Kurs: Programiranje igara sa programskim jezikom Pajton
012 čas: Program Kviz.py
Teme: tekstualni fajlovi, binarni fajlovi, izuzeci

Čitanje iz tekstualni fajlova

Sa Pajton programskim jezikom je lako čitati stringove iz čistih tekst fajlova – fajlova koji su napravljeni samo od ASCII karaktera. Tekst fajlovi su dobar izbor za trajno smeštanje informacija: one su nezavisne od platforme i lako se koriste.

001 Sadržaj citaj.txt tekstualnog fajla

Ovaj program prikazuje nekoliko načina za čitanje stringova iz tekst fajlova, od jednog znaka do čitavog fajla, ili čitanje linija po linija. Čitanje se izvodi pomoću fajla citaj.txt. Ovo je sadržaj fajla citaj.txt:

Linija 1
Ovo je linija 2
Onda je ovo linija 3

program Citaj.py

```
#Citaj
#Prikazuje citanje podataka iz tekst fajla
print("Otvaranje i zatvaranje fajla.")
tekst_fajl = open("citaj.txt", "r")
tekst_fajl.close()

print("\nCitanje znakova iz fajla.")
tekst_fajl = open("citaj.txt", "r")
print(tekst_fajl.read(1))
print(tekst_fajl.read(7))
tekst_fajl.close()

print("\nCitanje celog fajla odjednom.")
tekst_fajl = open("citaj.txt", "r")
celo = tekst_fajl.read()
print(celo)
tekst_fajl.close()

print("\nCitanje znakova po liniji.")
tekst_fajl = open("citaj.txt", "r")
print(tekst_fajl.readline(1))
print(tekst_fajl.readline(5))
tekst_fajl.close()

print("\nCitanje linija po linija.")
tekst_fajl = open("citaj.txt", "r")
print(tekst_fajl.readline())
print(tekst_fajl.readline())
print(tekst_fajl.readline())
tekst_fajl.close()

print("\nCitanje celog fajla kao lista.")
tekst_fajl = open("citaj.txt", "r")
linije = tekst_fajl.readlines()
print(linije)
print(len(linije))
for linija in linije:
    print(linija)
tekst_fajl.close()
```

```
print("\nPetljom kroz fajl, linija po linija.")
tekst_fajl = open("citaj.txt", "r")
for linija in tekst_fajl:
    print(linija)
tekst_fajl.close()
```

002 Otvaranje i zatvaranje fajla

Pre nego što se može pročitati (ili upisati u) fajl, potrebno je otvoriti fajl:

```
tekst_fajl = open("citaj.txt", "r")
```

Koristi se open() funkcija za otvaranje fajla i rezultat pokušaja otvaranja se dodeljuje promenjivoj tekst_fajl. U pozivu funkcije, postoje dva string argumenta: naziv tekstualnog fajla koji se otvara i mod pristupa.

Argument "citaj.txt" ukazuje na naziv fajla. Pošto se u nazivu ne uključuje putanja do fajla, Pajton traži u trenutnom direktorijumu taj tekstualni fajl. Tekstualnom fajlu se može prići u bilo kakvom direktorijumu ako je pravilno navedena putanja do tog direktorijuma.

Argument "r" je mod pristupa, koji govori Pajtonu da treba otvoriti fajl za čitanje. Fajlovi se mogu otvoriti za čitanje, upisivanje ili za obe radnje istovremeno. Sledeća tabela prikazuje validne modove pristupa tekst fajlova.

Mod	**Opis**
"r"	Čita iz tekstualnog fajla. Ako fajl ne postoji, Pajton prikazuje grešku.
"w"	Upisuje u tekstualni fajl. Ako fajl postoji, originalni sadržaj fajla se briše. Ako fajl ne postoji, automatski se kreira.
"a"	Pridodaje sadržaj u tekstualni fajl. Ako fajl postoji, novi podaci se pridodaju na kraju fajla. Ako fajl ne postoji, kreira se automatski.
"r+"	Čita iz i upisuje u tekst fajl. Ako fajl ne postoji, Pajton javlja grešku.
"w+"	Upisuje i čita iz tekstualnog fajla. Ako fajl postoji, sadržaj se briše. Ako fajl ne postoji, automatski se kreira.
"a+"	Pridodaje sadržaj i čita iz tekstualnog fajla. Ako fajl postoji, novi podaci se pridodaju na kraju postojećeg sadržaja u fajlu. Ako fajl ne postoji, automatski se kreira.

Posle otvaranja fajla, pristupa mu se kroz promenjivu tekst_fajl, koja predstavlja **fajl objekat**. Postoji veliki broj korisnih metoda objekata koji se mogu koristiti, ali najjednostavniji je close(), koji zatvara fajl, i ne dopušta dalje čitanje ili upisivanje sve dok se ponovo ne otvori:

```
tekst_fajl.close()
```

Kada se završi sa radom na tekstualnom fajlu, treba ga uvek zatvoriti.

003 Čitanje znakova iz fajla

Da bi fajl bio od bilo kakve koristi, treba nešto uraditi sa njegovim sadržajem između otvaranja i zatvaranja fajla. U programu, fajl je otvoren a zatim i pročitan njegov sadržaj sa read() objekat metodom.

Objekat metoda read() omogućava čitanje određenog broja znakova iz fajla, koje metoda vraća kao stringove. U programu se ponovo otvara fajl, a zatim i čita tačno jedan znak iz njega:

```
tekst_fajl = open("citaj.txt", "r")
print(tekst_fajl.read(1))
```

I na ekranu se pojavljuje: L

Broj u zagradi kaže koliko sledećih znakova treba pročitati, zato sa:

```
print(tekst_fajl.read(7))
```

se čitaju i štampaju sledećih sedam znakova: inija 1

Pajton pamti gde se stalo sa čitanjem u prethodnoj naredbi read() i odatle će nastaviti sa čitanjem sa sledećom naredbom read(). Ako se pročita kraj fajla, sledeće read() vraća prazan string.

