Ravni kotari

IV.

Damir Jakšić
Ivan Mudri

RADIONICA PROGRAMIRANJA

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Sadržaj priručnika isključiva je odgovornost Udruge za ruralni razvoj Ravni kotari

Ravni kotari

Damir Jakšić & Ivan Mudri

Radionica programiranja

Sadržaj:

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Uvod

Sadržaj edukativnog modula *Radionica programiranja* jedna je od pet cjelina priručnika za AgroSTEM koji su izradili vanjski stručnjaci IT Praxisa u sklopu provedbe Ugovora o pružanju usluga organizacije radionica: Osnove programiranja, Radionica programiranja, Digitalizacija kao pokretač ruralnog razvoja, Razvoj pedagoških vještina radi popularizacije STEM-a, Korištenje brzim terenskim metodama i inovativnim alatima u poljoprivredi te izrade AgroSTEM-ova priručnika za Udrugu za ruralni razvoj Ravni kotari. Objavljen je na internetu i dostupan svima na edukativnoj mrežnoj platformi naručitelja.

Tekst pod nazivom *Radionica programiranja* nastao je na temelju provedene tri desetodnevne radionice (po četiri školska sata) za tri skupine korisnika (učenici, studenti i predstavnici OPG-ova) u razdoblju od 13. siječnja do 15. veljače 2023. (Modul I.), od 13. siječnja do 17. veljače 2023. (Modul II.) i od 23. siječnja do 17. veljače 2023. (Modul III.). Radionice su bile namijenjene krajnjim korisnicima i članovima organizacija civilnog društva (OCD-a) uključenih u projekt AgroSTEM (UP.04.2.1.10.0021) i organizirane u tri modula, svaki s po deset radionica: 1. modul za djecu osnovnoškolske dobi od petog do osmog razreda, 2. modul za studente i 3. modul za predstavnike obiteljskih poljoprivrednih gospodarstava (OPG-ova).

Modul I.

Ovaj dio priručnika nastao je na temelju deset radionica. Radionice su bile namijenjene djeci osnovnoškolske dobi od petog do osmog razreda i članovima OCD-a, a polaznici su se upoznali s osnovama programskog jezika Pyhton s naglaskom na primjenu u području AgroSTEM-a.

Python je programski jezik koji se lako uči, a može se primijeniti za različite namjene – od stvaranja igara i aplikacija do analize podataka i umjetne inteligencije. Ovaj je tekst namijenjen svima koji žele naučiti osnove programiranja u Pythonu, a posebno je prilagođen djeci školske dobi. U njemu ćete pronaći sve što vam je potrebno da počnete programirati – od osnovnih pojmova i sintakse Pythona do složenijih programa koje možete sami stvoriti. Uz ovaj priručnik uzbuđenje i kreativnost su zajamčeni!

Radionice su bile podijeljene u 10 tematskih cjelina s pomoću kojih se lako mogu usvojiti osnovni koncepti programiranja u programskom jeziku Python. Te su tematske cjeline: Uvod u Python, Ulazne i izlazne vrijedno-

sti, Uvjeti u programiranju, Grananje IF-ELSE, Grananje IF-ELIF-ELSE, Ponavljanje s uvjetom, Tipovi podataka, Sekvencijsko pretraživanje, Rekurzija, Sortiranje podataka.

Modul II.

Tekst u ovom dijelu priručnika nastao je na temelju provedene desetodnevne radionice o programiranju u Pythonu namijenjene studentima Veleučilišta „Marko Marulić" iz Knina te svim drugim zainteresiranim studentima tog područja. Svrha je radionice bila upoznavanje polaznika s osnovama programskog jezika Python te mogućnostima njegove primjene u okružju AgroSTEM-a. Zahvaljujući svojoj jednostavnosti i svestranoj primjeni, Python je postao jedan od najprihvaćenijih programskih jezika te se primjenjuje u raznim industrijama i obrazovanju.

Učenje programiranja u Pythonu smatra se jednostavnim zbog njegove jednostavne sintakse. Python omogućuje programerima različite stilove pisanja programa, uključujući strukturno, objektno orijentirano i aspektno orijentirano programiranje. Usto, Python je popularan i zbog svoje primjene u području analize podataka, računalnog vida, strojnog učenja, interneta stvari i drugih tehnologija.

Tijekom desetodnevne radionice polaznici su imali priliku upoznati se s osnovama Pythona, uključujući varijable, tipove podataka, operatore, uvjete i petlje. Upoznali su i koncepte funkcija, modula i paketa te objektno orijentiranog programiranja. Poseban je naglasak stavljen na primjenu Pythona u okružju AgroSTEM-a, uključujući uporabu Pythona u analizi podataka, radu sa senzorima i mikrokontrolerima te njegovu primjenu u poljoprivredi i agrotehnologiji.

Radionica se održavala u mrežnom formatu, uz upotrebu edukacijskih platformi za učenje programiranja kao što su Codeacademy, Coursera ili Udemy. Na kraju svakog modula polaznicima su bili dostupni mrežni testovi kako bi se provjerilo razumijevanje gradiva. Tijekom radionice polaznici su mogli sudjelovati na sastancima, postavljati pitanja i dobiti potporu u svladavanju gradiva.

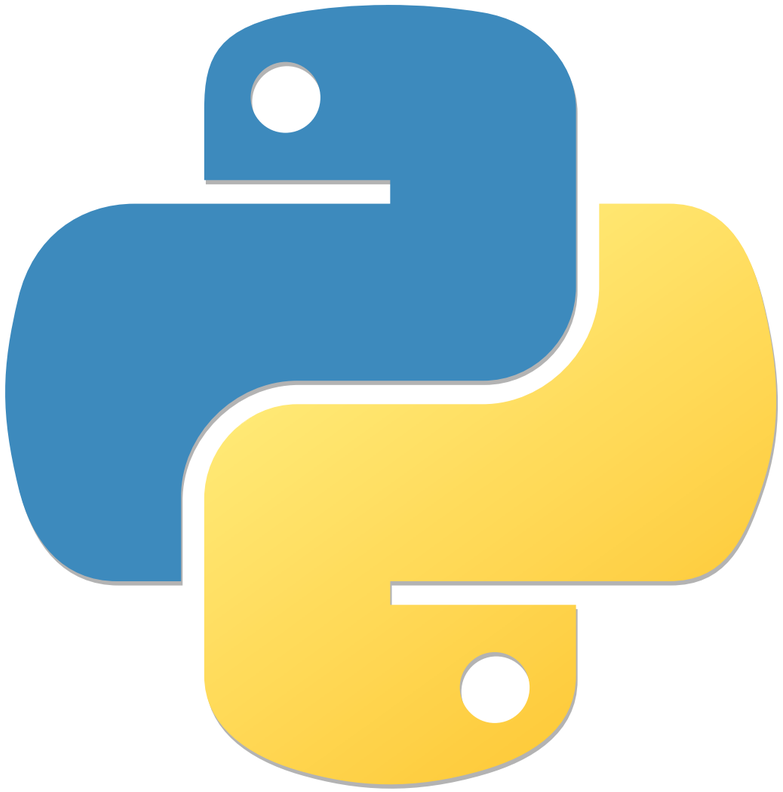
Modul III.

Tekst u ovom dijelu priručnika nastao je na temelju provedene desetodnevne radionice o programiranju u Pythonu, namijenjene članovima OPG-ova te svim drugim zainteresiranim građanima tog područja. Svrha radionice bilo je upoznavanje polaznika s osnovama programskog jezika.

Tijekom desetodnevne radionice polaznici su, jednako kao i studenti, imali priliku upoznati se s osnovama Pythona uključujući varijable, tipove podataka, operatore, uvjete i petlje. Upoznali su se s konceptima funkcija, modula i paketa te objektno orijentiranog programiranja. Naravno, i u ovom je modulu poseban naglasak stavljen na primjenu Pythona u okružju AgroSTEM-a uključujući uporabu Pythona u analizi podataka, radu sa senzorima i mikrokontrolerima te primjeni u poljoprivredi i agrotehnologiji.

Radionica se održavala također u mrežnom formatu uz primjenu edukacijskih platformi za učenje poput Google Classroma. Na kraju svakog modula polaznicima su bili dostupni mrežni testovi kako bi se provjerilo njihovo razumijevanje gradiva. Tijekom radionice polaznici su, kao i studenti, mogli sudjelovati na mrežnim sastancima na kojima su mogli postavljati pitanja i dobiti potporu u svladavanju gradiva.

Svaka nastavna tema počinje ishodima učenja koji se očekuju od polaznika i poveznicom za prezentaciju u PowerPointu i videozapis radionice, a završava pitanjima i zadatcima za provjeru ostvarenih ishoda učenja. Osnovni sadržaj obogaćen je metodičko-didaktičkim sastavnicama (izdvojenim i istaknutim tekstom – natuknicom na rubu stranice, primjerima, pitanjima i zadatcima) te dodatnim sadržajima za samostalno istraživanje i učenje (za one koji žele znati više) iz digitalnog okružja s pomoću poveznice za edukativnu platformu ili internetske stranice (tekstove, videozapise, primjere).



python™

1. Uvod u Pyhton

Nakon što ste u prvoj radionici, a potom i u ovom priručniku naučili sve što trebate znati u vezi s korištenjem programskim jezikom Python, moći ćete:

- razlikovati načine pokretanja programskog jezika Python
- prepoznati radno okružje programskog jezika Python
- koristiti se osnovnim alatima programskog sučelja.

PowerPoint prezentacija

Snimka 1. radionice

Python je programski jezik s pomoću kojega računalo razumije napisane naredbe. Stvorio ga je Guido van Rossum i prvi put objavio 1991. godine. Python **omogućuje izradu**:

Python kao programski jezik

- jednostavnih programa
- programa s grafičkim sučeljem (GUI)
- računalne igre
- integraciju na mrežne stranice…

Python se može upotrebljavati:

mogućnosti upotrebe Pythona

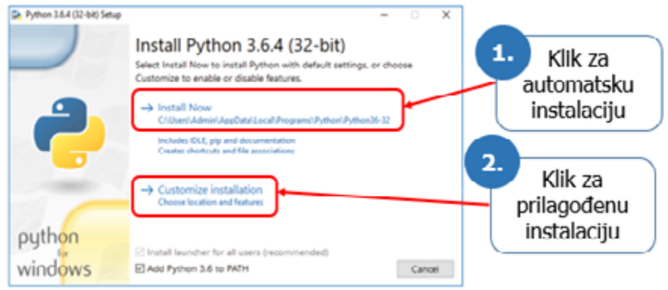
- lokalno instaliran na računalo
- alatom vizualizacije u mrežnom okružju.

Python se najlakše upotrebljava kad se instalira lokalno. Potrebno je posjetiti mrežnu stranicu https://www.python.org/downloads

koraci u lokalnom instaliranju

Nakon pokretanja instalacijske datoteke pojavljuje se dijaloški okvir čarobnjaka za instalaciju.

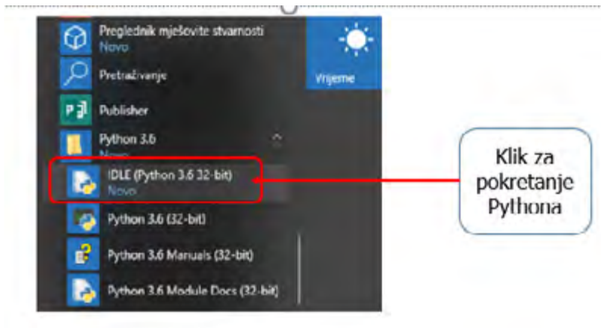
dijaloški okvir čarobnjaka za instalaciju



Nakon pokretanja instalacije klikom na **Install Now** slijedi prikaz tijeka instalacije.

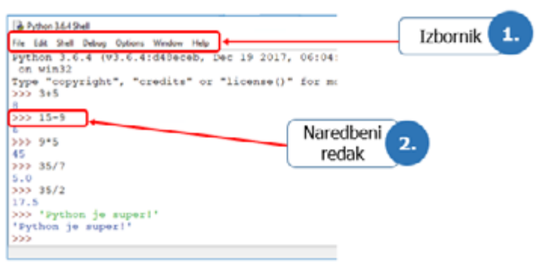
Python pokrećemo klikom na IDLE (Python GUI) iz popisa programa.

pokretanje Pythona

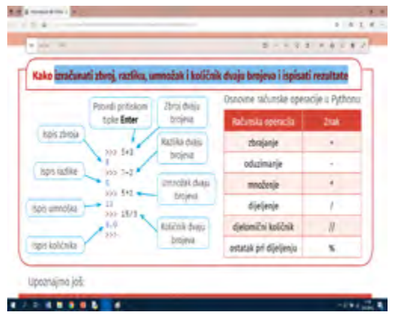


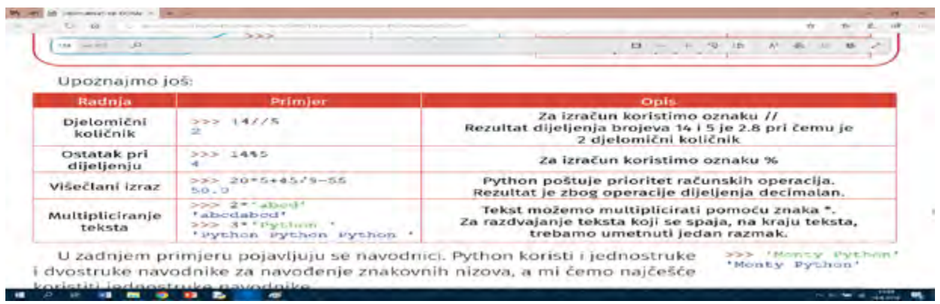
Python se može programirati u tzv. ljusci (engl. *shell*) ili konzoli u kojoj se naredbe izvršavaju redak po redak kako piše. Sve što se programira u Pythonu odmah se i vidi nakon unesene **naredbe**.

programiranje Pythona



Zadatak 1. Izračunaj zbroj, razliku, umnožak i količnik dvaju brojeva i ispiši rezultate.





Upoznajmo još:

Radnja	Primjer	Opis
Djelomični količnik	>>> 14//5 2	Za izračun koristimo oznaku // Rezultat dijeljenja brojeva 14 i 5 je 2.8 pri čemu je 2 djelomični količnik
Ostatak pri dijeljenju	>>> 14%5 4	Za izračun koristimo oznaku %
Višečlani izraz	>>> 20*5+45/9-55 50.0	Python poštuje prioritet računskih operacija. Rezultat je zbog operacije dijeljenja decimalan.

Multipliciranje teksta	>>> 2*'abcd'
	'abcdabcd'
	>>> 3*'Python '
	'Python Python Python '
		Tekst možemo multiplicirati pomoću znaka *.
		Za razdvajanje teksta koji se spaja, na kraju teksta, trebamo umetnuti jedan razmak.

U zadnjem primjeru pojavljuju se navodnici. Python koristi i jednostruke i dvostruke navodnike za navođenje znakovnih nizova, a mi ćemo najčešće koristiti jednostruke navodnike.

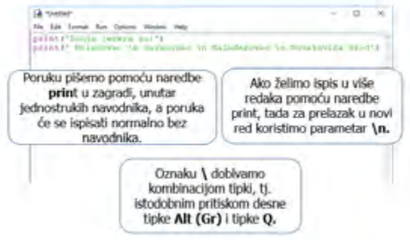
Zadatak 2. Izračunaj koliki je ostatak pri dijeljenju brojeva 234561 i 25.

```
   234561
% 25
   >>> 11
```

Zadatak 3. Napiši kod koji ponavlja riječ hop 10 puta.

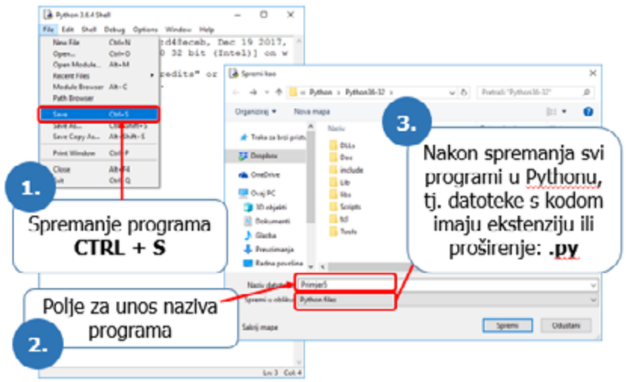
```
10*
'“hop”'
```

pisanje programa



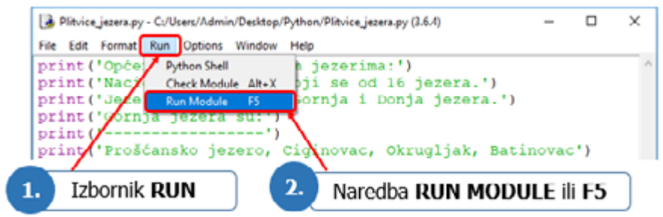
Prije pokretanja programa potrebno ga je spremiti. To se radi na sljedeći način.

spremanje programa



Nakon spremanja preostaje pokretanje programa klikom na *Run* pa *Run Module* ili F5. Pokretanje programa je postupak kojim počinje izvršavanje naredaba programskoga koda.

pokretanje programa



Programiranje u kojemu se program izvodi redoslijedom kojim je napisan, naredbu po naredbu, zove se **slijedno programiranje**. Za prikaz teksta upotrebljava se naredba print.

Zadatak 4. Izradite program koji zbraja i dijeli dva broja te na zaslonu ispisuje zbroj i ostatak pri dijeljenju dva broja.

zadatci za programiranje

```
print('Zbroj 40 + 50 = ', 40 + 50)
print('Ostatak pri dijeljenju 140 i 50 = ', 140%50)
```

Zadatak 5. Dodaj u prethodni zadatak množenje, dijeljenje i oduzimanje brojeva te djelomični količnik brojeva.

```
print ('Zbroj brojeva 50 i 40 je ', 50+40)
print ('Razlika brojeva 50 i 40 je ', 50-40)
print ('Umnožak brojeva 50 i 40 je ', 50*40)
print ('Količnik brojeva 50 i 40 je ', 50/40)
print ('Ostatak kod djeljenja brojeva 50 i 40 je', 50%40)
print ('Djelomični količnik brojeva 50 i 40 je', 50//40)
```

Zadatak 6. Izradite program koji ispisuje: „Danas je divan dan” te ispisuje današnji datum i zbraja brojeve datuma (primjer 23.4.2019. = 23+4+2019).

```
print ('Danas je divan dan')
print ('31.08.2022., a zbroj znamenaka je ', 31+8+2022)
```

Zadatak 7. Izradite program koji ispisuje raspored sati tvojeg razreda za prva tri dana (ponedjeljak, utorak i srijedu).

```
print (35*'=')
print ('    Ponedjeljak|Utorak|Srijeda \n \t HRV\t ENG\t \n \t GK\t MAT\t PID\t\n \t PID\t MAT\t GEO\t')
print (35*'=')
```

Ako želiš otvoriti prethodno snimljeni program, u izborniku File odaberi Recent Files, a zatim željeni program s popisa.

otvaranje prethodno snimljenog programa

Provjerite

1. Dan je sljedeći izraz:

82//9

Što je od navedenoga točno?

A. Oznaka // označava operaciju cjelobrojnog dijeljenja brojeva 82 i 9.

B. Oznaka // označava operaciju dijeljenja brojeva 82 i 9.

C. Oznaka // ne označava nikakvu operaciju u Pythonu.

2. Dan je sljedeći izraz u Pythonu:

23%3

Koja je od navedenih tvrdnji točna?

A. Oznaka % označava operaciju ostatka dijeljenja brojeva 23 i 3.

B. Oznaka % označava množenje brojeva 23 i 3.

Primijenite

Sljedeće računske operacije upišite u Python i odgovorite na sljedeća pitanja.

1. Što je rezultat navedenog izraza? 6 // 2

2. Što je rezultat navedenog izraza? 225 % 100

3. Što je rezultat sljedećeg izraza? 33 / –11

4. Koji će od dva matematička izraza dati veći rezultat?

A. (11 + 2) * 5 + 16

B. 11 + 2 * 5 + 16

5. Što će biti rezultat sljedećeg matematičkog izraza: 134 – 3 * (8 – 2) = ?

Provjerite

3. Kako biste u Pythonu zapisali sljedeći matematički izraz:
$a^2 + b^2 = c^2$

A. a*2 + b*2 = c*2

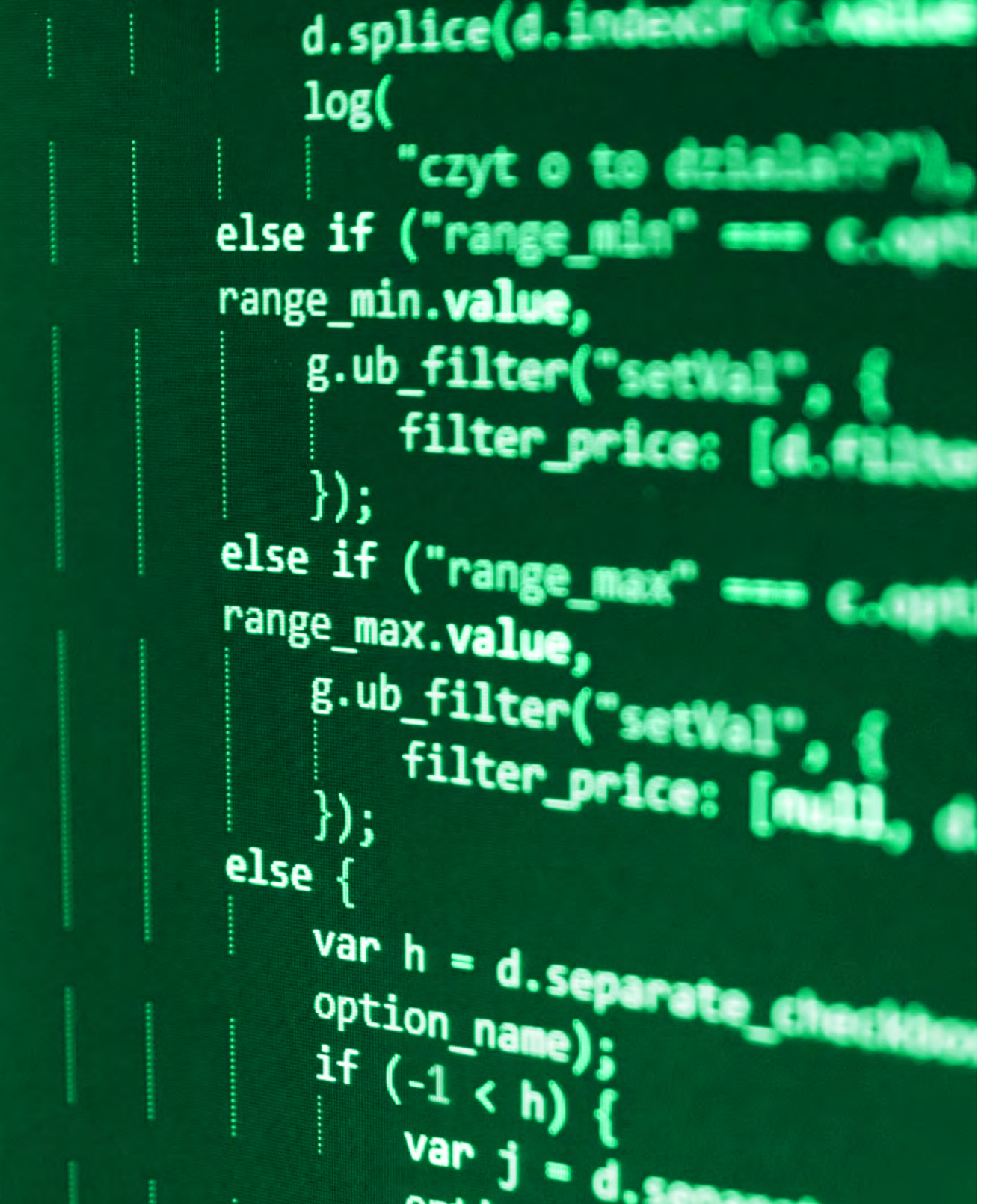
B. a**2 + b**2 = c**2

4. Kako biste u Pythonu zapisali matematički izraz:
18 : (–15) + (–37) * 6?

A. 18 / -15 + -37 * 6

B. 18 // - 15 + -37 * 6

C. 18 / (-15) + (-37) * 6



2. Ulazne i izlazne vrijednosti programa

Nakon što ste u drugoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o ulaznim i izlaznim programima, moći ćete:

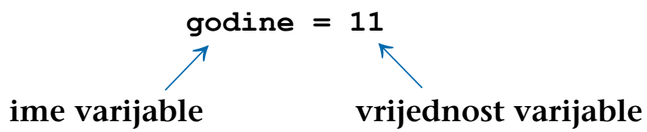
- izraditi program koji traži unos ulaznih podataka te ispisuje izlazne podatke
- primjenjivati varijable u programima
- razlikovati naziv varijable i njezinu vrijednost

PowerPoint prezentacija

Snimka 2. radionice

Varijabla označava nešto promjenjivo. Ona je dio memorije u koju se pohranjuje neki izmjenjivi podatak.

varijabla



Svake godine vrijednost će se promijeniti, stoga su godine varijabla, odnosno veličina koja poprima različite vrijednosti.

Svaki program radi s pomoću ulaznih i izlaznih vrijednosti.

vrijednosti varijable

ulazne vrijednosti	To je ono što korisnik unosi u računalo (tipkovnicom, mišem, prstom ili drugom ulaznom jedinicom).
izlazne vrijednosti	Rezultat su rada nekog programa (pomak pokazivača na ekranu, kretanje lika u računalnoj igri, prikaz slike na zaslonu itd.).

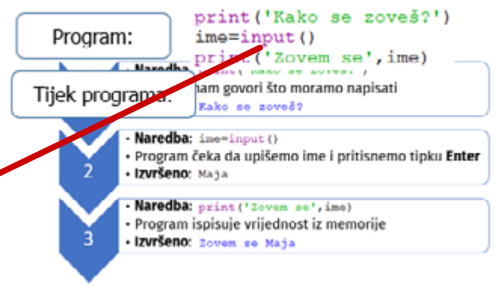
Rad svakog programa mogao bi se predočiti u **tri** osnovna koraka:

osnovni koraci programiranja

ulaz	To su ulazne vrijednosti (zadaju se naredbom Pridruživanje).
obrada	Pridruživanje je ulaznih vrijednosti, a rezultat toga su izlazne vrijednosti.
izlaz	Izlazne su vrijednosti nastale kao rezultat obrade tih podataka (naredba Print).

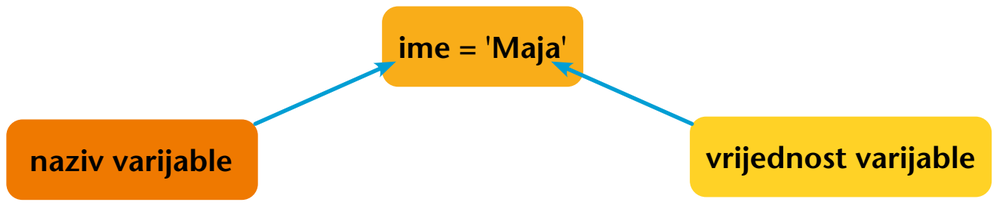
primjer

Zadatak 1. Izradite program koji unosi i ispisuje imena.



Za unos ulaznog podatka imena upotrijebljena je samo naredba **Input**. Naredbom pridruživanja ime=input() omogućeno je upisivanje imena čija se vrijednost sprema u memoriju ime. U memoriju ime mogu se upisati različita imena i različite vrijednosti, pa je ime promjenjiva veličina, tj. **varijabla**.

Varijabla je lokacija u memoriji koja ima simboličan naziv, a može joj se pridružiti neka vrijednost (broj, tekst, simboli). Ime varijable je nepromjenjivo, a vrijednost se može u bilo kojem trenutku promijeniti u neku drugu vrijednost. Svakim unošenjem nove vrijednosti u varijablu briše se stara vrijednost. Varijabla je jednoznačno određena svojim nazivom i vrijednošću kao na sljedećoj slici.



