



CS324 - SKRIPTING JEZICI

Datoteke i izuzeci

Lekcija 06

PRIRUČNIK ZA STUDENTE

CS324 - SKRIPTING JEZICI

Lekcija 06

DATOTEKE I IZUZECI

- ✓ Datoteke i izuzeci
- ✓ Poglavlje 1: Čitanje iz datoteka
- ✓ Poglavlje 2: Pisanje u datoteku
- ✓ Poglavlje 3: Čuvanje podataka
- ✓ Poglavlje 4: Izuzeci i obrada izuzetaka
- ✓ Poglavlje 5: Debagovanje u Python 3.x jeziku
- ✓ Poglavlje 6: Pokazna vežba #6
- ✓ Poglavlje 7: Individualna vežba #6
- ✓ Poglavlje 8: Domaći zadatak #6
- ✓ 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

Uvod u datoteke i obradu izuzetaka

Kao i kod većine programskih jezika, postoje nekoliko načina se da izlaz iz programa pokaže korisniku. Podaci se mogu štampati u obliku koji korisnik može da pročita, ali, možda još bitnije, može se napisati u datoteku koja se može kasnije iščitati ili iskoristiti za druge svrhe. Bilo da radite u Python-u za razvoj web ili desktop aplikacija, sigurno ćete se susretati sa datotekama, i potrebno je znati imati pravilnu interakciju sa datotekama.

Python, naravno, ima ugrađene funkcije za ulaz i izlaz. Ugrađenu funkciju open najčešće ćete koristiti za otvaranje datoteka, naravno uz odgovarajuće parametre.

Nije potrebno samo da otvorite datoteku, već da u nju nešto i upišete. Na primer, za upisivanje tekstualnih sadržaja koristite funkciju write. Konačno, posle završetka rada sa datotekom, zatvorite je funkcijom close.

Pre ili kasnije, desiće se da imate greške u kodu. Kao budući programeri i inženjeri računarske tehnike, veoma je bitno da znate kako te greške obratiti. U nastavku lekcije govoriće se o obradi izuzetaka i grešaka.

Proces debagiranja ili debagovanja, kako već želite da izgovarate, je sličan kao i u ostalim programskim jezicima, ali ovde ćemo spomenuti specifičnosti Python jezika koji ima try, except i finally komande za obradu izuzetaka.

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

✓ Poglavlje 1

Čitanje iz datoteka

UVOD U RAD SA DATOTEKAMA

Pisanje programa koji koriste datoteke za čitanje i pisanje podataka rešava problem korišćenjem tastature kao jedinog ulaza, i konzole kao jedinog izlaza.

Programi koji su bili pisani do sada čitali su sav ulaz sa tastature korisnika. Kao rezultat, ovaj pristup je zahtevao da se svi ulazni parametri ukucavaju svaki put kada se pokrene program. Ovaj pristup je neefikasan, naročito za programe koji zahtevaju unos velike količine podataka.

Na sličan način, programi koji su do sada pisani prikazivali su rezultate isključivu na ekranu (tačnije na konzoli ekrana). Ovaj pristup jeste adekvatan kada se štampa izlaz koji sadrži samo nekoliko linija teksta, ali je nepraktičan za veće rezultate, jer se prebrzo ispisuju na ekran. Još bitnije ovaj pristup je neefikasan kada je potrebno da izlaz bude bio naknadno analiziran od strane drugih programa.

Pisanje programa koji koriste datoteke za čitanje i pisanje podataka rešava problem korišćenje tastature kao jedinog ulaza, i konzole kao jedinog izlaza.

Datoteka (en. **file**) predstavlja skup podataka sadržanu u jednoj celini, identifikovana po imenu i tipu (ekstenziji, en. **extension**). Datoteka može biti tekstualni dokument, slika, audio ili video podatak, biblioteka podataka, izvršni program ili bilo koji skup podataka.

Datoteke se mogu otvoriti, sačuvati, obrisati i prenesti u različite direktorijume unutar sistema datoteka.

Direktorijum (en. **directory**, **folder**) predstavlja kataloški sistem koji sadrži reference na datoteke.

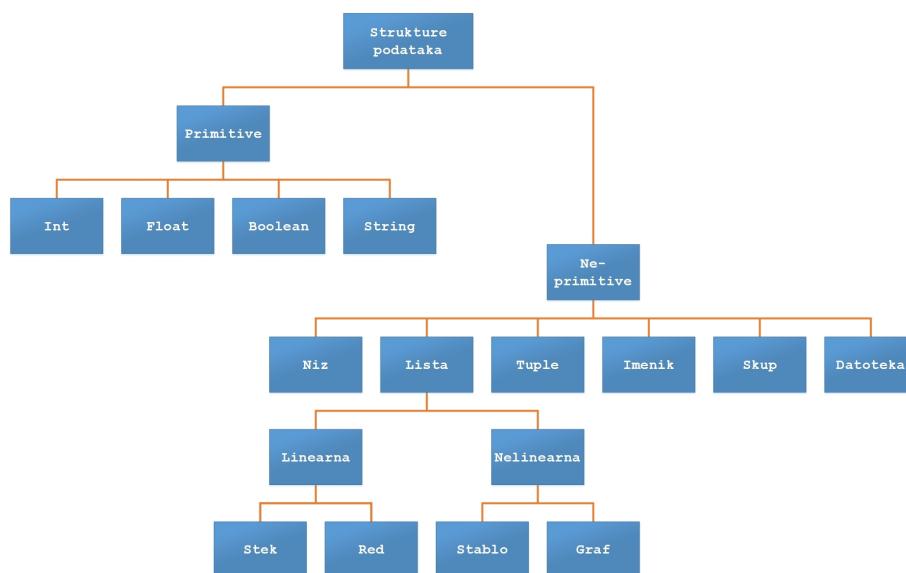
Sistem datoteka (en. **file system**) kontroliše kako se podaci smeštaju i pristupaju unutar računarskog sistema.

Bez sistema datoteka, podaci koji se nalaze u medijumu za smeštanje podataka bili bi jedan veliki skup podataka bez početka i kraja svake datoteke. Sistem datoteka služi da svaku datoteku zasebno smesti i imenuje, da bi se podaci mogli lako identifikovati i da bi im se moglo lako pristupiti.

RAD SA DATOTEKAMA U PYTHON PROGRAMSKOM JEZIKU

*Datoteke se najčešće dele na tekstualne datoteke i binarne datoteke.
Većina principa važi za oba tipa datoteka.*

Ako se pogleda struktura podataka koji Python 3.x jezik podržava, datoteka spada u ne-primitivu.



Slika 1.1.1 Struktura podataka koji Python 3.x jezik podržava. [Izvor: Autor]

Datoteke su relativno stalne po sadržaju. Vrednosti koje se čuvaju u datotekama se zadržavaju nakon što se program izvrši i nakon prestanka napajanja računara električnom energijom. Zbog ove osobine, datoteke jesu jako pogodne za smeštanje rezultata koji su potrebni za duži vremenski period, kao i za smeštanje ulaznih podataka za program koji treba izvršiti više puta.

Datoteke se najčešće dele na tekstualne datoteke (en. text files) i binarne datoteke (en. binary files).

Tekstualne datoteke sadrže samo sekvence bitova koji predstavljaju alfanumeričke karaktere, i koriste sistem kodovanja **ASCII** ili **UTF-8**. Ove datoteke se mogu otvoriti i modifikovati sa bilo kojim editorom teksta.

Dosadašnji Python programi su bile tekstualne datoteke.

Kao i tekstualne datoteke, binarne datoteke sadrže sekvence bitova. Međutim, za razliku od tekstualnih datoteka, ove sekvence mogu predstavljati bilo koji tip podataka. Tačnije, nisu ograničene na alfanumeričke karaktere. Većina principa koji važe za tekstualne datoteke važiće i za binarne datoteke.

1.1 Funkcija open()

OSNOVE FUNKCIJE OPEN()

Najjednostavniji načina otvaranje datoteke u Python 3.x jeziku jeste korišćenje funkcije `open()`, koja prima 3 argumenta (od kojih su dva opciona).

Najjednostavniji načina otvaranje datoteke u Python 3.x jeziku jeste korišćenje funkcije [open\(\)](#).

Najpre treba imati datoteku sa podacima. U nastavku je dat sadržaj u deset linija koda koji treba prekopirati i smestiti u **test_podaci.txt** datoteku i smestiti je u radni direktorijum Python programa.

Ukoliko se radi sa datotekama koje nisu u radnom direktorijumu, potrebno je naći relativnu ili absolutnu putanju do datoteke.

```
1] Ovo je datoteka sa test podacima
2] Takodje ima u sebi broj linija
3] Ovo je treca linija datoteke
4] Ovo je cetvrta linija datoteke
5] Ovo je, tako je, peta linija
6] Sesta linija
7] Sedma linija
8] Osma linija
9] Deveta i predzadnja linija
10] Zaustavicemo se kod deseta linije
```

Pitanje:

Šta je relativna, a šta je absolutna putanja do datoteke?

Funkcija open()

Sintaksa funkcije open je sledeća:

```
file object = open(file_name [, access_mode][, buffering])
```

Prvi argument jeste string koji je ime datoteke (sa putanjom ukoliko je potrebno)

Funkcija open drugim argumentom specificira da li se datoteka koja se otvara koristi za pregled, čitanje, pisanje, dodavanje, ili čitanje i pisanje. Ukoliko se drugi argument ne specificira, podrazumevano se datoteka otvara za čitanje sadržaja.