Ako se ne napiše broj znakova koji će se pročitati, Pajton vraća ceo fajl kao string. Zato u sledećem delu programa:

```
tekst_fajl = open("citaj.txt", "r")
celo = tekst_fajl.read()
print(celo)
tekst_fajl.close()
```

Ovako se čita ceo tekstualni fajl, dodeli se vraćeni string promenjivoj celo, i štampa se sadržaj promenjive celo:

Linija 1
Ovo je linija 2
Onda je ovo linija 3

Ovo ima smisla ako je fajl malog sadržaja. Pošto je pročitan ceo fajl, bilo koje sledeće čitanje samo vraća prazan string. Zato se na kraju zatvara fajl.

004 Čitanje znakova iz linije

Često je potrebno raditi sa jednom po jednom linijom teksta. Metoda readline() omogućava čitanje karaktera sa trenutne linije teksta. Samo se priključi broj karaktera koliko se želi pročitati sa trenutne linije a metoda ih vraća kao string. Ako se ne priključi nikakav broj, metod čita od trenutne pozicije do kraja linije. Kada se pročitaju svi karakteri na liniji, sledeća linija je trenutna linija.

Posle ponovnog otvaranja fajla, prvo se čita prvi karakter na trenutnoj liniji:

```
tekst_fajl = open("citaj.txt", "r")
print(tekst_fajl.readline(1))
```

što na izlazu daje: L

A zatim se čitaju sledećih sedam karaktera sa trenutne linije:

```
print(tekst_fajl.readline(7))
```

što na izlazu daje: inija 1

```
tekst_fajl.close()
```

čime se zatvara fajl.

Na prvi pogled se ne vidi razlika između metoda read() i readline(). Metoda readline() čita karaktere samo sa trenutne linije, dok read() čita karaktere iz celog fajla. Zato se readline() koristi prilikom čitanja linije po linije teksta, a to se i radi u sledećem delu programa:

```
tekst_fajl = open("citaj.txt", "r")
print(tekst_fajl.readline())
print(tekst_fajl.readline())
print(tekst_fajl.readline())
```

gde se prvo pročita i štampa: Linija 1

zatim: Ovo je linija 2

i na kraju: Onda je ovo linija 3

posle svakog štampanja se dobija i prazan red na ekranu. To je zato što svaka linija u tekstualnom fajlu se završava sa karakterom novog reda (“\n“).

005 Učitavanje svih linija u listu

Drugi način rada sa linijama u tekst fajlu je pomoću readlines() metode, koja učita tekst fajl u listu, pri čemu svaka linija fajla postaje string element u listi:

```
tekst_fajl = open("citaj.txt", "r")
linije = tekst_fajl.readlines()
```

```
print(linije)
print(len(linije))
for linija in linije:
    print(linija)
```

Promenjiva linije sada upućuje na listu sa elementima koji su string linije fajla. Pa kada se linije štampa, dobija se:

```
['Linija 1\n', 'Ovo je linija 2\n', 'Onda je ovo linija 3\n']
```

Promenjiva linija je kao i svaka druga lista. Može se naći dužina liste, pa i proći petljom kroz listu:

```
print(len(linije))
```
daje : 3

```
for linija in linije:
    print(linija)
```

daje:

Linija 1

Ovo je linija 2

Onda je ovo linija 3

006 Prolazak petlje kroz tekstualni fajl

Kroz tekstualni fajl se može proći petljom direktno kroz linije fajla:

```
tekst_fajl = open("citaj.txt", "r")
for linija in tekst_fajl:
    print(linija)
```

daje:

Linija 1

Ovo je linija 2

Onda je ovo linija 3

Prema tome, promenjiva linija dobija svaku liniju iz fajla, jednu po jednu. Prva iteracija petlje dobija prvu liniju, druga iteracija dobija drugu liniju itd. Ovo je najelegantnije rešenje ako se želi proći kroz tekstualni fajl linija po linija.

Upisivanje u tekstualni fajl

Da bi tekstualni fajl bio koristan vid čuvanja podataka, mora postojati način da se u fajl upisuju podaci. Sa Pajtonom, upis stringova u tekstualni fajl je jednostavno. Postoje dva osnovna načina za to.

program Pisi.py

Program kreira tekst fajl sa istim sadržajem kao u citaj.txt fajlu.

```
#Pisi
#prikazuje upisivanje u tekst fajl
print("Kreiranje tekst fajla sa metodom write().")
tekst_fajl = open("pisi.txt", "w")
tekst_fajl.write("Linija 1\n")
tekst_fajl.write("Ovo je linija 2\n")
tekst_fajl.write("Onda je ovo linija 3\n")
tekst_fajl.close()
```

```
print("\nCitanje upravo kreiranog fajla.")
tekst_fajl = open("pisi.txt", "r")
print(tekst_fajl.read())
tekst_fajl.close()

print("\nKreiranje tekst fajla sa metodom writelines().")
tekst_fajl = open("pisi.txt", "w")
linije = ["Linija 1\n", "Ovo je linija 2\n", "Onda je ovo linija 3\n"]
tekst_fajl.writelines(linije)
tekst_fajl.close()

print("\nCitanje upravo kreiranog fajla.")
tekst_fajl = open("pisi.txt", "r")
print(tekst_fajl.read())
tekst_fajl.close()
```

```
Citanje upravo kreiranog fajla.
Linija 1
Ovo je linija 2
Onda je ovo linija 3


Kreiranje tekst fajla sa metodom writelines().

Citanje upravo kreiranog fajla.
Linija 1
Ovo je linija 2
Onda je ovo linija 3
```

007 Upisivanje stringova u fajl

Vidi se da se mora koristiti pristupni mod “w“ da bi se mogao koristiti fajl za upisivanje podataka. Fajl pisi.txt se pojavljuje kao prazan tekst fajl koji samo čeka da program upiše stringove u njega. Ako je pisi.txt već postojao, bio bi zamenjen sa novim, praznim fajlom i sav sadržaj bi bio izbrisan.

