-elif-else naredbi. [Izvor: Autor]

PRIMER IF/ELIF/ELSE: POENI NA PREDISPITNIM OBAVEZAMA I ISPITU VIDEO

Video na kojem je objašnjen primer korišćenja if-elif-else naredbi

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

UGNJEŽDENE USLOVNE NAREDBE

Ukoliko želimo da imamo dodatno grananje unutar grane koda, korsitićemo ugnježđeno grananje.

budući da je sintaksa if-else naredbi takva:

```

if (uslov):
    Deo koda ako uslov vrati True
else:
    Deo koda ako uslov vrati False
    
```

ništa ne sprečava programere da u *Delu koda ako uslov vrati True/False* ubaci ponovo neke uslove i if-else naredbe.

Ovo se nazivaju ugnježdene uslovne naredbe. Opšti primer bio bi:

```

if (uslov1):
    #Deo koda ako uslov1 vrati True
    if (uslov2)
        #Deo koda ukoliko uslov2 vrati True
    else:
        #Deo koda ukoliko uslov2 vrati False
    
```

```
else:  
    #Deo koda ako uslov1 vrati False  
    if (uslov3)  
        #Deo koda ukoliko uslov3 vrati True  
    else:  
        #Deo koda ukoliko uslov3 vrati False
```

Podestnik:

Bitno je voditi računa da kod koji se piše bude pregledan za kasnije čitanje, bilo od Vas kao autora koda, bilo od strane drugih. Poštujte principe čistog koda (en. clean code).

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

Pitanje:

Ukoliko želimo više uslova ispitati u jednoj proveru, kako je to moguće uraditi?

▼ Poglavlje 5

Petlje

UVOD U PETLJE

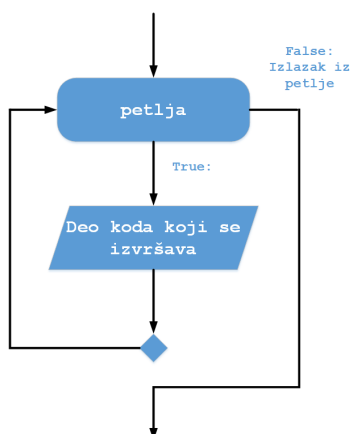
Petlja predstavlja naredbu u programu koja ponavlja deo koda dok se ne ostvari neki uslov.

Petlja (en. **loop**) predstavlja jedan od najosnovnijih koncepata programiranja.

Petlja predstavlja naredbu u programu koja ponavlja deo koda dok se ne ostvari neki uslov.

Opšta struktura petlje

Može se reći da petlja postavlja pitanje. Ukoliko odgovor na to pitanje zahteva neku radnju, ta radnja se izvršava. Isto pitanje se ponovo pita sve dok nije potrebno ništa više uraditi. Svaki put kada se postavlja pitanje čija je posledica izvršenje koda, dešava se jedna iteracija (en. **iteration**).



Slika 5.1 Opšta struktura petlje. [Izvor: Autor]

Najveća prednost korišćenja petlji jeste da programer ne mora više puta koristiti isti kod.

Svi programski jezici sadrže neki vid petlji.

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

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

PYTHON 3.X WHILE PETLJA

While petlja u Python jeziku identična je kao i kod drugih programskih jezika.

Python 3.x podržava dva osnovna tipa petlji: *while* i *for*.

Sintaksa while petlje je sledeća:

```
while (uslov):
    Deo koda kada je uslov ispunjen
```

Sve dok je ispunjen uslov deo koda koji ide neposredno ispod while (tačnije, nakon znaka :) izvršavaće se u iteracijama. Dobra (pravilna) praksa jeste unutar samog koda uneti naredbu koja će menjati neku promenljivu koja se ispituje u uslovu.

Ukoliko ne postoji naredba koja menja stanje uslova, javiće se tzv. beskonačna petlja (en. *infinite loop*) koja uglavnom izaziva probleme prilikom izvršenja programa.

Pitanje:

Kako najjednostavnije napraviti beskonačnu petlju?

Koji problemi se javljaju ukoliko postoji u programu beskonačna petlja?