Treći argument služi za opciju baferovanja, tj. da li se podaci čitaju preko operativnog sistema ili bafera.

https://www.tutorialspoint.com/python/python_files_io.htm

```
r - Čitanje datoteke.
rb - Čitanje datoteke u binarnom formatu.
r+ - Čitanje i pisanje u datoteku.
rb+ - Čitanje i pisanje u datoteku u binarnom fomatu.
w - Samo pisanje u datoteku
wb - Samo pisanje u binarnom formatu.
w+ - Čitanje i pisanje u datoteku. Preimenuje staru datotku ili kreira novu sa tim imenom.
wb+ - Čitanje i pisanje u datoteku u binarnom formatu. Preimenuje staru datotku ili kreira novu sa tim imenom.
a - Otvara datoteku za dodavanje sadržaja.
ab - Otvara datoteku za dodavanje sadržaja u binarnom formatu.
a+ - Otvara datoteku za dodavanje sadržaja i za čitanje.
ab+ - Otvara datoteku za dodavanje sadržaja i za čitanje, u binarnom formatu.
```

OTVARANJE DATOTEKE ZA ČITANJE FUNKCIJOM OPEN()

Datoteka se može otvoriti za čitanje pozivom `ime_promenljive = open("ime_datoteke", "r")`

Otvaranje datoteke

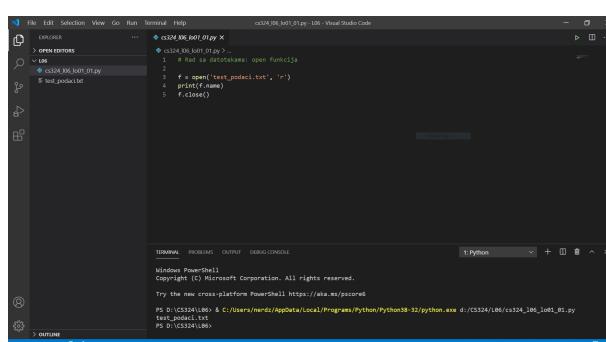
Otvaranje datoteke `test_podaci.txt` se može izvršiti sledećom komandom:

```
f = open('test_podaci.txt', 'r')
```

Zatvaranje datoteke

Nakon završetka rada sa datotekom, potrebno je da eksplisitno zatvorimo datoteku da ne bi ostala u memoriji. Zatvaranje datoteke se može izvršiti pozivom metode `.close()`

```
f = open('test_podaci.txt', 'r')
.
.
.
f.close()
```



Slika 1.2.1 Rad sa datotekama - `open()` funkcije. [Izvor: Autor]

Metode za rad za datotekama

Ime datoteke se može videti pozivom metode **.name()**

```
f = open('test_podaci.txt', 'r')
print(f.name)
f.close()
```

Režim otvaranja datoteke se može videti pozivom metode **.mode()**

Ukoliko nije sigurno kako je datoteka otvorena, uvek se može proveriti da li je otvorena za čitanje, pisanje ili dodavanje pozivom ove metode.

```
f = open('test_podaci.txt', 'r')
print(f.mode)
f.close()
```

Provera zatvaranja datoteke se može videti pozivom atributa **.closed**

```
f = open('test_podaci.txt', 'r')
print(f.mode)
print(f.name)
print(f.closed)

f.close()

print(f.closed)
```

PRIMER RADA SA OTVARANJEM DATOTEKE KORIŠĆENJEM OPEN() FUNKCIJE

Sledi video za otvaranje datoteke pomoću funkcije open()

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.

- ✓ 1.2 Kontekstni menadžer prilikom otvaranja datoteka

OTVARANJE DATOTEKE ZA ČITANJE KONTEKSTNIM MENADŽEROM

Kontekstni menadžer dozvoljava alociranje i oslobođanje resursa u tačno preciziranim trenucima, i automatizuje kontrolu resursa.

Do sada je bilo reči o eksplisitnom otvaranju i zatvaranju datoteka prilikom rada sa datotekama. Ukoliko se datoteka pravilno ne zatvori, doći će do tzv. curenja memorije (en. **memory leakage**), što može dovesti do usporenje rada ili do potpunog prestanka rada računara.

Kontekstni menadžer dozvoljava alociranje i oslobođanje resursa u tačno preciziranim trenucima, i automatizuje kontrolu resursa. Upotreba kontekstnog menadžera u Python jeziku postiže se ključnom rečju **with**, a sintaksa prilikom otvaranja datoteke na ovaj način je sledeća:

```
with open('test_podaci.txt', 'r') as f:  
    ...
```

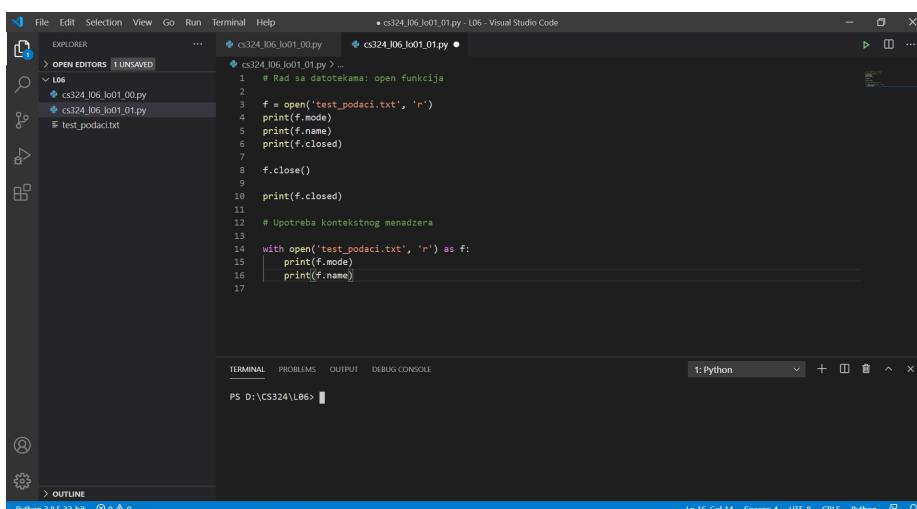
Treba obratiti pažnju da je ime promenljive dato na kraju izjave, a ne na početku kao kod korišćenja `open()` funkcije.

Prednost korišćenja kontekstnog menadžera jeste da omogućavaju rad sa datotekama unutar bloka koda, a kada se izade iz bloka koda, datoteka se automatski zatvara.

Korišćenje kontekstnih menadžera za rad sa datotekama predstavlja dobru praksu pisanja koda u Python jeziku.

Otvaranje prethodne datoteke korišćenjem kontekstnog menadžera ima sledeću sintaksu:

```
# Upotreba kontekstnog menadzera  
  
with open('test_podaci.txt', 'r') as f:  
    print(f.mode)  
    print(f.name)
```



Slika 1.3.1 Opotreba kontekstnog menadžera za otvaranje datoteka. [Izvor: Autor]

ČITANJA DATOTEKE DOK JE OTVORENA, I NAKON ZATVARANJA

Čitanje datoteke je moguće samo kada je datoteka otvorena.

Ime promenljive postoji i nakon zatvaranja datoteke, i biće tipa `IO.TextWrapper`. `io.TextWrapper` je izvedena klasa iz `TextIOBase`, koja je osnovna klasa za rad sa tekstualnim tokovima (en. **streams**) podataka.

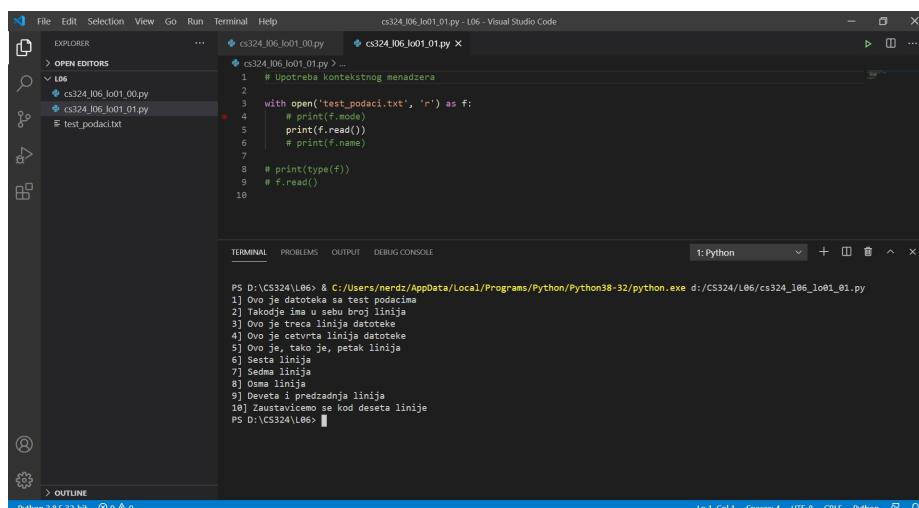
```
>>>print(type(f))

console output:
<class '_io.TextIOWrapper'>
```

Iako ime promenljive i dalje postoji, nije moguće pročitati sadržaj datoteke.

Čitanje datoteke je moguće pozivom metode `.read()`

```
with open('test_podaci.txt', 'r') as f:
    print(f.read())
```



Slika 1.3.2 Čitanje podataka iz datoteke. [Izvor: Autor]

Čitanje datoteke je moguće samo kada je datoteka otvorena, tj. dok je funkcija za čitanje unutar bloka koda kontekstnog menadžera.

Ukoliko se pokuša čitanje van bloka koda, interpreter će vratiti grešku.

```
# van kontekstnog menadzera
>>>f.read()

ValueError: I/O operation on closed file.
```

PRIMER RADA SA OTVARANJEM DATOTEKE KORIŠĆENJEM KONTEKSTNOG MENADŽERA

Sledi video za otvaranje datoteke pomoću kontekstnog menadžera

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

✓ 1.3 Metoda za čitanje podataka

METODE .READ(), .READLINES() I .READLINE()

Najjednostavniji način čitanja podataka iz datoteke jeste upravo `.read()` metoda, ali nije jedina koja se koristi, pogotovo ukoliko se radi o datoteci velikog kapaciteta.

Najjednostavniji način čitanja podataka iz datoteke jeste upravo `.read()` metoda koja je objašnjena u prethodnoj sekciji.

Umesto samog štampanja sadržaja, sadržaj se može smestiti u novu promenljivu:

```
with open('test_podaci.txt', 'r') as f:  
    f_sadrzaj = f.read()  
    print(f_sadrzaj)  
    print(type(f_sadrzaj))
```

Poslednji print vratiće da je `f_sadrzaj` tipa string.

Ukoliko se želi napraviti promenljiva koja bi čitala sadržaj liniju po liniju, onda je bolje da se koristiti `.readlines()` metoda.

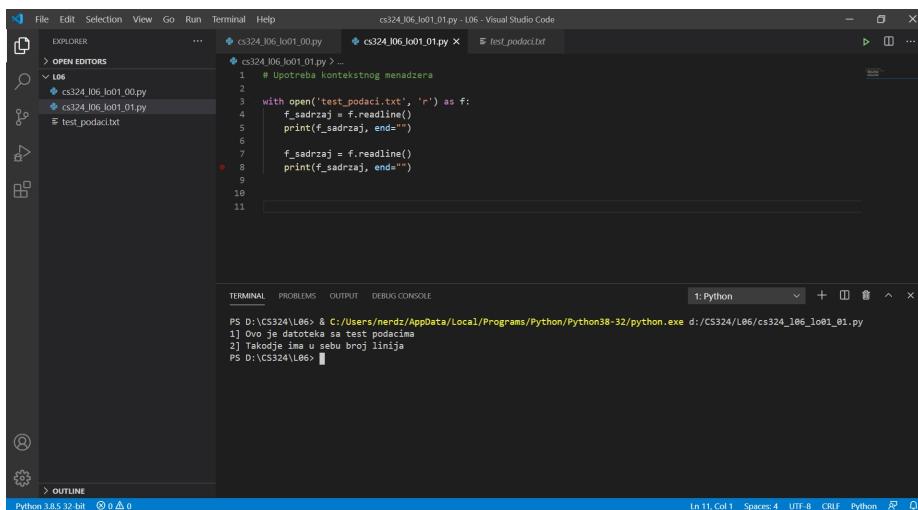
```
with open('test_podaci.txt', 'r') as f:  
    f_sadrzaj = f.readlines()  
    print(f_sadrzaj)  
    print(type(f_sadrzaj))
```

U ovom slučaju, `f_sadrzaj` biće lista, sa onoliko elemenata koliko ima linija u datoteci.