Broj varijabli u stvaranju programa je **neograničen**, ali svaka varijabla mora imati drukčiji naziv. Za naziv varijable ne smiju se upotrebljavati naredbe i funkcije Pythona (npr. Print, Input, int). Valja znati da su *ime* i *Ime* različite varijable jer Python razlikuje velika i mala slova.

primjer

Zadatak 2. Zadatak je izraditi tri varijable i program koji zbraja dva broja i ispisuje rezultat na zaslonu.

Program:

```
a=input('Upiši prvi broj:')
b=input('Upiši drugi broj:')
zbroj=a+b
print('Zbroj je:',zbroj)
```

Rezultat:

```
Upiši prvi broj:14
Upiši drugi broj:24
Zbroj je: 1424
```

- Program je umjesto zbroja izvršio spajanje brojeva.
- Python podatke doživljava kao riječi, a ne kao brojeve.

Ulazne podatke treba pretvoriti u brojeve, a potom ih zbrojiti. To se radi s pomoću funkcije `int`. Umjesto a+b, pišemo int(a)+int(b):

obrada

```
a=input('Upiši prvi broj:')
b=input('Upiši drugi broj:')
zbroj=int(a)+int(b)
print('Zbroj je:', zbroj)
```

Naredba **Input** sada se može zapisati i ovako: a=int(Input(„Upiši prvi broj”)). Takav način zapisa zove se ugniježđena naredba. **Ugniježđena naredba** znači naredba u naredbi. Pri njezinu provođenju najprije će se provesti unutarnja, a onda vanjska naredba.

odrednice ugniježđene naredbe

```
a=int(input('Upiši prvi broj:'))
b=int(input('Upiši drugi broj:'))
zbroj=a+b
print('Zbroj',a,'+',b,'=',zbroj)
```

Petlja

Petlja je dio programa koji se ponavlja. Postoje **dvije vrste** petlji:

petlja i vrste petlji

- petlje bez logičkog uvjeta
- petlje s logičkim uvjetom.

Petlje bez logičkog uvjeta su vrsta petlje u kojoj se unaprijed postavlja broj ponavljanja. One imaju **konačan broj ponavljanja** te uvijek počinju određenim naredbama kojima je definiran broj ponavljanja. U Pythonu se za petlju bez logičkog uvjeta upotrebljava naredba **For**.

petlje bez logičkog uvjeta

primjer

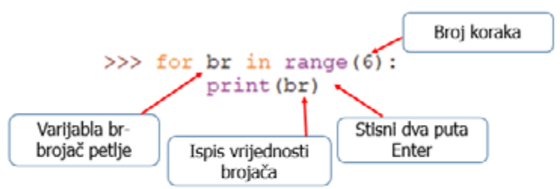
Zadatak 3. Ispiši brojeve od 0 do 10.

primjer petlje bez logičkog uvjeta

Program

```
print(0)
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

Kraći program



- Nakon dvotočke prelaskom u novi red sljedeći redak se uvukao.
- Da bi se u konzoli izvršila gornja naredba, nakon drugog retka treba dva puta stisnuti tipku Enter.

Da bi petlja za koju se upotrebljava naredba For u svakom trenutku znala na kojem je koraku ponavljanja, pomaže joj **kontrolna varijabla** petlje ili **brojač**. Npr. br in range (6). Varijabla **br** je brojač petlje. Brojač se **upotrebljava tako** da se postavi **početna vrijednost** i **broj koraka** izvođenja petlje. Broj ponavljanja može se zadavati izravnim upisivanjem broja ponavljanja u naredbi petlje ili zadavanjem s pomoću varijable.

Program može sadržavati onoliko petlji koliko je potrebno uz napomenu da pri radu s petljama treba paziti kako se ne bi stvorila **beskonačna petlja – program sam sebe pokreće i ne završava**. Beskonačna petlja je korisna u nekim sustavima koji svoj posao obavljaju bez stajanja.

Treba uvijek paziti na to da u većini programskih jezika, pa tako i u Pythonu, petlja for počinje od broja 0.

Zadatak 4. Ispiši brojeve od 0 do 10.

```
for br in range(11):
    print(br*2)
```

Broj ponavljanja u petlji može se dati izravno ili zadavanjem varijable unesene u program (tipkovnicom ili iz samog programa).

Zadatak 5. Izradite program koji računa zbroj prvih n brojeva.

```
print('Koliko brojeva želiš zbrojiti?')
n=int(input('Upiši broj:'))
zbroj=0
for br in range(1, n+1):
    zbroj=zbroj+br
print('Zbroj prvih',n,'prirodnih brojeva je:', zbroj)
```

Ako na memorijskoj lokaciji zbroj možda već postoji neka vrijednost, program bi tu vrijednost zbrojio s vrijednošću varijable br pa upisivanjem:

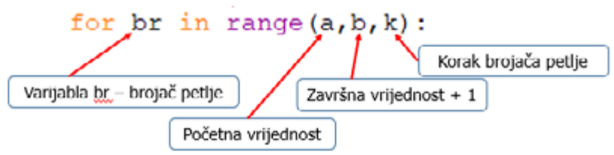
zbroj = 0

osiguravamo da se to ne dogodi.

U rasponu **range(1, n+1)** završna vrijednost n treba biti povećana za 1 kako bi se dobio točan raspon jer Python uvijek završnu vrijednost smanji za 1.

Zadatak 6. Izradite program koji ispisuje prvih n neparnih brojeva.

```
n=int(input('Upiši broj n:'))
for br in range (1,2*n,2):
    print(br)
```



n=3	n=5	n=10
1,3,5	1,3,5,7,9	1,3,5,7,9,11,13,15,17,19

Zadatak 7. Izradite program koji ispisuje prvih n parnih brojeva.

```
n=int(input('Upiši broj n:'))
for br in range (0,2*n,2):
    print(br)
```

Zadatak 8. Izradite program koji ispisuje brojeve od 9 do 0.

```
for br in range (9,0,-1):
    print(br)
```

Zadatak 9. Izradite program u kojemu se korisniku postavlja pitanje koliki je broj ponavljanja i nakon toga program ponovi poruku „HOP” toliko puta.

```
n=int(input('Koliko puta želiš ponoviti "HOP"?'))
for br in range(0,n):
    print('HOP')
```

Provjerite

1. Koliko ulaznih vrijednosti najmanje mora biti?

A. nijedna
B. jedna
C. dvije

2. Koja vrijednost nastaje obradom ulaznih podataka?

A. ulazna
B. izlazna

3. Koja se od navedenih naredaba u Pythonu upotrebljava za unos podataka?

A. sum
B. input
C. print

Primijenite

1. Izradite program koji vas pita koji su vam najdraži školski predmeti i ispisuje ih jedan ispod drugoga.

2. Izradite program s pomoću kojega ćete pretvarati sate u sekunde (unosite sate).

3. Izradite program koji će pretvarati kilometre u metre (unosite kilometre).

4. Izradite program za računanje površine kružnice (unosite polumjer r).

Provjerite

4. Varijabla je vrijednost koju ne možete promijeniti.

A. točno

B. netočno

5. Petlja je dio programa koji se _____ .

A. prepravlja

B. ponavlja

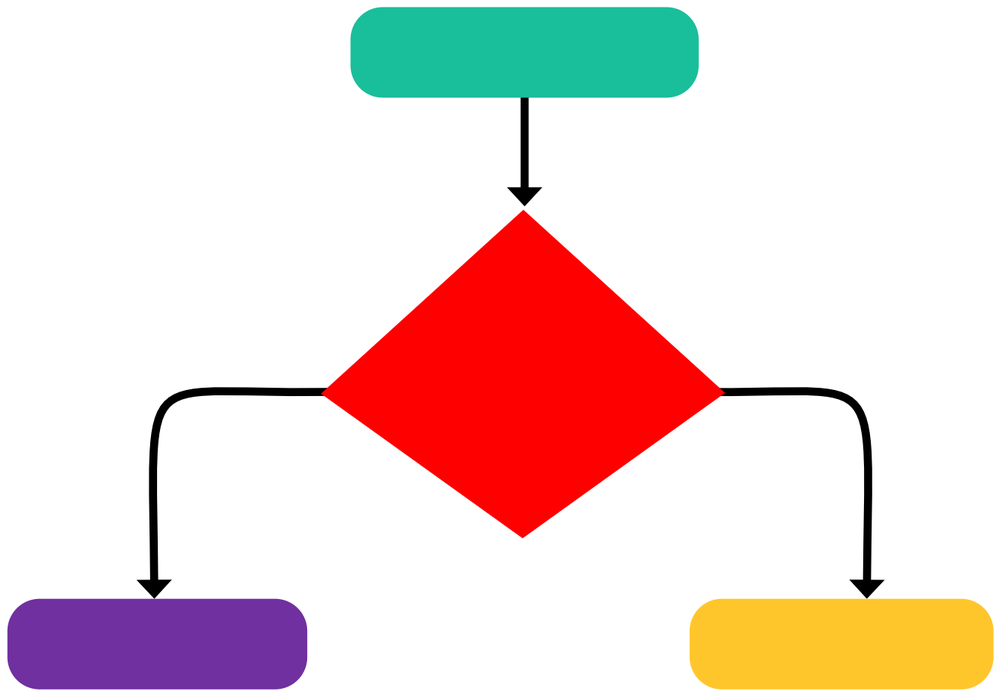
C. pretvara

6. U Pythonu se za petlju bez logičkog uvjeta koristimo naredbom For.

A. točno

B. netočno

Za one koji žele saznati više



3. Uvjet u programiranju

Nakon što ste u trećoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o uvjetu u programiranju, moći ćete:

- samostalno ili uz pomoć učitelja/učiteljice opisati i analizirati zadani problem, planirati rješavanje problema te predložiti i prepoznati algoritamsko rješenje
- prepoznati ulazne i izlazne vrijednosti
- prikazati rješenje problema dijagramom tijeka, riječima govornog jezika i naredbama programskog jezika
- primjenjivati algoritamsku strukturu grananja u rješavanju problema
- stvarati, pratiti i preurediti programe koji sadržavaju strukture grananja.

PowerPoint prezentacija

Snimka 3. radionice

Naredba If

naredba If

If je složena naredba koja ovisi o logičkom uvjetu koji se postavlja. **Logički uvjet** je pitanje koje se postavlja primjenom operatora usporedbe i provjerava se istinitost izraza. Ako je logički uvjet **istinit**, provodi se niz naredaba, koji se postavlja nakon uvjeta. Ako **nije istinit**, zaobilazi se grananje i nastavlja se izvođenje programa.

U Pythonu a=0 znači „varijabli a pridruži vrijednost nula”, a a==0 znači „usporedi vrijednost varijable s nulom”.

Operator	Značenje
=	jednako
≠	različito (nije jednako)
<	manje
≤	manje ili jednako
>	veće
≥	veće ili jednako

Dva se uvjeta mogu provjeriti istodobno s pomoću jednog bloka **naredbe If** tako da se u logički uvjet postavi operator **and** (I) ili **or** (ILI).

operator and	provjerava dva uvjeta istodobno kad su oba ispunjena, provodi naredbe unutar uvjeta
operator or	provjerava dva uvjeta istodobno i, kad je barem jedan ispunjen, provodi naredbe unutar uvjeta

primjeri upotrebe naredbe If

primjer

Zadatak 1. Izradite program koji provjerava je li broj pozitivan.

```
a=int(input('Upiši broj: '))
if a>0:
    print('Broj je pozitivan')
```

Zadatak 2. Izradite program koji provjerava je li broj veći od 5.

```
a=int(input('Upiši broj: '))
if a>5:
    print('Broj je  veći od 5')
```

Zadatak 3. Izradite program koji ispisuje da je broj dvoznamenkast ako je a veći od 9 i ako je a manji od 100.

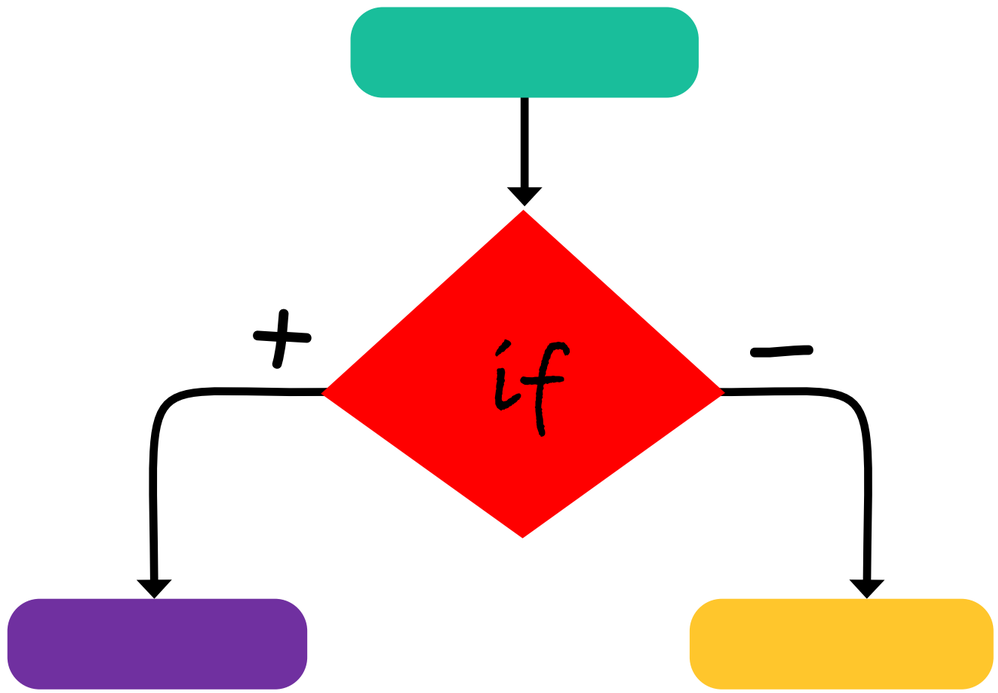
```
a=int(input('Upiši broj: '))
if a>9 and a<100:
    print('Broj je dvoznamenkast')
```

Zadatak 4. Izradite program koji će tražiti unos dvaju brojeva (a i b) i, ako je uvjet ispunjen, ispisati da je prvi broj jednak drugom.

```
a=int(input('Unesi prvi broj:'))
b=int (input('Unesi drugi broj:'))
if a=b:
    print('Prvi broj je jednak drugom broju'
```

Zadatak 5. Izradite program koji će tražiti unos dvaju brojeva i, ako je uvjet ispunjen, ispisati da je prvi broj veći od drugoga ili jednak njemu.

```
a=int(input('Upiši broj: '))
b=int(input('Upiši broj: '))
if a>=b:
    print ('Prvi broj je veći ili jednak od drugog')
```



4. Grananje If-else

Nakon što ste u četvrtoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o grananju If-else, moći ćete:

- primijeniti naredbu If-else u rješavanju problema
- samostalno ili uz pomoć učitelja/učiteljice opisati i analizirati zadani problem, planirati rješavanje problema te predložiti i prepoznati algoritamsko rješenje
- prikazati algoritamsko rješenje riječima govornog jezika, dijagramom tijeka ili programskim jezikom
- predviđati ponašanje algoritma i provjeravati ispravnost algoritma
- stvarati, pratiti i preurediti programe koji sadržavaju strukture grananja

PowerPoint prezentacija

Snimka 4. radionice

naredba If-else

Logička naredba grananja **If-else** složena je naredba koja služi za donošenje odluke kada postoje **dva ili više ishoda**, ovisno o odgovoru na pitanje postavljeno u uvjetu.

Za razliku od naredbe If, ako uvjet nije ispunjen, dio skripta unutar odluke neće se zaobići, nego će se provesti neki drugi niz naredaba.


```
if uvjet:
        naredba1
        naredba2
        ...
else:
    naredba1
    naredba2
    ...
```

primjeri upotrebe naredbe If-else

primjer

Zadatak 1. Izradite program koji provjerava je li korisnik dobro zbrojio brojeve.

```
odg=int(input('Koliko je 9+11?'))
if odg==20:
    print('Bravo! Točan odgovor!')
else:
    print('Žao mi je, netočan odgovor!')
```

Taj primjer detaljno pokazuje korisnost naredbe If-else. Korak po korak u dijagramu tijeka može se vidjeti kako svaki red naredbe izvodi i provjerava uvjete postavljene u naredbi**.** Umjesto dviju naredaba If, upotrebljava se jedna naredba If-else.

Je li moguće postaviti više naredaba If-else te na taj način riješiti više slučajeva?

zadatci za uvježbavanje upotrebe naredbe If-else

Zadatak 2. Izradite program koji od korisnika zahtijeva da pritisne tipku A za nastavak. Ako korisnik klikne na tipku A, računalo će ispisati tekst: „Bravo! Pritisnuo si tipku A.” Ako pak klikne na neku drugu tipku ili učini bilo što drugo, računalo će ispisati tekst: „Šteta, pogriješio si! Nisi pritisnuo tipku A.”

```
slovo=str(input('Pritisni tipku a:'))
if slovo=='a':
    print ('Bravo! Pritisnuo si tipku a.')
else:
    print ('Šteta, pogriješio si. Nisi pritisnuo tipku a.')
```

Zadatak 3. Izradite program koji unosi broj te provjerava je li broj paran.

```
a=int(input('Upiši broj:'))
ostatak=a%2
if ostatak==0:
    print(a, 'je prani broj')
else:
    print(a, 'je neparni broj')
```

Zadatak 4. Izradite program koji unosi dva broja te provjerava njihovu djeljivost.

```
a=int(input('Upiši broj a:'))
b=int(input('Upiši broj b:'))
ostatak=a%b
if ostatak==0:
    print(a, 'je djeljiv s' ,b)
else:
    print(a, 'nije djeljiv s', b)
```

Zadatak 5. Napišite program koji unosi broj te provjerava je li djeljiv sa sedam.

```
a=int(input('Unesi broj'))
if a%7==0:
    print('Broj',a, 'je djeljiv sa sedam.')
else:
    print('Broj',a,'nije djeljiv sa sedam')
```

Zadatak 6. Izradite program koji unosi broj te provjerava je li broj pozitivan.

```
num=float(input('Unesite broj'))
if num > 0:
    print('Broj', num, 'je pozitivan')
else:
    print('Broj', num, 'je negativan.')
```

Zadatak 7. Izradite program koji unosi dva broja, zbraja ih te provjerava je li zbroj veći od 20.

```
a=int(input('Unesite 1. broj:'))
b=int(input('Unesite 2. broj:'))
zbroj=a+b
if zbroj > 20:
    print('Zbroj brojeva',a,'i',b,'je',zbroj,'i veći je od 20.')
else:
    print('Zbroj brojeva',a,'i',b,'je',zbroj,'i manji je od 20.')
```

Zadatak 8. Izradite program koji unosi duljine dviju stranica. Ako su unesene duljine jednake, izračunat će površinu kvadrata, ako su različite izračunat će površinu pravokutnika, rezultat je sljedeći.

```
a=int(input('Unesite stranicu a:'))
b=int(input('Unesite stranicu b:'))
p=0
if a==b:
    p=a*b
    print('Površina kvadrata iznosi:',p)
else:
    p=a*b
    print('Površina pravokutnika iznosi:',p)
```

Uporabom naredbe If-else može se kombinirati nekoliko uvjeta. Provode se samo naredbe koje slijede prvi uvjet za koji se ustanovi istinitost, a sve se druge preskaču. Naredbe nakon konačne If-else provode se ako nijedan od uvjeta nije istinit.

prednosti uporabe naredbe If-else

Provjerite

1. Što označava znak „ = “ u Pythonu?
   A. jednakost
   B. pridruživanje
   C. uspoređivanje
2. Što označava znak „ == “ u Pythonu?
   A. jednakost
   B. pridruživanje
   C. uspoređivanje

Primijenite

1. Izradite program koji će od korisnika tražiti upis broja 5 i provjeriti je li upisan traženi broj.
2. Izradite program u koji će se učitati dva broja te se uvijek oduzimati veći od manjeg broja.
3. Izradite program u koji će se unositi riječ. Program će provjeravati nalazi li se samoglasnik a u riječi te, ako ga nađe, ispisati da se samoglasnik nalazi u napisanoj riječi. U suprotnom će ispisati da nema samoglasnika a.

Provjerite

3. Koristeći se operatorima **or** i **and** u naredbi **If** možemo provjeravati više uvjeta.

A. točno

B. netočno

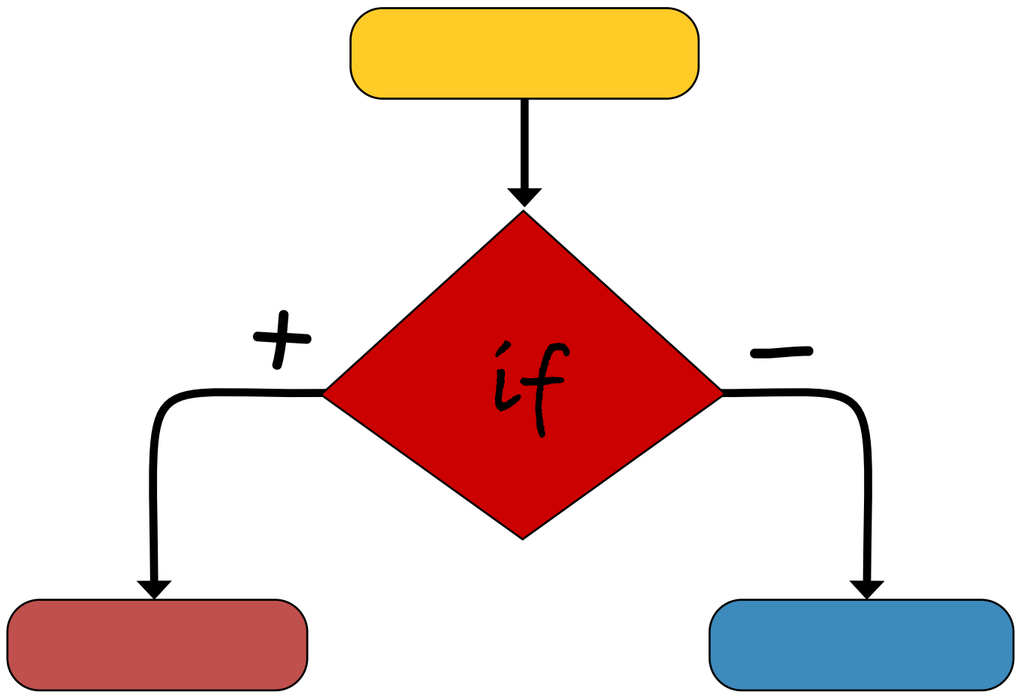
4. Koliko uvjeta treba postojati da bismo se služili naredbom If-else?

A. nijedan

B. jedan

C. dva ili više

Za one koji žele saznati više



5. Naredba Elif

Nakon što ste u petoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o naredbi Elif, moći ćete:

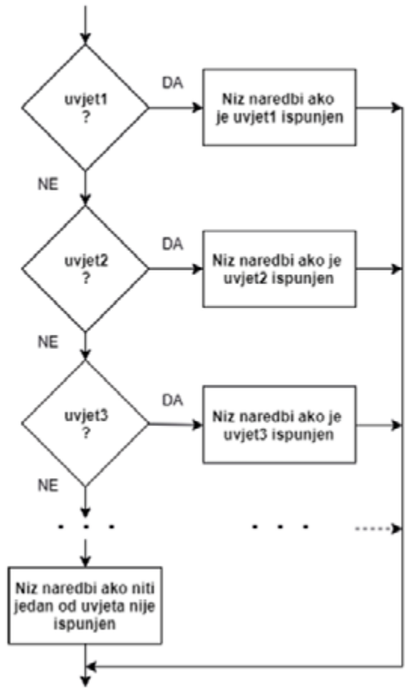
- prepoznati ulazne i izlazne vrijednosti
- primijeniti naredbu If-then-else u rješavanju problema
- samostalno ili uz pomoć učitelja/učiteljice opisati i analizirati zadani problem, planirati rješavanje problema te predložiti i prepoznati algoritamsko rješenje
- prikazati algoritamsko rješenje riječima govornog jezika, dijagramom tijeka ili programskim jezikom
- predviđati ponašanje algoritma
- provjeravati ispravnost algoritma
- stvarati programe koji sadržavaju strukture grananja
- pratiti programe koji sadržavaju strukture grananja
- preurediti programe koji sadržavaju strukture grananja

PowerPoint prezentacija

Snimka 5. radionice

Ako treba provjeriti samo jedan uvjet, naredba If-else bit će sasvim dovoljna. Ako treba provjeriti više uvjeta, pri čemu svaki od njih može imati drukčiji ishod, to se može učiniti s pomoću više naredaba If ili s pomoću naredbe Elif. Naziv naredbe Elif kratica je od Else-if (inače-ako).

odrednice naredbe Elif



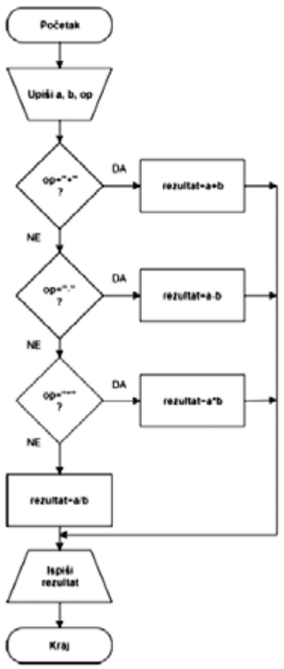

```
if uvjet1:
     naredba1
     naredba2
     ...
elif uvjet2:
     naredba1
     naredba2
     ...
elif uvjet3:
     naredba1
     naredba2
     ...
     ...
else:
     naredba1
     naredba2
     ...
```

Na primjeru izrade jednostavnog kalkulatora može se uvidjeti osnovna primjena naredaba If, Elif i Else. Nakon unesenih brojeva i računske operacije utvrđuje se koja je operacija upisana te se na temelju odgovora provodi pripadajuća računska operacija nad upisanim brojevima i ispisuje rezultat.

primjer

Zadatak 1. Izradite program u kojemu će korisnik unijeti dva broja i birati operaciju koju će obaviti nad ta dva broja

primjer upotrebe naredaba If, Elif i Else




```
a=int(input('Upiši prvi broj:'))
b=int(input('Upiši drugi broj:'))
op=input('Upiši operaciju (+,-,*,/):')
if op=="+":
    rezultat=a+b
elif op=="-":
    rezultat=a-b
elif op=="*":
    rezultat=a*b
else:
    rezultat=a/b
print('Rezultat je:',rezultat)
```

```
Upiši prvi broj:5
Upiši drugi broj:9
Upiši operaciju (+,-,*,/):*
Rezultat je: 45
>>>
```

1. Postaviti pitanje i unos odgovora:

```
a=int(input('Upiši prvi broj:'))
b=int(input('Upiši drugi broj:'))
op=input('Upiši operaciju (+,-,*,/):')
```

2. Postaviti uvjet: `if op=="+":`

   Ako je uvjet ispunjen, izračunaj: `rezultat=a+b`

3. Ako uvjet nije ispunjen, provjeriti sljedeći uvjet:
   Ako je uvjet ispunjen, izračunaj

```
rezultat=a-b
```

4. Ako ni taj uvjet nije ispunjen, provjeriti sljedeći uvjet

```
elif op=="*":
```

   Ako je uvjet ispunjen, izračunaj: `rezultat=a*b`

5. Inače izračunaj

```
else:
    rezultat=a/b
```

6. Ispiši rezultat: `print('Rezultat je:',rezultat)`

Prethodni primjer detaljno pokazuje korisnost naredaba If, Elif i Else . Korak po korak dijagramom tijeka može se vidjeti kako svaki red naredbe izvodi i provjerava uvjete postavljene u naredbi. Umjesto četiriju naredaba If, koristi se naredbama If, Elif i Else.

Je li moguće postaviti više naredaba If, Elif i Else te na taj način riješiti više slučajeva?

primjer

Zadatak 2. Izradite program koji unosi dva broja, provjerava koji je broj veći te ispisuje je li veći broj a ili b ili su jednaki.

```
a = int(input("Upišite broj a: "))
b = int(input("Upišite broj b: "))

if (a<b):
    print("broj b je veći")
clif (a==b):
    print("brojevi su jednaki")
else:
     print("broj a je veći")
```

Zadatak 3. Izradite program koji unosi broj i provjerava pozitivnost/negativnost broja.