Primer: Ispisivanje celih brojeva

Napisati program koji ispisuje sve cele brojeve od 0 do proizvoljnog broja n.

```
n = int(input("Uneti poslednji broj: "))
i = 0
while i <= n:
    print(i)
    i += 1
```

BESKONAČNA PETLJA

Kod while petlje treba voditi računa na pojavu beskonačne petlje.

Primer: Beskonačna petlja

U ovom primeru samo je neznatno promenjen uslov, koji dovodi do beskonačne petlje. Pri izvršenju ovog programa moraćemo ručno da prekinemo.

```
n = int(input("Uneti poslednji broj: "))
i = 0
while n:
```

```
print(i)
i += 1
```

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

PYTHON 3.X FOR PETLJA

For petlje u Python 3.x jeziku kreću od nultog indeksa.

For petlje u Python 3.x jeziku nemaju sintaksu nalik na C jeziku u njemu slične jezike, već je sintaksa

```
for in n
```

For petlje se mogu koristiti za iteraciju ne samo za numeričke tipove podataka, već i za liste, stringove, tuple-ove i nizove.

Ukoliko je dat skup elemenata u listi, for petlja se može iskoristiti da izvrši iteraciju nad svakim elementom liste.

Da bi petlja prošla svaku iteraciju, koristi se funkcija *range()*.

Funkcija range() vraća novu listu sa brojevima specifičnog opsega u odnosu na dužinu sekvence.

Kroz iteraciju kroz sekvencu mogu se takođe iskoristiti i indeksi elemenata sekvence za iteraciju, nali najpre treba izračunati kolika je dužina liste i tek onda proći kroz iteracije unutar opsega te dužine.

For petlje u Python 3.x jeziku kreću od nultog indeksa.

Osnovni primer korišćenja for petlje

Primer: Štampanje brojača (od nule)

```
for i in range(10):
    print(i)
```

Primer: Štampanje brojača (od jedinice)

```
for j in range(10):
    print(j+1)
```

Primer: Štampanje brojača sa korakom 2

```
for j in range(1,20,2):
    print (j)
```

Primer: Štampanje karaktera i u tekstu "CS324 Skripting jezici"

```
for k in "CS324 Skripting jezici":  
    if k == 'i':  
        print (k)
```

Primer: Štampanje elemenata liste

```
predmeti = ["CS220", "CS225", "CS324", "IT331", "IT335"]  
for ii in predmeti:  
    print(ii)
```

Primer: Štampanje elemenata heterogene liste

```
rnd_list = [1, 2, 20.54, 100 + 2.3j, 324, 'L03', 'Skripting Jezici']  
for jj in rnd_list:  
    print(jj)
```

DODATNI PRIMERI I VIDEO SA PYTHON 3.X FOR PETLJAMA

Primer sa listama i petljama

Do sada je prikazano više osnovnih primera u radu sa for petljom.

Može se primetiti da je for petlja jako korisna za dobijanje elemenata liste.

Međutim, ukoliko nekada želimo samo da saznamo koliko je lista dugačka, tj. koliko elemenata ima u sebi, možemo koristiti funkciju len().

Ova funkcija vraća kao izlaz element liste.

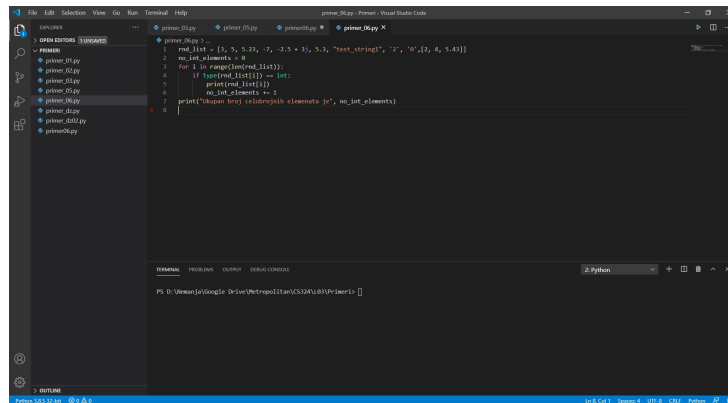
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Primer:

Napraviti listu koja sadrži 10 elemenata, a da sadrži celobrojne, kompleksne, razlomljene brojeve, stringove, i jednu listu kao elemente.