Sa metodom write(), argumenti unutar zagrade su sadržaj koji se upisuje kao string u fajl. Metod write() ne ubacuje automatski karakter nove linije na kraj stringa koji se upisuje. Programer mora sam uneti karaktere nove linije ako su neophodni. Da u primeru nisu uneti karakteri nove linije, svi stringovi bi bili upisani kao jedna dugačka string linija u fajlu.

Takođe, nije neophodno uneti tri metode write() za tri string linije, već može i:

```
tekst_fajl.write("Linija 1\n Ovo je linija 2\n Onda je ovo linija 3\n")
```

Sledi dao koda kojim se proverava, tj čita sadržaj tekst fajla pisi.txt.

008 Upisivanje liste stringova u fajl

Zatim se kreira isti fajl korišćenjem writelines() fajl objekt metode. Metoda writelines() radi sa listom stringova i upisuje listu stringova u fajl.

Na početku se otvara fajl istog imena kao i prethodni, pisi.txt, što znači da je prethodni obrisan i kreiran novi prazan. Zatim se kreira lista stringova za upis u fajl:

```
linije = ["Linija 1\n", "Ovo je linija 2\n", "Onda je ovo linija 3\n"]
```

I ovde je unet karakter novog reda u stringove.

Zatim se upisuje cela lista stringova u fajl sa writelines() metodom:

```
tekst_fajl.writelines(linije)
```

Posle se proverava urađeni posao čitanjem sadržaja fajla pisi.txt.

009 Neke fajl objekat metode

Metode	Opis
close()	Zatvara fajl. Zatvoreni fajl se ne može čitati i u njega se ne može upisivati sve dok se ne otvori.
read([velicina])	Čita ukupno velicina karaktera iz fajla i vraća ih kao string. Ako velicina nije definisana, metod vraća sve karaktere od trenutne pozicije sve do kraja fajla.
readline([velicina])	Čita ukupno velicina karaktera iz trenutne linije fajla i vraća ih kao string. Ako velicina nije definisana, metoda vraća sve karaktere od trenutne pozicije do kraja trenutne linije fajla.
readlines()	Čita sve linije u fajlu i vraća ih kao elemente liste.
write(izlaz)	Upisuje string izlaz u fajl.
writelines(izlaz)	Upisuje stringove iz liste izlaz u fajl.

Smeštanje kompleksnih podataka u fajlove

Tekst fajlovi su pogodni pošto se mogu čitati i manipulisati sa bilo kojim tekst editorom, ali oni su ograničeni u smeštanju nizova karaktera. Ponekad je potrebno čuvati kompleksnije podatke, poput liste ili rečnika i u istom ih obliku sačuvati u fajlu.

Program Sacuvaj.py služi da prikaže kako se kompleksni podaci smeštaju (pikluju, pickling), čuvaju u binarnim fajlovima.

program Sacuvaj.py

```
#Sacuvaj
#Prikazuje pickle i shalve module
import pickle, shelve
print("Piklovanje lista.")
opis = ["slatko", "ljuto", "gorko"]
oblik = ["ceo", "siljast", "tanak"]
brend = ["Claussen", "Heinz", "Vlassic"]
f = open("piklovi1.dat", "wb")
pickle.dump(opis, f)
pickle.dump(oblik, f)
pickle.dump(brend, f)
f.close()

print("\nDepiklovanje lista.")
f = open("piklovi1.dat", "rb")
opis = pickle.load(f)
oblik = pickle.load(f)
brend = pickle.load(f)
print(opis)
print(oblik)
print(brend)
f.close()

print("\nSelovanje lista.")
s = shelve.open("piklovi2.dat")
s["opis"] = ["slatko", "ljuto", "gorko"]
s["oblik"] = ["ceo", "siljast", "tanak"]
s["brend"] = ["Claussen", "Heinz", "Vlassic"]
s.sync()    #potvrdjuje da su podaci upisani

print("\nDobijanje lista iz selovanih fajlova:")
print("brend -", s["brend"])
print("oblik -", s["oblik"])
print("opis -", s["opis"])
s.close()
```

```
Piklovanje lista.

Depiklovanje lista.
['slatko', 'ljuto', 'gorko']
['ceo', 'siljast', 'tanak']
['Claussen', 'Heinz', 'Vlassic']

Selovanje lista.

Dobijanje lista iz selovanih fajlova:
brend - ['Claussen', 'Heinz', 'Vlassic']
oblik - ['ceo', 'siljast', 'tanak']
opis - ['slatko', 'ljuto', 'gorko']
```

010 Piklovanje podataka i upisivanje podataka u fajl

Prvo se importuju dva nova modula, pickle i shelve.

Modul pickle omogućava dohvatanje i smeštanje kompleksnijih podataka u fajl. Modul shelve omogućava smeštanje i direktan pristup piklovanim objektima iz fajla.