```
a=int(input("Unesi broj:"))
if a>0:
    print(a,"je pozitivan.")
elif a<0:
    print(a,"je negativan.")
else:
    print("Upisani broj je 0.")
```

Zadatak 4. Izradite program koji postavlja pitanje te od korisnika traži upis rješenja. Pitanje mora imati dva točna rješenja i jedno pogrešno. Ako se odabere točno rješenje, program ispisuje: „Točan odgovor." Ako se odabere pogrešan odgovor, program ispisuje: „Odgovor je pogrešan."

```
print ("a)4")
print ("b)8")
print ("c)6-2")
odgovor = str(input("Koliko je 2+2 ?: "))

if (odgovor=='a'):
    print("Točan odgovor")
elif (odgovor=='c'):
    print("Točan odgovor")
else:
     print("Odgovor je kriv")
```

Zadatak 5. Izradite program u koji ćete unijeti stranice trokuta. Program provjerava vrstu trokuta (jednakostraničan, jednakokračan, raznostraničan).

```
a = int(input('a: '))
b = int(input('b: '))
c = int(input('c: '))
if a == b and a==c:
    print ('Jednakostranican')
elif a == b or a==c or b==c:
    print ('Jednakokracan')
else:
    print('Raznostranican')
```

Zadatak 6. Izradite program koji će provjeravati djeljivost unesenog broja s 2 i 3.

```
a = int(input("Unesi broj: "))
if a % 2 == 0 and a % 3 == 0:
    print ("Broj je deljiv sa 2 i 3")
elif a % 2 == 0:
    print ("Broj je deljiv samo sa 2")
elif a % 3 == 0:
    print ("Broj je deljiv samo sa 3")
else:
    print ("Broj nije deljiv ni sa 2, ni sa 3")
```

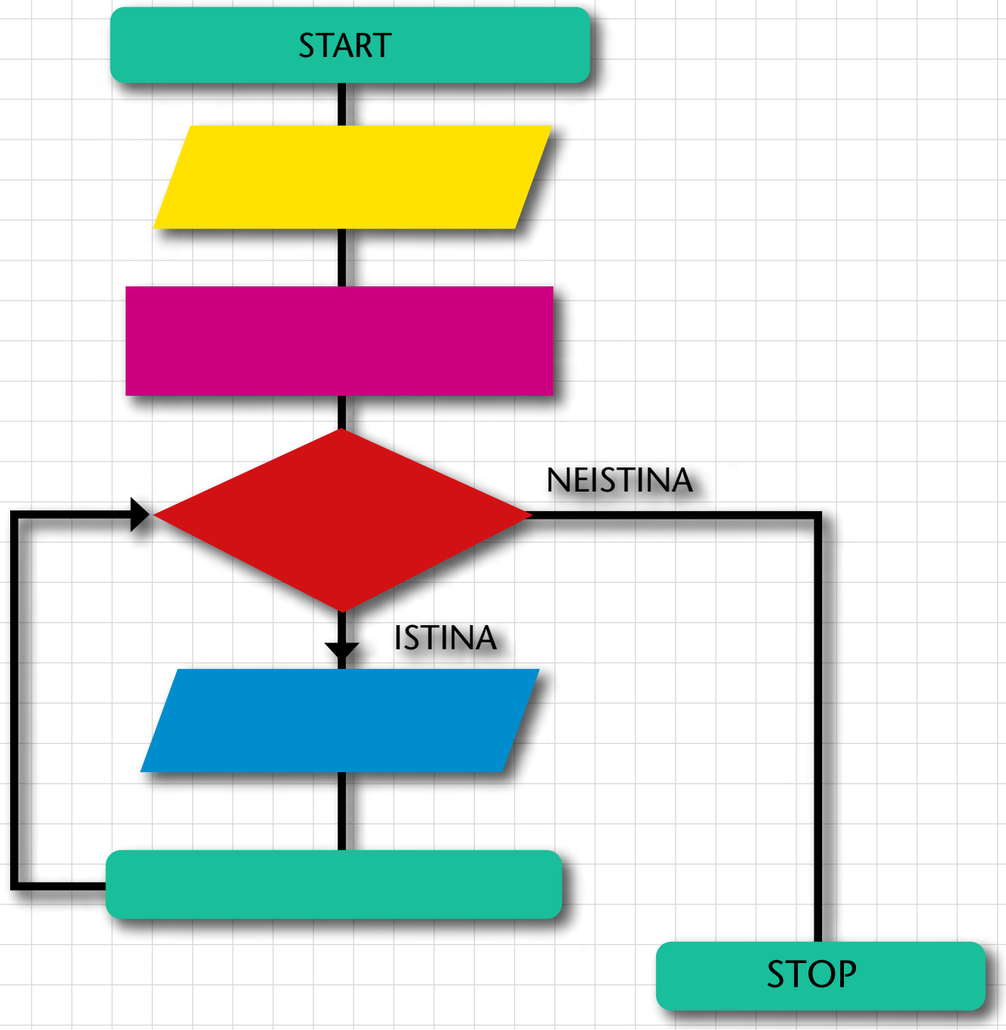
Zadatak 7. Izradite program koji će omogućiti korisniku da odabere jednu od četiri aritmetičke operacije (+, –,*,/). Nakon što korisnik odabere jednu od operacija, moći će unijeti dva broja nad kojima će se provesti operacija. Nakon toga ispisat će se rezultat operacije.

```
n = int(input('Unesi prvi broj:'))
m = int(input('Unesi drugi broj'))
o = input ('Unesi operaciju')
if o == '+':
    print(n+m)
elif o == '-':
    print(n-m)
elif o == '*':
    print(n*m)
elif o == '/':
    print(n/m)
```

Primijenite

1. Izradite program koji će od korisnika tražiti unos broja u intervalu od 1 do 12. Kad korisnik odabere broj, program treba riječima ispisati naziv mjeseca koji odgovara broju unutar intervala 1 – 12. Na primjer, nakon unosa broja 10 ispisat će se naziv mjeseca listopada.

2. Izradite program koji će korisniku omogućiti unos ocjena od 1 do 5. Ako korisnik odabere prolaznu ocjenu (2, 3, 4, 5), ispisat će se „Ocjena ispita je (ocjena riječima), prošli ste.” Ako je ocjena negativna, ispisat će se: „Ocjena ispita je nedovoljan, niste prošli.” Ako korisnik odabere broj koji nije u rasponu ocjena, ispisat će se: „Nepostojeća ocjena.”
3. Izradite program u koji ćete upisati dva različita cijela broja. Ako su oba parna, potrebno je podijeliti veći broj s manjim. Ako su oba neparna, potrebno je od većeg broja oduzeti manji. Inače je potrebno zbrojiti ta dva broja.



6. Ponavljanje s uvjetom

Nakon što ste u šestoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o ponavljanju s uvjetom, moći ćete:

- razlikovati petlje s uvjetom i bez uvjeta
- izraditi skripta koja se koriste petljom s uvjetom

PowerPoint prezentacija

Snimka 6. radionice

Petlja While

U petlji For, bez logičkog uvjeta, poznat je broj ponavljanja, zna se koliko će puta program ponoviti određeni dio programa.

U naredbi petlje While broj ponavljanja ovisi o postavljenom uvjetu te će se provoditi dio programa sve dok uvjet bude zadovoljen. Onog trenutka kad uvjet ne bude zadovoljen, program će izaći iz petlje i nastaviti izvoditi sljedeće blokove naredaba.

odrednice naredaba petlje While

primjer

Prikazana je svakidašnja aktivnost jedenja s pomoću blok-dijagrama te pokazuje mjesto odluke u kojemu se program, kad je odgovor niječan, vraća na pitanje sve dok odgovor ne postane potvrdan.

primjer blok-dijagrama

To je slučaj kod petlja za koju se broj ponavljanja ne definira na početku jer broj ponavljanja ovisi o uvjetu u samoj petlji. Za ponavljanje s uvjetom upotrebljava se naredba ponavljaj dok je (*while*).

algoritam naredbe While


```
while uvjet:
    naredba1
    naredba2
    ...
```

Važno je znati da program svaki put kad prođe kroz petlju provjeri je li uvjet zadovoljen.

primjer

Zadatak 1. Izradite program za bacanje kockice koja ima šest strana. Nakon što baciš kockicu, program te pita želiš li baciti još jednu. Program se ponavlja sve dok odgovor ne bude niječan.

primjeri petlje s logičkim uvjetom


```
File  Edit  Format  Run  Options  Window  Help
import random
baci = 'd'
while baci == "d":
    print('Bacam kockicu...')
    print('Dobiveni broj je...')
    broj=random.randint(1, 6)
    print (broj)
    baci = input('Baci kockicu d/n ? ')
Bacam kockicu...
Dobiveni broj je...
6
Baci kockicu d/n ? n
>>>
```

Zadatak 2. Izradite program koji će ispisati riječ škola pet puta i broj ispisa ispred riječi škola koristeći se naredbom While.

```
i=0
while i < 5:
    print(i+1,' škola')
    i +=1
```

Zadatak 3. Izradite program u kojemu korisnik unosi brojeve sve dok ne unese 0. Odmah ispisujte samo dvoznamenkaste brojeve.

```
while True:
    broj = int(input('Upisi broj: '))
    if broj > 9 and broj < 100:
        print (broj)
    elif broj == 0:
        break
```

Zadatak 4. Izradite program koji traži upis broja u rasponu između 10 i 20. Ako upišete brojeve u tom rasponu, ispisat će se: „Broj je pogođen, bravo!" U suprotnom će program ispisati: „Broj nije u rasponu."

```
while True:
    broj = int (input ("Unesi broj: "))
    if broj >= 10 and broj <=20:
        print ("Broj je pogodjen, bravo")
    elif broj==0:
        break
    else:
        print ("Broj nije u rasponu")
```

Zadatak 5. Izradite program koji unosi dva broja, provjerava koji je broj veći te ispisuje je li veći broj a ili b ili su jednaki.

```
password = ""
while password != "tajno":
    password = input("Unesite lozinku: ")
    if password == "tajno":
        print("Upisali ste  tocnu lozinku")
    else:
        print("upisali ste pogresnu lozinku-pokusajte ponovno")
```

Zadatak 6. Izradite program u kojemu će korisnik pokušati pogoditi o kojem je programu riječ. Ako pogodi, ispisat će se: „Broj je pogođen, bravo!” Ako broj bude veći ili manji, ispisat će se: „Broj je veći/manji od zadanog, pokušaj ponovno!”

```
print ("Igra pogadjanja brojeva")
print ("Za prekid igre unesi broj 0")

# zadani broj je x

x = 17

while 1:
    broj = int (input ("Unesi broj: "))
    if broj == x:
        print ("Broj je pogodjen, bravo!")
    elif broj == 0:
        break
    elif broj > x:
        print ("Broj je veci od zadanog, pokusaj ponovno!")
    elif broj < x:
        print ("Broj je manji od zadanog, pokusaj ponovno!")
```

Provjerite

1. Koja je od ovih naredaba petlja s logičkim uvjetom?

A. For
B. If
C. While

2. Postoji li razlika između petlje While i petlje For?

A. da
B. ne

Primijenite

1. Izradite program u kojemu će korisnik pogađati jednoznamenkasti nasumično generiran broj sve dok ga ne pogodi, a tada će Python odgovoriti: „Bravo, pogađali ste n puta!”

2. Izradite program koji će korisniku omogućiti pogađanje brojeva. Ako korisnik upiše bilo koji broj, izvest će se blok naredaba ispod petlje While. Ako odabere 0, dogodit će se prekid programa. Prekid programa omogućit će se naredbom Break. Ako korisnik upiše točan broj, ispiše se poruka o pogođenom broju i program se dalje izvodi. Ako korisnik napiše broj koji je veći ili manji od traženog, ispisat će se prigodna poruka te će se i dalje izvoditi program sve dok korisnik sam ne odabere opciju 0.

Provjerite

3. Koja je razlika između petlje While i petlje For?

A. petlja While nema brojač, ali ima uvjet

B. petlja While ima i brojač i uvjet

C. petlja While nema ni brojač ni uvjet

4. Kojim od navedenih operatora može biti postavljen uvjet u petlji While?

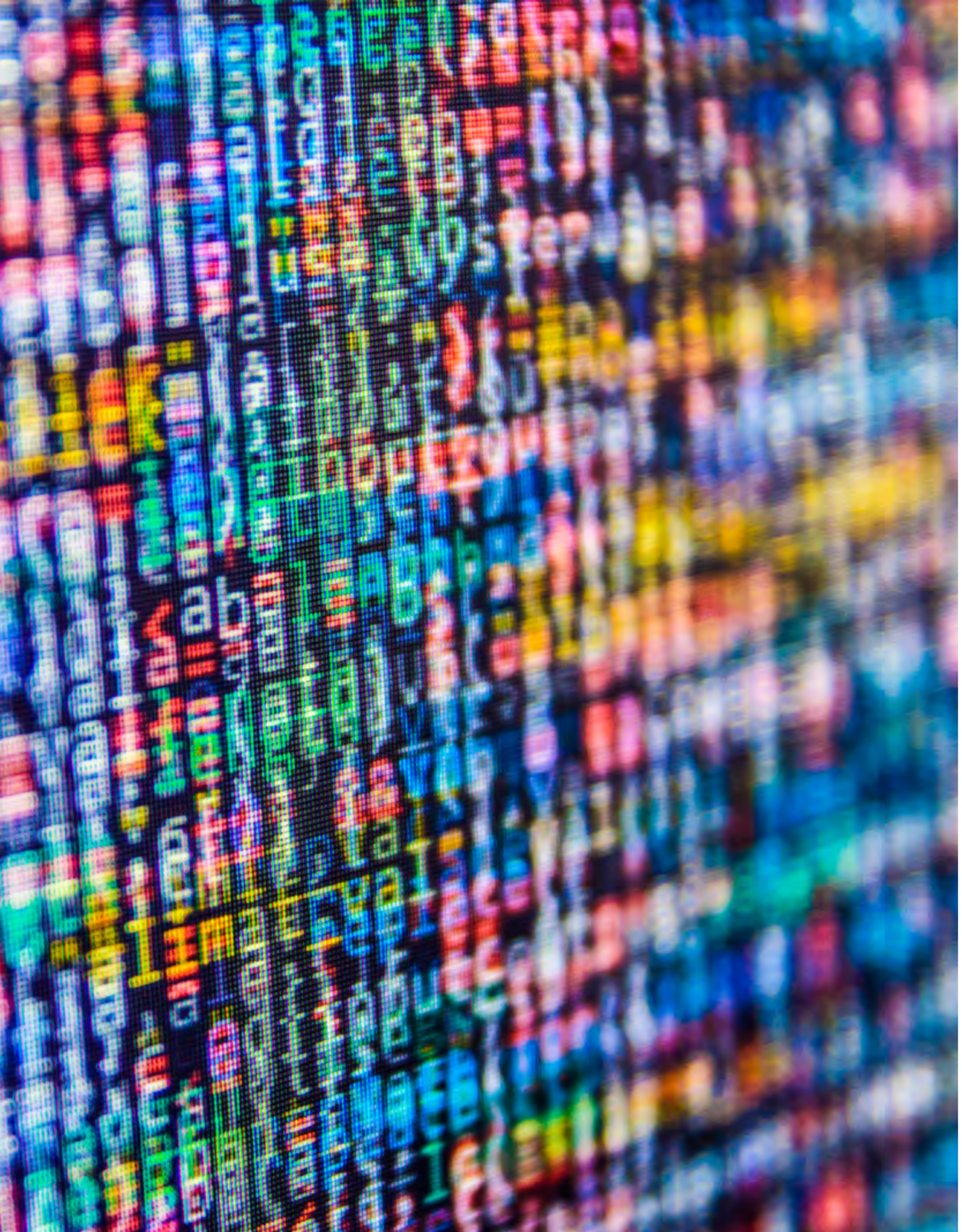
A. operatorom usporedbe

B. aritmetičkim operatorom

C. logičkim operatorom

D. svima od navedenih

Za one koji žele saznati više



7. Tipovi podataka

Nakon što ste u sedmoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o tipovima podataka, moći ćete:

- analizirati problem i primijeniti određeni tip podataka
- primijeniti različite tipove podataka za rješavanje specifičnog problema.

PowerPoint prezentacija

Snimka 7. radionice

U Pythonu se svi tipovi podataka mogu podijeliti na jednostavne i složene. Jednostavni tipovi podataka zovu se jednostavni jer obično predstavljaju jednu vrijednost (jedan podatak). Složeni tipovi podataka predstavljaju više različitih vrijednosti.

vrste tipova zadataka

Jednostavni tipovi podataka u Pythonu:

jednostavni tipovi podataka u Pythonu

- **int** – cjelobrojni tip podataka – rabi se za prikaz i rad s cijelim brojevima
- **string** – znakovni tip podataka – pohranjuje niz znakova
- **float** – omogućuje uporabu decimalnog zapisa broja
- **bool** – logički tip podataka koji može imati vrijednost True (istina) ili False (laž).

Vrijednost varijable

input() – naredba koja unesene podatke prepoznaje kao znakovni niz

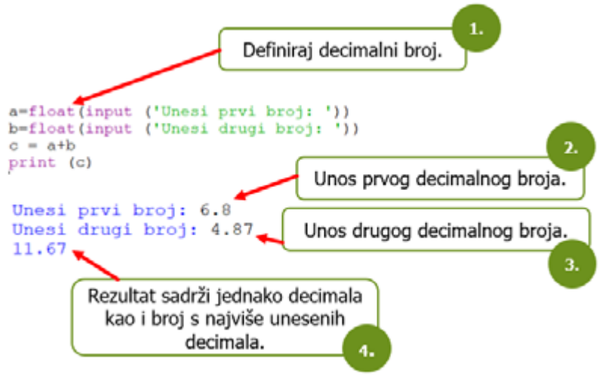
int – naredba koja uneseni znakovni niz pretvara u cijeli broj

Želimo li izvesti matematičke operacije nad unesenim podatcima, potrebno ih je definirati kao broj. Ako se unutar kôda ne postavi da je uneseni podatak broj, program ga prepoznaje kao niz i u tom slučaju operator + obavlja operaciju pridruživanja podataka.

primjer unosa varijable kao cijelog broja

```
a=int(input('Unesi prvi broj: '))
b=int(input('Unesi drugi broj: '))
c=a+b
print(c)
```

primjer unosa varijable kao decimalnog broja



primjer

Bool(ean) je logički tip podataka koji poprima vrijednost True (istina) ili False (nije istina – laž).

primjer podatka kao logičkog tipa podatka

```
>>> 4==5
False
>>> 2>9
False
>>> 'petak'<'PETAK'
False
>>> 7>5
True
```

primjeri zadataka s različitim tipovima podataka

Zadatak 1. Izradite sigurnosni sustav za unošenje četveroznamenkaste lozinke za otključavanje. U program se mogu unositi kombinacije sve dok se ne unese ispravna lozinka.

```
a=int(input('Unesi lozinku: '))
while True:
    if (a==1234):
        a=bool(a)
        print(a)
        print('Ulaz dozvoljen.')
        break
    a=int(input('Ulaz odbijen. \n\nUnesi lozinku: '))
```

Zadatak 2. Izradite program koji provjerava je li troznamenkasti broj paran te ispisuje znamenke stotica, desetica i jedinica.

```
broj=int(input("unesi troznamenkasti broj"))

if (broj % 2 == 0):
    print("broj je paran")
    stotice=int(broj / 100)
    print("stotice su:",stotice)
    desetice=int((broj - stotice*100)/10)
    print("desetice su:",desetice)
    jedinice= int(broj - stotice *100 - desetice *10)
    print("jedinice su:",jedinice)
else:
    print("broj je neparan")
```

Znakovni niz

Svaki znak znakovnog niza ima svoje mjesto određeno indeksom.

određivanje na znakovnom nizu

Na znakovnom nizu može se:

- određivati duljina znakovnog niza
- provjeravati položaj određenog znaka
- određivati najmanja i najveća vrijednost u nizu
- pretvarati niz u niz s velikim ili malim slovima
- pronalaziti određeni izraz unutar niza.

Duljina znakovnog niza može se odrediti naredbom Len koja prikazuje koliko znakova sadržava znakovni niz.

Osnovne funkcije.

Funkcija	Opis djelovanja
sum(a)	vraća sumu svih članova liste a
len(a)	vraća duljinu liste a
min(a)	vraća najmanju vrijednost elementa liste a
max(a)	vraća najveću vrijednost elementa liste a
del(a[i])	element s indeksom i uklanja ga iz liste **a**, lista je kraća za jedan
del(a[i:j])	Briše se isječak koji započinje indeksom i, a završava indeksom j - 1. Lista se skraćuje za j – i

primjer

primjer znakovnog niza podataka

Zadatak 3. Izradite program koji učitava listu a=[3,5,7,8,3,2,5] i briše sve duplikate s te liste te ispisuje novu listu s obrisanim duplikatima.

primjeri zadataka sa znakovnim nizovima

```
a=[3,5,7,8,3,2,5]
pr=[]
for i in a:
    if i not in pr:
        pr.append(i)
print pr
```

Zadatak 4. Izradite program koji upisuje pet brojeva (pet skokova Ivice Kostelića na skijaškom natjecanju) te ispisuje njegov najbolji i najlošiji rezultat.

```
lista=[]
for i in range(5)
        a= input ('upisi broj')
        lista.append(a)

print max(lista)
print min(lista)
```

Zadatak 5. Izradite program koji upisuje bodove n plesnih natjecanja. Ispišite zbroj svih bodova tako da odbacite najbolji i najlošiji rezultat.

```
lista=[]
for i in range(5)
        a=input ('upisi broj')
        lista.append(a)

print sum(lista)-min(lista)-max(lista)
```

Zadatak 6. Izradite program koji unosi bodove pet najboljih natjecatelja u trčanju na 100 m te ispisuje koliko je bodova imao drugi najbolji natjecatelj.

```
lista=[]
for i in range(5):
        a= input ('upisi broj')
        lista.append(a)
lista.sort()
print lista[-2]
```

Zadatak 7. Izradite program koji s liste imena = [’Pero’, ’Ivo’,’Ana’,’Katarina’,’Masa’,’Darko’,’Tin’] ispisuje najdulju riječ.

```
imena = ['Pero', 'Ivo','Ana','Katarina','Masa','Darko','Tin']
print (max(imena, key=len))
```

Zadatak 8. Izradite program koji unosi pet ocjena te ih sprema u listu, a zatim računa i ispisuje njihov prosjek.

```
lista = []

while len(lista) < 5:
    ocjena = int(input('Upisi ocjenu: '))
    if ocjena < 0 or ocjena > 5:
        print ('Neispravan unos.')
        print (' Upisi ocjenu od 1-5')
        continue
    else:
        lista.append(ocjena)


print ('Ocjene:', lista)
print ('Prosjek ocjena:', "{0:.2f}".format(sum(lista)/float(len(lista))))
```

Provjerite

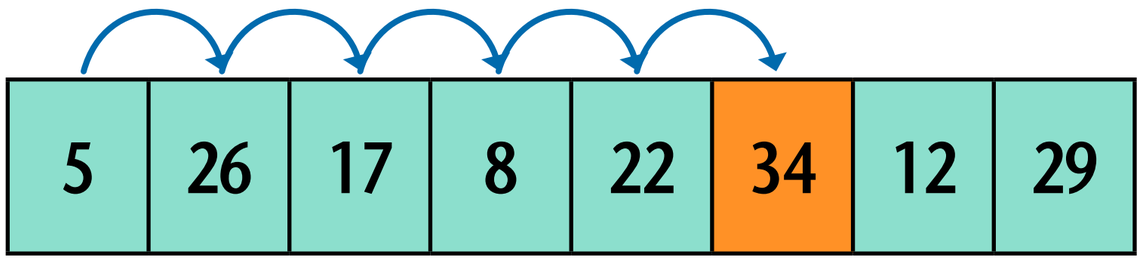
1. Koji su jednostavni tipovi podataka u Pythonu?
2. Pretpostavimo da imamo sljedeću naredbu: a = 7.7. Koji smo tip podataka dodijelili toj varijabli?
3. Pretpostavimo da imamo sljedeću naredbu: b = '28'. Koji smo tip podataka dodijelili toj varijabli?
4. Pretpostavimo da imamo sljedeću naredbu: c = False. Koji smo tip podataka dodijelili toj varijabli?
5. Kojim znakom upisujemo niz u Pythonu?
   A. ( )
   B. [ ]
   C. { }
   D. < >

Primijenite

1. Izradite program koji za učitane dvije liste (prezimena, godine) sortira podatke s obje liste prema prezimenu.
2. Izradite program koji će za upisani znakovni niz provjeriti postoji li riječ „da" u zadanom nizu.
3. Izradite program koji će za upisani znakovni niz izbrojiti koliko ima samoglasnika.

8. Sekvencijsko pretraživanje

Traži: 34



Nakon što ste u osmoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o sekvencijskom pretraživanju, moći ćete:

- izraditi jednostavan program koji se koristi sekvencijskim pretraživanjem
- analizirati i objasniti metodu sekvencijskog pretraživanja
- primijeniti sekvencijsko pretraživanje za prebrojavanje ponavljanja sastavnih dijelova

PowerPoint prezentacija

Snimka 8. radionice

Naredba **In** rabi se za provjeru postajanja određenog znaka unutar znakovnog niza, a naredbu **Count** za prebrojavanje pojavljivanja određenog znaka. Te naredbe prolaze znakovnim nizom te uspoređuju svaki znak u nizu prema određenom uvjetu. To je sekvencijsko pretraživanje.

naredbe **In** i **Count**

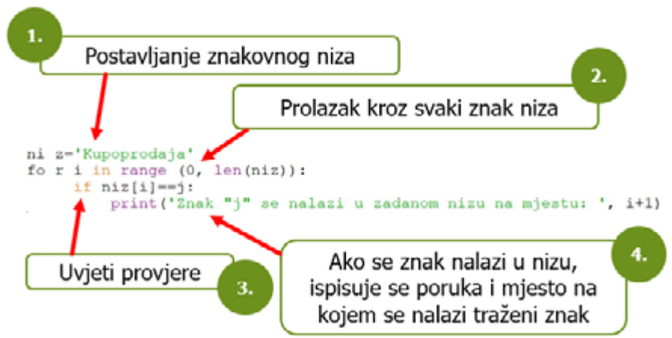
Ako je uvjet ispunjen, naredba In vraća vrijednost *True* da postoji traženi znak, a naredba Count, ako je uvjet ispunjen, povećava vrijednost brojača i prelazi na provjeru sljedećeg znaka.

Isto pretraživanje može se obaviti s pomoću naredaba For i If. Naredbom For pregledava se svaki znak u znakovnom nizu, a s pomoću naredbe If provjerava se uvjet.

primjer

Zadatak 1. Izradite program koji u nizu pretražuje na kojem se mjestu nalazi slovo j.

primjer korištenja sekvencijskim pretraživanjem znakovnog niza



Osim provjere postojanja jednog znaka, može se provjeriti i postojanje određenog izraza. Tada se mora postaviti više uvjeta, koliko mjesta sadržava traženi izraz.

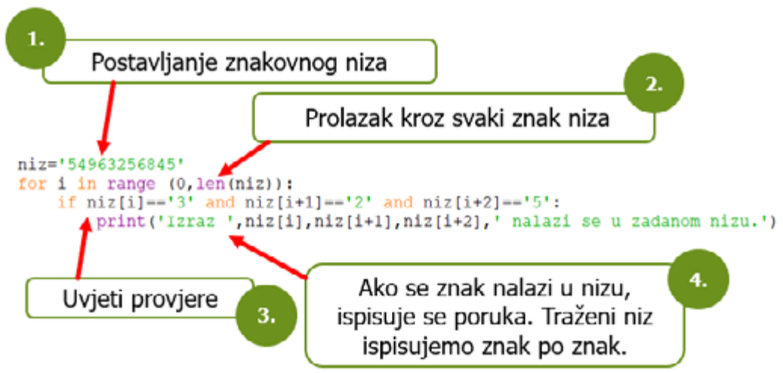
primjer

Zadatak 2. Izradite program koji u znakovnom nizu „PoduZetnišTvo" provjerava postoji li znak t.

primjeri zadataka u kojima se primjenjuje sekvencijsko pretraživanje

```
niz='PoduZetnišTvo'
for brojač in range (0, len(niz)):
    if niz[brojač]=='t':
        print('znak "t" nalazi se u zadanom nizu na mjestu', brojač+1)
```

Zadatak 3. Izradite program koji u zadanom nizu pretražuje izraz 325.



Naredba Count

naredba **Count**

Naredbom **Count** prebrojava se broj pojavljivanja određenog znaka. Kako bi se s pomoću sekvencijskog pretraživanja prebrojilo pojavljivanje traženog znaka, morat će se upotrijebiti brojač.

Brojač je na početku programa postavljen na vrijednost 0, no svaki put kad naiđe na traženi znak, povećat će vrijednost brojača. Na kraju program ispiše dobiveni rezultat.

primjeri zadataka primjenom naredbe Count

primjer

Zadatak 4. Izradite program koji u zadanom nizu provjerava koliko se puta pojavljuje znak a.

Zadatak 5. Izradite program koji provjerava postoji li izraz „STAV” u znakovnom nizu „dostava”. Koristi se naredbom Upper kako bi promijenio znakove niza u velika slova.