Napisati program koji će ispisati broj elemenata koji su celi brojevi, ali i te elemente štampati na izlazu.

```
rnd_list = [3, 5, 5.23, -7, -2.5 + 3j, 5.3, "test_string1", '2', '0', [2, 4, 5.43]]  
no_int_elements = 0  
for i in range(len(rnd_list)):  
    if type(rnd_list[i]) == int:  
        print(rnd_list[i])  
        no_int_elements += 1  
print("Ukupan broj celobrojnih elemenata je", no_int_elements)
```

```

1 my_list = [1, 5, 5.01, -1, -2.5 * 31, 5.1, "test_string", "2", "0"[2, 4, 5.43]]
2
3 my_int_elements = 0
4 for i in range(len(my_list)):
5     if type(my_list[i]) == int:
6         print(my_list[i])
7         my_int_elements += 1
8     print("ukupan broj celobrojnih elemenata je", my_int_elements)
9

```

Terminal Output:

```

PS D:\Metropol\poglavje-05\petlje\01\primer1>

```

Slika 5.2 Primer programa koji broji celobrojne elemente liste. [Izvor: Autor]

▼ Poglavlje 6

Pokazna vežba #3

UVOD U POKAZNU VEŽBU #3

Izlaz, tačnije print() funkcija Python 3.x programskog jezika se može formatirati na više načina.

U pokaznoj vežbi biće reči o formatiranju izlaza u različitim formatima.

Nakon toga, biće obrađeno par primera koji opisuju operacije nad numeričkim i ne-numeričkim podacima.

Procenjeno trajanje pokazne vežbe iznosi 45 minuta.

FORMATIRANJE IZLAZA

Izlaz (štampa) se može različito formatirati u Python 3.x jeziku

Formatiranje izlaza u Python 3.x jeziku može se ostvariti na klasični %način (poznat korisnicima C familije jezika), korišćenjem `.format()` metode, ali i i putem novog, jednostavnijeg *f-string* načina.

U nastavku vežbe biće dati primeri za svaki od tipova formatirana izlaza.

Klasični način formatiranja izlaza

U Python 3.x jeziku koristi se % placeholder za formatiranje izlaza, i može se iskoristiti za više tipova podataka. Ovaj način formatiranja se u literaturi naziva i *printf-stil formatiranja*, jer podseća na *printf* funkciju iz programskog jezika C.

U savremenom programiranju, ovakav način pisanja izlaza u Python 3.x jeziku je zastareo, i najviše se koriste sledeća dva načina - `.format()` metoda, i f-string formatiranje.

Spisak %-formata

- %d - označeni decimalni ceo broj
- %o - označeni oktalni broj
- %x - označeni heksadecimalni broj (mala slova)
- %X - označeni heksadecimalni broj (velika slova)
- %e - razlomljeni broj u eksponencijalnom obliku (mala slova)
- %E - razlomljeni broj u eksponencijalnom obliku (velika slova)
- %f - razlomljeni decimalni broj

- %g - razlomljeni decimalni broj (automatski bira da li koristi eksponencijalnu notaciju ili ne)
- %c - karakter
- %s - string
- %% - vraća karakter "%"

FORMATIRANJE IZLAZA POMOĆU KLASIČNOG NAČINA - PRIMER

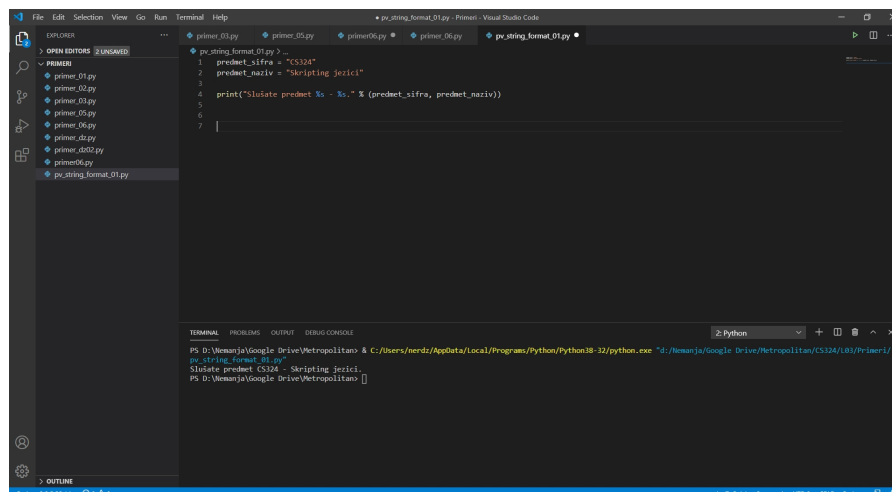
Izlaz se može formatirati u Python 3.x jeziku na klasični način, tj. korišćenjem %