Ukoliko se želi pročitati linija po linija, koristi se `.readline()` metoda, koja pri svakom pozivu čita sledeću liniju unutar datoteke.

```
with open('test_podaci.txt', 'r') as f:  
    f_sadrzaj = f.readline()  
    print(f_sadrzaj, end="")
```

```
f_sadrzaj = f.readline()  
print(f_sadrzaj, end="")
```



Slika 1.4.1 Čitanje sadržaja datoteke liniju po liniju. [Izvor: Autor]

PRIMER .READLINE() NASPRAM .READLINES()

Metode `.readline()` i `.readlines()` se mogu koristiti u zavisnosti da li je potrebno čitati sve ili samo neke linije iz datoteke.

Ukoliko je datoteka velika, nije efikasno koristiti `.readline()` mnoštvo puta dok se svi redovi ne
iščitaju.

Efikasnije je iterativno čitati red po red:

```
with open('test_podaci.txt', 'r') as f:  
    for line in f:  
        print(line, end='')
```

Ukoliko postoji prazna lista koju treba populisati liniju po liniju iz datoteke, može se koristiti sličan pristup:

```
sadrzaj = []

with open('test_podaci.txt', 'r') as f:
    for line in f:
        sadrzaj.append(line)

print(sadrzaj)
```

Pitanje:

Kako se može isto (smeštanje sadržaja svih linija iz datoteke u listu) postići bez iteracije?

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

DODATNI ARGUMENT ZA .READ() METODU

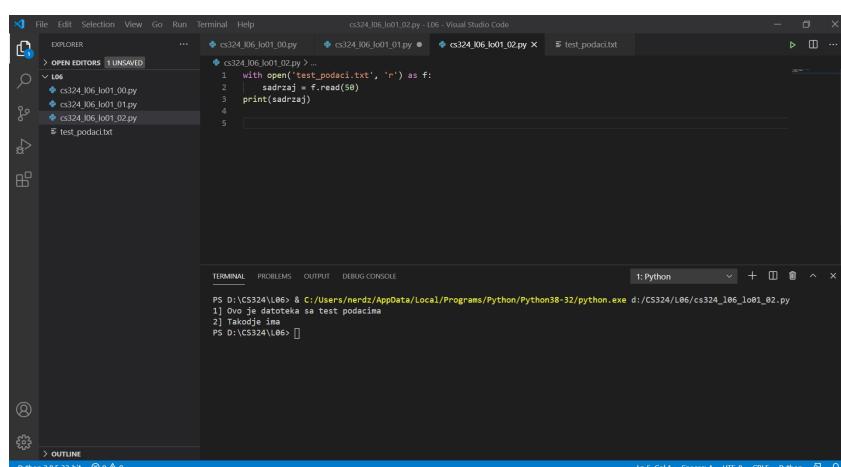
Moguće je dodati argument u .read() metodi, i onda se čita samo toliko karaktera.

Podrazumevano, metoda **.read()** pročitaće ceo sadržaj datoteke.

Dodavanjem argumenta u metodu, moguće je podesiti broj karaktera koji se čita pozivom ove metode:

```
with open('test_podaci.txt', 'r') as f:  
    sadrzaj = f.read(50)  
print(sadrzaj)
```

U primeru iznad broj karaktera jeste 50, a izlaz biće kao na slici ispod.

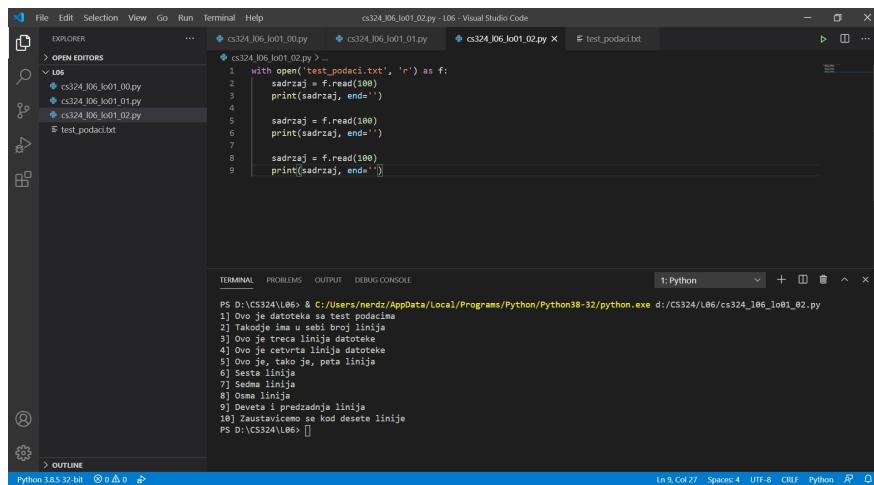


Slika 1.4.2 Izbor broja karaktera koji se čita. [Izvor: Autor]

Ukoliko se više puta pozove **.read()** metoda sa ovim argumentom, svaki sledeći put nastaviće sa čitanjem.

```
with open('test_podaci.txt', 'r') as f:  
    sadrzaj = f.read(100)  
    print(sadrzaj)  
  
    sadrzaj = f.read(100)  
    print(sadrzaj)  
  
    sadrzaj = f.read(100)  
    print(sadrzaj)
```

Kada **.read()** nađe na kraj datoteke, vratiće prazan string.



Slika 1.4.3 Ponavljanje metode .read() sa argumentom, dok se ne pročita cela datoteka. [Izvor: Autor]

ČITANJE IZ VELIKIH TEKSTUALNIH DATOTEKA

Za veću kontrolu nad čitanjem podataka iz datoteke, bolje je koristiti parametar koji kontroliše čitanje svake iteracije.

U opštem slučaju, veličina datoteke ili broj redova datoteke neće biti poznati. Zbog toga je potrebno napraviti petlju koja će učitavati deo po deo datoteke, a koliko će se učitati u jednoj iteraciji kontroliše se parametrom, a ne direktnim ubacivanjem vrednosti.

```
with open('test_podaci.txt', 'r') as f:
    broj_karaktera = 20
    sadrzaj = f.read(broj_karaktera)

    while len(sadrzaj) > 0:
        print(sadrzaj, end=' ')
        sadrzaj = f.read(broj_karaktera)
```

Pitanje:

Šta bi se desilo ukoliko nema poslednjeg reda unutar while petlje?

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

▼ Poglavlje 2

Pisanje u datoteku

OSNOVE FUNKCIJE WRITE()

Za razliku od čitanja iz datoteke, nije neophodno imati unapred napravljenu datoteku. Ukoliko datoteka ne postoji, ova metoda će napraviti.

Metoda .write() predstavlja najjednostavniji način upis podataka u datoteku.

Za razliku od čitanja iz datoteke, nije neophodno imati unapred napravljenu datoteku. Ukoliko datoteka ne postoji, ova metoda će napraviti.

Sintaksa je sledeća:

```
fileObject.write( str )
```

Naravno, najpre treba otvoriti datoteku za čitanje, i to se može uraditi na više načina, kao i kod otvaranja datoteke za čitanje.

Otvaranje preko open() funkcije

Datoteka se može otvoriti za pisanje preko open() funkcije, ali da je drugi argument string 'w'. Nakon toga, može se pisati u datoteku pozivom metode .write()

```
f = open('test_podaci.txt', 'w')
f.write('Ovo su novi test podaci.')
f.close()
```

Primer - Otvaranje datoteke za čitanje i pokušaj pisanja

Ukoliko se pokuša sledeći kod:

```
with open('test_podaci.txt', 'r') as f:
    f.write('Ovo su novi test podaci.')
```

Konzola vraća grešku da se u datoteci ne mogu upisati podaci

```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS primer_za_write.py novi_test_podaci.txt
primer_za_write.py ...
1 f = open('test_podaci.txt', 'r')
2 f.write('Ovo su novi test podaci. \n')
3 f.write('Ovo je drugi red test podataka. \n')
4 f.close()
5
6

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
PS D:\CS324\106> & D:\anaconda3\python.exe d:/CS324/106/primer_za_write.py
Traceback (most recent call last):
File "d:/CS324/106/primer_za_write.py", line 2, in module
f.write('Ovo su novi test podaci.')
io.UnsupportedOperation: not writable
PS D:\CS324\106>

```

Slika 2.1 Pokušaj pisanja u datoteku dok je otvorena za čitanje. [Izvor: Autor]

PISANJE U DATOTEKU KORIŠĆENJEM KONTEKSTNOG MENADŽERA

Efikasnije je otvoriti datoteku korišćenjem kontekstnog menadžera i u slučaju pisanja u datoteku.

Otvaranje preko kontekstnog menadžera

Kao i u ranijim primerima, efikasnije je otvoriti datoteku kroz kontekstni menadžer, jer onda ne treba voditi računa da se datoteka eksplisitno zatvori.

```

# Upotreba kontekstnog menadzera

with open('novi_test_podaci.txt', 'w') as f:
    f.write('Ovo su novi test podaci. \n')
    f.write('Ovo je drugi red test podataka. \n')

```

```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS primer_za_write.py novi_test_podaci.txt
primer_za_write.py ...
1 # Upotreba kontekstnog menadzera
2
3 with open('novi_test_podaci.txt', 'w') as f:
4     f.write('Ovo su novi test podaci. \n')
5     f.write('Ovo je drugi red test podataka. \n')

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
PS D:\CS324\106> & D:\anaconda3\python.exe d:/CS324/106/primer_za_write.py
1 Ovo su novi test podaci.
2 Ovo je drugi red test podataka.
3

PS D:\CS324\106>

```

Slika 2.2 Korišćenje kontekstnog menadžera za pisanje u datoteku. [Izvor: Autor]

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

METODE POZICIONIRANJA: .SEEK() I .TELL()

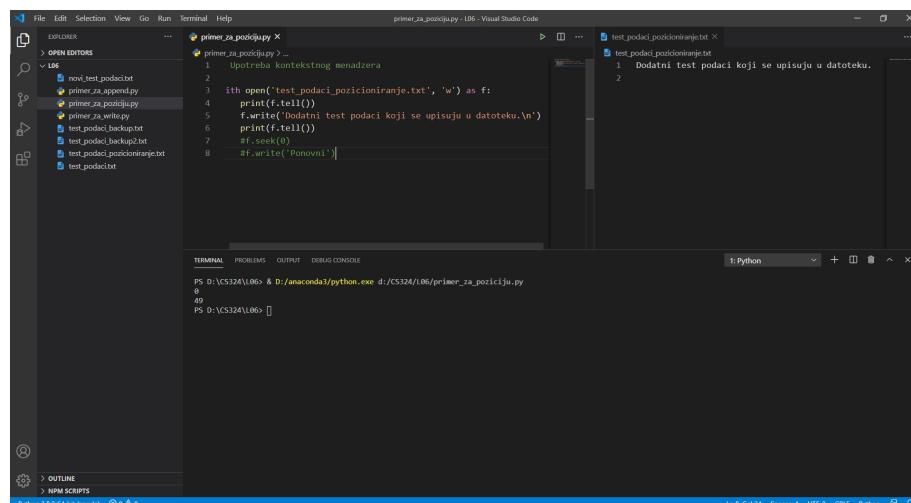
Korišćenjem metoda .tell() i .seek() moguće je kontrolisati poziciju toka podataka za upis u datoteku.