```
niz='dostava'
veliki=niz.upper()
brojač=0

for i in range (0, len(veliki)):
    if veliki[i]=='S' and veliki[i+1]=='T' and veliki[i+2]=='A' and veliki[i+3]=='V' :
        brojač+=1


print('Riječ "STAV" pojavila se ',brojač,'puta')
```

Zadatak 6. Deklarirajte niz naziva niza i ograničite ga na 10 sastavnih dijelova. Omogućite unos elemenata s pomoću tipkovnice. Ispišite samo parne sastavne dijelove niza.

```
niz=[0]*10 #deklaracija niza

#unos podataka u niz
for i in range (0,10):
    niz[i]=int(input("unesite  broj: " ))

for i in range (1,10,2):
    print(niz[i])
```

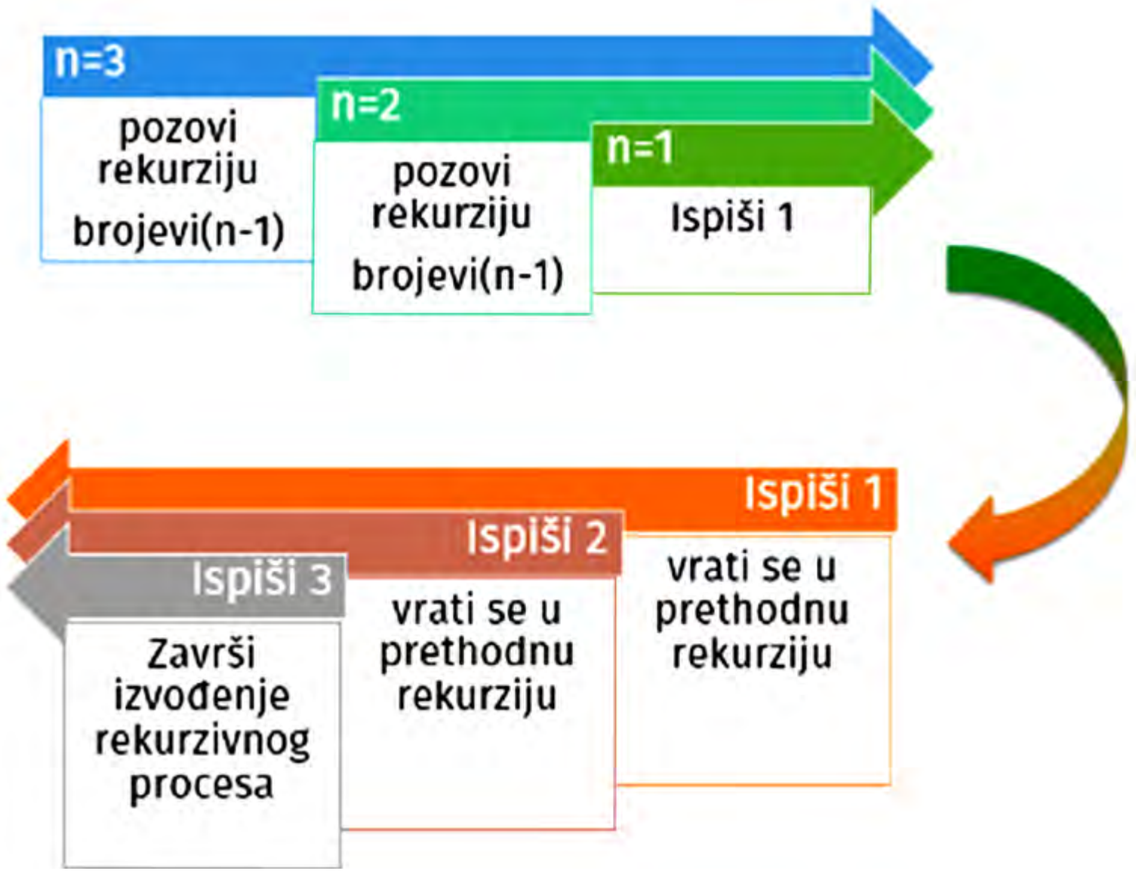
Provjerite

1. Kojom se naredbom koristimo za provjeru postojanja određenog znaka unutar znakovnog niza?
   A. In
   B. Or
   C. Out
2. Kojom se naredbom koristimo za prebrojavanje pojavljivanja određenog znakovnog niza?
   A. Sum
   B. Count
   C. Input

Primijenite

1. Deklarirajte niz naziva niza i ograničite ga na 10 sastavnih dijelova. Omogućite unos sastavnih dijelova s pomoću tipkovnice. Ispišite svaki četvrti sastavni dio niza.
2. Deklarirajte niz naziva niza i ograničite ga na 10 sastavnih dijelova. Omogućite unos sastavnih dijelova s pomoću tipkovnice. Ispišite neparne sastavne dijelove niza.
3. Deklarirajte niz naziva niza i ograničite ga na 10 sastavnih dijelova. Napunite niz bez unosa s pomoću tipkovnice. Ispišite niz u obrnutom redoslijedu, od 10 prema 1.
4. Deklarirajte niz naziva niza i ograničite ga na 10 sastavnih dijelova. Napunite unosom s pomoću tipkovnice.
5. Zbrojite sastavne dijelove nizova i ispišite njihov zbroj.
6. Kreirajte dva niza od po pet sastavnih dijelova. Zatim napunite prvi niz s pomoću tipkovnice. Nakon toga premjestite sastavne dijelove prvog niza u drugi kreirani niz i ispišite taj niz.

9. Rekurzija



Nakon što ste u devetoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o rekurziji, moći ćete:

- prepoznati jednostavne rekurzivne procese
- opisati jednostavne rekurzivne procese
- primijeniti rekurziju na primjerima matematičkih problema i problema iz stvarnog života.

PowerPoint prezentacija

Snimka 9. radionice

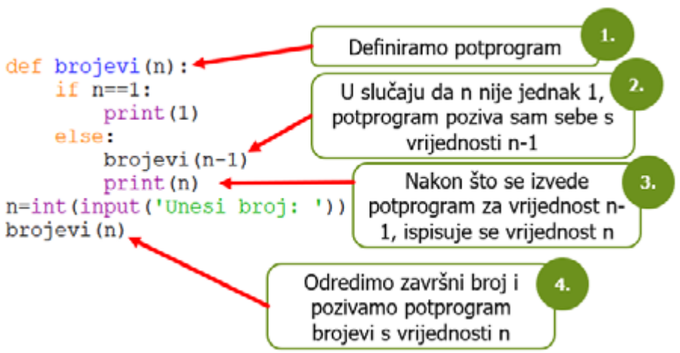
U programiranju rekurzivnim procesima nazivaju se procesi (funkcije) koji pozivaju sami sebe. Rekurzija se najčešće definira kao potprogram koji vraća određene vrijednosti i ponovno se izvodi s novim vrijednostima.

programiranje rekurzivnim procesima

Kad funkcija poziva sama sebe, ona oponaša petlju. U rekurzivnim procesima petlja se stvara tako da potprogram poziva sam sebe. Ako se žele ispisati svi brojevi od 1 do željenog broja, rekurzija će sama pozivati svoj potprogram mijenjajući mu ulazne vrijednosti.

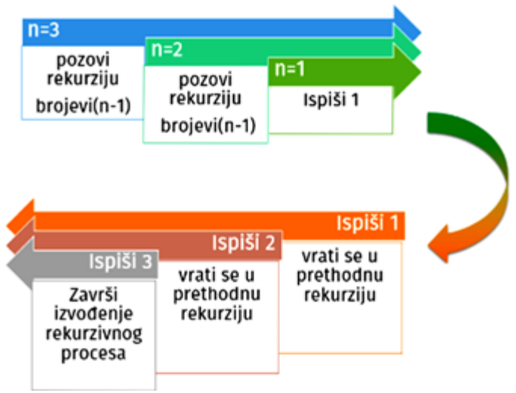
Zadatak 1. Izradite program koji ispisuje brojeve do unesenog broja koristeći se rekurzijom.

primjer ispisa brojeva od 1 do željenog broja s pomoću rekurzije



Objašnjenje rada rekurzije

Potprogram poziva sam sebe smanjujući ulaznu vrijednost n-1 svaki put kad se pozove. Kad dosegne vrijednost n = 1, stvaraju se uvjeti za prekid rekurzije, više potprogram ne poziva sam sebe, već ispisuje vrijednost. Nakon ispisa vrijednosti n = 1 izlazi se iz posljednje rekurzije i vraća u rekurziju n = 2 u kojoj se nastavlja izvođenje, odnosno ispisuje se broj n = 2. Nakon toga ta se rekurzija vraća u prethodnu gdje je n = 3, ispisuje se vrijednost n. To je posljednji korak, pa program završava. Uvjet za prekid rekurzije je uvjet koji prekida pozivanje rekurzivnog potprograma.



primjer rekurzivnog procesa

Ovaj primjer pokazuje kompleksnost izvođenja rekurzije. Računalo **izvodi niz koraka u kratkom vremenu**. To je najveća razlika između računanja i računala. Računala su sagrađena tako da velikom brzinom obrađuju informacije i matematičke operacije, pa za njih rekurzija nije nimalo teža od rješavanja jedne po jedne operacije.

zadatci za uvježbavanje rekurzije

Zadatak 2. Izradite program koristeći se rekurzijom koja računa umnožak svih prirodnih brojeva od 1 do zadanog broja. Potrebno je razmišljati o tome koje vrijednosti rekurzija vraća u svakom koraku.

```
def umnožak(k):
    if k==1:
        return 1
    else:
        return k*umnožak(k-1)
k=int(input('Do kojeg broja množiš: '))
umnožak(k)
print('Umnožak svih brojeva do ',k,' je: ',umnožak(k))
```

Zadatak 3. Izradite skripta koja određuju vrijednost Fibonaccijeva niza na željenome mjestu.

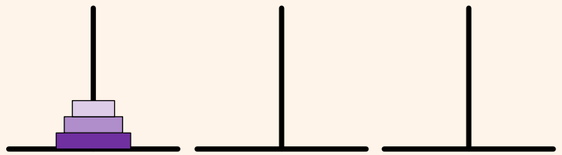
```
def fib(n):
    if n==0:
        return 1
    elif n==1:
        return 1
    else:
        return (fib(n-1)+fib(n-2))
n=int(input('Koje mjesto u Fibonaccijevu nizu želiš provjeriti: '))
print('Na ', n, '. mjestu Fibonaccijeva niza nalati se broj: ', fib(n))
```

Zadatak 4. Napišimo rekurzivno funkciju koja će ispisivati rastav prirodnog broja n na jednostavne faktore.

```
def r(n, k):
 if n == k:
 return str(k)
 elif n % k == 0:
 return str(k) + ' * ' + r(n // k, k)
 else:
 return r(n, k + 1)
```

Primijenite

1. Napišimo rekurzivno funkciju za rješavanje Hanojskih tornjeva.



Hanojski tornjevi su zagonetka koju je smislio francuski matematičar Lucas 1883. godine. Sastoje se od tri tornja. Na jednom se nalazi n diskova poredanih po veličini. Na dnu je najveći, a na vrhu najmanji.

Cilj je prebaciti sve diskove na neki od preostalih tornjeva, i to tako da oni zadrže originalni poredak, a pritom se valja držati dva jednostavna pravila:

1. Prebacuje se jedan po jedan disk.
2. Ne smije se staviti veći disk na manji.

Za one koji žele saznati više

10. Sortiranje podataka

Nakon što ste u desetoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o sortiranju podataka, moći ćete:

- primijeniti jednostavan algoritam sortiranja podataka u listi
- samostalno opisati i analizirati zadani problem, planirati rješavanje problema te predložiti i prepoznati algoritamsko rješenje.

PowerPoint prezentacija

Snimka 10. radionice

odrednice sortiranja

Precizno određen redoslijed sastavnih dijelova pomaže tomu da se lakše pronađe i izdvoji pojedini sastavni dio. Često se stvari organiziraju prema nekom redoslijedu kako bismo ih lakše pronašli. Kad dođete u knjižnicu po knjigu za lektiru, jednostavno ćete je pronaći na polici jer su knjige složene prema prezimenu autora. Takav popis organiziranja nazivamo sortiranje.

Sortiranje je postavljanje sastavnih dijelova određenim redoslijedom prema nekom kriteriju.

Bubble sort – algoritam sortiranja

U programiranju se primjenjuje sortiranje, a za to postoje različiti algoritmi i postupci sortiranja podataka. Jedan od najjednostavnijih algoritama sortiranja jest metoda *bubble sort* koja omogućuje sortiranje podataka liste ili niza prema veličini sastavnih dijelova.

metoda *bubble sort*

U toj metodi primjenjuje se sekvencijsko pretraživanje sastavnih dijelova liste i sortiranje. Provjerava se dio liste, uspoređuje svaki član s članom pokraj njega i, ako je potrebno, zamjenjuje im se mjesto. Stvara se privremena varijabla kojoj se pridružuje vrijednost sastavnog dijela koji se trenutačno sortira. Stvara se „mjehur" u koji se privremeno unose podatci. U toj se metodi zatim uspoređuje vrijednost određenog sastavnog dijela sa sastavnim dijelom liste pokraj njega, sortira se prema vrijednosti te se prelazi na sljedeći sastavni dio.

slikoviti prikaz *bubble sort* metode

primjer

Zadatak 1. Sortirajte listu [4, 2, 1, 3].

1. Postavi početnu listu

```
lista = [4,2,1,3]
```

2. Pomoću petlje for pregledavaj segment (dio) liste. Svaki prolaz provjeri jedan element manje. Prvu iteraciju (prolaz) pregledavaj sve elemente, zatim u drugom prolazu jedan element manje i tako redom dok ne bude više segmenata za pregledavanje.

```
lista = [4,2,1,3]
for i in range(len(lista)-1,0,-1):
```

3. Pomoću druge petlje for provjeri svaki element liste za svaki korak prve petlje for.

```
lista = [4,2,1,3]
for i in range(len(lista)-1,0,-1):
    for j in range(i):
```

4. Pomoću naredbe if provjeri svaki element liste. Ako je vrijednost elementa koji se provjerava veća od vrijednosti sljedećeg elementa, postavi vrijednost u varijablu privremeno i zamijeni im mjesta.

```
lista = [4,2,1,3]
for i in range(len(lista)-1,0,-1):
    for j in range(i):
        if lista[j]>lista[j+1]:
            privremeni=lista[j]
            lista[j]=lista[j+1]
            lista[j+1]=privremeni
```

5. Ispiši novi poredak liste.

```
lista = [4,2,1,3]
for i in range(len(lista)-1,0,-1):
    for j in range(i):
        if lista[j]>lista[j+1]:
            privremeni=lista[j]
            lista[j]=lista[j+1]
            lista[j+1]=privremeni
print(lista)
```

Naredba Sort()

U programskom jeziku Python postoji funkcija sortiranja podataka koja omogućuje jednostavno sortiranje podataka bez izrade algoritma i složenoga kôda. Funkcija sortiranja liste poziva se naredbom **Sort()**.

Unutar naredbe Sort mogu se postaviti određeni uvjeti sortiranja podataka kao što su redoslijed sortiranih podataka od manjeg prema većem ili obratno te sortiranje prema duljini znakovnog niza u listi, od sastavnog dijela s najmanje znakova prema onome s najviše znakova i obratno.

primjer sortiranja naredbom Sort

primjer

Zadatak 2. Sortirajte zanimanja.

```
zanimanja = ['profesor', 'čuvar', 'liječnik']
print('Početna lista zanimanja: ', zanimanja)
zanimanja.sort()
print('Sortirana lista zanimanja: ', zanimanja)
```

Sortiranje liste zanimanja

```
Početna lista zanimanja:  ['profesor', 'čuvar', 'liječnik']
Sortirana lista zanimanja:  ['liječnik', 'profesor', 'čuvar']
```

U tablici kodova ASCII znak č ima veću vrijednost od ostalih znakova iz engleske abecede. Tablica kodova ASCII najprije dodjeljuje manje vrijednosti slovima engleske abecede, a zatim dodjeljuje vrijednosti preostalim znakovima (slovima) hrvatske abecede.

tablica kodova ASCII

Ako se naredbi Sort ne postave nikakve vrijednosti unutar zagrade, lista će se sortirati od najmanje vrijednosti prema najvećoj. Ako se želi promijeniti redoslijed sortiranih podataka od najvećeg prema najmanjem, unutar zagrade naredbe Sort unese se **reverse=True**.

```
['liječnik', 'profesor', 'čuvar']
```

Uzlazno sortirani podatci

```
['čuvar', 'profesor', 'liječnik']
```

Silazno sortirani podatci

Funkcija `key`

Osim uzlaznog i silaznog sortiranja sastavnih dijelova liste, moguće je sortiranje liste i prema duljini znakova pojedinog sastavnog dijela. Za sortiranje sastavnih dijelova prema duljini znakovnog niza upotrebljava se funk-

cija key unutar zagrade naredbe Sort i postavi se tako da ključ (način sortiranja) prema kojemu se sortira lista ovisi o duljini znakovnog niza len.

primjer

primjeri sortiranja uzlazno i silazno

Zadatak 3. Sortirajte zanimanja uzlazno.

```
zanimanja = ['profesor', 'čuvar', 'liječnik']
print('Početna lista zanimanja: ', zanimanja)
zanimanja.sort(key=len)
print('Sortirana lista zanimanja: ', zanimanja)
```

```
Početna lista zanimanja:  ['profesor', 'čuvar', 'liječnik']
Sortirana lista zanimanja:  ['čuvar', 'profesor', 'liječnik']
```

Zadatak 4. Sortirajte listu zanimanja iz prethodne aktivnosti prema duljini znakovnog niza silazno.

```
zanimanja = ['profesor','čuvar','liječnik']
print('Početna lista zanimanja: ', zanimanja)
zanimanja.sort(key=len)
print('sortirana lista zanimanja: ', zanimanja)
```

Zadatak 5. Sortirajte listu [54, 26, 93, 17, 77, 31, 44, 55, 20] (metoda mjehurića) uzlazno.

```
niz = [54,26,93,17,77,31,44,55,20]
for i in range(len(niz)):
        for j in range(i):
            if niz[j]>niz[j+1]:
                temp = niz[j]
                niz[j] = niz[j+1]
                niz[j+1] = temp

print(niz)
```

Zadatak 6. Zadatak 5. Sortirajte listu [54, 26, 93, 17, 77, 31, 44, 55, 20] (metoda mjehurića) silazno.

```
niz = [54,26,93,17,77,31,44,55,20]
for i in range(len(niz)):
        for j in range(i):
            if niz[j]<niz[j+1]:
                temp = niz[j]
                niz[j] = niz[j+1]
                niz[j+1] = temp