Piklovanje je jednostavno. Umesto upisivanja karaktera u fajl, može se upisati piklovan objekat u fajl. Piklovani objekti su smešteni u fajlove poput karaktera; mogu se smestiti i mogu se dobiti sekvencijalno. Prvo, kreiraju se tri liste koje će se piklovati i upisati u fajl:

```
print("Piklovanje lista.")
opis = ["slatko", "ljuto", "gorko"]
oblik = ["ceo", "siljast", "tanak"]
brend = ["Claussen", "Heinz", "Vlassic"]
```

Zatim se otvara novi fajl za smeštanje piklovanih lista:

```
f = open("piklovi1.dat", "wb")
```

Piklovani objekti moraju biti smešteni u binarnim fajlovima, nikako u tekstualnim fajlovima. Zato se otvara novi binarni fajl piklovi1.dat za upisivanje upotrebom “wb“ kao fajl pristupnog moda.

Lista korisnik pristupnih modova za binarne fajlove:

Mod	**Opis**
“rb“	Čita iz binarnog fajla. Ako fajl ne postoji, Pajton prikazuje grešku.
“wb“	Upisuje u binarni fajl. Ako fajl postoji, originalni sadržaj fajla se briše. Ako fajl ne postoji, automatski se kreira.
“ab“	Pridodaje sadržaj u binarni fajl. Ako fajl postoji, novi podaci se pridodaju na kraju fajla. Ako fajl ne postoji, kreira se automatski.
“rb+“	Čita iz i upisuje u binarni fajl. Ako fajl ne postoji, Pajton javlja grešku.
“wb+“	Upisuje i čita iz binarnog fajla. Ako fajl postoji, sadržaj se briše. Ako fajl ne postoji, automatski se kreira.
“ab+“	Pridodaje sadržaj i čita iz binarnog fajla. Ako fajl postoji, novi podaci se pridodaju na kraju postojećeg sadržaja u fajlu. Ako fajl ne postoji, automatski se kreira.

Zatim se pikluju i smeštaju tri liste, opis, oblik i brend u fajl piklovi1.dat korišćenjem pickle.dump() funkcije. Funkcija zahteva dva argumenta: podatak za piklovanje i fajl u koji će se smestiti:

```
pickle.dump(opis, f)
pickle.dump(oblik, f)
pickle.dump(brend, f)
```

To znači da kod pikluje listu opis i upisuje je celu kao jedan objekat u fajl piklovi1.dat. Zatim to isto sa oblik pa sa brend listama. Zatim se fajl zatvara.

Piklovati se mogu brojevi, stringovi, ntorke, liste i rečnici.

011 Čitanje podataka iz fajlova i depiklovanje podataka

Zatim se preuzimaju i depikluju tri liste sa pickle.load() funkcijom. Funkcija ima jedan argument, fajl iz kojeg se učitava sledeći piklovani objekat:

```
print("\nDepiklovanje lista.")
f = open("piklovi1.dat", "rb")
opis = pickle.load(f)
oblik = pickle.load(f)
brend = pickle.load(f)
```

Program čita prvi piklovani objekat iz fajla, depikluje ga i realizuje listu `["slatko", "ljuto", "gorko"]`i dodeljuje listu promenjivoj opis. Slično se radi sa sledeće dve liste.

Postoje dva značajne pickle funkcije:

dump(objekat, fajl, [,bin]) – upisuje piklovanu verziju objekat u fajl; ako je bin true, objekat je upisan u binarnom formatu; ako je bin false, objekat je upisan u manje efikasnom ali lakšem za čitanje tekstualnom formatu; difolt vrednost za bin je false

load(fajl) – depikluje i vraća sledeći piklovani objekat u fajl

012 Šelfovanje za smeštanje piklovanih podataka

Ideja o piklovanju se unapređuje šelfovanjem lista zajedno u jedan fajl korišćenjem shelve modula. Tako se kreira shelf koja služi kao rečnik, koji omogućava direktan pristup listama.

Prvo se kreira shelf s:

```
s = shelve.open("piklovi2.dat")
```

Funkcija shelve.open() radi slično kao open() funkcija. Ipak shelve.open() funkcija radi sa fajlom u kojem su smešteni piklovani objekti a ne karakteri. U ovom slučaju, dodeljena je rezultujuća shelf s, koja sada radi kao rečnik čiji sadržaj je smešten u fajl ili grupe fajlova.

Kada se pozove shelve.open() Pajton može dodati ekstenziju u ime fajla koji je programer odredio. Pajton takođe može kreirati dodatne fajlove za podržavanje novo kreirane shelf.

Funkcija shelve.open() traži jedan argument: ime fajla. Takođe opciono može imati i pristupni mod. Ako se ne ubaci pristupni mod, po difoltu to je “c“.

Mod	**Opis**
“c“	Otvara fajl za čitanje ili upisivanje. Ako fajl ne postoji, kreira se.
“n“	Kreira novi fajl za čitanje ili upisivanje. Ako fajl ne postoji, sadržaj je obrisan.
“r“	Čita iz fajla. Ako fajl ne postoji, Pajton daje grešku.
“w“	Upisuje u fajl. Ako fajl ne postoji, Pajton daje grešku.

Dodate su tri liste u šelf:

```
s["opis"] = ["slatko", "ljuto", "gorko"]
s["oblik"] = ["ceo", "siljast", "tanak"]
s["brend"] = ["Claussen", "Heinz", "Vlassic"]
```

Piklovi rade poput rečnika. Ključ “opis“ se uparuje sa vrednostima `["slatko", "ljuto", "gorko"]`. Ključ “oblik“ se uparuje sa vrednostima `["ceo", "siljast", "tanak"]` . Bitno je primetiti da šelf ključ može biti samo string.