Primer #1: %-formatiranje stringova klasičnim načinom (15 minuta)

Napisati program koji ispisuje šifru predmeta i naziv predmeta koji se trenutno sluša, korišćenjem % formatiranja izlaza.

```
predmet_sifra = "CS324"  
predmet_naziv = "Skripting jezici"  
print("Slušate predmet %s - %s." % (predmet_sifra, predmet_naziv))
```

Primećuje se da u okviru izlaznog stringa koji ispisuje vrednost neke promenljive stoji karakter kojem prethodi znak procenta %. To predstavlja **placeholder** za promenljivu, koja se redom navodi nakon kraja stringa, ponovo sa % znakom pre navođenja.



Slika 6.1 Formatiranje izlaza % načinom. [Izvor: Autor]

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

FORMATIRANJE POMOĆU .FORMAT() METODE

Metoda .format() olakšava štampanje izlaza u odnosu na % način, jer se ne mora brinuti o tome kakva je promenljiva

.format() način formatiranja izlaza

Umesto eksplicitnog navođenja tipa podataka kroz % placeholder, u izlaznom stringu se može navesti "mesto" za promenljivu, tako što će promenljiva biti "smeštena" u velike zagrade {}.

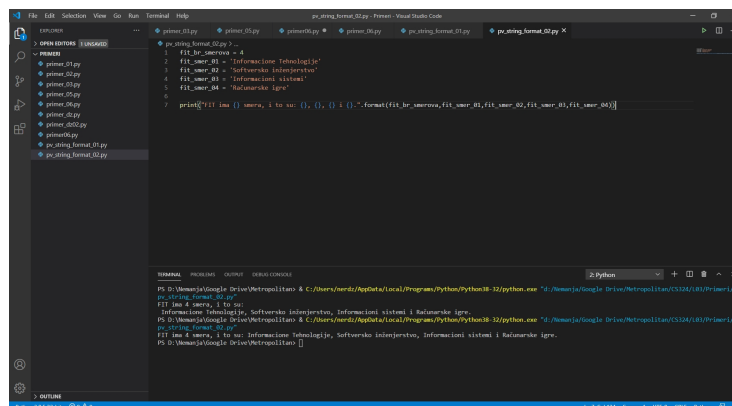
Nakon pisanja samog stringa, treba pozvati metodu .format(), koja u svojoj listi argumenata ima spisak promenljivih.

Primer #2: formatiranje stringova .format() metodom (15 minuta)

Napisati program koji ispisuje broj i nazive smerova na Fakultetu Informacionih Tehnologija Univerziteta Metropolitan. Koristiti .format() metodu za formatiranje izlaznog stringa.

```
fit_br_smerova = 4
fit_smer_01 = 'Informacione Tehnologije'
fit_smer_02 = 'Softversko inženjerstvo'
fit_smer_03 = 'Informacioni sistemi'
fit_smer_04 = 'Računarske igre'

print("FIT ima {} smerova, i to su: {}, {}, {} i  
{}, {}".format(fit_br_smerova, fit_smer_01, fit_smer_02, fit_smer_03, fit_smer_04))
```



Slika 6.2 Formatiranje izlaza .format() načinom. [Izvor: Autor]

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

FORMATIRANJE POMOĆU F-STRING METODE

Najnoviji format koji podržava Python 3.x programski jezik jeste tzv. f-string formatiranje

f-string formatiranje izlaza

F-string formatiranje uvedeno je u Python 3.6 verziji avgusta 2015. godine.