print(niz)
```

Provjerite i primijenite

1. Postoje dvije jednostavne metode sortiranja, a to su metoda izbora i metoda umetanjem. U čemu je razlika između metode „bubble sort”, metode izbora i metode umetanjem?

Izazov 1. Sortirajte listu [54, 26, 93, 17, 77, 31, 44, 55, 20] (sortiranje umetanjem)

Izazov 2. Sortirajte listu [54, 26, 93, 17, 77, 31, 44, 55, 20] (sortiranje izborom)

Izazov 3. Prethodnu listu sortirajte s pomoću obje metode, ali prema rastućem redoslijedu.

1. Uvod u programiranje

Nakon što ste u prvoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o osnovama programiranja, moći ćete:

- objasniti kontekst programskih jezika
- razlikovati strojni jezik od programskih jezika više razine
- objasniti što su dijagrami tijeka i pseudokod te kako se oni upotrebljavaju u izradi računalnih programa
- opisati povijest programskog jezika Python
- prepoznati osnovne faze ili korake u izradi računalnog programa: analiza problema, crtanje dijagrama tijeka, pisanje programskoga koda i unos programskoga koda u računalo te pokretanje programa
- obrazložiti što obuhvaća prvi korak u izradi računalnog programa: analizu potreba, definiranje preko jednačina, uvjeta i pseudoalgoritama te kako se on izvodi kod jednostavnijih i složenijih programa.
- razviti algoritamski način razmišljanja i primijeniti ga u rješavanju problema uporabom dijagrama tijeka i pseudokoda.

PowerPoint prezentacija

Snimka 1. radionice

Programiranje

Tehnike programiranja

odrednice programiranja

Programiranje je vještina koja omogućuje korisniku da stvori i izvede algoritme koristeći se određenim programskim jezicima kako bi izradio računalni program. Programiranje uključuje sastavnice umjetnosti, znanosti, matematike i inženjerstva. Strojni jezik (strojni kod ili binarni kod) jedini je programski jezik koji računalo može izravno primijeniti.

povijesni razvoj računalnog pisma, programiranja

U početcima računalstva programeri su pisali u strojnom kodu što je bilo vrlo komplicirano i zamorno. Zatim je slijedila upotreba simboličnih jezika, poznatih pod zajedničkim nazivom asembler, koji se sastoje od jednostavnih instrukcija koje se izravno i jednoznačno mogu prevesti u strojni kod. Iako je mnogo pogodnije od strojnog programiranja, asemblersko programiranje karakterizira velika količina posla koju programer mora obaviti zbog činjenice da su operacije i dalje elementarne.

Zbog toga su stručnjaci stvorili programske jezike više razine s pomoću kojih se piše izvorni kod koji se prevodi u strojni kod posredovanjem specijalnih programa – prevoditelja poput tumača i kompajlera.

koraci za izradu programa

Pri izradi svakog programa potrebno je proći četiri osnovne faze ili koraka: analizu problema, crtanje dijagrama tijeka, pisanje programskoga koda i unos programskoga koda u računalo te pokretanje programa.

analiza potreba

Prvi korak u izradi računalnog programa jest analiza potreba koja obuhvaća razmatranje situacije i problema koji treba riješiti, definiranje preko jednadžbi, uvjeta i tzv. pseudoalgoritama. Kod jednostavnijih programa taj je korak u domeni usmene analize, a kod složenijih programa radi u pisanoj i simboličnoj formi uz uredno dokumentiranje svih promjena (taj korak pripada softverskoj domeni).

Algoritam

Algoritam je precizno zapisan niz postupaka, radnji ili naredaba koje služe tomu da se obavi neki posao uz potrebne resurse. Programiranjem se razvija algoritamski način razmišljanja što znači da se uči rješavati probleme na strukturiran način. Dijagram tijeka grafički je prikaz algoritma koji olakšava razumijevanje postavljenog zadatka.

Dijagram tijeka

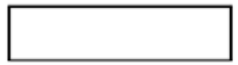
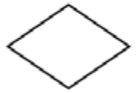
Dijagram tijeka u Pythonu se može izraditi uporabom programskog jezika Python i njegovih sastavnica.

Dijagram tijeka u Pythonu prikazuje korake u programu u obliku grafičkog prikaza, a služi za vizualizaciju procesa izvođenja programa i olakšavanje njihova razumijevanja.

odrednice dijagrama tijeka u Pythonu

Za crtanje dijagrama tijeka koristimo se sljedećim simbolima

simboli dijagrama tijeka

Simbol	Značenje
	simbol koji označava početak i kraj algoritma
	simbol za ulaz podataka
	simbol obrade podataka
	simbol odluke
	simbol za izlaz podataka
	simbol za nastavak dijagrama (spojna točka, poveznica)

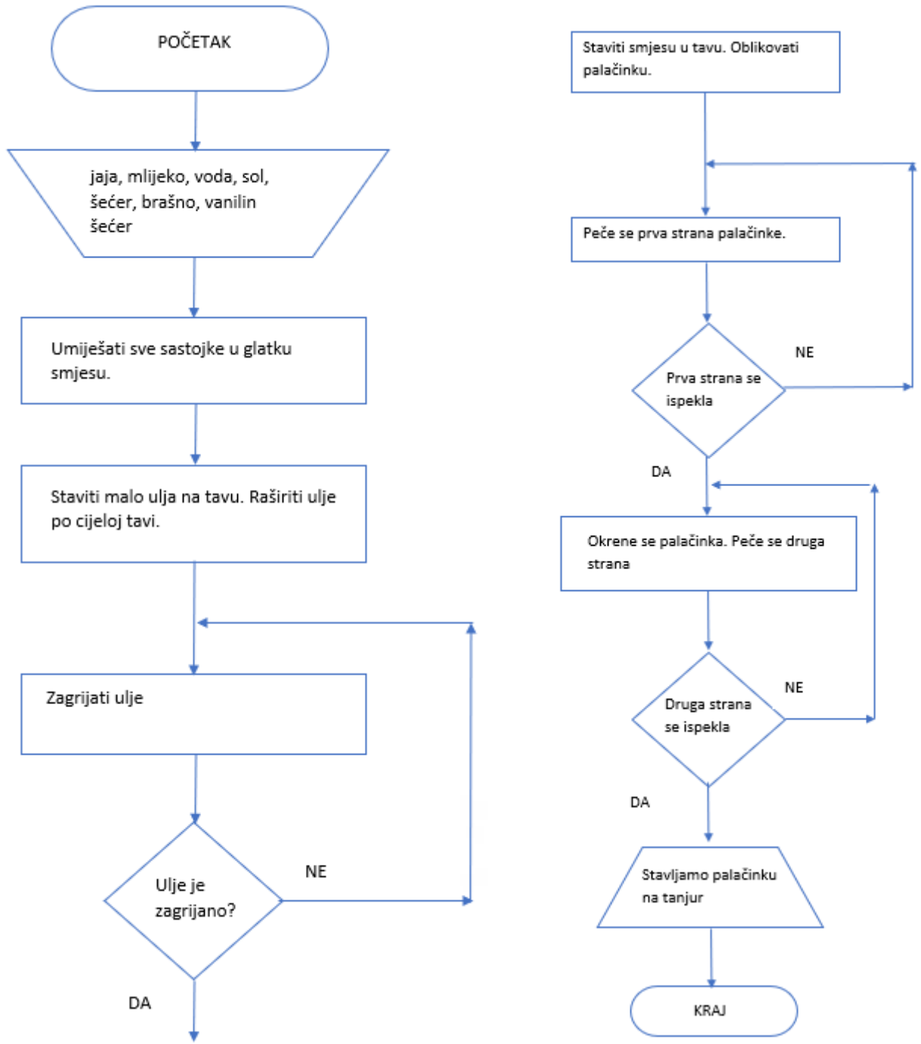
Koraci dijagrama tijeka na primjeru pečenja palačinki:

dijagrama tijeka u Pythonu na primjeru pečenja palačinki

- pokrenite program
- pripremite sastojke

- pomiješajte sastojke u zdjeli (brašno, jaja, mlijeko, sol, šećer)
- zagrijte tavu na srednje jakoj vatri
- dodajte malo ulja u tavu
- malom kutlačom zagrabite smjesu i ulijte je u tavu
- pecite palačinku sve dok ne dobije smeđu boju s jedne strane
- okrenite palačinku i pecite je sve dok ne dobije smeđu boju s druge strane
- ponovite korake 6 – 9 sve dok ne potrošite svu smjesu
- isključite vatru i poslužite palačinke sa šećerom, medom ili drugim preljevom prema želji
- završite program.

U nastavku je prikazan dijagrama tijeka u Pythonu koji prikazuje korake u programu za pečenje palačinki što olakšava razumijevanje i izvođenje tog procesa.

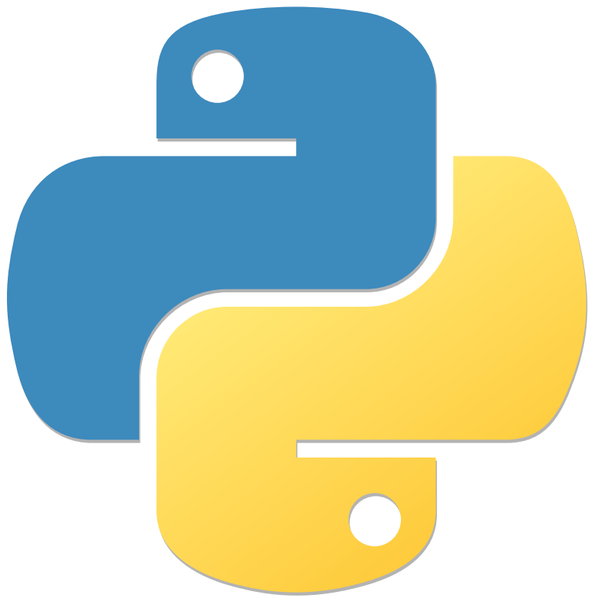

POČETAK
jaja, mlijeko, voda, sol, šećer, brašno, vanilin šećer
Umiješati sve sastojke u glatku smjesu.
Staviti malo ulja na tavu. Raširiti ulje po cijeloj tavi.
Zagrijati ulje
Ulje je zagrijano?
NE
DA
Staviti smjesu u tavu. Oblikovati palačinku.
Peče se prva strana palačinke.
Prva strana se ispekla
NE
DA
Okrene se palačinka. Peče se druga strana
Druga strana se ispekla
NE
DA
Stavljamo palačinku na tanjur
KRAJ

Provjerite i primijenite

1. Napravite algoritam koji će izračunati prosjek ocjena studenata iz tri predmeta.
2. Izradite program Python koji će omogućiti korisniku unos triju brojeva (ocjena) i izračunati prosjek tih brojeva te ispisati rezultat.
3. Izradite dijagram tijeka za program Python koji će omogućiti korisniku unos riječi te će program ispisati broj samoglasnika i suglasnika u toj riječi.

Za one koji žele saznati više

Izradite dijagram tijeka za rješavanje složenijeg problema – kako organizirati svoj radni tjedan ako su predavanja svakog parnog dana poslijepodne, a neparnog ujutro, a vježbe su utorkom i petkom?



python™

2. Instaliranje Pythona i razvojno okružje

Nakon što ste u drugoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o instaliranju Pythona i razvojnom okružju, moći ćete:

- razumjeti mogućnosti upotrebe programskog jezika Python kao što su lokalno instaliranje i upotreba mrežnog okružja
- poznavati korake u lokalnom instaliranju programskog jezika Python
- razumjeti načine programiranja u Pythonu uključujući ljusku (engl. *shell*) i konzolu
- riješiti jednostavne primjere zadataka programiranja u Pythonu kao što su izračunavanje matematičkih operacija i ponavljanje riječi
- spremiti program Python i pokrenuti ga
- razumjeti koncept slijednog programiranja i koristiti se naredbom Print za prikaz teksta.

PowerPoint prezentacija

Snimka 2. radionice

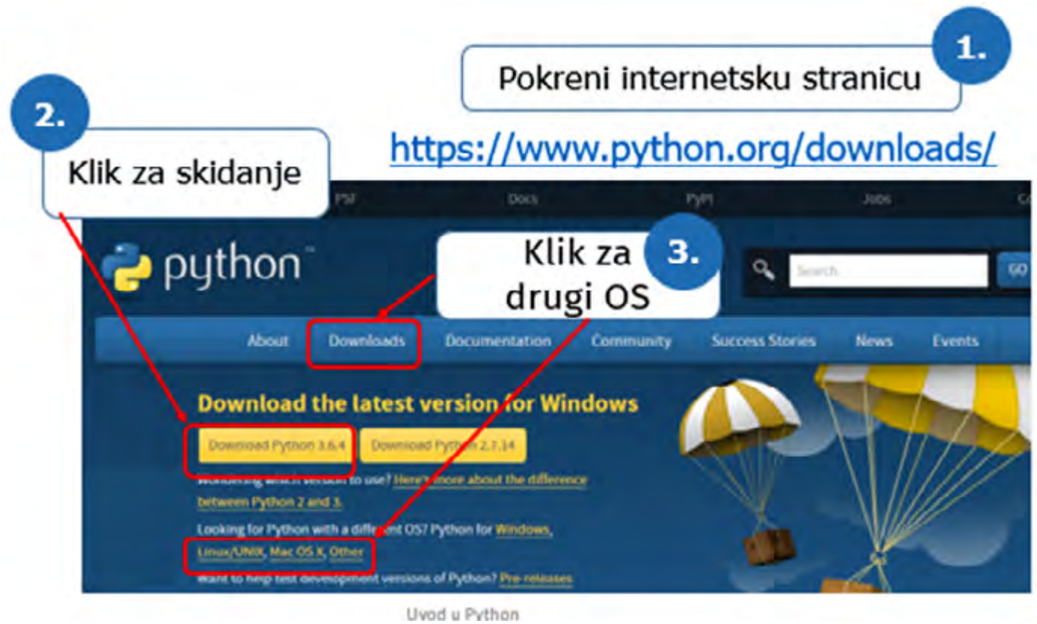
Mogućnosti upotrebe Pythona

Python se može upotrebljavati:

- lokalno instaliran na računalo
- alatom vizualizacije u mrežnom okružju.

Koraci u lokalnom instaliranju

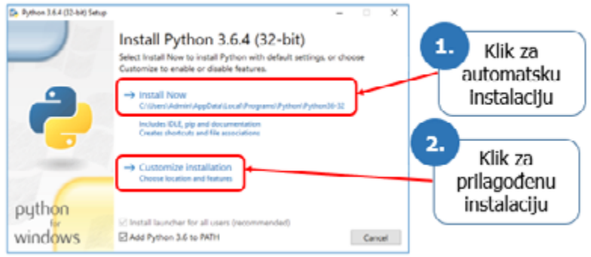
Python se najlakše upotrebljava kad se instalira lokalno. Potrebno je posjetiti mrežnu stranicu https://www.python.org/downloads



Uvod u Python

Nakon pokretanja datoteke za instaliranje pojavljuje se dijaloški okvir čarobnjaka za instaliranje.

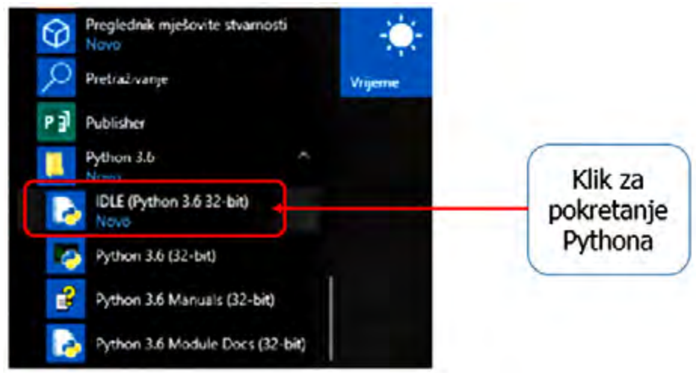
dijaloški okvir čarobnjaka za instaliranje



Nakon pokretanja instaliranja klikom na **Install Now**, slijedi prikaz tijeka instaliranja.

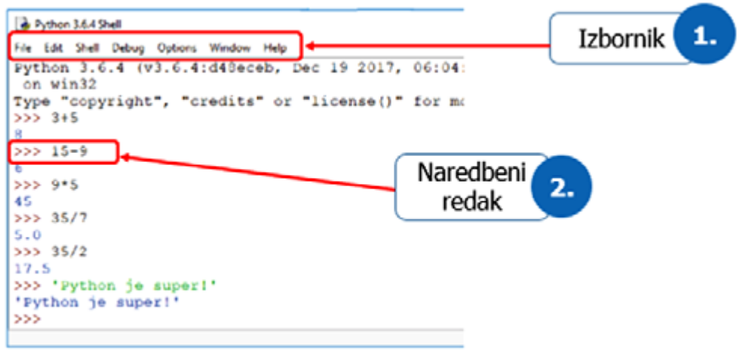
Python pokrećemo klikom na IDLE (Python GUI) s popisa programa.

pokretanje Pythona klikom



Programiranje Pythona

Python se može programirati u tzv. ljusci (engl. *shell*) ili konzoli u kojoj se naredbe provode redak po redak kako piše. Sve što se programira u Pythonu odmah se i vidi nakon unesene **naredbe**.



primjer

Zadatak 1. Izračunajte zbroj, razliku, umnožak i količnik dvaju brojeva i ispišite rezultate.

Osnovne računske operacije u Pythonu:

Računska operacija	Znak
zbrajanje	+
oduzimanje	-
množenje	*
dijeljenje	/
djelomični količnik	//
ostatak pri dijeljenju	%

Radnja	Primjer	Opis
Djelomični količnik	`>>> 14//5` `2`	Za izračun koristimo oznaku // Rezultat dijeljenja brojeva 14 i 5 je 2.8 pri čemu je 2 djelomični količnik
Ostatak pri dijeljenju	`>>> 14%5` `4`	Za izračun koristimo oznaku %
Višečlani izraz	`>>> 20*5+45/9-55` `50.0`	Python poštuje prioritet računskih operacija. Rezultat je zbog operacije dijeljenja decimalan.

Multipliciranje teksta	`>>> 2*'abcd'`
	`'abcdabcd'`
	`>>> 3*'Python '`
	`'Python Python Python '`
		Tekst možemo multiplicirati pomoću znaka *.
		Za razdvajanje teksta koji se spaja, na kraju teksta, trebamo umetnuti jedan razmak.

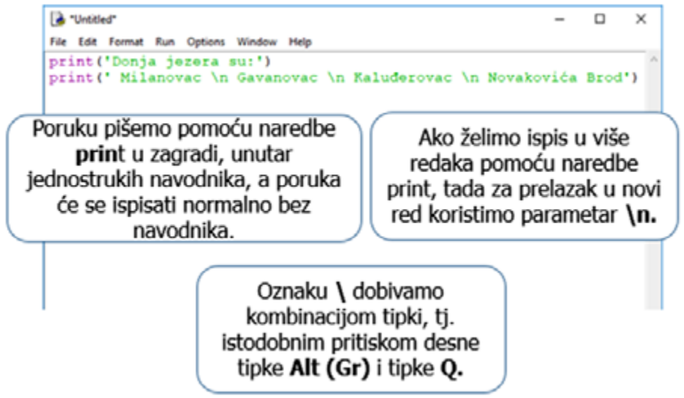
Zadatak 2. Izračunajte koliki je ostatak pri dijeljenju brojeva 1234567 i 25.

```
1234567%25
>>> 11
```

Zadatak 3. Napišite kod koji 10 puta ponavlja riječ „student”.

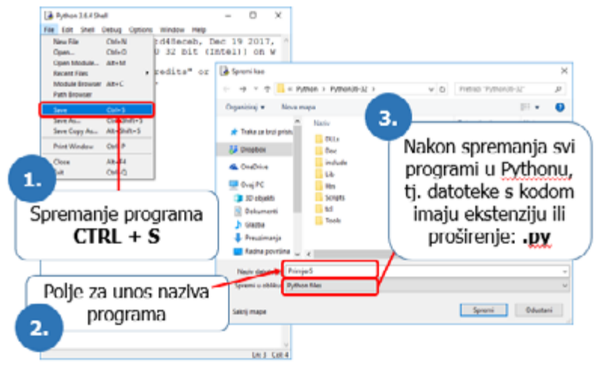
Postupci u programiranju

pisanje programa



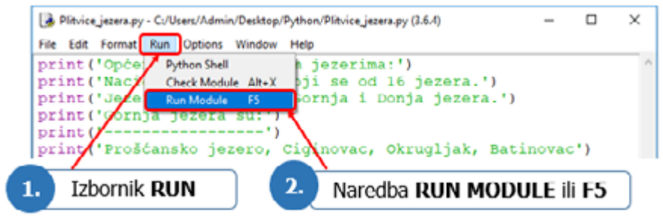
Prije pokretanja programa potrebno ga je spremiti.
To se radi na sljedeći način.

spremanje programa



Nakon spremanja preostaje pokretanje programa klikom na *Run*, pa *Run Module* ili F5. Pokretanje programa je postupak kojim počinje provođenje naredaba programskoga koda.

pokretanje programa



Programiranje u kojemu se program izvodi redoslijedom kojim je napisan, naredbu po naredbu, zove se slijedno programiranje**.** Za prikaz teksta upotrebljava se naredba Print.

slijedno programiranje

primjer

primjer slijednog programiranja

Zadatak 4. Izradite program koji zbraja i dijeli dva broja te na zaslonu ispisuje zbroj i ostatak pri dijeljenju dvaju brojeva.

```
print('Zbroj 40 + 50 = ', 40 + 50)
print('Ostatak pri dijeljenju 140 i 50 = ', 140%50)
```

Zadatak 5. Dodajte u prethodni zadatak množenje, dijeljenje i oduzimanje brojeva te djelomični količnik brojeva.

```
print ('Zbroj brojeva 50 i 40 je ', 50+40)
print ('Razlika brojeva 50 i 40 je ', 50-40)
print ('Umnožak brojeva 50 i 40 je ', 50*40)
print ('Količnik brojeva 50 i 40 je ', 50/40)
print ('Ostatak kod djeljenja brojeva 50 i 40 je', 50%40)
print ('Djelomični količnik brojeva 50 i 40 je', 50//40)
```

Provjerite i primijenite

Zadatak 1. Izradite program koji ispisuje „Danas je:” te navodi određeni datum i zbraja brojeve datuma (primjer 23. 4. 2019. = 23+4+2019).

Zadatak 2. Izradite program koji ispisuje raspored sati razreda za prva tri dana (ponedjeljak, utorak i srijedu).

```
print (35*'=')
print ('    Ponedjeljak|Utorak|Srijeda \n \t HRV\t ENG\t \n \t GK\t MAT\t PID\t\n \t PID\t MAT\t GEO\t')
print (35*'=')
```

Za one koji žele saznati više

Izradite program za rješavanje složenijeg problema – kako organizirati svoj radni tjedan ako su svakoga parnog dana predavanja poslijepodne, a svakoga neparnog ujutro, a vježbe su utorkom i petkom.

3. Osnovne i elementarne jedinice i pojmovi

Nakon što ste u trećoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o osnovnim i elementarnim jedinicama i pojmovima, moći ćete:

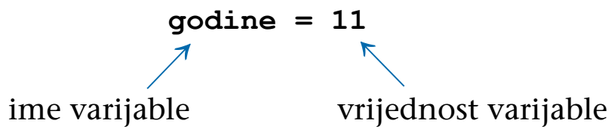
- razumjeti pojam varijable i njezinu ulogu u programiranju
- prepoznati ulazne i izlazne vrijednosti
- objasniti ulogu ulazne i izlazne vrijednosti u programiranju
- opisati osnovne korake u programiranju
- primijeniti pridruživanje varijablama
- koristiti se pridruživanim varijablama za pohranu i upravljanje podatcima
- objasniti što je petlja
- razumjeti dvije vrste petlji
- primijeniti petlje bez logičkog uvjeta za iteraciju s pomoću nizova podataka i ponavljanja naredaba
- koristiti se brojačem petlje za praćenje ponavljanja petlje.

PowerPoint prezentacija

Snimka 3. radionice

Varijabla

Varijabla označava nešto promjenjivo. Ona je dio memorije u koju se pohranjuje neki izmjenjivi podatak.



Svake godine vrijednost će se promijeniti, stoga su godine varijabla, odnosno veličina koja poprima različite vrijednosti.

Svaki program radi s pomoću ulaznih i izlaznih vrijednosti.

vrijednosti varijable

ulazne vrijednosti	Ono su što korisnik unosi u računalo (tipkovnicom, mišem, prstom ili drugom ulaznom jedinicom).
izlazne vrijednosti	Rezultat su rada nekog programa (pomak pokazivača na ekranu, kretanje lika u računalnoj igri, prikaz slike na zaslonu itd.).

Rad svakog programa mogao bi se predočiti u **tri** osnovna koraka:

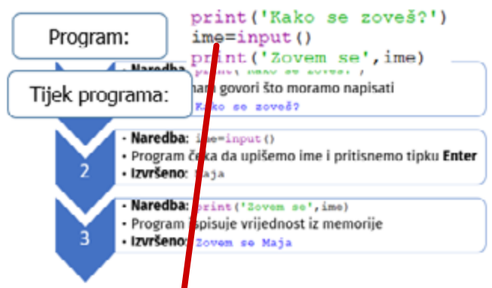
osnovni koraci u programiranju

ulaz	To su ulazne vrijednosti (zadaju se naredbom Pridruživanje).
obrada	Pridruživanje je ulaznih vrijednosti, a rezultat toga su izlazne vrijednosti.
izlaz	Izlazne su vrijednosti nastale kao rezultat obrade tih podataka (naredba Print).

primjer

Zadatak 1. Izradite program koji unosi i ispisuje imena.

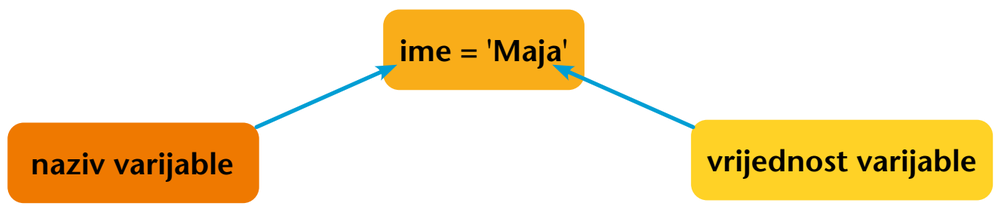
primjer uporabe varijable



Za unos ulaznog podatka imena upotrijebljena je samo naredba **Input**. Naredbom Pridruživanje ime=input() omogućeno je upisivanje imena čija se vrijednost sprema u memoriju ime. U memoriju ime mogu se upisati različita imena i različite vrijednosti, pa je ime promjenjiva veličina, tj. **varijabla**.

Varijabla je lokacija u memoriji koja ima simboličan naziv, a može joj se pridružiti neka vrijednost (broj, tekst, simboli). Ime varijable je nepromjenjivo, a vrijednost se u bilo kojem trenutku može promijeniti u neku drugu vrijednost. Svakim unošenjem nove vrijednosti u varijablu briše se stara vrijednost. Varijabla je jednoznačno određena svojim nazivom i vrijednošću kao na sljedećoj slici.

odrednice varijable



Broj varijabla u stvaranju programa je **neograničen**, ali svaka varijabla mora imati drukčiji naziv. Za naziv varijable ne smiju se upotrebljavati naredbe i funkcije Pythona (npr. Print, Input, `int`). Valja znati da su „ime” i „Ime” različite varijable jer Python razlikuje velika i mala slova.

određivanje naziva varijable

Zadatak 2. Zadatak je izraditi tri varijable i program koji zbraja dva broja te ispisuje rezultat na zaslonu.

zadatak za uvježbavanje uporabe varijable

Program:

```
a=input('Upiši prvi broj:')
b=input('Upiši drugi broj:')
zbroj=a+b
print('Zbroj je:',zbroj)
```

Rezultat:

```
Upiši prvi broj:14
Upiši drugi broj:24
Zbroj je: 1424
```

- Program je umjesto zbroja izvršio spajanje brojeva.
- Python podatke doživljava kao riječi, a ne kao brojeve.

Ulazne podatke treba pretvoriti u brojeve, a potom ih zbrojiti. To se radi s pomoću funkcije `int`. Umjesto a+b, piše se int(a)+int(b):

obrada podataka

```
a=input('Upiši prvi broj:')
b=input('Upiši drugi broj:')
zbroj=int(a)+int(b)
print('Zbroj je:', zbroj)
```

Naredba **Input** sada se može zapisati i ovako: a=int(input('Upiši prvi broj')). Taj način zapisa zove se ugniježđena naredba. **Ugniježđena naredba** znači „naredba u naredbi". Pri njezinoj provedbi najprije će se provesti unutarnja naredba, a onda vanjska.

funkcija `int` i ugniježđena naredba

```
a=int(input('Upiši prvi broj:'))
b=int(input('Upiši drugi broj:'))
zbroj=a+b
print('Zbroj',a,'+',b,'=',zbroj)
```

Petlja

Petlja je dio programa koji se ponavlja. Postoje **dvije vrste** petlji:

petlja i vrste petlji

- petlje bez logičkog uvjeta
- petlje s logičkim uvjetom.

Petlje bez logičkog uvjeta su vrsta petlje u kojoj se unaprijed postavlja broj ponavljanja. One imaju **konačan broj ponavljanja** te uvijek počinju određenim naredbama kojima je definiran broj ponavljanja. U Pythonu se za petlju bez logičkog uvjeta upotrebljava naredba **For**.

petlje bez logičkog uvjeta

primjer

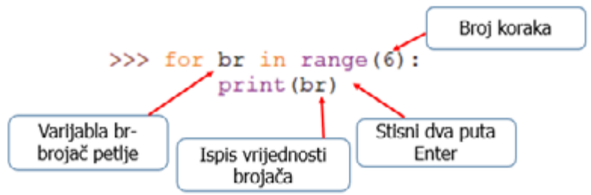
Zadatak 3. Ispišite brojeve od 0 do 10

primjer petlje bez logičkog uvjeta

Program

```
print(0)
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

Kraći program



- Nakon dvotočke prelaskom u novi red sljedeći redak se uvukao.
- Da bi se u konzoli izvršila gornja naredba, nakon drugog retka treba dva puta stisnuti tipku Enter.

Da bi petlja za koju se upotrebljava naredba For u svakom trenutku znala na kojem je koraku ponavljanja, pomaže joj **kontrolna varijabla** petlje ili **brojač**, npr. br in range (6). Varijabla **br** je brojač petlje. Brojač se **upotrebljava tako** da se postavi **početna vrijednost** i **broj koraka** izvođenja petlje. Broj ponavljanja može se zadavati izravnim upisivanjem broja ponavljanja u naredbi za petlju ili zadavanjem s pomoću varijable.

primjena brojača

Program može sadržavati onoliko petlji koliko je potrebno uz napomenu da u radu s petljama treba paziti kako se ne bi stvorila **beskonačna petlja – program koji pokreće sam sebe i ne završava**. Beskonačna petlja je korisna u nekim sustavima koji svoj posao obavljaju bez stajanja.

ograničenje nastajanja beskonačne petlje

Treba uvijek paziti na to da u većini programskih jezika, pa tako i u Pythonu, naredba For počne od broja 0.

Zadatak 4. Ispišite brojeve od 0 do 10.

zadatak za uvježbavanje petlje bez logičkog uvjeta

```
for br in range (11):
    print (br*2)
```

Broj ponavljanja u petlji može se dati izravno ili zadavanjem varijable unesene u program (tipkovnicom ili iz samog programa).

Zadatak 5. Izradite program koji računa zbroj prvih n brojeva.

```
print('Koliko brojeva želiš zbrojiti?')
n=int(input('Upiši broj:'))
zbroj=0
for br in range(1, n+1):
    zbroj=zbroj+br
print('Zbroj prvih',n,'prirodnih brojeva je:', zbroj)
```

Ako na memorijskoj lokaciji zbroj možda već postoji neka vrijednost, program bi tu vrijednost zbrojio s vrijednošću varijable **br**, pa upisivanjem:

zbroj = 0 *osiguravamo da se to ne dogodi.*

U rasponu **range (1, n+1)** završna vrijednost **n** treba biti povećana za 1 kako bi se dobio točan raspon jer Python završnu vrijednost uvijek smanji za 1.

Zadatak 6. Izradite program koji ispisuje prvih n neparnih brojeva.

```
n=int(input('Upiši broj n:'))
for br in range (1,2*n,2):
    print(br)
```

```
for br in range(a,b,k):
```

Varijabla br – brojač petlje

Početna vrijednost

Završna vrijednost + 1

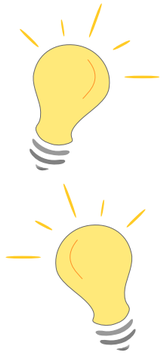
Korak brojača petlje

n=3	n=5	n=10
1,3,5	1,3,5,7,9	1,3,5,7,9,11,13,15,17,19

Provjerite i primijenite

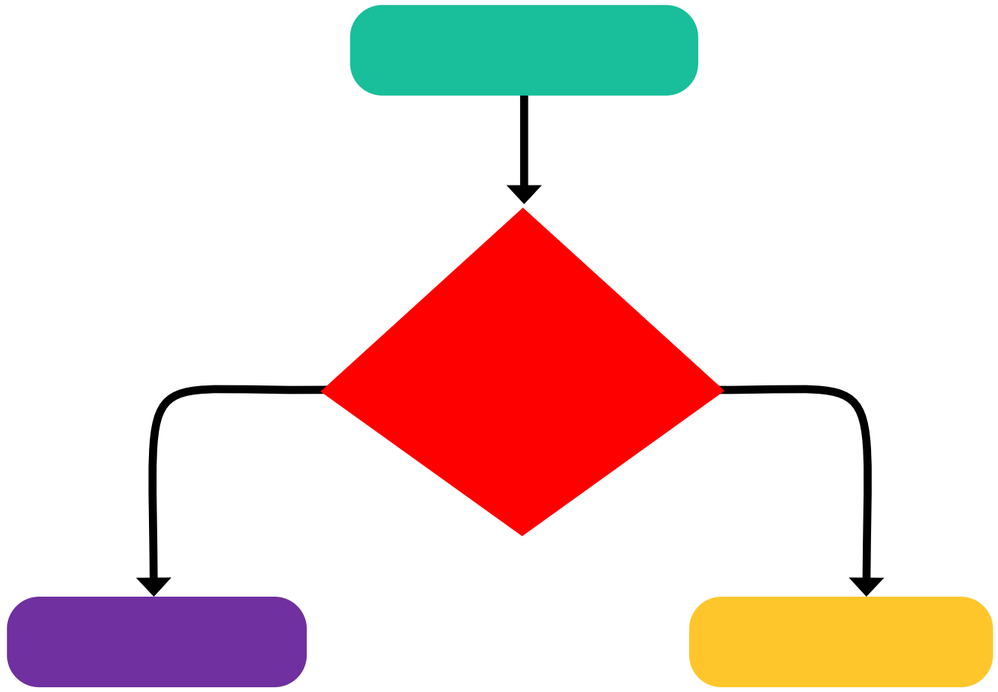
Zadatak 1. Izradite program koji ispisuje prvih n parnih brojeva.

Zadatak 2. Izradite program koji ispisuje brojeve od 9 do 0.



Za one koji žele saznati više

Zadatak: Izradite program u kojemu ćete korisnika upitati koliki je broj ponavljanja uz uvjet pogađanja i nakon toga program će ponoviti poruku „Pogodio” toliko puta uz provjeru rezultata.



4. Osnovni sastavni dijelovi kontrole tijeka

Nakon što ste u četvrtoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o osnovnim sastavnim dijelovima kontrole tijeka, moći ćete:

- razumjeti što je naredba If
- objasniti ulogu naredbe If u programiranju
- prepoznati logičke uvjete
- koristiti se operatorima usporedbe za postavljanje uvjeta u naredbi If
- razumjeti kako se provjeravaju dva uvjeta istodobno uporabom operatora and i or u naredbi If
- objasniti kako se koristiti naredbom If-else za donošenje odluke kad postoji više ishoda, ovisno o uvjetu
- primijeniti naredbu If-else za provjeru uvjeta i izvođenje odgovarajućeg niza naredaba
- kombinirati nekoliko uvjeta u naredbi If-else
- objasniti ispunjavanje uvjeta u naredbi If-else.

PowerPoint prezentacija

Snimka 4. radionice

Naredba If

If je složena naredba koja ovisi o logičkom uvjetu koji se postavlja. **Logički uvjet** je pitanje koje se postavlja primjenom operatora usporedbe i provjerava se istinitost izraza. Ako je logički uvjet **istinit**, izvršava se niz naredaba koji se postavlja nakon uvjeta. Ako **nije istinit**, zaobilazi se grananje i nastavlja izvođenje programa.