Na kraju se podiže sync() metoda: s.sync()

Pajton upisuje promene u šelf fajlu u bafer a zatim periodično upisuje sadržaj bafera u fajl. Da bi se obezbedilo da fajl sadrži sve promene u šelfu, može se pozvati sync() metoda. Šelf fajl je takođe updejtovan kada se zatvori sa close() metodom.

Iako se može simulirati šelf piklovanjem rečnika, shelve modul je memorijski efikasniji. Zato, kada je potrebno direktno pristupiti piklovanim objektima, kreira se šelf.

013 Korišćenje šelfova za dobijanje piklovanih podataka

Pošto šelf radi kao rečnik, može se direktno pristupiti piklovanim objektima u njemu korišćenjem ključeva. Za proveru:

```
print("brend -", s["brend"])
print("oblik -", s["oblik"])
print("opis -", s["opis"])
```

U realnosti piklovanje i depiklovanje su dobar način za smeštanje i dobijanje strukturisanih informacija ali još kompleksnije informacije mogu tražiti još više snage i fleksibilnosti. Baze podataka i XML su dve popularne metode za smeštanje i dobijanje kompleksnih podataka, a Pajton ima module koji rade sa obe metode.

Rad sa izuzecima

Kad aPajton naleti na grešku, on stopira trenutni program i prikazuje poruku o grešci. Tačnije, podiže izuzetak (exception), ukazujući da se nešto neočekivano desilo. Ako se ništa ije preduzelo sa izuzetkom, Pajton stopira izvršenje programa i prikazuje poruku o grešci sa detaljima o izuzetku.

Primer:

```
>>> num = float("Hi!")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'Hi!'
```

U ovom primeru, Pajton pokušava da konvertuje string u float broj. Pošto to ne može da se uradi, Pajton podiže izuzetak i prikazuje detalje o izuzetku.

Upotrebom funkcionalnosti rukovanja sa izuzecima programer može preuprediti pojavu izuzetaka tako da program se ne prekida čak ni pri nemogućim kodovima. Može se urediti da program makar uputi korisnika o problemu i zatraži prekid programa.

program Uredi.py

Program radi do pojave izuzetka koji se dešava posle unosa od strane korisnika a zatim samostalno izaziva nekoliko izuzetaka. Ali, umesto da se prekine, program radi do kraja. To je zato što program rukuje izuzecima koji su se podigli.

```
#Uredi
#Prikazuje rukovanje izuzecima

#try/except
try:
    broj = float(input("Unesi broj: "))
except:
    print("Nesto nije u redu!")

#odredjivanje tipa izuzetka
try:
    broj = float(input("\nUnesi broj: "))
except ValueError:
    print("To nije broj!")

#rukovanje sa vise tipova izuzetaka
print()
for vrednost in (None, "Zdravo!"):
    try:
        print("Pokusaj konvertovanja", vrednost, "-->", end=" ")
        print(float(vrednost))
    except (TypeError, ValueError):
        print("Nesto nije u redu!")
```

```
print()
for vrednost in (None, "Zdravo!"):
    try:
        print("Pokusaj konvertovanja", vrednost, "-->", end=" ")
        print(float(vrednost))
    except TypeError:
        print("Mogu konvertovati samo string ili broj!")
    except ValueError:
        print("Mogu konvertovati samo string od cifara!")

#dobijanje argumenta izuzetaka
try:
    broj = float(input("\nUnesi broj: "))
except ValueError as e:
    print("To nije bio broj! Ili sto bi Pajton rekao...")
    print(e)

#try/except/else
try:
    broj = float(input("\nUnesi broj: "))
except ValueError:
    print("To nije bio broj!")
else:
    print("Uneo si broj", broj)
```

014 Iskaz try sa iskazom except

Najosnovniji način da se rukuje (ili uhvati) izuzetak je upotreba try iskaza sa except iskazom. Korišćenjem try iskaza, sigurno je da se jedan deo koda, koji bi potencijalno podigao izuzetak, neće realizovati. Zato se piše except iskaz sa blokom iskaza koji se realizuju samo ako se podigne izuzetak.

U programu se prvo pita korisnik za broj. Ako korisnik unese string a zatim se pokuša konverzija stringa u float. Koristi se try i except za rukovanje bilo kojim izuzecima koji bi se mogli podići u procesu.

```
try:
    broj = float(input("Unesi broj: "))
except:
    print("Nesto nije u redu!")
```

Daje:

```
Unesi broj: Zdravo!
Nesto nije u redu!
```

Ako poziv float() podigne izuzetak (kao rezultat korisnikovog unosa stringa koji se ne može konvertovati), izuzetak je uhvaćen i korisnik je informisan da nešto nije u redu. Ako se ne podigne nikakav izuzetak, promenjiva broj dobija vrednost koju je korisnik uneo i program preskače except iskaz, nastavljajući sa ostatkom koda.

015 Određivanje tipa izuzetka

Različite vrste grešaka izazivaju različite tipove izuzetaka. Npr, pokušaj konvertovanja stringa “Zdravo!“ sa float() rezultuje sa ValueError izuzetkom pošto karakteri u stringu au pogrešnog tipa (nisu cifre). Postoji preko 20 tipova izuzetaka, a na tabeli su prikazani najčešći:

Tip izuzetka	**Opis**
IOError	Podiže se kada I/O operacije propadne, poput pokušaja otvaranja nepostojećeg fajla u modu čitanja.