Metoda .tell()

Metoda .tell() vraća poziciju gde je stao tok podataka koji se upisuje.

```
# Kontekstni menadzer

with open('test_podaci_pozicioniranje.txt', 'w') as f:
    print(f.tell())
    f.write('Dodatni test podaci koji se upisuju u datoteku.\n')
```



Slika 2.3 Poziv metode .tell(), koja vraća trenutnu poziciju toka podatka za pisanje. [Izvor: Autor]

Metoda .seek()

Metoda .seek() podešava trenutnu poziciju toka podataka koji se upisuje.

```
# Kontekstni menadzer

with open('test_podaci_pozicioniranje.txt', 'w') as f:
    print(f.tell())
    f.write('Dodatni test podaci koji se upisuju u datoteku.\n')
    print(f.tell())
    f.seek(0)
    f.write('Ponovni')
```

```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS
primer_za_posiziju.py
1 # Upotreba kontekstnog menadzera
2
3 with open('test_podaci_posicioniranje.txt', 'w') as f:
4     print(f.tell())
5     f.write('Dodatajni test podaci koji se upisuju u datoteku.\n')
6     print(f.tell())
7     f.seek(0)
8     f.write('Ponovni')

test_podaci_posicioniranje.txt
1 Ponovni test podaci koji se upisuju u datoteku.

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
PS D:\CS324\U06 & D:/anaconda3/python.exe d:/CS324/U06/primer_za_posiziju.py
0
49
PS D:\CS324\U06 & D:/anaconda3/python.exe d:/CS324/U06/primer_za_posiziju.py
0
49
PS D:\CS324\U06 > []

```

Slika 2.4 Poziv metode .seek() podešava se pozicija toka podataka za pisanje. [Izvor: Autor]

DODAVANJE PODATAKA U DATOTEKU

Upotrebom argumenta 'a' prilikom poziva funkcije open(), sadržaj koji upisujemo se dodaje, a ne presnimava u datoteku.

Može se primetiti da svaki put kada se otvori datoteka za pisanje, datoteka presnimava sadržaj, i stari sadržaj datoteke se gubi. Da se to ne bi dešavalо, prilikom poziva open() funkcije, koristi se karakter 'a' kao drugi argument.

Upotreba ovog argumenta dozvoljava da sav upis ide nakon već postojećeg sadržaja.

```

# Upotreba kontekstnog menadzera

with open('test_podaci.txt', 'a') as f:
    f.write('11] Ipak se dodaje i jedanaesti red. \n')
    f.write('12] A moze i dvanaesti.')

```

Na ovaj način moguće je zadržati sadržaj datoteke, i dodati novi sadržaj.

```

File Edit Selection View Go Run Terminal Help
OPEN EDITORS
primer_za_append.py
1 # Upotreba kontekstnog menadzera
2
3 with open('test_podaci.txt', 'a') as f:
4     f.write('11] Ipak se dodaje i jedanaesti red. \n')
5     f.write('12] A moze i dvanaesti.')

test_podaci.txt
1 Ovo je datoteka sa test podacima
2 Takođe ima u sebi broj linija
3 Ovo je treća linija datoteke
4 Ovo je četvrt linija datoteke
5 Ovo je petnaesti red. Je, petnaesti
6 Šesta linija
7 Sedma linija
8 Osmi linija
9 Deveta i predzadnja linija
10 Zauštavimo se kod deseta linije

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE
PS D:\CS324\U06 & D:/anaconda3/python.exe d:/CS324/U06/primer_za_append.py
PS D:\CS324\U06 > []

```

Slika 2.5 Sadržaj datoteke test_podaci.txt preizvršenja programa. [Izvor: Autor]

The screenshot shows the Visual Studio Code interface with two open files: `primer_na_append.py` and `test_podaci.txt`. The `primer_na_append.py` file contains Python code for appending to a file. The `test_podaci.txt` file contains a text file with 12 lines of text. The terminal window shows the execution of the script and its output, which is the content of the `test_podaci.txt` file.

Slika 2.6 Sadržaj datoteke `test_podaci.txt` nakon izvršenja programa. [Izvor: Autor]

PRIMER: DODAVANJE SADRŽAJA U DATOTEKU SA 'R+' ARGUMENTOM

Primer koji sledi opisuje mogućnost dodavanja sadržaja u datoteku (bez brisanja prethodnog) iako se kao argument za otvaranje koristi 'r+'.

Moguće je (ali je manje efikasno od append režima) koristiti read+write režim za dodavanje sadržaja na kraju datoteke.

Razmotriti sledeći kod:

```
# Upotreba kontekstnog menadzera

with open('test_podaci_primer_w_a.txt', 'r+') as f:
    print(f.tell())
    sadrzaj = f.read()
    broj_karaktera = len(sadrzaj)
    # print(sadrzaj)
    print(broj_karaktera)

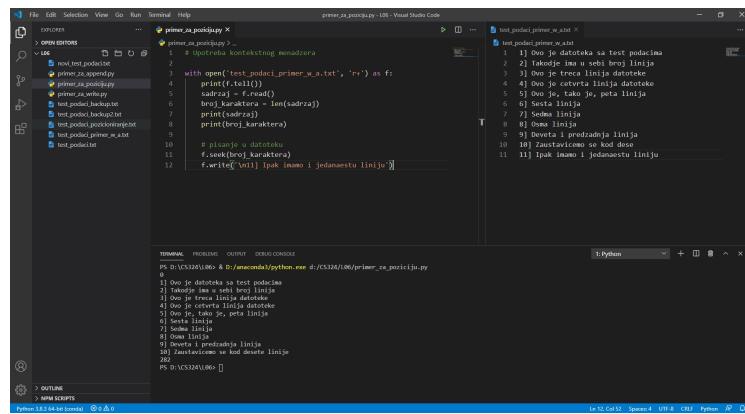
    pisanje u datoteku
    f.seek(broj_karaktera)
    print(f.tell())
    f.write('\n11] Ipak imamo i jedanaestu liniju')
```

Nakon čitanja sadržaja u liniji 5, uzima se **broj_karaktera**.

Onda se poziva **.seek()** metoda koja pozicionira tok podataka na **broj_karaktera**. Konačno, piše se u datoteku od pozicioniranja.

Zanimljivost:

Šta vraća napisani program? Proveriti sadržaj datoteke.



```
primer_sa_posicijom.py
1 # Upotreba kontekstnog menadzera
2
3 with open('test_podaci_i_primer_w_n.txt', 'r+') as f:
4     print(f.tell())
5     sadrzaj = f.read()
6     print(sadrzaj)
7     print(len(sadrzaj))
8     print(broj_karaktera)
9
10    # pisanje u datoteku
11    f.seek(broj_karaktera)
12    f.write('\n') i pak inamo i jedanaestu liniju!
```

```
test_posledi.primer_w.txt
1 1] Ovo je datoteka sa test podacima
2 2] Takođe ima i neku drugu
3 3] liniju u datoteci
4 4] Ovo je četveta linija datoteke
5 5] Ovo je, takođe, peta linija
6 6] u datoteci
7 7] Sedma linija
8 8] Osma linija
9 9] Deveta i predzadnja linija
10 10] Zastavljenu se kod doček
11 11] spisak imena i jedanaestu liniju!
```

TERMINAL: PROBLEMS: OUTPUT: DOKUMENCI

```
PS D:\CS320\106 & D:\me\code\python.exe d:\CS320\106\primer_sa_posicijom.py
```

```
1] Ovo je datoteka sa test podacima
2] Takođe ima i neku drugu
3] liniju u datoteci
4] Ovo je četveta linija datoteke
5] Ovo je, takođe, peta linija
6] u datoteci
7] Sedma linija
8] Osma linija
9] Deveta i predzadnja linija
10] Zastavljenu se kod doček
11] spisak imena i jedanaestu liniju!
```

Python 3.8.5 64-bit (Python) D:\CS320\106

Slika 2.7 Primer dodavanja sadržaja u datoteku sa `r+` argumentom. [Izvor: Autor]

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

▼ Poglavlje 3

Čuvanje podataka

ČUVANJE PODATAKA U CSV DATOTEKU

CSV datoteka jeste tabelarno uređena datoteka koja se često koristi za razmenu podataka između aplikacija.

Ukoliko je potrebno sačuvati veliku količinu podataka, obično su ti podaci nekako uređeni.

Iako sadrži tekstualne podatke, CSV datoteka (en. comma separated values file) jeste tabelarno uređena datoteka koja se koristi za razmenu podataka između aplikacija.

Struktura CSV datoteke

CSV datoteka ima relativno jednostavnu strukturu. Predstavlja listu podataka koje deli zapeta. Prednost CSV datoteke jeste u tome što je čitljiva i od strane korisnika (čoveka) ali i da može da se otvori u bilo kom editoru teksta.

Primer: Telefonski imenik

Name,Email,Phone Number,Address

Bob Smith,bob@example.com,123-456-7890,123 Fake Street

Mike Jones,mike@example.com,098-765-4321,321 Fake Avenue

Pojedine CSV datoteke ne moraju imati ni liniju za zaglavlje na vrhu.

Rad sa CSV datotekama u Python jeziku

Python ima ugrađene module za rad sa CSV datotekama. Potrebno je pre korišćenja ukucati:

```
import csv
```

Nakon toga, mogu se koristiti metode za pisanje i čitanje CSV datoteka.

U nastavku je tekst u CSV formatu koji se može koristiti za primere čitanja.

```
CS220,CS225,CS324,IT331,IT335,IT376,SE321,SE325
```

```
2,3,4,2,3,3,4,4
```

```
2020,2019,2018,2017,2020,2018,2017
```

```
Jun A,Jun A,Januar B,Jun B,Septembar,Januar A,Oktobar II,Novembar
```

9,8,10,10,8,7,6,9

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

PISANJE U CSV DATOTEKU

Otvaranje CSV datoteke isto je kao i otvaranje bilo koje datoteke, a za pisanje kreira se novi objekat iz klase csv.writer

Otvaranje CSV datoteke isto je kao i otvaranje bilo koje datoteke. Moguće je otvoriti direktno (ali onda treba voditi računa da se zatvori), ili preko kontekstnog menadžera.

Primer: Broj radnih sata u nedelji

Napisati CSV datoteku koja će sadržati u jednom redu radne dane (ponedeljak - petak), a u drugom broj radnih sata za svaki dan.