U Pythonu a=0 znači „varijabli a pridruži vrijednost nula”, a a==0 znači „usporedi vrijednost varijable s nulom”.

Operator	Značenje
=	jednako
≠	različito (nije jednako)
<	manje
≤	manje ili jednako
>	veće
≥	veće ili jednako

Dva se uvjeta mogu provjeriti istodobno s pomoću jednog bloka **naredbe If** tako da se u logički uvjet postavi operator **and** (I) ili **or** (ILI).

operator ***and***	provjerava dva uvjeta istodobno kad su oba ispunjena, provodi naredbe unutar uvjeta
operator ***or***	provjerava dva uvjeta istodobno i, kad je barem jedan ispunjen, provodi naredbe unutar uvjeta

Grananje If-else

Logička naredba grananja **If-else** složena je naredba koja služi za donošenje odluke kad postoje **dva ili više ishoda**, ovisno o odgovoru na pitanje postavljeno u uvjetu.

Za razliku od naredbe If, ako uvjet nije ispunjen, dio skripta unutar odluke neće se zaobići, nego će se provesti neki drugi niz naredaba.


```
If uvjet:
        naredba1
        naredba2
        ...
else:
    naredba1
    naredba2
    ...
```

primjeri upotrebe naredbe If-else

primjer

Zadatak 1. Izradite program koji provjerava je li korisnik dobro zbrojio brojeve.

```
odg=int(input('Koliko je 9+11?'))
if odg==20:
    print('Bravo! Točan odgovor!')
else:
    print('Žao mi je, netočan odgovor!')
```

1. Postaviti pitanje i unos odgovora:

```
odg=int(input('Koliko je 9+11?'))
```

2. Postaviti uvjet:

```
if odg==20:
```

3. Ako je uvjet ispunjen, tj. odgovor točan, ispiši poruku "Bravo! Točan odgovor!":

```
if odg==20:
    print('Bravo! Točan odgovor!')
```

4. Ako uvjet nije ispunjen, tj. odgovor je netočan, ispiši poruku "Žao mi je, netočan odgovor!":

```
else:
    print('Žao mi je, netočan odgovor!')
```

Taj primjer detaljno pokazuje korisnost naredbe If-else. Korak po korak u dijagramu tijeka može se vidjeti kako svaki red naredbe izvodi i provjerava uvjete postavljene u naredbi. Umjesto dviju naredaba If, upotrebljava se jedna naredba If-else.

Može li se postaviti više naredaba If-else te na taj način riješiti više slučajeva?

zadatci za uvježbavanje upotrebe naredbe If-else

Zadatak 2. Izradite program koji od korisnika zahtijeva da pritisne tipku A za nastavak. Ako se klikne ili učini bilo što drugo, lik ispisuje tekst pritisni tipku A. Ako korisnik klikne na tipku A, računalo ispisuje tekst: „Bravo! Pritisnuo si tipku A!", a ako se klikne ili učini bilo što drugo, računalo ispisuje tekst: „Šteta, pogriješio si. Nisi pritisnuo tipku A."

```
slovo=str(input('Pritisni tipku a:'))
if slovo=='a':
    print ('Bravo! Pritisnuo si tipku a.')
else:
    print ('Šteta, pogriješio si. Nisi pritisnuo tipku a.')
```

Zadatak 3. Izradite program koji unosi broj te provjerava je li broj paran.

```
a=int(input('Upiši broj:'))
ostatak=a%2
if ostatak==0:
    print(a, 'je prani broj')
else:
    print(a, 'je neparni broj')
```

prednosti uporabe naredaba If-else

Uporabom naredaba If-else može se kombinirati nekoliko uvjeta. Provode se samo naredbe koje slijede prvi uvjet za koji se ustanovi istinitost, a sve se druge preskaču. Nakon konačne naredbe Else, naredbe se provodeako nijedan od uvjeta nije istinit.

Provjerite i primijenite

Zadatak 1. Provjerite kod programa koji unosi dva broja te provjerava njihovu djeljivost.

```
a=int(input('Upiši broj a:'))
b=int(input('Upiši broj b:'))
ostatak=a%b
if ostatak==0
    print(a,'je djeljiv s', b)
else:
    print(a,'nije djeljiv s', b)
```

Zadatak 2. Provjerite kod programa koji unosi broj te provjerava je li djeljiv sa sedam.

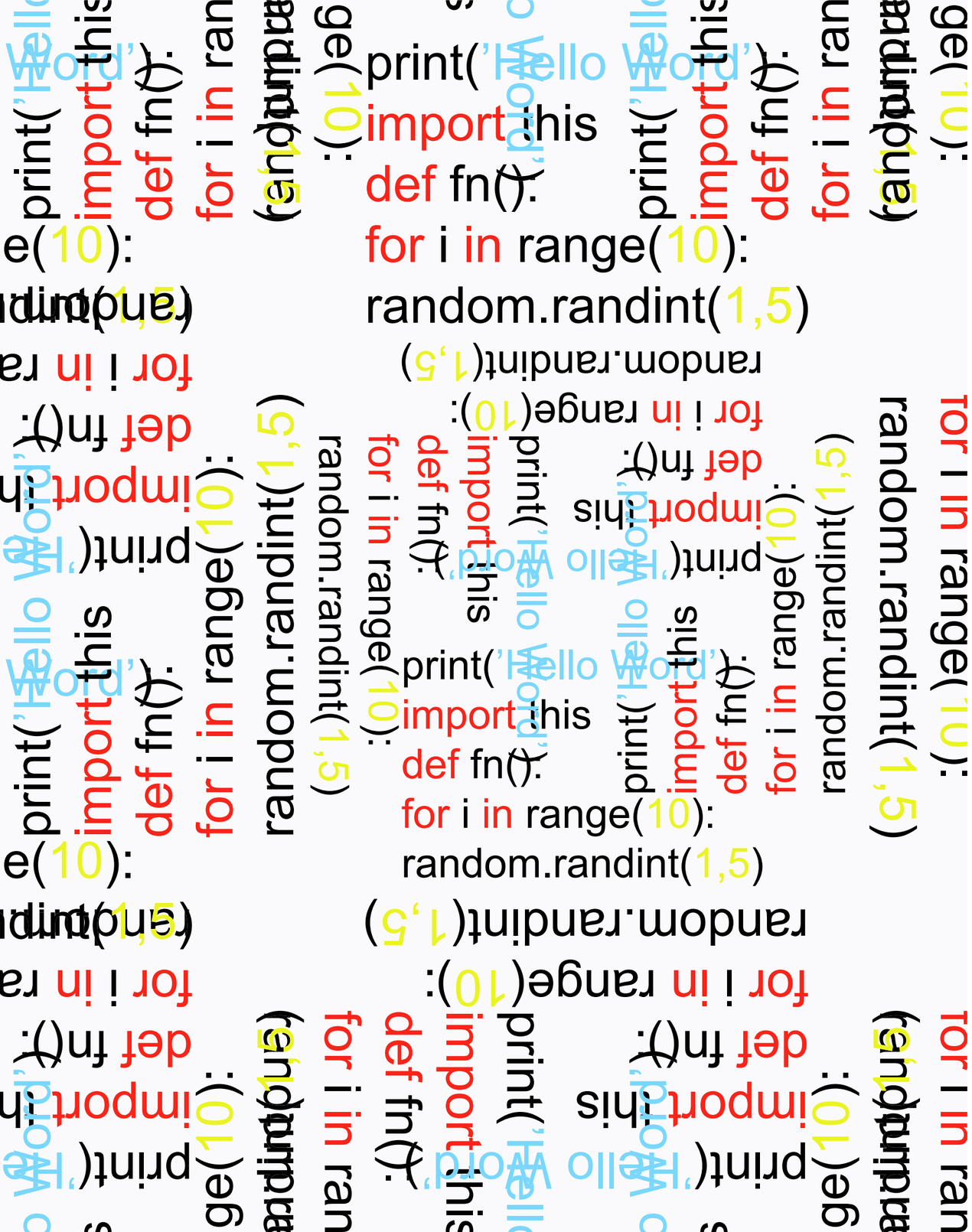
```
a=int(input('Unesi broj'))
if a%7==0
    print('Broj',a, 'je djeljiv sa sedam.')
else:
    print('Broj',a, 'nije djeljiv sa sedam')
```

Zadatak 4. Izradite program koji unosi broj te provjerava je li broj pozitivan.

Za one koji žele saznati više

Zadatak 1. Izradite program koji unosi duljine dviju stranica. Ako su unesene duljine jednake, izračunat će površinu kvadrata, a ako su različite izračunat će površinu pravokutnika.

Zadatak 2. Izradite program koji unosi dva broja, zbraja ih te provjerava je li zbroj veći od 20.



5. Funkcije

Nakon što ste u petoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o funkcijama, moći ćete:

- definirati funkciju i njezin naziv
- razumjeti i koristiti se različitim vrstama parametara u funkcijama (obvezatni parametri, opcionalni parametri i parametri s pretpostavljenom vrijednošću)
- razumjeti različite vrste povratnih vrijednosti u funkcijama (jedna vrijednost, više vrijednosti i nula vrijednosti)
- koristiti se različitim vrstama povratnih vrijednosti u funkcijama (jedna vrijednost, više vrijednosti i nula vrijednosti)
- pozivati funkciju
- prosljeđivati argumente
- razumjeti što su globalne i lokalne varijable i kako se njima koristiti u funkcijama
- koristiti se funkcijama za strukturiranje koda i izbjegavanje ponavljanja koda
- razumjeti što su rekurzivne funkcije i kako se koristiti rekurzivnim funkcijama za rješavanje problema
- razumjeti koncept modula i kako se koristiti funkcijama iz drugih modula u Pythonu
- koristiti se ugrađenim funkcijama u Pythonu
- razumjeti funkcionalnost ugrađenih funkcija u Pythonu.

PowerPoint prezentacija

Snimka 5. radionice

Odrednice funkcije

svrha funkcije

Funkcija je dio programskoga kôda koji je organiziran prema određenoj logičkoj cjelini. Svrha je korištenja funkcijom to da se dijelovi programske logike koji se ponavljaju u programskom kôdu napišu samo jednom, i to unutar funkcije. Uporabom funkcija jedan te isti kôd može se iskoristiti neograničen broj puta jednostavnim pozivom funkcije koja je u nekom trenutku potrebna. Takav pristup omogućuje veću preglednost i organizaciju programskoga kôda. Isto tako znatno je lakše nakon pisanja pročitati programski kod koji je zapisan koristeći se funkcijama. Funkcijama se koristimo kao osnovnim građevnim sastavnicama te se na temelju njih jednostavnije mogu slagati složeniji blokovi koda a da se pritom zadrži čitkost.

Dakle, može se reći da funkcija služi:

- minimiziranju koda
- postizanju veće razine apstrakcije/čitkosti koda
- ponovnoj uporabi koda / njegovu optimiranju.

savjet za kreiranje funkcije

Glavna je misao vodilja pri kreiranju funkcija da odsječak koda unutar funkcije bude što jednostavniji, tj. podijeljen na što manje logičke cjeline. Takva organizacija omogućuje veću ponovnu iskoristivost programskoga kôda.

unaprijed napisane funkcije

Funkcije mogu biti unaprijed napisane. Ako su funkcije unaprijed napisane, to su funkcije koje su ugrađene u sâm programski jezik ili su dio standardnoga ili uključenog paketa. Pojam paketa detaljnije je obrađen u poglavlju *8. Moduli i paketi*

obvezatne sastavnice funkcije

Funkcija nužno mora imati naziv, nula ili više ulaznih parametara te nula ili više izlaznih vrijednosti.

```
def ime_funkcije (popis parametara)
        blok_naredbi
        return vrijednosti
```

Funkcija se sastoji od zaglavlja funkcije i tijela funkcije. U nastavku slijedi primjer definicije funkcije:

sastavnice i primjer funkcije

primjer

```
def sumaBrojeva():
    # tijelo funkcije
```

Svaka funkcija koju želimo primijeniti počinje ključnom riječju def nakon čega slijedi naziv funkcije. U navedenu slučaju naziv funkcije je sumaBrojeva(). Naziv funkcije može biti proizvoljan, no treba pripaziti da se funkcija ne zove poput neke od ključnih riječi u Pythonu ili pak imenom neke druge funkcije. Nakon naziva funkcije obvezatne su zagrade i nakon zagrada dvotočka.

opis funkcije u navedenu primjeru

Tijelo funkcije mora biti uvučeno – u suprotnom Pythonov tumač javlja pogrešku pri pokretanju.

savjet za pisanje funkcije

Za razliku od drugih programskih jezika koji se koriste vitičastim zagradama ili ključnim riječima za razlikovanje programskih blokova, *Python* se koristi uvlačenjem. U navedenu primjeru tijelo funkcije je programski blok koji je na neki način potrebno omeđiti kako bi se pri izvršavanju znalo koje sve linije programskoga kôda potpadaju pod funkciju.

razlikovanje programskih blokova uvlačenjem

Python se koristi uvlačenjem kao načinom razlikovanja programskih blokova. Povećanje uvlačenja znači da dolazi novi, ugniježđeni blok, a smanjenje označava kraj trenutačnog bloka programskoga kôda.

Poziv funkcije

Nakon što je funkcija definirana i upisana u tumač te sadržava ispravno napisane naredbe (logički i sintaksom), može se pozivati tako da se napiše njezino ime te u oblim zagradama navedu odgovarajući argumenti (vrijednosti parametara koje su ulaz u funkciju). Argumenti funkcije bit će obrađeni u nastavku.

Sintaksa poziva funkcije bez parametara i bez povratnih tipova prikazana je u sljedećem primjeru.

primjer

```
>>> def moja_funkcija():
...     print("Pozdrav iz funkcije")
...
...
>>> moja_funkcija()
Pozdrav iz funkcije
>>>
```

primjer poziva funkcije

Kao što se može vidjeti iz navedenog primjera, najprije je napisana funkcija (zaglavlje i tijelo funkcije) te je tek nakon toga izveden sâm poziv funkcije `moja_funkcija()`.Bitno je primijetiti to da definicija funkcije mora biti napisana ispred samog poziva funkcije. Ako je poziv funkcije u programskom kôdu ispred definicije funkcije, pri pokretanju programa dogodit će se pogreška. U sljedećem primjeru može se vidjeti pozivanje funkcije prije njezine definicije.

Korišten je IDLE u interaktivnom načinu rada, no ista bi se pogreška pojavila u pokretanju iz konzolnog načina rada.

primjer

```
>>> moja_funkcija()
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    moja_funkcija()
NameError: name 'moja_funkcija' is not defined
>>>
>>> def moja_funkcija():
...     print("Pozdrav iz funkcije")
...
...
>>>
```

primjer pisanja poziva funkcije prije njezine definicije

Za davanje naziva funkcijama preporučuju se sljedeća pravila:

pravila za davanje naziva funkcijama

- sve treba pisati malim slovima
- ključne riječi i razmaci nisu dopušteni
- umjesto razmaka upotrebljava se donja crta (engl. *underscore*).

Parametri funkcije

Da bismo funkcijama mogli slati različite vrijednosti na temelju kojih će one provoditi određenu obradu, u zaglavlju funkcije u oblim zagradama upisuju se parametri koje funkcija prima.

Formalni parametri funkcije mogu se mijenjati ovisno o potrebi. U istom programu može se nekoliko puta pozvati jedna te ista funkcija, primjerice `funkcija kvadrat()`.

formalni parametri

primjer

Funkcija `kvadrat()` prima samo jedan parametar i, ovisno o vrijednosti tog parametra, izračunava kvadrat unesena broja. Prvi put se poziva funkcija `kvadrat(3)`, a drugi put `kvadrat(5)`. Primjećuje se da rezultat funkcije pri prvom pozivu mora biti 9, a rezultat funkcije pri drugom pozivu mora biti 25, tj. rezultat ovisi o vrijednostima koje je funkcija primila.

Pozicija u popisu varijabla važna je za pozivanje. Tip se implicitno dinamički zaključuje.

„a”,2,

„a”,”b”

Primjer

primjer sljedivosti

primjer

```
>>> def zbroji(a,b):
...     return a+b
...
>>> zbroji(3,2)
5
```

Formalni parametri su parametri, tj. varijable koje se nalaze u definiciji funkcije.

formalni parametri

```
def kvadrat(x):
```

Varijabla x je formalni parametar. Ovdje je prikazan primjer definicije funkcije kvadrat(). Vidljivo je da funkcija prima jedan parametar pod imenom x. U tijelu funkcije na temelju vrijednosti u varijabli x izračunava se i ispisuje kvadrat prenesene vrijednosti.

```
def kvadrat(x):
    print(x*x)
```

Funkcije mogu primati i više parametara. Na primjer, ako funkcija mora vratiti zbroj tri broja, a sve tri vrijednosti su funkciji predane kao parametri, definicija funkcije izgleda ovako:

```
def suma(var1, var2, var3):
    print(var1 + var2 + var3)
```

U idućem je primjeru vidljiva definicija funkcije suma() s dva formalna parametra var1 i var2. Nakon same funkcije (zaglavlja i tijela funkcije) nalazi se poziv funkcije suma(10, 20). U pozivu se prenose u funkciju vrijednosti 10 i 20. Vrijednost 10 sprema se u varijablu var1, a vrijednost 20 sprema se u varijablu var2. Pri pozivu funkcije formalni parametri zamjenjuju se stvarnim argumentima, tj. konkretnom vrijednošću.

primjer poziva funkcije s formalnim parametrima

primjer

```
def suma(var1, var2):
    print(var1 + var2)

suma(10, 20)
```

```
Izlaz:
30
```

U gornjem dijelu objašnjenja uveden je novi pojam – argument. Riječi parametar i argument često se miješaju iako je zapravo riječ o jednom te istom pojmu koji se gleda iz različitih perspektiva.

argument

Parametri funkcije – to su varijable koje su napisane i rabe se u samoj funkciji (zaglavlje funkcije i tijelo funkcije). Parametri funkcije mogu se smatrati običnim varijablama.

parametri funkcije

Argumenti funkcije – to su vrijednosti koje se rabe pri pozivu funkcije.

argumenti funkcije

Python omogućuje da se unaprijed zadaju predefinirane vrijednosti parametara funkcije. To je omogućeno jer broj parametara funkcije ne mora uvijek odgovarati broju argumenata koji se šalju pri pozivu funkcije.

unaprijed zadane vrijednosti parametara

U nastavku je prikazan način s pomoću kojega se parametru funkcije postavlja unaprijed zadana vrijednost. U donjem slučaju ta je unaprijed zadana vrijednost pridružena varijabli (formalnom parametru funkcije) var3 na

vrijednost 0. Varijabla var3 poprimit će vrijednost 0 samo ako se pri pozivu funkcije prenesu dva argumenta, a ne sva tri koliko ih najviše može biti. Ako se prenesu tri argumenta, tada varijabla var3 poprima vrijednost trećega prenesenog argumenta.

primjer načina s pomoću kojega se parametru funkcije postavlja unaprijed zadana vrijednost

primjer

```
def suma(var1, var2, var3=0):
    print(var1 + var2 + var3)

suma(10, 20)
```

```
Izlaz:
      30
```

U navedenu primjeru pri pozivu funkcije suma() prenesene su vrijednosti 10 i 20, tj. prenesena su samo dva argumenta. U ovom slučaju varijabla var3 poprima predefiniranu vrijednost, a to je vrijednost 0.

Varijable var1 i var2 poprimaju vrijednost iz poziva funkcije – 10 i 20.

primjer

```
def suma(var1, var2, var3=0):
    print(var1 + var2 + var3)

suma(10, 20, 500)
```

```
Izlaz:
      530
```

U ovom primjeru prenose se tri argumenta, tj. vrijednosti. Kako funkcija suma() može poprimiti najviše tri parametra, pri pozivu funkcije suma() prenijeli smo tri argumenta u funkciju. Varijabla var3 neće poprimiti predefiniranu vrijednost 0, već će poprimiti vrijednost trećega prenesenog argumenta iz poziva funkcije suma(), tj. vrijednost 500.

ulančavanje poziva više funkcija

Nekoliko funkcija može se pozivati iz tijela jedne funkcije te se tako postiže ulančavanje i bitno čitkiji kod cijelog programa. Princip s ovim primjerom prikazan je na sljedećoj slici.

```
>>> def vece(broj):
...     print(f"{broj} je veći")
...
...
>>> def manje(broj):
...     print(f"{broj} je manji")
...
```

Najprije se definiraju funkcije na „najnižoj razini”, drugim riječima one funkcije koje sadržavaju najmanje funkcionalnosti i na kojima se grade druge funkcije.

Nakon toga tako definirane i isprobane funkcije mogu se pozivati iz drugih funkcija kao što je prikazano na sljedećem primjeru.

primjer

primjer načina pozivanja iz drugih funkcija

```
>>> def usporedi(a,b):
...     if a>b:
...         vece(a)
...         manje(b)
...     else:
...         vece(b)
...         manje(a)
...
...
>>> usporedi (3,10)
10 je veći
3 je manji
```

Provjerite i primijenite

1. Funkcije možemo podijeliti na dva dijela. Koji su to dijelovi?
2. Ako funkcija prima pet parametara, je li pri pozivu funkcije potrebno navesti svih pet argumenata ili postoji način da se neki argumenti izostave, a ako postoji,koji bi to način bio?
3. Napišite funkciju koja prima dva parametra. Rezultat izračuna funkcije ovaj se put ne ispisuje izravno u funkciji, nego u glavnom dijelu programa. Funkcija mora izračunati rezultat formule:

   (a*a) + (b*b)

   Rezultat spremite u varijablu koja se nalazi u dijelu programskoga kôda u kojem se funkcija poziva te ispišite tu varijablu.
4. Pozovite funkciju koja ne prima nijedan parametar, ali mora izračunati i ispisati zbroj dvaju brojeva. Brojevi neka se dohvate iz glavnog dijela programa preko globalnih varijabli.
5. Prethodni zadatak napišite tako da se ne koristite globalnim varijablama (korištenjem parametara funkcije).
6. ** Izradite program koji će inicijalizirati u varijable n i m dva cijela broja proizvoljnih vrijednosti. Provjerite zadovoljavaju li inicijalizirane vrijednosti uvjet: `0 ≤ n ≤ m`. Ako uvjet nije zadovoljen, tada ispišite poruku: „Nedopuštene vrijednosti!" Ako je uvjet zadovoljen, izračunajte i ispišite binomni koeficijent „m povrh n" pri čemu se koristite sljedećim izrazom:

$$\binom{m}{n} = \frac{m!}{[n! * (m-n)!]}$$

Primjer: 5! izračunava se na način 5*4*3*2.

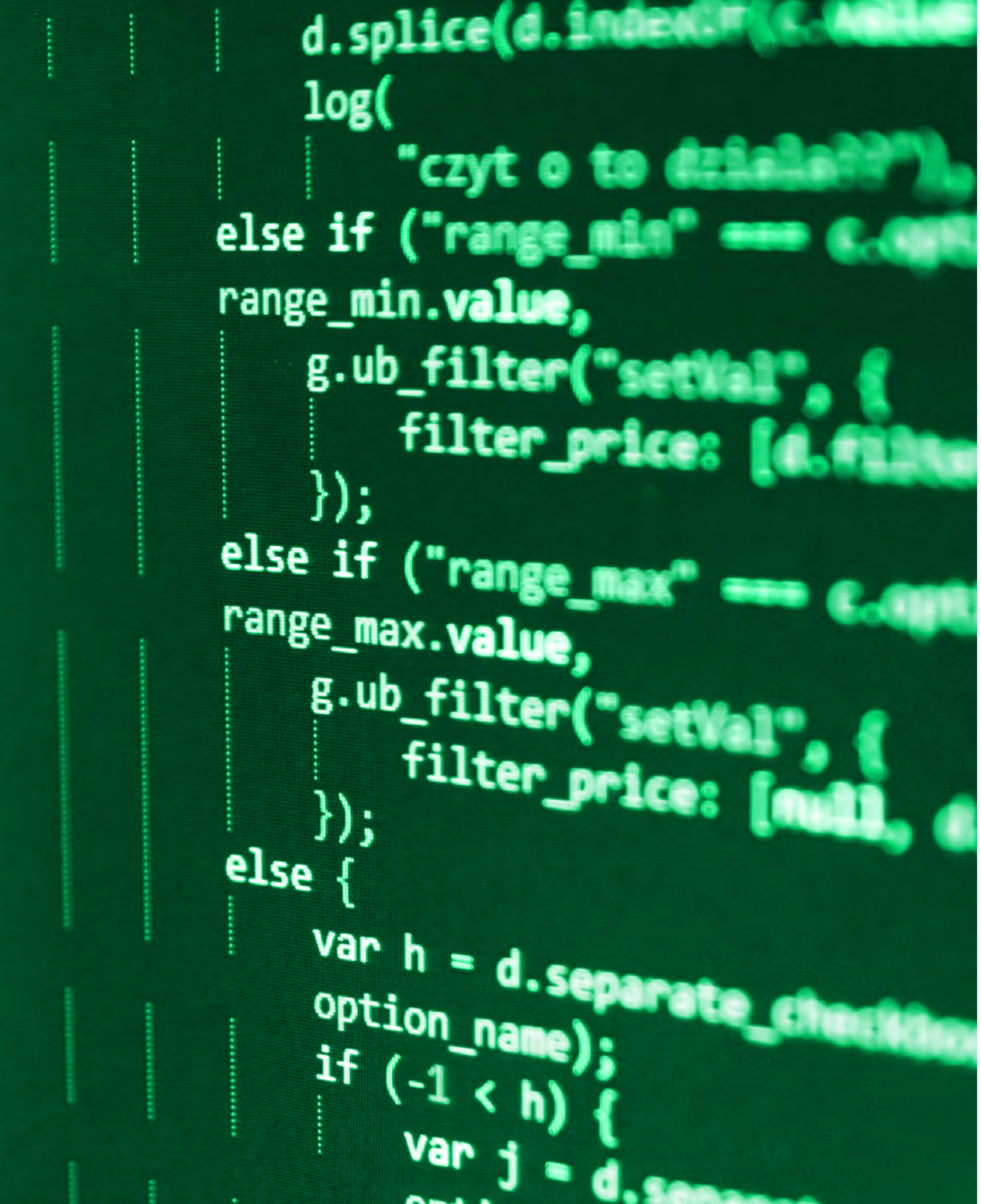
(Napomena: Primijenite funkciju fakt(). Tako primijenjena funkcija izračunava faktorijele, na primjer za poziv funkcije fakt(5) povratna vrijednost je 120. Funkcija se mora pozivati iz glavnog programa za sve tri vrijednosti: m!, n!, (m-n)!).

Za one koji žele saznati više

Nekoliko tema preporučuje se kao samostalno istraživanje za polaznike koji su s lakoćom svladali dosadašnje gradivo dubljom obradom funkcija.

Promjena redoslijeda parametara **Ugnježđivanje funkcija**

Rekurzija



6. Skladišta podataka – varijable

Nakon što ste u šestoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o skladištu podataka – varijablama, moći ćete:

- objasniti pojam vidljivosti varijable u programiranju
- razlikovati lokalne i globalne varijable
- opisati životni vijek varijable
- napraviti dijagram odnosa između globalne i lokalne varijable
- objasniti vidljivost lokalnih varijabla i njihov životni vijek
- objasniti vidljivost globalnih varijabla i njihov životni vijek
- definirati predugrađene varijable
- objasniti preklapanje vidljivosti i nadjačavanje
- primijeniti ključnu riječ global za pristup globalnim varijablama.

PowerPoint prezentacija

Snimka 6. radionice

Vidljivost varijable

Vidljivost varijable je pojam koji određuje iz kojih je dijelova programa neka varijabla dohvatljiva, tj. dostupna. Varijable mogu biti lokalne ili globalne.

Životni vijek varijable je pojam koji je određen trenutkom u kojemu je varijabla nastala u memoriji i trenutkom u kojemu se ta ista varijabla briše iz memorije. Životni vijek varijable ograničen je završetkom bloka programskoga kôda u kojemu je ta varijabla nastala, na primjer, funkcija.