IndexError	Podiže se kada je sekvenca indeksirana sa brojem nepostojećeg elementa.
KeyError	Podiže se kada nije pronađen ključ u rečniku.
NameError	Podiže se kada ime promenjive ili funkcije nisu pronađeni.
SyntaxError	Podiže se pri sintaksnoj grešci.
TypeError	Podiže se kada se primeni ugrađena operacija ili funkcija na objekat neodgovarajućeg tipa.
ValueError	Podiže se kada ugrađena operacija ili funkcija primi argument koji je dobrog tipa ali neodgovarajuće vrednosti.
ZeroDivisionError	Podiže se kada je drugi argument operacije deljenja ili modula nula.

Iskaz except omogućava određivanje sa kojim tipom izuzetka će da rukuje. Da bi se odredio jedan tip izuzetka, mora se izlistati određeni tip izuzetka posle except.
U kodu, pita se korisnik za broj ali sada se hvata VelueError izuzetak:

```
try:
    broj = float(input("\nUnesi broj: "))
except ValueError:
    print("To nije broj!")
```

Daje:

```
Unesi broj: Zdravo!
To nije broj!
```

Sada, funkcija print se izvršava samo ako se podigne ValueError izuzetak. Kao rezultat, može se biti još određeniji u ispisivanju poruke poput: To nije broj! Ipak, ako bi se podigao bilo koji drugi tip izuzetka unutar try iskaza, except iskaz ga neće uhvatiti i program će se zaustaviti.
Dobra praksa u programiranju je da se specificira tip izuzetka i tako rukuje svaki pojedinačni slučaj. Zapravo je loše pokušati hvatati sve izuzetke kao u prvom primeru i takav način rada treba izbegavati.
Kada pokušati hvatanje izuzetaka? Svaki deo koda gde je spoljna interakcija sa programom je dobro mesto za to. Dobro mesto je pri otvaranju fajlova za čitanje, čak iako se veruje da fajl već postoji. Takođe, pri pokušaju konverzije podatka iz spoljnog izvora, poput korisnika.
Neka treba uhvatiti neki izuzetak ali nije sigurno koji niti kako se zove. Način da se to pronađe je sledeći: kreirati izuzetak. Npr, treba uhvatiti izuzetak deljenja nulom, ali nije sigurno kako se zove taj izuzetak, samo u interpreteru napisati operaciju deljenja 0:

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

Ovako se vidi da je naziv izuzetka ZeroDivisionError.

016 Rukovanje sa više tipova izuzetaka

Jedan kod može dovesti do različitih tipova izuzetaka. Srećom, moguće je uhvatiti više tipova izuzetaka. Jedan način hvatanja više tipova izuzetaka je izlistati ih u jednom except iskazu kao grupu odvojenu zarezima unutar zagrade:

```
for vrednost in (None, "Zdravo!"):
    try:
        print("Pokusaj konvertovanja", vrednost, "-->", end=" ")
        print(float(vrednost))
    except (TypeError, ValueError):
        print("Nesto nije u redu!")
```

```
Pokusaj konvertovanja None --> Nesto nije u redu!
Pokusaj konvertovanja Zdravo! --> Nesto nije u redu!
```

Ovaj kod pokušava konverziju dva različita tipa u float. Oba ne uspevaju, ali svaki podiže različit tip izuzetka. Kod float(None) podiže se TypeError pošto funkcija može samo konvertovati stringove i brojeve. Kod float("Zdravo!") podiže se ValueError pošto je u pitanju string ali znakovi nisu sifre. Kao rezultat except iskaza, svaki tip izuzetka se rukuje.
Drugi način hvatanja više izuzetaka je sa više except izkaza. Može se izlistati koliko god treba except iskaza posle try iskaza:

```
for vrednost in (None, "Zdravo!"):
    try:
        print("Pokusaj konvertovanja", vrednost, "-->", end=" ")
        print(float(vrednost))
    except TypeError:
        print("Mogu konvertovati samo string ili broj!")
    except ValueError:
        print("Mogu konvertovati samo string od cifara!")
```

```
Pokusaj konvertovanja None --> Mogu konvertovati samo string ili broj!
Pokusaj konvertovanja Zdravo! --> Mogu konvertovati samo string od cifara!
```

Sada, svaki tip izuzetka ima svoj blok. Tako da kada je vrenost None, TypeError se podiže i string se štampa.
Korišćenje više except iskaza omogućava definisanje jedinstvene reakcije na različite tipove izuzetaka iz istog try bloka. U ovom slučaju, omogućava se detaljnija poruka o zasebnoj grešci.

017 Dobijanje argumenta izuzetaka

Kada se desi izuzetak, on može imati pridruženu vrednost, argument izuzetka. Argument je obično zvanična poruka od Pajtona kojime se opisuje izuzetak. Može se dobiti argument ako se odredi promenjiva posle tipa izuzetka, posle koje sledi ključna reč as.
Na primeru, dobija se argument izuzetka u promenjivoj e i štampa zajedno sa uobičajenom porukom o grešci:

```
try:
    broj = float(input("\nUnesi broj: "))
except ValueError as e:
    print("To nije bio broj! Ili sto bi Pajton rekao...")
    print(e)
```

```
Unesi broj: Zdravo!
To nije bio broj! Ili sto bi Pajton rekao...
could not convert string to float: 'Zdravo!'
```

018 Dodavanje komande else

Može se dodati komanda else posle exept bloka u try iskazu. Blok else se izvršava samo ako nije podignut nijedan izuzetak u try bloku:

```
try:
    broj = float(input("\nUnesi broj: "))
except ValueError:
    print("To nije bio broj!")
else:
    print("Uneo si broj", broj)
```

```
Unesi broj: 5.6
Uneo si broj 5.6
```

U kodu, broj se štampa u bloku else samo ako iskaz dodele u try bloku ne podigne izuzetak. Ovo je dobro pošto to znači da će broj biti odštampan samo ako je iskaz dodele bio uspešan i promenjiva postoji.