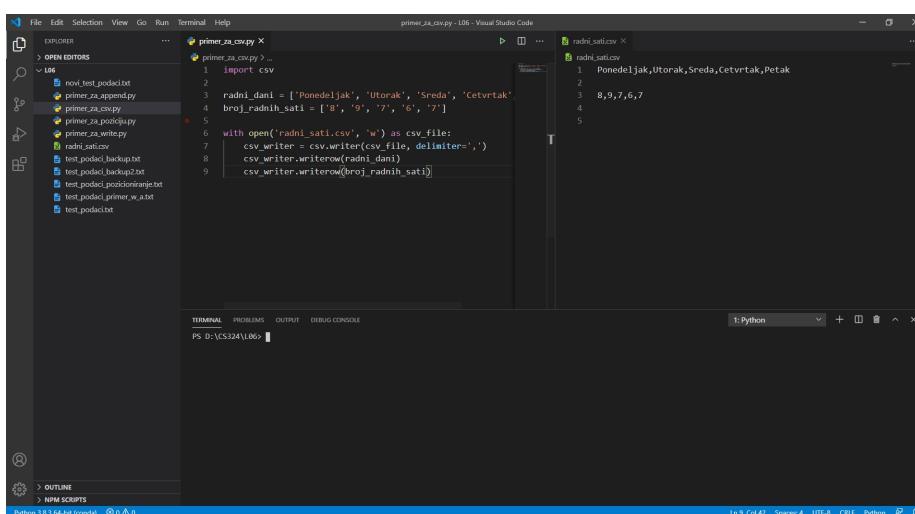
```
import csv

radni_dani = ['Ponedeljak', 'Utorak', 'Sreda', 'Cetvrtak', 'Petak']
broj_radnih_sati = ['8', '9', '7', '6', '7']

with open('radni_sati.csv', 'w') as csv_file:
    csv_writer = csv.writer(csv_file, delimiter=',')
    csv_writer.writerow(radni_dani)
    csv_writer.writerow(broj_radnih_sati)
```

Najpre treba uvesti csv biblioteku zbog dodatnih funkcija.

Kreiramo novi objekat iz klase **csv.writer** gde unosimo dva parametra, prvi je ime datoteke, a drugi jeste *separator* (en. **delimiter**), koji je u CSV datotekama najčešće *zapeta*. Zatim, koristimo metodu *.writerow()* za svaki red koji želimo da unesemo. Parametar ove metode jeste lista sa prethodno unetim elementima.



Slika 3.1.1 Sadržaj radni_sati.csv datoteke nakon njene kreacije. Izvor: Autor.

ČITANJE IZ CSV DATOTEKE

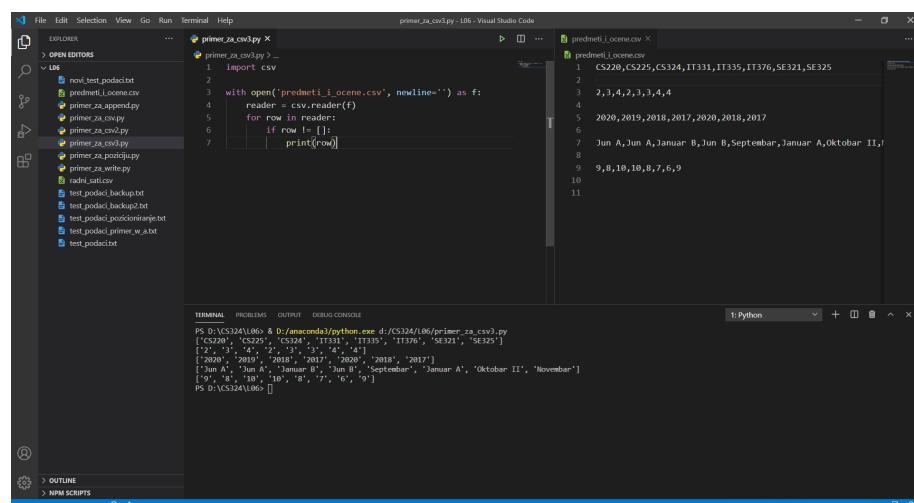
Čitanje iz CSV datoteke je jednostavnije od pisanja u CSV datoteci, ali je svakako potrebno napraviti objekat klase csv.reader()

Čitanje iz CSV datoteke je takođe jednostavno.

Potrebno je kreirati objekat klase **csv.reader()** koji može pročitati sve redove CSV datoteke.

```
import csv

with open('predmeti_i_ocene.csv', newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        if row != []:
            print(row)
```



Slika 3.1.2 Čitanje iz CSV datoteke. [Izvor: Autor]

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

ČUVANJE PODATAKA KAO JSON

JSON podataka predstavlja otvoreni standard za smeštanje i razmenu podataka, koji koristi razumljivi tekst (en. human-readable text) za smeštanje i razmenu objekata podataka.

JSON podataka (en. **JavaScript Object Notation**) predstavlja otvoreni standard za smeštanje i razmenu podataka, koji koristi razumljivi tekst (en. **human-readable text**) za smeštanje i razmenu objekata podataka.

Sastoji se od parova atribut-vrednost i tipova podataka koji imaju više elemenata.

Ovaj format se jako često koristi u raznim aplikacijama, i služi kao zamena za [XML](#) u [AJAX](#) sistemima.

JSON tip podataka je nezavistan od programske jezike iako je izведен iz JavaScript jezika. Zvanični tip podataka za JSON jeste application/json

<https://www.json.org/json-en.html>

Rad sa JSON datotekama u Python jeziku

Python ima ugrađene module za rad sa JSON datotekama. Potrebno je pre korišćenja ukucati:

```
import json
```

Nakon toga, mogu se koristiti metode za pisanje i čitanje JSON datoteka.

U nastavku je tekst u JSON formatu koji se može koristiti za primere čitanja.

```
{"predmeti": [{"ime": "Skripting jezici", "sifra": "CS324", "godina studija": "IV", "smer": "RI"}, {"ime": "Operativni sistemi", "sifra": "CS220", "godina studija": "III", "smer": "IT"}, {"ime": "Arhitektura racunara", "sifra": "CS220", "godina studija": "II", "smer": "IT"}, {"ime": "Racunarske mreze i komunikacije", "sifra": "IT331", "godina studija": "II", "smer": "IT"}, {"ime": "Administracija racunarskih sistema i mreza", "sifra": "IT335", "godina studija": "III", "smer": "IT"}]}
```

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

PISANJE U JSON DATOTEKU

Prilikom pisanja podataka u JSON datoteku potrebno je kreirati objekat klase `json.dump()`, a kao parametre staviti promenljivu sa podacima koji se štampaju.

Otvaranje JSON datoteke isto je kao i otvaranje bilo koje datoteke. Moguće je otvoriti direktno (ali onda treba voditi računa da se zatvori), ili preko kontekstnog menadžera.

Primer: Imenik koji sadrži ime aplikacije, web adresu i državu odakle potiče

Napraviti prazan imenik podaci. Zatim dodati ključ "[apps](#)" i vrednost prazan string. Dodati kao elemente string nove imenike, sa parovima ključ: vrednost na sledeći način:

`'ime': 'ime aplikacije'`

`'website': 'web sajt aplikacije'`

`'drzava': 'drzava odakle potice aplikacija'`

Zatim, sačuvati imenik kao podaci.json

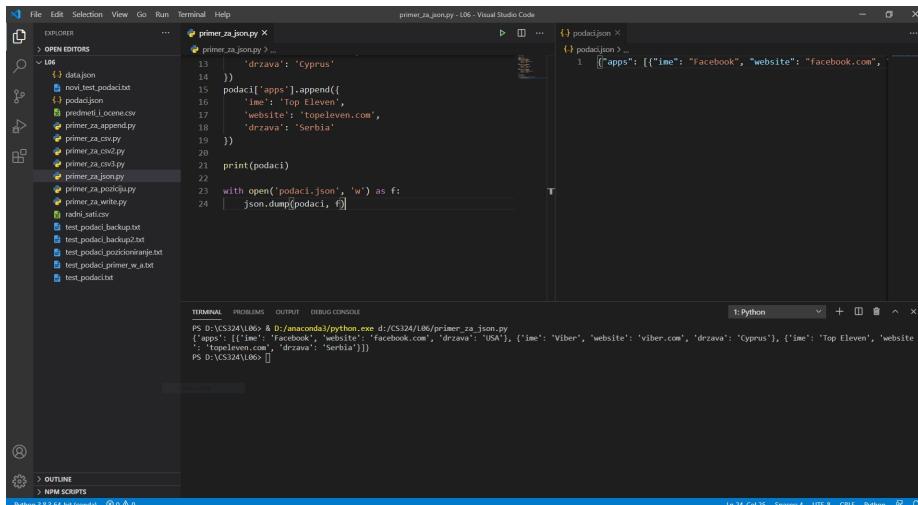
Prilikom otvaranja podataka potrebno je kreirati objekat klase `json.dump()` a kao parametre staviti koji se podaci štampaju i u koju otvorenu datoteku.

```
import json

podaci = []
podaci['apps'] = []
podaci['apps'].append({
    'ime': 'Facebook',
    'website': 'facebook.com',
    'drzava': 'USA'
})
podaci['apps'].append({
    'ime': 'Viber',
    'website': 'viber.com',
    'drzava': 'Cyprus'
})
podaci['apps'].append({
    'ime': 'Top Eleven',
    'website': 'topeleven.com',
    'drzava': 'Serbia'
})

print(podaci)

with open('podaci.json', 'w') as f:
    json.dump(podaci, f)
```



Slika 3.1.3 Primer pisanja podataka u JSON format. [Izvor: Autor]

ČITANJE IZ JSON DATOTEKE

Čitanje iz JSON datoteke je vrlo slično čitanju podataka iz bilo koje datoteke, i direktno će populisati imenik iz datoteke.

Čitanje iz JSON datoteke je vrlo slično čitanju podataka iz bilo koje datoteke.

Ponovo je potrebno uvesti `json` biblioteku da bi se JSON string parsirao iz objekta datoteke.

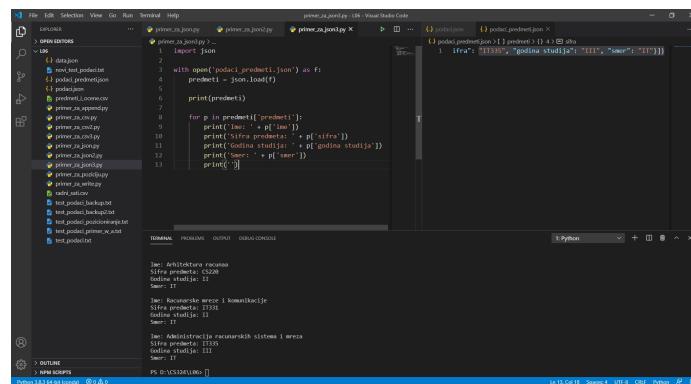
Za čitanje podataka treba kreirati objekat klase **`json.load()`** koji kao parametar uzima objekat JSON datoteke. Direktno će populisati imenik iz datoteke.

```
import json

with open('podaci_predmeti.json') as f:
    predmeti = json.load(f)

    print(predmeti)

    for p in predmeti['predmeti']:
        print('Ime: ' + p['ime'])
        print('Sifra predmeta: ' + p['sifra'])
        print('Godina studija: ' + p['godina studija'])
        print('Smer: ' + p['smer'])
        print('')
```



Slika 3.1.4 Čitanje iz CSV datoteke. [Izvor: Autor]

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

3.1 Rad sa binarnim datotekama

PYTHON I BINARNE DATOTEKE

Rad sa binarnim datotekama je sličan kao sa tekstualnim. Treba samo obratiti pažnju na drugi argument prilikom otvaranja datoteke.