životni vijek varijable

Ugrađene varijable

U širem kontekstu riječ je o varijablama dostupnima unutar razine jedne instancije tumača (engl. *shell*) ili varijablama koje su dostupne u svim instancijama tumača ili konzolnog načina rada.

Dijagram odnosa između ovih pojmova prikazan je u sljedećem primjeru.

dijagram odnosa između globalne i lokalne varijable

Lokalne varijable

Lokalne varijable su varijable koje su korištene unutar tijela funkcija, a u tu kategoriju mogu se svrstati i parametri funkcija. Načelno se može smatrati da se parametri funkcija ponašaju identično kao i obične varijable unutar tijela funkcije.

Lokalne su varijable vidljive samo unutar funkcije u kojoj su prvi put upotrijebljene. Prvom upotrebom može se smatrati izraz pridruživanja vrijednosti varijabla, na primjer var=10.

vidljivost lokalnih varijabla

Ako je varijabla definirana unutar funkcije, nakon što se izvođenje funkcije završi (bilo s pomoću naredbe Return ili pak završetkom tijela funkcije), varijabla se trajno uništava. To znači da sljedeći poziv iste funkcije više ne može vidjeti vrijednost varijable iz prethodnog poziva (jer je varijabla pri izlasku iz funkcije trajno uništena).

životni vijek varijable

primjer programa

primjer

```
8.  def test():
9.  var=5
10.
11. test()
    print(var)
```

```
Izlaz:
     NameError: name 'var' is not defined
```

U glavnom dijelu navedenog programa pozvana je funkcija test(), unutar funkcije test() definirana je varijabla var s vrijednošću 5. Nakon što se izađe iz funkcije (funkcija ništa ne ispisuje, već samo postavlja vrijednost varijable var), u glavnom se programu pokušava ispisati vrijednost varijable var, no varijable var nakon izlaska iz funkcije više nema. Nakon izlaska iz funkcije životni vijek varijable je završio, pa je više nema ni u memoriji, a usto je varijabla var lokalna varijabla te joj se ne može pristupiti iz glavnog dijela programa jer je vidljiva samo unutar funkcije.

Globalne varijable

Uz lokalne varijable postoje i globalne. Globalne su varijable kreirane i korištene u glavnom dijelu programa i njihov životni vijek traje dokle god se program izvršava. Globalne varijable zovu se tako jer im se može pristupati ne samo iz glavnog dijela programa nego i iz funkcija.

primjer

```
def test():
    print(var)

var=5
test()
```

```
Izlaz:
      5
```

primjer programa globalne varijable

U navedenu programu kreirana je varijabla `var` i pridružena joj je vrijednost 5. Nakon toga u sljedećoj je liniji pozvana funkcija `test()`. Funkcija `test()` ispisuje sadržaj varijable var, no primjećuje se da funkcija nema ni formalne parametre ni varijable koje bi bile definirane u njoj, ali svejedno uspijeva ispisati vrijednost varijable var. To znači da je varijabla var globalna varijabla koja je vidljiva iz svih dijelova programa (što uključuje i funkcije).

deklaracija globalnih varijabla

Deklaracija globalnih varijabla može biti napravljena izravno u glavnom tijelu programa, bez stvaranja modularnog i organiziranog koda, u funkcijama/metodama kao u sljedećem primjeru.

primjer

```
>>> globalna_varijabla=100
>>>
>>> print(globalna_varijabla)
100
```

primjer deklaracije globalnih varijabla

Alternativno je moguće učiniti to i unutar bloka funkcije kao što je prikazano u sljedećem primjeru.

primjer

```
>>> def misli_globalno():
...     global pero
...     pero=250
...
...
>>> pero
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    pero
NameError: name 'pero' is not defined
>>> misli_globalno()
>>> pero
250
```

Predugrađene varijable

To su varijable koje su u istom obliku dostupne za čitanje iz bilo koje konzole ili tumača.

Primjer takve varijable je „credits” koja ispisuje niz string s osnovnim informacijama o organizacijama koje podržavaju rad Pythonove zajednice.

primjer predugrađene varijable

primjer

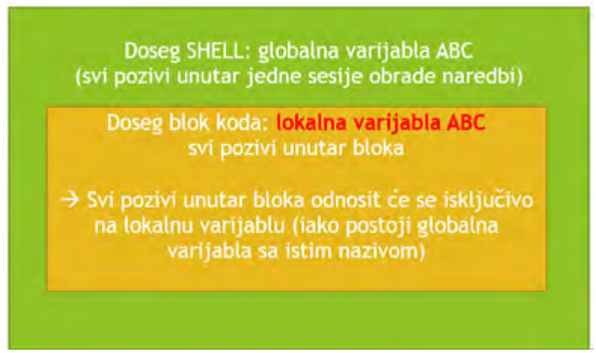
```
>>> credits
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast of thousands
    for supporting Python development.  See www.python.org for more information.
```

Popis svih predugrađenih varijabla i imena prikazan je ovdje:

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning',
 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning',
'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError',
 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError',
'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',
'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError',
'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError',
'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError',
'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError',
'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning',
 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '__build_class__',
 '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__',
 '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray',
'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright',
 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec',
'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter',
'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range',
'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod',
'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

Preklapanje vidljivosti

Ako se u različitim dosezima rabi isti naziv za varijablu, vrijednost varijable šireg dosega prekriva se varijablom užeg dosega.



Ako globalna i lokalna varijabla imaju isti naziv, događa se nadjačavanje, tj. preklapanje vidljivosti u kojoj ime koje je deklarirano „bliže" lokaciji poziva ima veću važnost te se stoga varijabla s manjom važnošću u tom trenutku ne vidi preko poziva istog imena.

Kao što je vidljivo u sljedećem primjeru, definiranjem globalne varijable ona postaje dostupna u kompletnom tumaču.

primjer definiranja globalne varijable

primjer

```
>>> globalna_varijabla=100
>>>
>>> globalna_varijabla
100
```

Kad se globalna varijabla upotrebljava unutar funkcije, ona također preko imena ima istu vrijednost kao i pozivom izvan funkcije.

primjer korištenja ključnom riječju global

primjer

```
>>> def pisi():
...     print (globalna_varijabla)
...
...
>>> pisi()
100
```

U situaciji kad se unutar tijela funkcije definira globalna varijanta s istim imenom kao i globalna varijabla, lokalna će varijabla isključivo u toj funkciji nadjačati poziv i dodjelu vrijednosti kao što je prikazano u sljedećem primjeru.

primjer

```
>>> def pisi2():
...     globalna_varijabla=200
...     print(globalna_varijabla)
...
...
>>> pisi()
100
>>> pisi2()
200
>>> pisi()
100
```

primjer definiranja globalne varijante s istim imenom kao i globalna varijabla

Kako bi se eksplicitno odredilo da se želi iz tijela funkcije djelovati nad globalnom varijablom ili joj se pristupiti, potrebno je koristiti se ključnom riječju global.

primjer

```
>>> def pisi2_bez_preklapanja():
...     global globalna_varijabla
...     globalna_varijabla=200
...     print(globalna_varijabla)
...
...
>>>
```

primjer korištenja s ključnom riječju global

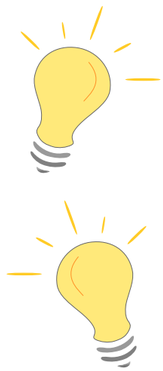
Uzimajući u obzir prethodni primjer, kad se pozove ova funkcija, ona iz svojega tijela promijeni vrijednost globalne varijable.

```
>>> pisi()
100
>>> pisi2_bez_preklapanja()
200
>>> pisi()
200
```

izlazak izvan okvira dosega funkcije

Provjerite i primijenite

1. Kako Python razlikuje globalne i lokalne varijable?
2. Koje su prednosti i mane korištenja globalnim varijablama u Pythonu?
3. Može li se lokalna varijabla iz jedne funkcije upotrijebiti u drugoj funkciji? Zašto?
4. Kako se može spriječiti preklapanje vidljivosti u Pythonu?
5. Provjerite vježbe dostupne na mrežnoj poveznici https://www.w3schools.com/python/gloss_python_local_scope.asp



Za one koji žele saznati više

Polaznicima koji žele proširiti svoje znanje s područja skladišta podataka – varijabla preporučuje se sljedeća tema kao samostalno istraživanje unutar Pythonova razvojnog okružja.

- Koncept *namespace*
- Ugnježđivanje i enkapsulacija
- Metoda nadjačavanja (engl. *override*) – u objektnom programiranju

7. Proces upravljanja memorijom računala – napredno korištenje Pythonom

Nakon što ste u sedmoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o procesu upravljanja memorijom računala – naprednom korištenju programskim jezikom, moći ćete:

- razumjeti kategorije memorije dostupne na računalu
- razumjeti razlike između fizičke radne memorije i radne memorije operativnog sustava
- razumjeti važnosti upravljanja potrošnjom memorije u programiranju
- razumjeti automatski mehanizam za upravljanje potrošnjom memorije u modernim programskim jezicima
- pokazati životni ciklus varijabli/objekata u programskom jeziku Python
- objasniti globalnu varijablu i njezino zauzimanje memorije u shellu
- poznavati naredbe Del za ručno uklanjanje varijabla
- razumjeti podjele objekata na tri klase (generacije) u automatskom mehanizmu za upravljanje memorijom u Pythonu

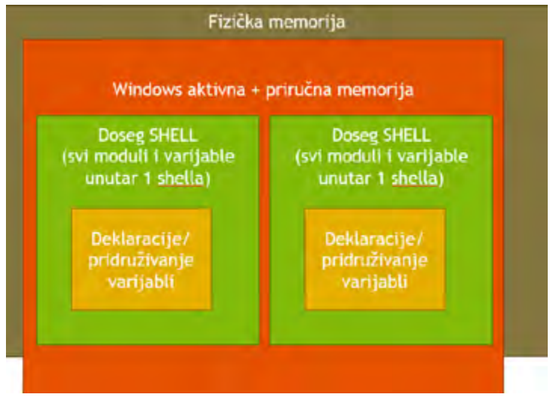
PowerPoint prezentacija

Snimka 6. radionice

Upravljanje memorijom računala

kategorije memorije

Sva memorija koja je dostupna na računalu dijeli se na nekoliko kategorija što je prikazano u sljedećem dijagramu.



razlika između fizičke radne memorije i radne memorije Operativnog sustava

Fizička radna memorija je konačnog dosega, no radna memorija kojom se koristi Operativni sustav (u ovom slučaju Windows) ima veći zbirni iznos nego isključivo fizička radna memorija zbog korištenja straničnom datotekom.

Unutar memorije kojom se koristi Operativni sustav nalaze se sve aplikacije uključujući i tumač (engl. *shell*) u kojemu se nalaze sve varijable kojima se program koristi.

primjer

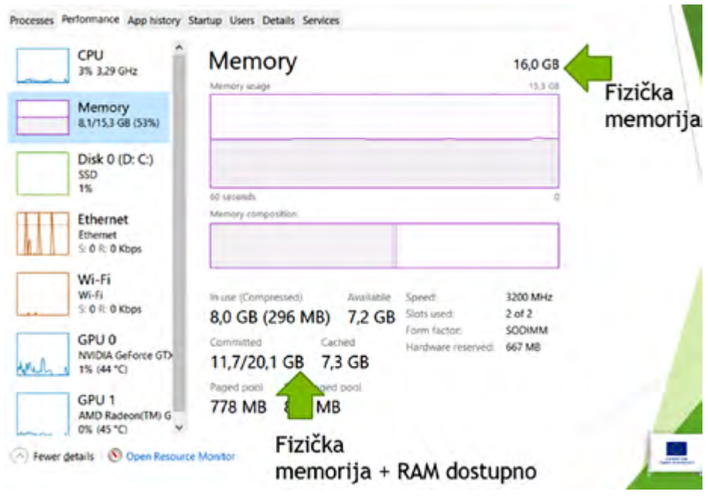
Ako se uporabom potroši više memorije od fizički dostupne i veličine definirane u postavkama za straničnu datoteku, a proces za uklanjanje memorije kojom se ne koristi, događaju se razne sistemske pogreške – primjer je prikazan na sljedećoj slici.

primjer sistemske pogreške

primjer

Jednako je vidljivo i na upravitelju zadataka unutar operativnog sustava.

primjer sistemske pogreške na upravitelju zadataka unutar operativnog sustava



Kako je već pokazano u prijašnjim dijelovima ovog poglavlja, ako se potrošnjom memorije ne upravlja na kvalitetan način, posljedice za rad našeg programa, ali i cijelog računala mogu biti izrazito negativne. Zbog toga je u današnjim modernim programskim jezicima upravljanje potrošnjom memorije automatsko. U prvim programskim jezicima upravljanje potrošnjom memorije bilo je ručno jer nisu postojali automatski mehanizmi.

upravljanje potrošnjom memorije

Ručno upravljanje potrošnjom memorije često je dovodilo do sljedećih situacija:

nedostatci ručnog upravljanja potrošnjom memorije

- vrlo česte pogreške u kojima se zaboravljalo oslobađati memoriju
- vrlo česte pogreške u kojima se memorija prerano oslobađala.

Kako je spomenuto, moderni programski jezici (uključujući Python) već imaju ugrađeno upravljanje potrošnjom memorije te se ti mehanizmi neovisno o korisniku brinu o:

prednosti programskog upravljanja potrošnjom memorije

- potrebama kada treba ukloniti neku varijablu koja se više ne rabi
- alociranju sistemske memorije kako bi bilo dovoljno memorije za objekte/podatke u memoriji tumača (engl. *shell*).

Skupni naziv za sve mehanizme za upravljanje potrošnjom memorije engleski je termin Garbage collection, tj. Uklanjanje nekorištenih objekata iz memorije.

uklanjanje nekorištenih objekata iz memorije

Automatski mehanizam za upravljanje potrošnjom memorije koji uklanja nekorištene objekte iz memorije može taj zadatak obavljati s nekoliko razina agresivnosti uklanjanja objekata koje variraju od minimalne do maksimalne. Svaki od tih pristupa ima svoje pozitivne i negativne strane.

automatski mehanizam za upravljanje potrošnjom memorije

preagresivno (maksimalno) uklanjanje objekata	Vodi do sporijeg izvršavanja zbog ponovnog učitavanja ako je objekt ponovno potreban, ali je zahvaljujući njemu gotovo uvijek dostupna dovoljna količina memorije.
konzervativno uklanjanje objekata (minimalno)	Vodi do velike uporabe resursa računala, ali će zahvaljujući njemu jednom stvorene varijable uvijek biti dostupne.

Automatski mehanizam za upravljanje potrošnjom memorije događa se prema unaprijed postavljenim postavkama umjerenim tempom u pozadini rada i izvršavanja programa. U situacijama u kojima programer unaprijed zna da će biti velika potrošnja memorije zbog jednokratne uporabe nekih objekata/varijabla moguće je ručno ubrzati oslobađanje memorije. Alati koji služe za praćenje potrošnje memorije/brzine izvršavanja programa zovu se na engleskom jeziku profiler.

djelovanje automatskog mehanizma za upravljanje potrošnjom memorije

Životni vijek varijabla/objekata u Pythonu

Svaki objekt koji se rabi u programskom jeziku Python prolazi tri faze u svojem životnom ciklusu: fazu stvaranja, fazu korištenja te fazu uništenja.

Stvaranje (engl. *Instantiation*) je prva faza u životnom ciklusu varijable, tj. objekta u programskom jeziku Python koja počinje čim se u programskom kodu prvi put navede ime te varijable. Prvi korak u ovoj fazi kroz koju prolaze svi objekti je konstrukcija. U primjeni to znači da se poziva funkcija, tj. metoda zvana konstruktor. Uobičajen sadržaj rada metode konstruktora je:

stvaranje objekata

- **prvi korak** – rezerviranje memorije
- **drugi korak** – dodjela vrijednosti.

Nakon ovoga slijedi aktivno korištenje varijablom, tj. objektom. Tijekom faze korištenja svaki objekt ima jednu ili nekoliko referencija, tj. poziva iz drugih odsječaka programskoga koda.

korištenje objektom

Na kraju životnog ciklusa svake varijable, tj. objekta u programskom jeziku Python javlja se faza koja se zove destrukcija.

uništavanje objekata

Slično kao kod metode, tj. funkcije koja se poziva kod stvaranja objekata, i kod uništavanja objekata poziva se metoda koja se zove destruktor i koja oslobađa memoriju koju je varijabla ili objekt zauzeo tijekom svojega životnog ciklusa. Destruktor se poziva ili ručno ili mehanizmom automatskog upravljanja memorijom nakon što u određenom razdoblju nema aktivnog korištenja tom varijablom ili objektom.

Mehanizam automatskog upravljanja memorijom u programskom jeziku Python

stanje mehanizma za upravljanje memorijom

Deklaracijom globalne varijable zauzima se dio memorije *shella*.

```
>>> globalna_varijabla=100
```

Moduli sys i GC sadržavaju metode koje daju uvid u rad sistemskog alata za oslobađanje memorije te rada sustava s varijablama.

način dobivanja uvida u stanje automatskog upravljanja memorijom

Na sljedećoj slici prikazan je način kako dobiti uvid u stanje automatskog upravljanja memorijom za aktualne referencije i za već „potrošene” referencije, tj. one koje se tijekom izvođenja programa više neće ponavljati.

```
>>> import sys
>>> import gc
>>>
>>> sys.getrefcount(globalna_varijabla)
11
>>>
>>> len(gc.get_referrers(globalna_varijabla))
2
```

Sve privremene reference (već „potrošene” i aktualne”)

Aktualne reference

ručno uklanjanje varijable naredbom `Del`

Varijable se, osim s pomoću ugrađenog mehanizma za upravljanje memorijom, mogu i ručno ukloniti naredbom `Del`.

```
>>> del globalna_varijabla
>>>
>>> globalna_varijabla
Traceback (most recent call last):
  File "<pyshell#170>", line 1, in <module>
    globalna_varijabla
NameError: name 'globalna_varijabla' is not defined
>>>
```

Ručno ”brisanje”

Važno je napomenuti da se atomarni tipovi (npr. samo varijable koje sadržavaju jednu vrijednost) i objekti koji zauzimaju malo memorije obično ne prate s pomoću ugrađenog mehanizma za upravljanje memorijom. Prikazana metoda daje nam uvid u to prati li se pojedini objekt, tj. varijabla, ili ne.

način rada ugrađenog mehanizma za upravljanje memorijom

```
>>> gc.is_tracked(globalna_varijabla)
False
```

Ugrađeni mehanizam za upravljanje memorijom radi na principu podjele svih objekata koje prati u tri klase, tj. generacije. Prva generacija sadržava objekte koji će posljednji biti uklonjeni, a treća generacija sadržava objekte koji će prvi biti uklonjeni pri sljedećem izvršavanju mehanizma.

djelovanje ugrađenog mehanizma za upravljanje memorijom

Provjera koliko je objekata trenutačno spremno za uklanjanje prema generacijama moguća je s pomoću sljedeće metode – svaki prolazak mehanizma uklanja posljednju generaciju objekata (krajnje desnu) i priprema sljedeću generaciju za uklanjanje ako nema referencija za nju.

```
>>> gc.get_count()
(29, 7, 3)
```

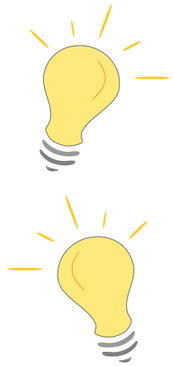
Automatsko pozivanje mehanizma za upravljanje memorijom ili ručno pozivanje s pomoću naredbe Gc.collect() uklanja objekte koji nemaju referencije.

uklanjanje objekata koji nemaju referencije

```
>>> gc.collect()
66
>>> gc.get_count()
(46, 0, 0)
```

Provjerite i primijenite

1. Koje su kategorije memorije dostupne na računalu i kako se dijele?
2. Koja je razlika između fizičke radne memorije i radne memorije operativnog sustava?
3. Koji su primjeri sistemske pogreške koja se može pojaviti zbog nedovoljnog upravljanja potrošnjom memorije?
4. Kako moderni programski jezici automatski upravljaju potrošnjom memorije?
5. Koje su tri faze u životnom ciklusu varijable/objekta u programskom jeziku Python i što se događa u svakoj fazi?



Za one koji žele saznati više:

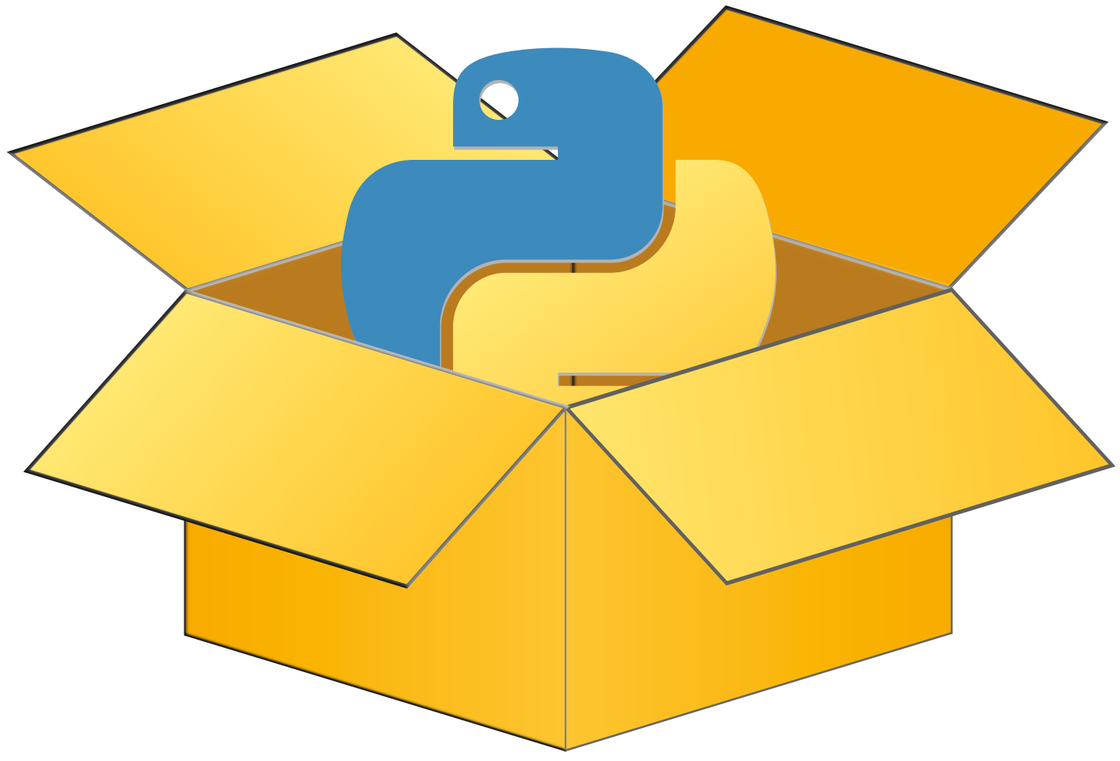
Polaznicima koji žele proširiti svoje znanje s područja procesa upravljanja memorijom računala preporučuju se sljedeće teme kao samostalno istraživanje unutar Pythonova razvojnog okružja:

- Postavke mehanizma za automatsko upravljanje memorijom
- Aktivno praćenje zauzimanja memorije u sesiji/sustavu
- Pravila za uklanjanje objekata.

Detalji o mehanizmu dostupni su na internetskoj adresi

☞ https://docs.python.org/3/library/gc.html

8. Moduli i paketi



Nakon što ste u osmoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o modulima i paketima, moći ćete:

- razlikovati module i pakete
- objasniti ulogu modula i paketa u programiranju
- objasniti koncept standardne biblioteke
- navesti primjere modula koji dolaze sa standardnom bibliotekom
- koristiti se ključnim riječima import, from i import za uporabu funkcija, konstanta i razreda koji se nalaze unutar modula
- napisati primjere uporabe funkcija, konstanta i razreda iz modula
- objasniti problem koji se može javiti pri uključivanju svih funkcija iz modula
- objasniti koncept kreiranja vlastitih modula
- demonstrirati modul na primjeru
- objasniti postupak instaliranja modula trećih strana s pomoću upravljača paketa pip.
- pretraživati odgovarajuće module trećih strana na platformi pypi.org
- odabrati odgovarajuće module trećih strana na platformi pypi.org.

PowerPoint prezentacija

Snimka 7. radionice

Moduli i paketi omogućuju lakši i brži razvoj programskoga kôda, a u konačnici i programa. Napisano je puno modula, tj. paketa za *Python* koji ubrzavaju razvoj programskoga kôda jednostavnim pozivanjem unaprijed napisanih funkcija, razreda, konstanta. Moduli, tj. paketi organizirani su u smislene cjeline radi što lakšeg snalaženja ne samo u korištenju već i radi što lakšeg pretraživanja dokumentacije.

doprinos modula i paketa razvoju programskoga kôda

Module i pakete koji dolaze sa standardnim instaliranjem *Pythona* nazivamo standardnom bibliotekom.

standardna biblioteka

Katkad su programerima potrebne funkcionalnosti koje ne dolaze s modulima, tj. paketima iz *standardne biblioteke*. Za takve funkcionalnosti potrebno je poslužiti se internetom i potražiti odgovarajuće pakete te ih instalirati na računalo na kojemu će se program izvršavati. U ovoj edukaciji obrađeni su samo moduli koji dolaze sa standardnim instaliranjem *Pythona*.

Potrebno je razlikovati module i pakete. Moduli su sastavni dijelovi programskoga kôda unutar kojih je primijenjena neka specifična funkcionalnost. Kao primjer navodi se nekoliko modula koji dolaze sa standardnom bibliotekom: `math`, `cmath`, `random`, `datetime`.

razlika između modula i paketa

Paketi su cjeline koje uključuju druge module i oni omogućuju njihovo hijerarhijsko grupiranje. Kao primjer navodimo paket koji dolazi sa standardnom bibliotekom, paket `urllib`, i sadržava nekoliko modula:

primjer

- `urllib.request`
- `urllib.error`
- `urllib.parse`
- `urllib.robotparser.`

Rad s modulima i paketima

Funkcijom, konstantom i razredom koji se nalaze unutar nekog modula možemo se koristiti na dva načina:

dva načina korištenja funkcijom, konstantom i razredom

- najavljivanjem korištenja
- uključivanjem u program.

Najavljivanje korištenja

Najavljivanje korištenja modulom ostvaruje se s pomoću ključne riječi import. Sintaksa:

```
import <modul>
```

primjer

```
import math

print(math.sin(55))
print(math.tan(55))
```

```
Izlaz:
      -0.9997551733586199
      0.022126756261955736
```

primjer najavljivanje korištenja modulom

U gornjem primjeru prikazan je način korištenja naredbom Import. Kod takvog načina korištenja naredbom Import najavljuje se primjena **svih** funkcija, razreda i konstanta (u daljnjem tekstu spominjat će se samo funkcije, no podrazumijeva se to da se u nekom modulu mogu nalaziti i razredi i konstante) iz nekog modula.

Ako se uporaba funkcija samo najavi kao u gornjem primjeru, tada u svakom dijelu programa u kojemu želimo iskoristiti funkciju iz najavljenog modula mora pisati ime modula i ime funkcije odvojene točkom. Sintaksa takvog načina korištenja najavljenim funkcijama jest:

```
modul.funkcija()
```

Uključivanje u program

korištenje ključnim riječima from i import

Ako želimo izbjeći gornju sintaksu i koristiti se samo imenom funkcije kao pozivom, a ne da ispred imena funkcije piše ime modula u kojem se funkcija nalazi, potrebno je željene funkcije uključiti u program. Uključivanje funkcija u program provodi se uporabom ključnih riječi `from` i `import`.