Igra izazovnog kviza

Program čita tekst fajl, kviz.txt koj mora biti postavljen u isti direktorijum gde će biti i .py fajl.

Epizoda koju ne mozes odbiti
Problem sa sisarom
Recimo da imas dokaz u istrazi i da moras "predati jagnje". Ako predugo cekas /sta ce se desiti?
Pojavice se ovca
Pojavice se krava
Pojavice se koza
Pojavice se emu
1
Jagnje je samo mlada ovca.
Kum ce sada da posedi sa tobom
Neka imas sastanak sa Kumom soula. Kako bi bilo /pametno da ga oslovis?
Gospodin Ricard
Gospodin Domino
Gospodin Braun
Gospodin Ceker
3
Dzejms Braun je Kum soula.
Ovo ce da te kosta
Ako si dao mafijasima novac za zastitu u rupijima, koji biznis pokusavas /da osiguras?
Farma lalama u Holandiji
Fabrika pudera od karija u Indiji
Destilerija votke u Rusiji
Skladiste vojnih nozeva u Svajcarskoj
2
Rupiji su standardna novcana jedinica u Indiji.
Sve ostaje u porodici
Ako je sin tvoje majke od tvog oca od tvoje sestre u "Familiji", u kakvom si ti /odnosu sa mafijom?
Preko tvog prvog rodjaka jednom sklonjenog
Preko tvog prvog rodjaka dvaput sklonjenog
Preko tvog drugog rodjaka jednom sklonjenog
Preko tvog drugog rodjaka dvaput sklonjenog
1
Sestra od tvoje majke tvoga oca je njena tetka -- a njen sin je od tvoje /majke prvi rodjak. Posto ste ti i tvoja majka jedna generacija /razlike, njen prvi rodjak je tvoj prvi rodjak jednom sklonjen.
Sluskinjac

Ako bukvalno peres pare, ne zelis da zeleno sa tvojih novcanica /se opere, koju temperaturu moras da koristis?
Vruce
Toplo
Umereno
Hladno
4
Prema mojoj boci sa deterdzentom, najbolje je koristiti hladno za boje koje se skidaju.

019 Izgled podataka u fajlu

Ovo je struktura podataka u kviz.txt fajlu. Prva linija u fajlu je naslov sesije. Ostatak fajla se sastoji od bloka sa sedam linija za svako pitanje. Može se koristiti neograničeni broj pitanja. Ovo je generička šema bloka:
<kategorija>
<pitanje>
<odgovor 1>
<odgovor 2>
<odgovor 3>
<odgovor 4>
<tačan odgovor>
<objašnjenje>
Vidi se iz teksta fajla da je unutar stringova uključen znak /. Sa njime se označava novi red pošto Pajton automatski ne deli tekst kada ga štampa. Kada program čita liniju iz fajla, zamenjuje sve / znake sa karakterom nove linije.

program Kviz.py

```
#Kviz
#Igra kviza koja cita tekstualni fajl
import sys
def otvoren_fajl(ime_fajla, mod):
    """Otvara fajl."""
    try:
        taj_fajl = open(ime_fajla, mod)
    except IOError as e:
        print("Ne moze se otvoriti fajl", ime_fajla, "Zavrsavam program.\n", e)
        input("\n\nPritisni ENTER za izlaz.")
        sys.exit()
    else:
        return taj_fajl

def sledeca_linija(taj_fajl):
    """Vraca sledecu formatiranu liniju iz kviz fajla."""
    linija = taj_fajl.readline()
    linija = linija.replace("/", "\n")
    return linija

def sledeci_blok(taj_fajl):
    """Vraca sledeci blok podataka iz kviz fajla."""
    kategorija = sledeca_linija(taj_fajl)
    pitanje = sledeca_linija(taj_fajl)
    odgovori = []
    for i in range(4):
        odgovori.append(sledeca_linija(taj_fajl))

    tacno = sledeca_linija(taj_fajl)
```

```
    if tacno:
        tacno = tacno[0]

    objasnjenje = sledeca_linija(taj_fajl)

    return kategorija, pitanje, odgovori, tacno, objasnjenje

def dobrodosli(naslov):
    """Dobrodoslica igracu i ime igraca."""
    print("\t\tDobrodosli u Kviz izazova!\n")
    print("\t\t", naslov, "\n")

def main():
    kviz_fajl = otvoren_fajl("kviz.txt", "r")
    naslov = sledeca_linija(kviz_fajl)
    dobrodosli(naslov)
    bodovi = 0

    #uzmi prvi blok
    kategorija, pitanje, odgovori, tacno, objasnjenje = sledeci_blok(kviz_fajl)
    while kategorija:
        #postavi pitanje
        print(kategorija)
        print(pitanje)
        for i in range(4):
            print("\t", i + 1, "-", odgovori[i])

        #uzmi odgovor
        odgovor = input("Koji je tvoj odgovor?: ")

        #provera odgovora
        if odgovor == tacno:
            print("\nTacno!", end=" ")
            bodovi += 1
        else:
            print("\nNetacno.", end=" ")
        print(objasnjenje)
        print("Bodovi:", bodovi, "\n\n")

        #uzmi sledeci blok
        kategorija, pitanje, odgovori, tacno, objasnjenje = sledeci_blok(kviz_fajl)

    kviz_fajl.close()

    print("To je bilo poslednje pitanje!")
    print("Tvoj ukupan rezultat je", bodovi)

main()
```

020 Funkcija otvoren_fajl()

Prvo se u programu definiše funkcija otvoren_fajl(). Ona prima ime fajla i mod a vraća odgovarajući fajl objekat.

```
import sys
def otvoren_fajl(ime_fajla, mod):
    """Otvara fajl."""
    try:
        taj_fajl = open(ime_fajla, mod)
    except IOError as e:
        print("Ne moze se otvoriti fajl", ime_fajla, "Zavrsavam program.\n", e)
        input("\n\nPritisni ENTER za izlaz.")
        sys.exit()
    else:
```

```
        return taj_fajl
```

Koristi se try i except za hvatanje IOError izuzetaka za ulazne/izlazne greške, koje se mogu pojaviti ako fajl npr ne postoji.