Rad sa binarnim datotekama je sličan kao sa tekstualnim.

Prilikom otvaranja datoteke, potrebno je u drugom argumentu dodati karakter 'b', što označava rad sa binarnim datotekama.

```
# otvaranje za citanje binarnih podataka
f_bin_read = open('Lenna.png', 'rb')

# otvaranje za pisanje binarnih podataka
f_bin_write = open('Lenna.png', 'wb')

#otvaranje kroz kontekstni menadzer za citanje
with open('Lenna.png', 'rb') as rfb:
    ...
    #otvaranje kroz kontekstni menadzer za pisanje
    with open('Lenna.png', 'wb') as wfb:
        ...
```

Primer: Kopiranje binarne datoteke

Otvoriti sliku i kopirati je u drugu sliku.

Link do slike:

<https://en.wikipedia.org/wiki/Lenna>

```
# rad sa binarnim datotekama

with open('Lenna.png', 'rb') as rf:
    with open('Lenna_copy.png', 'wb') as wf:
        vel_bloka = 4096
        rf_blok = rf.read(vel_bloka)
        while len(rf_blok) > 0:
            wf.write(rf_blok)
            rf_blok = rf.read(vel_bloka)
```

PRIMER RADA SA BINARNIM DATOTEKAMA

Sledi video koji detaljno opisuje prethodni primer.

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

▼ Poglavlje 4

Izuzeci i obrada izuzetaka

ŠTA SU IZUZECI?

Izuzetak predstavlja događaj koji se desi tokom izvršenja programa, a taj događaj poremeti normalni tok instrukcija.

Izuzetak (en. `exception`) predstavlja događaj koji se desi tokom izvršenja programa, a taj događaj poremeti normalni tok instrukcija.

Postoje mnoge vrste grešaka koje mogu prouzrokovati izuzetke: od problema hardverske prirode (prestanak rada spoljašnje memorije ili diska), do jednostavnih grešaka u programiranju, kao što je pokušaj pristupa elementu liste koji ne postoji.

Kada se desi izuzetak, Python napravi objekat klase `Exception` i predala ga interpreteru. Ovaj objekat sadrži informaciju o grešci koja je napravljena.

Kada se izuzetak *"bací"* (en. `throwing an exception`), interpreter pokušava naći način za obradu izuzetka. Ukoliko postoji deo koda koji je odgovoran za obradu određenog tipa greške, onda se taj deo koda poziva. U tom slučaju se *"hvata"* izuzetak (en. `catching an exception`)

Obrada izuzetaka u Python jeziku

U Python jeziku, obrada izuzetaka se sastoji od tri ključne reči:

- **try**
- **except**
- **finally**

Kombinacijom ovih ključnih reči moguće je detaljno obraditi greške koje se javljaju u Python programima, i omogućiti normalan tok rada programa.

U nastavku su dati konkretni primeri za upotrebu ovih ključnih reči prilikom obrade izuzetaka.

OBRADA IZUZETAKA KROZ KLJUČNU REČ TRY

U try bloku koda upisuje se deo koda koji može izazvati grešku, a u except bloku vršimo obradu izuzetka.

Primer: Pronalaženje greške u programu

Napisati program koji otvara nepostojeću datoteku za čitanje. Obraditi grešku **FileNotFoundException**.

U glavnom programu najpre treba napisati kod za otvaranje datoteke (u režimu za čitanje) koja ne postoji.

```
f = open('imedatoteke.txt', 'r')
```

Konzola vraća da datoteka ne postoji.

```
Traceback (most recent call last):
  File "d:/CS324/L06/Exceptions/test_exception.py", line 1, in <module>
    f = open('imedatoteke.txt', 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'imedatoteke.txt'
```

Konzola vraća koja je greška u pitanju i u kojoj je liniji koda, što je prvi korak ka obradi izuzetaka. Ukoliko se mogu predvideti delovi koda koji mogu vratiti grešku ("baciti" izuzetak), onda je moguće koristiti blokove za obradu izuzetaka ("uhvatiti" izuzetak).

```
try:
    f = open('imedatoteke.txt', 'r')
except Exception:
    print('Datoteka ne postoji!')
```

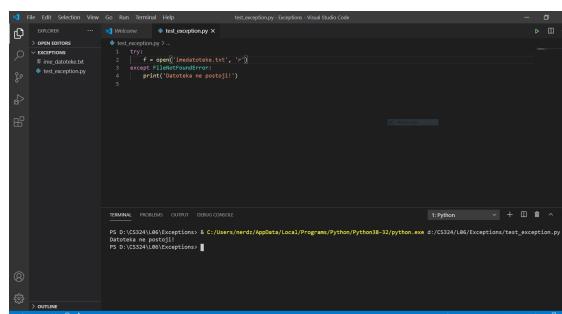
U **try** bloku koda upisuje se deo koda koji može izazvati grešku, a u **except** bloku vršimo obradu izuzetka.

Napomena:

Nakon ključne reči **except** piše se koji je izuzetak u pitanju. Ukoliko se želi opšti izuzetak, treba pisati **Exception**.

Ukoliko želimo specifičan izuzetak, posle **except** može se pisati *konkretni objekat klase exception*.

```
try:
    f = open('imedatoteke.txt', 'r')
except FileNotFoundError:
    print('Datoteka ne postoji!')
```



Slika 4.1 Hvatanje FileNotFoundError izuzetka. [Izvor: Autor]

TIPOVI GREŠAKA I REDOSLED HVATANJA IZUZETAKA

Najbolja praksa kod hvatanja izuzetaka jeste najpre napraviti kod za obradu sa konkretnim izuzecima, a nakon toga staviti opšti izuzetak.

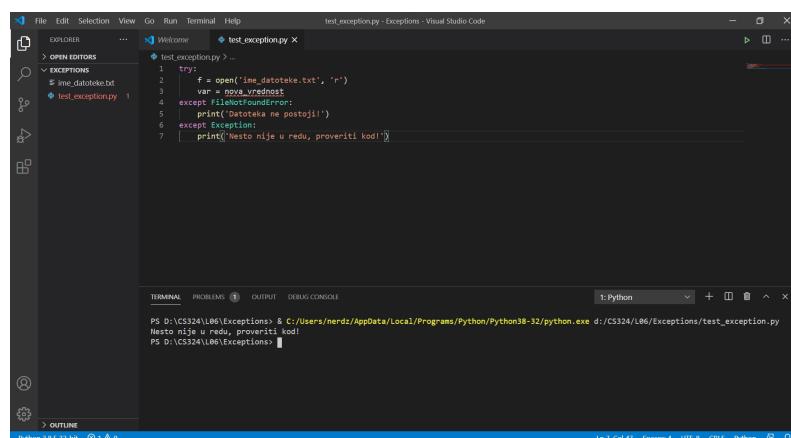
Budući da se greške mogu javiti zbog različitih uzroka, najbolja praksa jeste najpre napraviti kod za obradu sa konkretnim izuzecima, a nakon toga staviti opšti izuzetak.

```
# Cesti tipovi gresaka
EOFError - kada input() funkcija dodje do end-of-file uslova
FileNotFoundException - greska kada ne postoji datoteka kojoj se pristupa
FloatingPointError - greska u operaciji sa razlomljenim brojevima
ImportError - greska kod uvoza modula
IndexError - greska pri indeksiranju elementa iterabilnog objekta
KeyError - greska ako se ne pronadje kljuc u recniku ili skupu
KeyboardInterrupt - greska kada se pritisne Ctrl+C ili Delete na tastaturi
NameError - greska kada ime promenljive ne postoji u lokalnom ili globalnom oopsegu
SyntaxError - sintaksna greska
IndentationError - greska koja se javlja ukoliko nije kod dobro 'tabovan'
SystemError - interna greska interpretera
RuntimeError - ostale greske
```

Primer: Hvatanje opšteg izuzetaka

Napisati program koji ima dve except izjave, jednu za FileNotFoundException, a drugu za opšti izuzetak. U programu otvoriti datoteku (ispravno ime) ali unutar try dela dodeliti vrednost promenljivi var nekoj promenljivoj koja prethodno nije definisana.

```
try:
    f = open('ime_datoteke.txt', 'r')
    var = nova_vrednost
except FileNotFoundException:
    print('Datoteka ne postoji!')
except Exception:
    print('Nesto nije u redu, proveriti kod!')
```



Slika 4.2 Hvatanje opšteg izuzetka. [Izvor: Autor]

ISPISIVANJE PORAZUMEVANIH PORUKA ZA GREŠKE

Moguće je videti podrazumevane poruke za greške ukoliko se naredba except proširi navođenjem imena objekta koji želimo da uhvatimo.

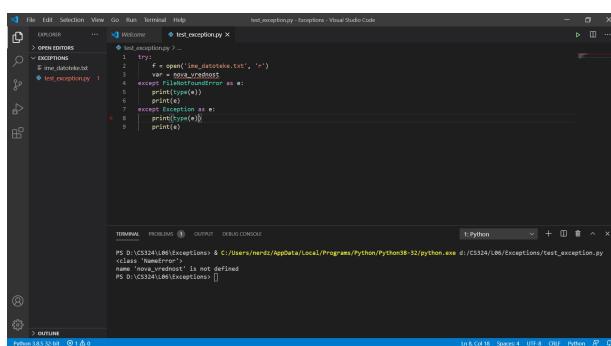
Moguće je videti podrazumevane poruke za greške ukoliko se naredba except proširi navođenjem imena objekta koji želimo da uhvatimo.

```
try:  
    f = open('ime_datoteke.txt', 'r')  
    var = nova_vrednost  
except FileNotFoundError as e:  
    print(type(e))  
    print(e)
```

U četvrtom redu koda dodajemo ključnu reč as, nakon čega pravimo objekat konkretnе klase izuzetka. Za deo koda u except bloku, konzola izbacuje sledeće:

```
<class 'FileNotFoundException'>  
[Errno 2] No such file or directory: 'imedatoteke.txt'
```

Na ovaj način može se videti koja je konkretna greška u pitanju, ali može se videti i šta je konkretno izazvalo grešku.



Slika 4.3 Hvatanje greške za podrazumevanom porukom. [Izvor: Autor]

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

IZUZECI I ELSE I FINALLY BLOKOVI KODA

Kod koji će se izvršiti kada nema izuzetaka biće u bloku koda nakon reči else, dok će se kod nakon reči finally izvršiti bilo da postoji greška ili ne.