Kod ovakvog formatiranja na početku stringa dodato je slovo *f*, a unutar velikih zagrada `{}` ubacuje se promenljiva čija se vrednost ispisuje.

Primer #3: Računanje vrednosti broja Pi (15 minuta)

Napisati program koji računa približnu vrednost broja Pi korišćenjem *Nilakanta* aproksimacije:

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \dots$$

Na izlazu koristiti f-string formatiranje.

```
var_pi_str = 'π'
pi_temp = 0

for i in range(1, 1 + int(input("Uneti željenu tačnost: "))):
    if i % 2 == 1:
        pi_temp += 4/((2*i) * ((2*i)+1) * ((2*i)+2))
    else:
        pi_temp -= 4/((2*i) * ((2*i)+1) * ((2*i)+2))
var_pi_approx = 3 + pi_temp

print(f"Vrednost broja {var_pi_str} izosi približno {var_pi_approx}.")
```

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

▼ Poglavlje 7

Individualna vežba #3

UVOD U INDIVIDUALNU VEŽBU #3

Individualna vežba #3 odnosi se na rad sa petljama i kontrolom toka.

Individualna vežba #3 daje tri primera u kojima se radi sa petljama, kontrolom toka, i formatiranjem izlaza.

Procenjeno trajanje individualnih vežbi iznosi 90 minuta.

INDIVIDUALNA VEŽBA #3

Napisati jednostavne programe u Python programskoj jeziku koristeći instalirana okruženja.

Zadatak #1 (25 minuta)

Napisati program koji ispisuje tablicu istinitosti za izraz

$x = (a \text{ OR } b) \text{ AND NOT}(c)$

u Bool / numeričkom obliku.

Koristiti:

1. štampanje izlaza samo sa promenljivima
2. .format() metodu za štampanje izlaza uz pravilni alignment

Zadatak #2 (30 minuta)

Napisati program koji broj četvorocifren broj indeksa pretvara u BCD format.

Zadatak #3 (35 minuta)

Napisati program koji računa vrednost broja φ po sledećem beskonačnom redu:

$$\varphi = \frac{13}{8} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1} (2n+1)!}{4^{2n+3} n! (n+2)!}$$

Umesto beskonačnosti uneti krajnji član reda.

Zatim, ispisati vrednost broja φ ako je krajnji član 10, 100 i 1000 puta veći od unetog.

▼ Poglavlje 8

Domaći zadatak #3

DOMAĆI ZADATAK

Domaći zadatak #3 se okvirno radi 2h

Domaci zadatak #3

Zadatak #1

Napisati program u kojem ubacujete broj trenutno položenih ispita. Ulazne podatke sačuvati u imenik koji sadrži šifru predmeta kao ključ predmeta, a vrednost jeste ocena na položenom predmetu.

Zatim, izvršiti upit za predmet po šifru predmeta.

Izbaciti kao izlaz da li je predmet položen ili ne, i ako jeste, sa kojom ocenom.

Zadatak #2

Napisati program koji će generisati n nasumičnih celih brojeva, gde je n broj indeksa studenta, i smestiti u listu, i sortirati listu u padajućem redosledu. Koristiti `import random` za dobijanje nasumičnih brojeva.

Zadatak #3

Napisati program koji će generisati m nasumičnih razlomljenih brojeva unifromne raspodele tako da je opseg od prve dve cifre indeksa do druge dve cifre, i $m = \text{broj_indeksa} // 3$. Smestiti brojeve u imenik, tako je ključ redni broj, a vrednost sam razlomljeni broj.

Zatim, izvući `broj_indeksa % 3` elementa imenika. Koristiti `import random`.

Primer: Ako je broj indeksa 1234, koristiti:

```
random.uniform(12, 34)
```

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 #3

Rezime:

U ovoj lekciji bilo je reči o standardnim tipovima podataka koje Python 3.x programski jezik koristi.

Bilo je reči o logičkim, numeričkim i ne-numeričkim tipovima podataka.

Zatim, govorilo se o kontroli toka i o if-else uslovnim naredbama.

Konačno, bilo je reči o petljama while i for.

Sve ove 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.