Ako se uhvati izuzetak, to znači da postoji problem u otvaranju kviz fajla. Ako se to desi, nema smisla nastaviti sa programom, tako da se ispisuje odgovarajuća poruka i poziva sys.exit() funkcija. Ova funkcija podiže izuzetak koji rezultuje prekidom programa. Trebalo bi koristiti sys.exit() samo kao poslednju mogućnost, kada se baš mora prekinuti program. Modul sys se mora importovati na početku koda.

021 Funkcija sledeca_linija()

Ova funkcija prima fajl objekat i vraća sledeću liniju teksta iz njega:

```
def sledeca_linija(taj_fajl):
    """Vraca sledecu formatiranu liniju iz kviz fajla."""
    linija = taj_fajl.readline()
    linija = linija.replace("/", "\n")
    return linija
```

Ipak postoji malo formatiranje linije pre nego se vraća. Zamenjeni su svi “/“ sa karakterom nove linije. To je stoga što Pajton automatski ne uređuje odštampani tekst. Ova procedura daje malo veću kontrolu izlaza programeru. Sada se može urediti gde se nalazi prelom u novi red teksta.

022 Funkcija sledeci_blok()

Ova funkcija čita sledeći blok linija za jedno pitanje.

```
def sledeci_blok(taj_fajl):
    """Vraca sledeci blok podataka iz kviz fajla."""
    kategorija = sledeca_linija(taj_fajl)
    pitanje = sledeca_linija(taj_fajl)
    odgovori = []
    for i in range(4):
        odgovori.append(sledeca_linija(taj_fajl))

    tacno = sledeca_linija(taj_fajl)
    if tacno:
        tacno = tacno[0]

    objasnjenje = sledeca_linija(taj_fajl)

    return kategorija, pitanje, odgovori, tacno, objasnjenje
```

Uzima fajl objekat a vraća četiri stringa i listu stringova. Vraća stringove za kategoriju, pitanje, tačan odgovor i objašnjenje kao i listu od četiri stringa kao moguće odgovore na pitanje.

Ako se dođe do kraja fajla, čitanje linije vraća prazan string. Tako da kada program dođe do kraja kviz.txt fajla, kategorija dobija prazan string. Kategorija se proverava u main() funkciji. Kada postane prazna, igra se završava.

023 Funkcija dobrodosli()

Funkcija pozdravlja igrača i najavljuje naslov sesije.

```
def dobrodosli(naslov):
    """Dobrodoslica igracu i ime igraca."""
    print("\t\tDobrodosli u Kviz izazova!\n")
    print("\t\t", naslov, "\n")
```

Funkcija dobija naslov sesije kao string i štampa ga zajedno sa porukom.

024 Postavka igre

Unutar main() funkcije, postavljaju se početni elementi igre:

```
def main():
    kviz_fajl = otvoren_fajl("kviz.txt", "r")
```

```
    naslov = sledeca_linija(kviz_fajl)
    dobrodosli(naslov)
    bodovi = 0
```

025 Postavljanje pitanja

Sledeće, učitavaju se prvi blokovi linija za prvo pitanje u promenjive. Onda, počinje while petlja:

```
while kategorija:
        #postavi pitanje
        print(kategorija)
        print(pitanje)
        for i in range(4):
            print("\t", i + 1, "-", odgovori[i])
```

ova petlja će se nastavljati sve dok se ne dobije prazan string u promenjivu kategorija. Ako je kategorija prazana string, to znači kraj kviz fajla i neće se više ulaziti u petlju. Pitanje se postavlja štampanjem kategorije za određeno pitanje, samog pitanja i četiri moguća odgovora.

026 Dobijanje odgovora

Odgovor od ighrača se dobija sa:

```
#uzmi odgovor
        odgovor = input("Koji je tvoj odgovor?: ")
```

027 Provera odgovora

Zatim se upoređuju odgovori igrača sa tačnim odgovorima. Ako su identični, igraču se čestita i povećava se suma bodova za jedan. Ako nisu isti, igraču se kaže da nije pogođen odgovor. U svakom slučaju, objašnjenje se prikaže, kojim se opisuje zašto je baš to tačan odgovor.

```
#provera odgovora
        if odgovor == tacno:
            print("\nTacno!", end=" ")
            bodovi += 1
        else:
            print("\nNetacno.", end=" ")
        print(objasnjenje)
        print("Bodovi:", bodovi, "\n\n")
```

Na kraju se ispisuje trenutni skor u bodovima.

Zatim se poziva funkcija sledeci_blok():

```
#uzmi sledeci blok
        kategorija, pitanje, odgovori, tacno, objasnjenje = sledeci_blok(kviz_fajl)
```

dobija se blok stringova sa sledećim pitanjem. Ako više nema pitanja, kategorija dobija prazan string i petlja se neće nastaviti.

Na kraju se zatvara kviz.txt fajl i prikazuje krajnji skor igrača.