Ukoliko deo koda koji se nalazi u try bloku ne baci ni jedan izuzetak, onda je moguće odvojiti kod koji će se izvršiti (nakon provere izuzetka) komandom `else`.

```
try:  
    f = open('ime_datoteke.txt', 'r')  
except FileNotFoundError as e:  
    print(type(e))  
    print(e)  
except NameError as e:  
    print(type(e))  
    print(e)  
except Exception as e:  
    print(e)  
else:  
    sadrzaj = f.read()  
    print(sadrzaj)  
    f.close()
```

Blok `finally` izvršiće kod u svom bloku bilo da kod baci izuzetak ili ne.

```
try:  
    f = open('imedatoteke.txt', 'r')  
except FileNotFoundError as e:  
    print(type(e))  
    print(e)  
except NameError as e:  
    print(type(e))  
    print(e)  
except Exception as e:  
    print(e)  
else:  
    sadrzaj = f.read()  
    print(sadrzaj)  
finally:  
    print('Finally blok se izvršava...')  
    f.close()
```

RUČNO DODAVANJE IZUZETAKA

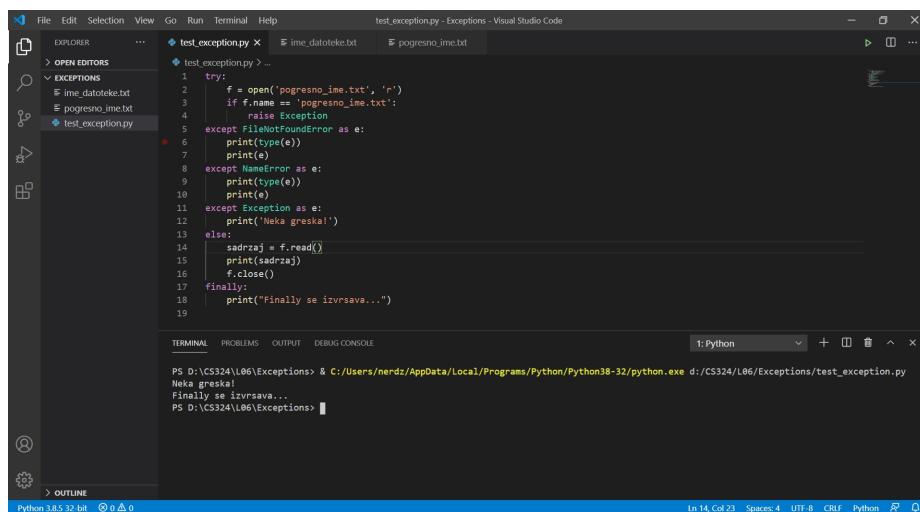
Moguće je ručno dodati izuzetak korišćenjem komande `raise`

Komanda `raise` služi za ručno dodavanja ("dizanje") izuzetaka.

Na sledećem primeru javlja se izuzetak ukoliko korisnik unese ime datoteke koje ima (u ovom primeru) ne treba da ima.

```
try:  
    f = open('pogresno_ime.txt', 'r')  
    if f.name == 'pogresno_ime.txt':  
        raise Exception  
except FileNotFoundError as e:
```

```
print(type(e))
print(e)
except NameError as e:
    print(type(e))
    print(e)
except Exception as e:
    print('Neka greska!')
else:
    sadrzaj = f.read()
    print(sadrzaj)
    f.close()
finally:
    print("Finally se izvrsava...")
```



Slika 4.4 Ručno dodavanje izuzetaka. [Izvor: Autor]

▼ Poglavlje 5

Debagovanje u Python 3.x jeziku

POJAM DEBAGOVANJA

Debagovanje jestе proces koji podrazumeva identifikovanja i ispravljanja grešaka u kodu programa.

Debagovanje (en. **debugging**) jestе proces koji podrazumeva identifikovanja i ispravljanja grešaka u kodu programa.

Postoje više metode debagovanja, i mogu uključivati interaktivno debagovanje, jedinično testiranje, integraciono testiranje, analiza log datoteka, kao i druge metode.

Debager (en. **debugger**) predstavlja softver ili softverski paket koji pruža pomoć u debagovanju.

Proces debagovanja

Proces debagovanja se sastoji iz pet koraka:

1. Identifikacija problema
2. Opis problema
3. Hvatanje problema
4. Analiza uhvaćenog problema
5. Popravka problema

Nakon identifikacije problema, potrebno je opisati problem da bi se našao tačan razlog za pojavu problema. Nakon toga, treba reprodukovati pojavu problema, što obuhvata podešavanja svih promenljivih koje bi izazvale problem. Na osnovu reprodukcije, pokušati naći izvor problema, i na kraju rešiti problem, uz proveru da se taj problem ponovo ne javlja.

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

DEBAGOVANJE U PYTHON JEZIKU KROZ MODUL PDB

Python jezik ima ugrađen modul za debagovanje pdb (Python DeBugger), ali nije pregledan niti napredan. Bolje je koristiti debager unutar konkretnog IDE-a.

Python jezik ima ugrađen modul za debagovanje pdb (Python DeBugger).

Dovoljno je uvesti ovaj modul, i na mestu gde se želi postaviti breakpoint, pozvati `pdb.set_trace()` funkciju. Kada program dođe do ove linije koda, u terminalu pokrenuće se Python Debugger.

```
import pdb

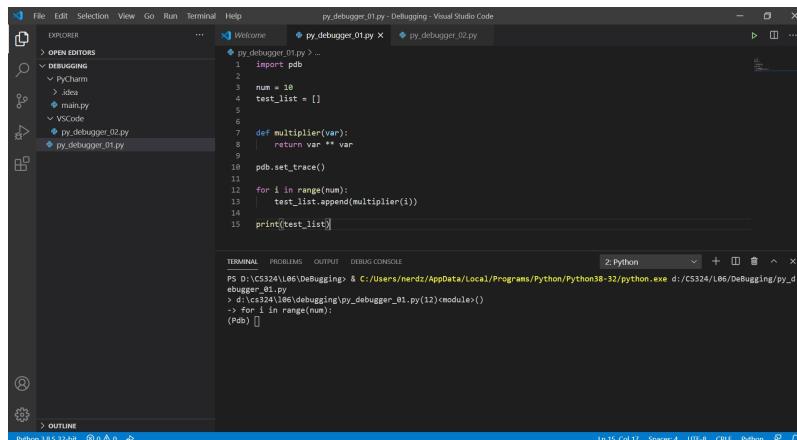
num = 10
test_list = []

def multiplier(var):
    return var ** var

pdb.set_trace()

for i in range(num):
    test_list.append(multiplier(i))

print(test_list)
```



Slika 5.1 Ugrađeni Python debugger pdb. Izvor: Autor.

Prilikom pokretanja pdb-a, na konzoli pojaviće se sledeći prompt:

```
(Pdb)
```

Kucanjem komandi može se izvršiti debagovanje. Komande su sledeće:

```
n - izvrsiti sledecu liniju koda
c - zavrsiti izvrsenje programa
l - listati prethodne i naredne tri linije koda u programu
s - uci u poziv funkcije
b - pokazati spisak svih breakpoint-ova
b[int] - postaviti breakpoint na liniji koda [int]
b[func] - postaviti breakpoint na ime funckije [func]
cl - izbrisati sve breakpoint-ove
cl[int] - izbrisati breakpoint na liniji koga [int]
p - stampa
```

Iako je modul pdb ugrađen u Python, nije najpregledniji, niti najnapredniji.

U nastavku obradiće se debuggeri u Visual Studio Code i PyCharm IDE.

DEBAGOVANJE U VISUAL STUDIO CODE IDE

Postavljanjem breakpoint-ova pored linije koda (u tzv. gutter-u) moguće je pokrenuti debager u Visual Studio Code-u.

Postavljanjem breakpoint-ova pored linije koda (u tzv. gutter-u) moguće je pokrenuti debager u Visual Studio Code-u.

Program staje, zapravo pauzira sa izvršenjem prilikom nailaska na breakpoint i onda je moguće videti u levom panelu promenljive u tom trenutku izvršenja, moguće je postaviti sopstveni "watch" ukoliko je potrebno nešto specifično pratiti što program ne javlja automatski.

Program nastavlja sa radom klikom na *step over*, ulazi u funkciju koja se poziva sa *step into*, a izlazi sa *step out*.

Primer:

Debugovati program koji ima klasu korisnik, sa atributima ime i email.

```
# Test VisualStudioCode

class Korisnik():
    # konstruktor
    def __init__(self, ime, email):
        self.ime = ime
        self.email = email

    # metoda koja vraca ime korisnika
    def vrati_ime(self):
        return self.ime

    # metoda koja vraca email korisnika
    def vrati_email(self):
        return self.email

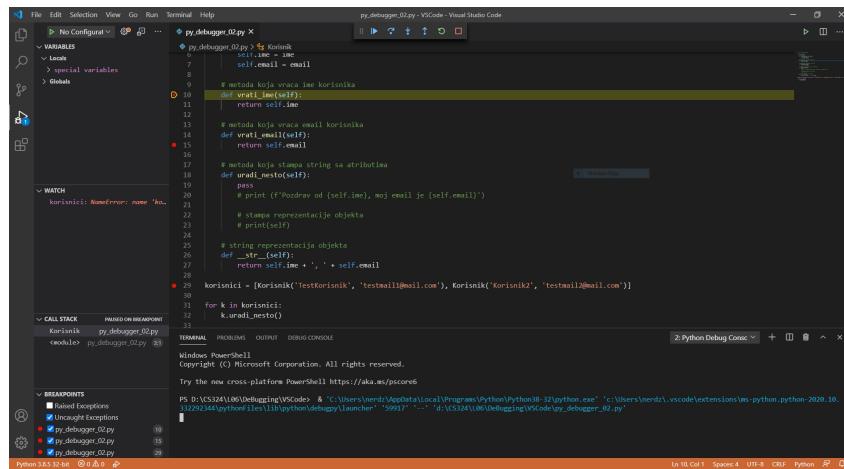
    # metoda koja stampa string sa atributima
    def uradi_nesto(self):
        pass
        # print (f'Pozdrav od {self.ime}, moj email je {self.email}')

        # stampa reprezentacije objekta
        # print(self)

    # string reprezentacija objekta
    def __str__(self):
        return self.ime + ', ' + self.email
```

```
korisnici = [Korisnik('TestKorisnik', 'testmail1@mail.com'), Korisnik('Korisnik2', 'testmail2@mail.com')]

for k in korisnici:
    k.uradi_nesto()
```



Slika 5.2 Debugovanje u Visual Studio Code-u. [Izvor: Autor]

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

DEBAGOVANJE U PYCHARM IDE

PyCharm debager automatski postavlja vrednosti promenljiva kao komentar nakon izjave u samom kodu!

Kao i kod Visual Studio Code IDE, u PyCharm-u je moguće postaviti breakpoint pored same linije koda.

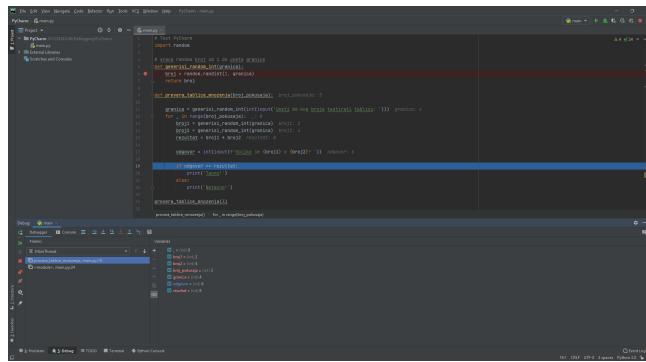
PyCharm debager otvara tab pored konzole za debagovanje, i prilikom nailaska na breakpoint moguće je videti sve promenljive u lokalnom opsegu, ali i pratiti ličnu unetu izjave.

Ostim toga, PyCharm debager automatski postavlja vrednosti promenljiva kao komentar nakon izjave u samom kodu! Na taj način moguće je nesmetano pratiti tok programa i uvek imati u vidu koje su vrednosti promenljiva, a da se pritom gleda u izvorni kod.

```
# Test PyCharm
import random

# vraca random broj od 1 do unete granice
def generisi_random_int(granica):
    broj = random.randint(1, granica)
    return broj
```

```
def provera_tablice_mnozenja(broj_pokusaja):  
  
    granica = generisi_random_int(int(input('Uneti do kog broja testirati tablicu: ')))  
    for _ in range(broj_pokusaja):  
        broj1 = generisi_random_int(granica)  
        broj2 = generisi_random_int(granica)  
        rezultat = broj1 * broj2  
  
        odgovor = int(input(f'Koliko je {broj1} x {broj2}? '))  
  
        if odgovor == rezultat:  
            print('Tacno!')  
        else:  
            print('Netacno!')  
  
provera_tablice_mnozenja(3)
```



Slika 5.3 Debugovanje u PyCharm-u. Izvor: [Izvor: Autor]

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

▼ Poglavlje 6

Pokazna vežba #6

POKAZNA VEŽBA #1

Prva zadatak pokazuje rad sa datotekama. Procenjeno vreme 20 minuta

U pokaznoj vežbi proći će se kroz zadatak koji obuhvata sledećim tematskim jedinicama ove lekcije:

- Rad sa datotekama
- Obrada Izuzetaka
- Debagovanje

Procenjeno vreme izrade pokaznih vežbi je 45 minuta.

ZADATAK #1

Prvi zadatak pokazuje rad sa datotekama. Procenjeno vreme 20 minuta

Zadatak #1 (20 minuta)

Napisati funkciju ascii_sifre(broj_sifri, duzina_sifre) koja će generisati nasumičnu šifru od proizvoljnog broja ASCII karaktera (velika i mala slova). Funkcija vraća duple stringova čiji su elementi šifre.

U glavnom programu napraviti CSV datoteku **sifre.csv** koja će sadržati dva reda. Prvi red sadrži stringove oblika "Sifra #", gde je # indeks liste, a drugi red sadrži šifre.

koristiti module random, string i csv.

Debug-ovati program.

```
import random
import string
import csv

def ascii_sifre(broj_sifri,duzina_sifre):
    lista_sifre = []
    letters = string.ascii_letters
    for i in range(broj_sifri):
```

```
sifra = ''  
for j in range(duzina_sifre):  
    sifra += random.choice(letters)  
lista_sifre.append(sifra)  
return tuple(lista_sifre)  
  
lista = ascii_sifre(8, 8)  
  
redni_broj_sifre = []  
sama_sifra = []  
  
for i in range(0, len(lista)):  
    redni_broj_sifre.append('Sifra #' + str(i + 1))  
    sama_sifra.append(lista[i])  
  
with open('sifre.csv', 'w') as csv_file:  
    csv_writer = csv.writer(csv_file, delimiter=',')  
    csv_writer.writerow(redni_broj_sifre)  
    csv_writer.writerow(sama_sifra)
```

ZADATAK #2

U drugom zadatku treba ručno podići izuzetke kad se ispune neki uslovi. 25 minuta

Zadatak #2 (25 minuta)

Napraviti klasu **geo_figura** koja može imati proizvoljan broj argumenata. Argument predstavljaju stranice geometrijske figure. Napraviti metodu **koja_je_figura** koja ispituje tip figure na sledeći način:

- Ukoliko je uneseno 3 stranice, štampati da je trougao, četiri za četvorougao, itd.
- Ukoliko je trougao pravougli (proveriti testiranjem Pitagorine teoreme) izračunati i štampati površinu.
- Ukoliko je četvorougao pravougaonik ili romb (testirati da li ima dva para identičnih stranica), izračunati i štampati površinu.
- Ukoliko je unet petougao ili veći, javiti da se površina ne može izračunati.
- Za bilo koju unetu figuru izračunati obim.

Podići izuzetke ako:

- Korisnik ne unese nikakvu vrednost za stranicu.
- Korisnik unese negativnu vrednost za stranicu.
- Korisnik unese nulu za stranicu.
- Korisnik unese samo jednu ili dve stranice.

Debug-ovati program.

```
try:
    class geo_figura():
        stranice = []
        def __init__(self, *duzina_stranice):
            self.stranice.extend(duzina_stranice)
            for i in range(0, len(self.stranice)):
                if self.stranice[i] == 0:
                    raise Exception('Stranica ne može biti jednaka nuli!')
                if self.stranice[i] < 0:
                    raise Exception('Stranica ne može biti negativna!')

                if len(self.stranice) == 0:
                    raise Exception("Ne postoji nultougao")
                elif (len(self.stranice) == 1) or (len(self.stranice) == 2):
                    raise Exception('Dvougao je emisija, a jednougao ne postoji.')

        def koja_je_figura(self):
            self.obim = sum(self.stranice)
            self.stranice.sort()
            if len(self.stranice) == 3:
                print("U pitanju je trougao!")
                print(f'Obim je {self.obim}')
                if (self.stranice[0]**2 + self.stranice[1]**2) ==
self.stranice[2]**2:
                    print(f'Povrsina je {((self.stranice[0] * self.stranice[1])/2)}')

            elif len(self.stranice) == 4:
                print("Cetvorougao")
                print(f'Obim je {self.obim}')
                if (self.stranice[0] == self.stranice[1]) and (self.stranice[2] ==
self.stranice[3]):
                    print(f'Povrsina je {self.stranice[0]*self.stranice[2]}')

            else:
                print(f"U pitanju je {len(self.stranice)}-ugao")
                print(f'Obim je {self.obim}')

    except Exception as e:
        print('Neka greska!')
```

▼ Poglavlje 7

Individualna vežba #6

UVOD U INDIVIDUALNU VEŽBU #6

U individualnoj vežbi 6 rade se tri zadatka koje bi studenti mogli samostalno da odrade u toku dva časa.

Individualna vežba #6 odnosi se na rad sa datotekama, upis, čitanje i čuvanje, ali i sa procesom debagovanja i obradom izuzetaka.

Procenjeno trajanje individualnih vežbi iznosi 90 minuta.

Zadatak #1 (30 minuta)

Napisati program koji će učitati CSV datoteku (data u prilogu) koja sadrži spisak svih predmeta (šifre u jednom redu, i naziva u drugom).

Napraviti listu koja u sebi sadrži predmete koje slušate u ovom semestru po šifri.

Iz CSV datoteke učitati nazine samo onih predmeta koje slušate. Sačuvajte predmete koje slušate u ovom semestru u novu CSV datoteku.

CS101,IT101,MA104,NT111,CS102,IT210,CS115,NT112,IT331,SE201,IT350,NT213,CS230,IT370,CS220,CS323,IT255,CS225,IT335,CS324,SE325,IT355,IT381,IT376,CS322,SE401,OM350,CS232,NT213

Uvod u objektno-orientisano programiranje,Osnove informacionih tehnologija,Matematika,Engleski 1,Objekti i apstrakcija podataka,Sistemi informacionih tehnologija,Diskretne strukture,Engleski 2,Računarske mreže i komunikacij,Uvod u softversko inženjerstvo,Baze podataka,Engleski za informatičare,Distribuirani sistemi,Interakcija čovek-računar,Arhitektura računara,C/C++ programski jezik,Veb sistemi 1,Operativni sistemi,Administracija računarskih sistema i mreža,Skripting jezici,Upravljanje projektima razvoja softvera,Veb sistemi 2,Zaštita i bezbednost informacija,Robotika,C# Programska jezik,Timski razvoj softvera,Preduzetništvo,Programiranje 2D igara,Engleski za informatičare

Zadatak #2 (30 minuta)

Napisati program koji će sumirati sve brojeve koje korisnik unese sa komandne linije, ignorajući svaki ulaz koji nije validni broj. Program treba da pokazuje trenutnu sumu posle svakog unetog broja.

Javiti poruku ukoliko se ne unese ne-numeričku ulaz, ali nastaviti sa unosom. Izaći iz programa kada se unese prazna linija (Enter). Program treba da radi za celobrojne i razlomljene brojeve.

Zadatak #3 (30 minuta)

Programe individualnih vežbi 5. i 6. lekcije debug-ovati pomoću debugger unutar IDE-a.
Diskutovati međusobno

✓ Poglavlje 8

Domaći zadatak #6

DOMAĆI ZADATAK

Domaći zadatak #6 se okvirno radi 3h

Zadatak #1

Napisati program koji će praviti Vaš raspored časova i sačuvati kao CSV datoteku.

Otvoriti CSV datoteku u programu Excel (ili sličnom programu) kao dokaz da je uspešno napravljena datoteka.

Zadatak #2

Napisati funkciju **srednja_vrednost(niz)** koja vraća srednju vrednost elemenata niza. Niz može biti lista celobrojnih ili razlomljenih brojeva. Koristiti pomoćnu promenljivu **tmp** koja akumulira vrednosti niza, element po element.

Uhvatiti izuzetak ukoliko element liste nije jedan od ovih tipova podataka (posebne izuzetke za tip string, list, dict, tuple), i vratiti posebne poruke.

U glavnom programu pozvati funkciju 4 puta sa različitim parametrima i pratiti vrednosti ulaznog parametra funkcije (**niz**) i **tmp** promenljive kroz debugger.

Screenshot-ovati nekoliko stanja debugger-a.

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!

Domaći zadatak poslati dr Nemanji Zdravkoviću: nemanja.zdravkovic@metropolitan.ac.rs

Obavezno koristiti uputstvo za izradu domaćeg zadatka.

Uz .doc dokument (koji treba sadržati i screenshot svakog urađenog zadatka kao i komentare za zadatak), poslati i izvorne i dodatne datoteke.

✓ Poglavlje 9

Zaključak

ZAKLJUČAK

Zaključak lekcije #6

Rezime:

U ovo lekciji bilo je reči o radu sa datotekama i obradom izuzetaka u Python 3.x jeziku.

Posebna pažnja je data na radu sa CSV i JSON datotekama, i kako čitati i pisati u takve datoteke.

Zatim, bilo je reči o obradi izuzetaka u Python jeziku, kako napisati pravilan kod koji može javiti poruku o izuzecima kada se desi neka greška.

Konačno, bilo je reči u procesu debug-iranja unutar Python koda, i kako koristiti ugrađen debugger, ali i kako koristiti napredne debugger-e unutar IDE-a.

Primeri u okviru lekcije kao i zadaci za individualni rad treba da osposobe studente za rad u Python 3.x jeziku kroz rad sa datotekama, obradom izuzetaka, kao i proces debug-ovanja.

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.