Sintaksa takvog načina korištenja funkcijama jest:

```
from <modul> import <funkcija>, <funkcija>, ...
```

Ključna riječ `from` govori koji će se modul primijeniti, a ključna riječ `import` govori koje će se sve funkcije iz modula uključiti u naš program. Kod takvog načina uključivanja funkcija u program pri korištenju uključenim funkcijama, za razliku od prijašnjeg primjera, više nije potrebno pisati u kojem se modulu nalazi korištena funkcija.

značenje ključne riječi from i import

primjer

```
from math import sin, cos

print(sin(55))
print(cos(55))
print(math.tan(55))
```

```
Izlaz:
      -0.9997551733586199
      0.022126756261955736
      Traceback (most recent call last):
        File "test.py", line 4, in <module>
          print(math.tan(55))
      NameError: name 'math' is not defined
```

primjer sintakse korištenja funkcijama tako da se one uključuju u programski kôd

U prethodnom primjeru prikazana je sintaksa korištenja funkcijama tako da se one uključuju u programski kôd. U program su uključene samo one funkcije koje će se upotrijebiti u programskom kôdu, a to su `sin()` i `cos()`. Primjećuje se da pri pozivu funkcija više nije potrebno ispred funkcije pisati ime modula u kojemu se ta funkcija nalazi.

U gornjem primjeru namjerno je izazvana pogreška. Funkcija `tan()` iz modula `math` nije uključena u programski kôd, a nije najavljeno ni korištenje njome u programskom kôdu. Za demonstraciju ona nije pozvana tako da se napiše samo ime funkcije, već je pozvana na način math.tan(), no svejedno se dogodila pogreška. Ona je nastala zbog toga što u navedenom načinu uključivanja funkcija iz modula math u program nije najavljeno i korištenje ostalim funkcijama iz tog paketa.

namjerno izazvana pogreška

Želimo li ispraviti tu pogrešku, to se može učiniti na dva načina:

ispravljanje namjerno izazvane pogreške

- dodavanjem funkcije `tan()` u uključene funkcije:

```
from math import sin, cos, tan
```

- najavljivanjem uporabe svih funkcija iz modula `math`: `import math`

Svejedno je koji će se način odabrati.

Ako se pri uporabi (pozivu) funkcija iz modula ne želi pisati iz kojeg modula dolazi funkcija, a ni pri uključivanju modula se ne želi sastavljati popis svih funkcija koje će se upotrebljavati, to se može napraviti tako da se u program uključi sve funkcije iz nekog modula.

razlog uključivanja svih funkcija iz nekog modula i mogući problem

```
from math import *
```

Problem koji se može pojaviti pri takvom načinu uporabe modula jest to da se pri uključivanju više različitih modula može dogoditi kolizija ako se u dva i više modula nalazi isto ime funkcije.

Konstante

U modulima se mogu nalaziti i razne konstante. Tako se u modulu `math` nalazi nekoliko matematičkih konstanta: *Pi*, *e*, *Tau*, *INF*, *nan*. U nastavku slijede primjeri programskoga kôda unutar kojih se dohvaća i ispisuje vrijednost konstante *Pi*.

primjeri programskoga kôda unutar kojih se dohvaća i ispisuje vrijednost konstante *Pi*.

primjer

Najavljivanje korištenja svim funkcijama i konstantama modula `math`:

```
import math print(math.pi)
```

```
Izlaz:
     3.141592653589793
```

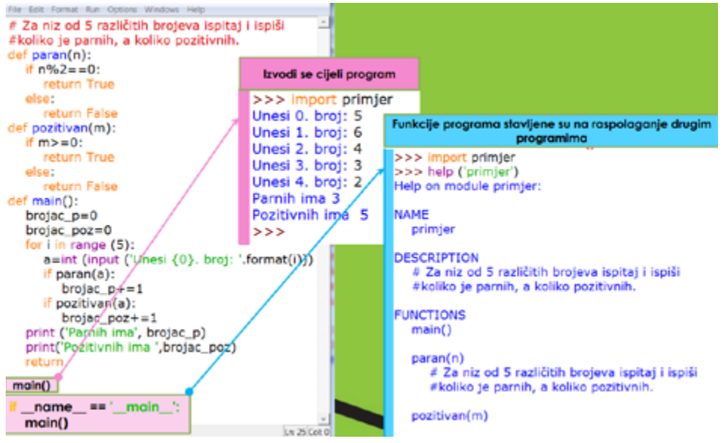
Uključivanje u program konstante Pi iz modula `math`:

```
from math import pi
print(pi)
```

```
Izlaz:
     3.141592653589793
```

Kreiranje vlastitih modula

Na slici je prikazan pristup kako napraviti vlastiti modul koji se nakon toga može uključiti u kod i upotrijebiti.

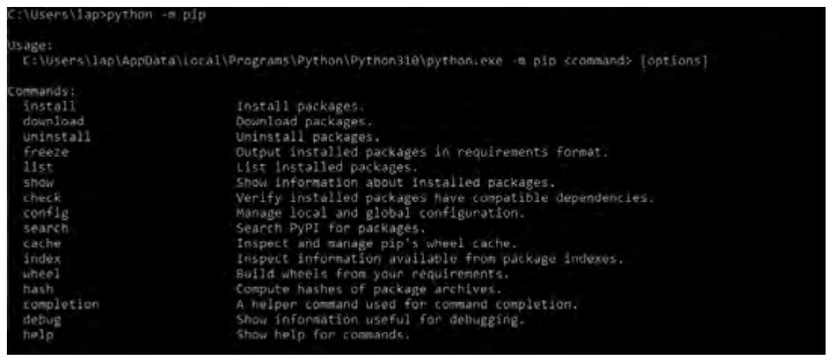


Instaliranje modula trećih strana

Postoji „upravljač paketa" – *packet manager za Python* – koji služi ažuriranju i instaliranju modula trećih strana.

Pokretanje iz komandnog retka

postupak pokretanja iz komandnog retka



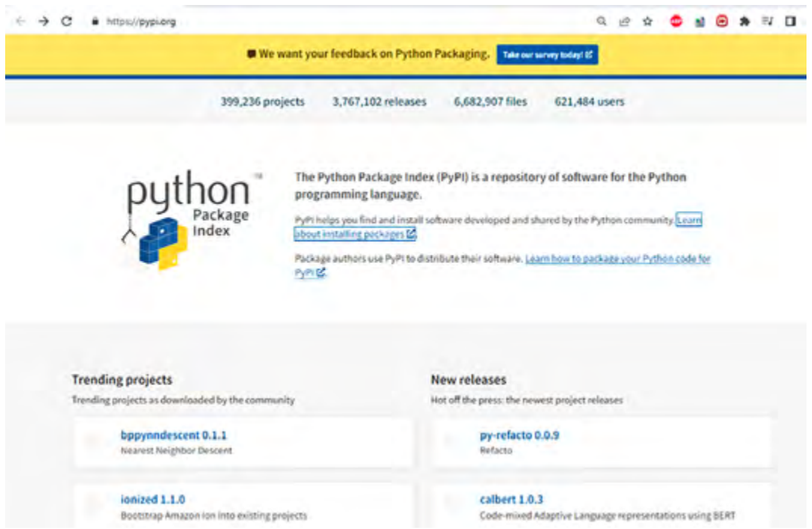
- Ako se već zna naziv modula i pip je instaliran

```
C:\Users\lap>python -m pip install IME
```

- Ako nema pip-instaliranja, najprije treba postaviti pip

```
C:\Users\lap>python -m ensurepip --default-pip
```

- Prikaz registra svih modula trećih strana s pomoću alata i kataloga pypi.org
  ☞ https://pypi.org/



Vježbe korištenja *Pythonovim* modulima i paketima

1. Izradite program koji najavljuje korištenje funkcijama iz modula `math` te izračunajte i ispišite rezultat:

$$\sin(2) + \cos(1)$$

2. Pronađite na internetu u službenoj dokumentaciji *Python 3* kako se zove funkcija koja se nalazi u modulu `math` i primjenjuje za korjenovanje. S pomoću tipkovnice unesite broj i ispišite njegov korijen.

Službenu dokumentaciju *Python 3* možete pronaći na URL-u: https://docs.python.org/3/library/math.html

3. Primjenom konstanta koje su pohranjene u modulu math ispišite vrijednost konstante *Pi*.

4. Korištenjem konstante *Pi* izračunajte i ispišite rezultat: sin(2*Pi*) + cos(*Pi*)

5. * U službenoj dokumentaciji *Python 3* pronađite ime funkcije za potenciranje (zamjena za aritmetički operator **). S tipkovnice učitajte cjelobrojne vrijednosti i spremite ih u varijable *a* i *b*. Nije potrebno provjeravati ispravnost učitanih brojeva. Na temelju tih vrijednosti izračunajte kvadrat zbroja $(a + b)^2$. Formula za kvadrat zbroja je $(a + b)^2 = a^2 + 2ab + b^2$. Funkcije kojima ćete se koristiti u ovom zadatku uključite u program.

Provjerite i primijenite

1. Napišite za što služe moduli te kako biste ih upotrijebili u svojem radu.

2. Provjerite koje sve sastavnice mogu biti uključene u module.

3. Provjerite mogu li se moduli koristiti drugim modulima.

4. Provjerite mogu li se nazivi između nekoliko modula preklapati i koje pravilo vrijedi ako se dogodi preklapanje.

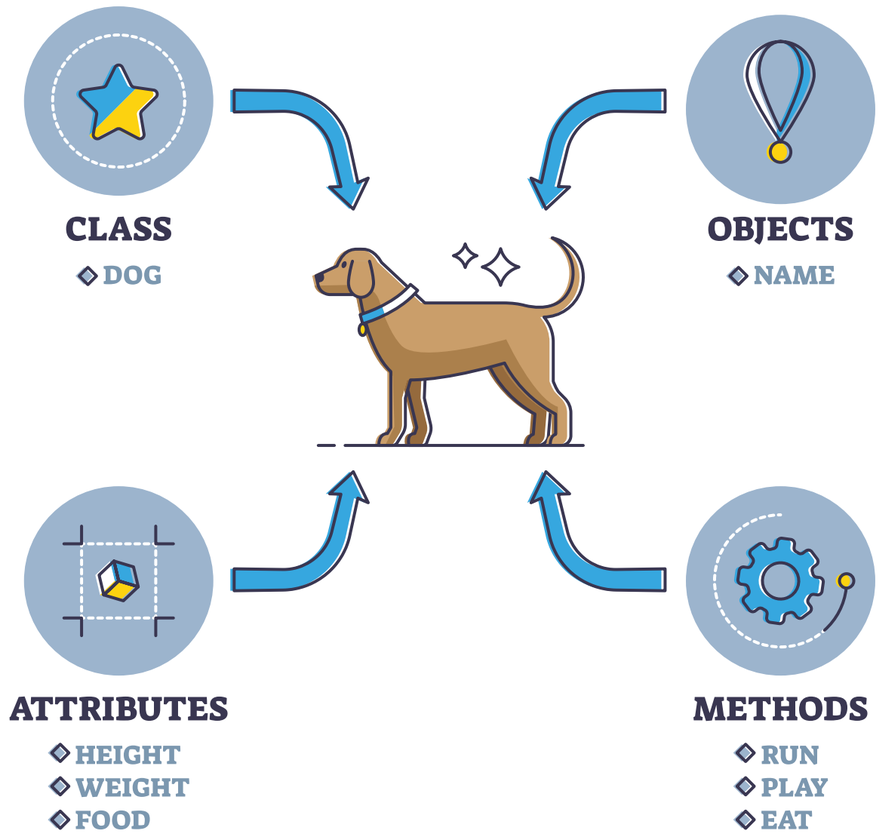
Za one koji žele saznati više

1. Provjerite definiciju modula na službenoj dokumentaciji Pythona

   ☞ https://docs.python.org/3/tutorial/modules.html

2. Riješite interaktivne zadatke o modulima dostupne na sljedećoj internetskoj poveznici

   ☞ https://www.w3schools.com/python/python_modules.asp



9. Objektno orijentirano programiranje

Nakon što ste u devetoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o objektno orijentiranom programiranju, moći ćete:

- navesti prednosti objektno orijentiranog programiranja u odnosu prema proceduralnom programiranju
- objasniti osnovne koncepte objektno orijentiranog programiranja kao što su razred, objekt, atribut i metoda
- definirati razred i objekt
- objasniti međusobnu povezanost razreda i objekta
- objasniti kako se pozivaju metode na objektima i kako se instancira razred
- napisati jednostavne primjere koda kojim se koristi objektno orijentirano programiranje
- prikazati osnovne koncepte kao što su definiranje razreda, instanciranje objekata i pozivanje metoda.

PowerPoint prezentacija

Snimka 9. radionice

Uvod u objektno orijentirano programiranje

proceduralno programiranje

Način razvoja programskoga kôda koji je prikazan u dosadašnjem dijelu priručnika naziva se proceduralno programiranje. To je način programiranja čija se logika zasniva na temelju funkcija, tj. blokova programskoga kôda.

Izrada programskoga kôda koja će biti objašnjena u ovom poglavlju zove se objektno orijentirano programiranje (kraće OOP). Većina programa može se kvalitetno napisati s pomoću proceduralnog načina razvoja programskoga kôda, no kod velikih projekata preporučljivo je koristiti se objektno orijentiranim programiranjem. Objektno orijentirano programiranje služi smanjenju kompleksnosti razvoja i održavanja programskih rješenja. OOP ne sprečava programere da pišu loš programski kôd, već ih usmjerava da razvijaju programski kôd u okvirima standarda ove paradigme.

prednosti objektno orijentiranog programiranja

U objektno orijentiranom programiranju nakana je da problem koji se rješava bude razdijeljen na što manje logičke cjeline.

primjer objektno orijentiranog programiranja

primjer

Na primjer, ako program treba obrađivati podatke o osobama, tada je osnovni tip objekta u tom programu **razred** Osoba. Svakoj osobi možemo pridružiti razne parametre ili pak u duhu objektno orijentiranog pristupa **atribute**. Atributi mogu biti, na primjer, ime, prezime, visina, težina, OIB. Jednako tako svakom razredu (instancija razreda naziva se objekt) pridružene su i pripadajuće akcije, tj. **metode**. Na primjer, osoba može hodati, sjesti, trčati itd., pa tako imamo metode: hodaj(), sjedni(), trci(), koje objektu Osoba daju „život". Ponuđen je ilustrativan prikaz razreda Osoba.

Ime razreda	Osoba
atributi	ime
	prezime
	visina
	tezina
	...
metode	hodaj()
	sjedni()
	trci()
	...

Definiranje objekta/klase

U nastavku slijedi tablica sinonima koji se često pojavljuju pri čitanju literature, a imaju isto značenje.

tablica istoznačnica ili sinonima

Hrvatski nazivi (sinonimi)			Engleski nazivi
razred	klasa	–	engl. (*class*)
atribut	svojstvo	–	engl. (*attribute*)
objekt	instancija	jedinka	engl. (*instance*)

Razred je osnovna programska cjelina kod objektno orijentiranog načina programiranja. Unutar razreda zapisan je shematski plan (engl. *blueprint*) te se na temelju njega kreiraju **objekti**. Razred sadržava **atribute** i **metode**. Atributi opisuju objekt raznim podatcima, a svaki objekt ima kopiju svih atributa. Metode su zapravo funkcije, no one se pozivaju nad objektom kojemu je neka metoda pridružena.

osnovna programska cjelina kod objektno orijentiranog načina programiranja

U nastavku se nalazi osnovni primjer kreiranja razreda.

primjer

```
#deklaracija klase/nema atributa u tijelu klase
#nije potrebno unositi atribut u klasu Posuda()
class Posuda:
#poziv funkcije unosa biskvita
    def insertBiskvit(self, biskvit):
        self.biskvit = biskvit
#poziv funkcije ispisa biskvita
    def getBiskvit(self):
        print (self.biskvit)
```

primjer kreiranja razreda

Pozivanje metoda

Svaki razred počinje ključnom riječju class, a nakon ključne riječi navodi se proizvoljno ime razreda. Prema *Pythonovoj* preporuci riječi naziva razreda pišu se velikim početnim slovom, na primjer: Osoba, KoordinatniSustav i slično. Unutar tako kreiranog razreda (klase) pišu se:

kreiranje razreda

- **atributi** – svojstva razreda
- **metode** – funkcionalnosti razreda.

programski kôd

U nastavku je prikazan programski kôd koji ostvaruje neke funkcionalnosti samih ljudi. Unutar razreda imena Osoba primijenjene su tri metode – `hodaj()`, `sjedni()`, `trci()`. U glavnom programu kreiran je jedan objekt (instancija) razreda Osoba imena o. U donjem primjeru može se vidjeti da se objekt nekog razreda može kreirati na sljedeći način:

imeObjekta = ImeRazreda()

Metode nekog objekta pozivaju se s pomoću sljedeće sintakse:

imeObjekta.imeMetode()

Napomena: unutar donjeg primjera vidljivo je da se upotrebljava parametar **self** koji označava varijable i funkcije jedne instancije objekta.

primjer upotrbe parametra self

primjer

```
class Osoba:

    def hodaj(self):

        print(„Hodaj!”)


    def sjedni(self):

        print(„Sjedni!”)


    def trci(self):

        print(„Trci!”)
```

```
   o = Osoba()


   o.hodaj() o.sjedni
() o.trci() Izlaz:

   Hodaj!
   Sjedni!
   Trci!
```

Ako je potrebno kreirati beskoristan razred koji ne sadržava programski kôd, to se može postići tako da se u tijelo kreiranog razreda napiše naredba Pass.

U donjem primjeru vidljivo je da je razred imena Osoba razred bez atributa i metoda. Unutar glavnog programa kreirana su tri objekta (instancije) razreda Osoba, a to su: o1, o2, o3. Na temelju ispisa može se zaključiti da sva tri objekta imaju različitu memorijsku lokaciju što znači da su sva tri objekta međusobno neovisna.

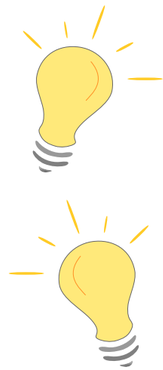
Na primjeru programskoga koda s početka na ovaj se način deklarira instancija klase, tj. objekt, te kako se poziva.

primjer

primjer deklariranja instancije klase

```
#pozivi klase i funkcija
mojaPosuda=Posuda()
mojaPosuda.insertBiskvit('mojBiskvit')
mojaPosuda.getBiskvit()
```

Provjerite i primijenite



1. Objasnite na svojem primjeru kako se koristiti razredima i objektima.
2. Objasnite čemu služe objekti i razredi.
3. Napravite model klase koja opisuje ponašanje i atribute vozila te njegovu interakciju s garažom.

Za one koji žele saznati više

1. Istražite službenu Pythonovu dokumentaciju o klasama i objektima

   ☞ https://docs.python.org/3/tutorial/classes.html

2. Riješite interaktivne vježbe dostupne na poveznici

   ☞ https://www.w3schools.com/python/python_classes.asp

10. Praktične primjene objektno orijentiranog programiranja

Nakon što ste u desetoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o praktičnoj primjeni objektno orijentiranog programiranja, moći ćete:

- razumjeti koncept objektno orijentiranog programiranja
- primijeniti koncept objektno orijentiranog programiranja u praktičnim primjenama
- razumjeti programske knjižnice za interakciju sa senzorima
- primijeniti programske knjižnice za interakciju sa senzorima
- koristiti se senzorima za detekciju i prikupljanje informacija, za interakciju sa zaslonom, sa zvučnim aktuatorima
- primijeniti objektno orijentirano programiranje u području poljoprivrede i poljoprivredne proizvodnje
- razumjeti i primijeniti koncepte programske podrške za uređaje poput mikrofona, kamere, tipkovnice i miša
- razumjeti i primijeniti koncepte regulacije visine i trajanja zvuka te reprodukcije zvuka iz datoteke

PowerPoint prezentacija

Snimka 10. radionice

Programske knjižnice za pristup ugrađenim senzorima

odrednice senzora

Senzor je pretvornik ili mjerno osjetilo i dio je mjernoga sustava koji je u izravnom dodiru s mjerenom veličinom te daje izlazni signal ovisan o njezinu iznosu. Zbog prevladavajuće primjene električnih i elektroničkih sustava, većina mjernih osjetila pretvara mjerenu veličinu u električki mjerljiv signal.

ugrađeni senzori i njihova primjena u poljoprivredi

Ugrađeni senzori su sljedeći.

mikrofon

Zvučni je senzor koji pretvara mehaničko titranje membrane koja čini jednu ploču električnoga kondenzatora u kondenzatorskome mikrofonu u promjenu električnoga kapaciteta.

Praktična primjena u poljoprivredi:

primjer otkrivanje štetnih događaja poput pada objekata na plastenik ili staklenik što može prouzročiti veliku štetu u poljoprivrednoj proizvodnji, otkrivanje i druge vrste buke koja može biti problem u uzgoju životinja, poput preglasnog kukanja ili plača, a to može upozoriti na zdravstvene probleme ili stres u životinja, otkrivanje i praćenje zvukova koje proizvode štetnici poput insekata i glodavaca

kamera

Vizualni je senzor koji uzima 25 – 30 slika u sekundi i prikazuje izlaz ili kao sliku ili kao video.

Praktična primjena u poljoprivredi:

primjer udaljeni nadzor nad nasadima vinograda, maslinicima itd., praćenje rasta i razvoja biljaka te otkrivanje bilo kakvih problema s bolestima ili štetnicima, snimanje fotografija nasada tijekom vremena te zatim analiza fotografija kako bi se utvrdilo kako biljke rastu i razvijaju se, prepoznavanje neželjenih promjena u boji ili veličini ploda što može upozoriti na prisutnost bolesti ili štetnika, praćenje životinjskog svijeta u poljoprivrednom okružju, praćenje kretanja oblaka što može pomoći u predviđanju kiše ili drugih vremenskih uvjeta koji mogu utjecati na usjeve, praćenje rada strojeva kako bi se osiguralo njihovo redovito održavanje

tipkovnica	Senzor je koji služi kao ulaz znakovnih i brojčanih simbola u računalo, no može biti samo nekoliko tipki koje pokreću pojedine funkcije. Praktična primjena u poljoprivredi: **pr.** pokretanje određenih akcija pritiskom na tipku
miš	Senzor je koji služi za prihvat informacije kretanja x i y komponenata koje se mogu translatirati na poziciju pokazivača na računalu te primarne i sekundarne tipke. Praktična primjena u poljoprivredi: **pr.** pokretanje određenih akcija pritiskom na tipku (pokretanje određenih pripremljenih Pythonovih programa)
dodatni senzori/ uređaji spojeni na računalo	Na primjer, senzori vlažnosti i temperature zraka ili zemlje itd. Praktična primjena u poljoprivredi: **pr.** prikupljanje ključnih informacija o stanju poljoprivrednih nasada itd.

Pristup uređajima – rezolucija zaslona

Ugrađeni uređaji za izlaz informacija su, na primjer, zaslon i zvučnici.

ugrađeni uređaji za izlaz informacija

primjena zaslona i zvučnika u poljoprivredi

praktična primjena zaslona u poljoprivredi	prikaz i nadzor nad stanjem nasada, prikaz upozorenja na štetne događaje ili događaje u kojima treba primijeniti akciju kako bi se izbjegao npr. razvoj bolesti u nasadu
praktična primjena zvučnika u poljoprivredi	služe kao upozorenje na pojedine važne događaje

```
D:\py>pip install screeninfo
Collecting screeninfo
  Downloading screeninfo-0.8.1-py3-none-any.whl (12 kB)
Installing collected packages: screeninfo
Successfully installed screeninfo-0.8.1
```

```
D:\py>python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from screeninfo import get_monitors
>>> for m in get_monitors():
...     print(str(m))
...
Monitor(x=0, y=0, width=1920, height=1080, width_mm=344, height_mm=194, name='\\\\.\\DISPLAY1', is_primary=True)
Monitor(x=-1920, y=-80, width=1920, height=1080, width_mm=477, height_mm=268, name='\\\\.\\DISPLAY4', is_primary=False)
>>>
```

Interakcija sa zaslonom i zvučnim aktuatorima

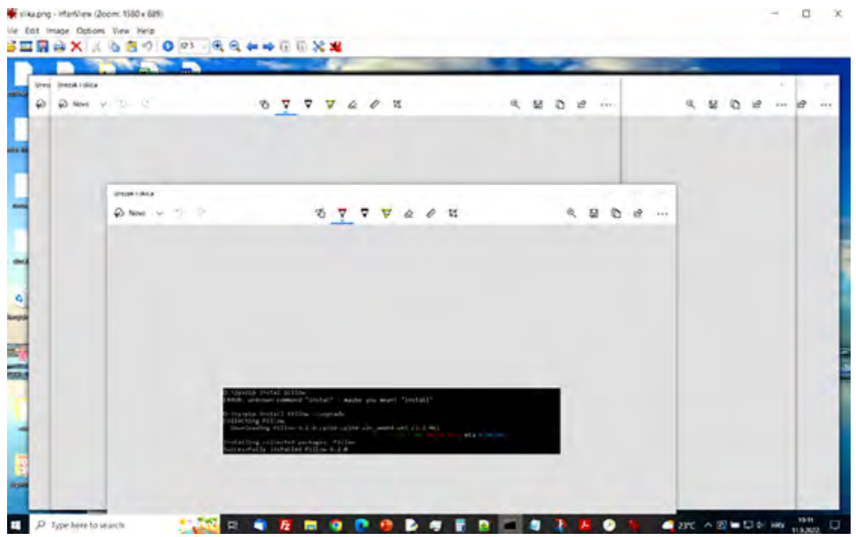
- Programske knjižnice za interakciju s ugrađenim zaslonom

```
D:\py>pip install pyautogui
Collecting pyautogui
  Downloading PyAutoGUI-0.9.53.tar.gz (59 kB)
     ---------------------------------------- 59.0/59.0 kB 774.5 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting pymsgbox
  Downloading PyMsgBox-1.0.9.tar.gz (18 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting PyTweening>=1.0.1
  Downloading pytweening-1.0.4.tar.gz (14 kB)
  Preparing metadata (setup.py) ... done
Collecting pyscreeze>=0.1.21
  Downloading PyScreeze-0.1.28.tar.gz (25 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting pygetwindow>=0.0.5
  Downloading PyGetWindow-0.0.9.tar.gz (9.7 kB)
  Preparing metadata (setup.py) ... done
Collecting mouseinfo
  Downloading MouseInfo-0.1.3.tar.gz (10 kB)
  Preparing metadata (setup.py) ... done
Collecting pyrect
  Downloading PyRect-0.2.0.tar.gz (17 kB)
  Preparing metadata (setup.py) ... done
Collecting pyperclip
  Downloading pyperclip-1.8.2.tar.gz (20 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: pyautogui, pygetwindow, pyscreeze, PyTweening, mouseinfo, pymsgbox, pyp
yrect
  Building wheel for pyautogui (setup.py) ... done
  Created wheel for pyautogui: filename=PyAutoGUI-0.9.53-py3-none-any.whl size=36614 sha256=e39fe81c4db557a453e
5d732780eca579efcffe0179d1bb7ff6f3d3
```

```
D:\py>pip instal pillow
ERROR: unknown command "instal" - maybe you meant "install"

D:\py>pip install Pillow --upgrade
Collecting Pillow
  Downloading Pillow-9.2.0-cp310-cp310-win_amd64.whl (3.3 MB)
     ---------------------------------------- 3.3/3.3 MB 885.6 kB/s eta 0:00:00
Installing collected packages: Pillow
Successfully installed Pillow-9.2.0
```

```
>>> import pyautogui
>>> slika = pyautogui.screenshot()
>>> slika.save(r'D:\py\slika.png')
```



- Programske knjižnice za interakciju sa zvučnikom računala

  Regulacija visine i trajanja zvuka

```
>>> import winsound
>>> winsound.Beep(440, 500)
```

Reprodukcija sistemskih zvukova

```
>>> winsound.PlaySound("SystemExclamation", winsound.SND_ALIAS)
```

Reprodukcija zvuka iz datoteke

```
>>> winsound.PlaySound("beep.wav", winsound.SND_FILENAME)
```

Završni zadatak

Vježba: Primjena naučenog te primjena jednog od ponuđenih zadataka

Provjerite i primijenite

1. Čemu služe senzori i aktuatori?
2. Kako se koristiti senzorima i aktuatorima iz Pythonova programskoga koda?
3. Koji su ugrađeni senzori i aktuatori kojima se može upravljati iz Pythona?
4. Koja je vrsta Pythonova modula potrebna za rad s ugrađenim senzorima i aktuatorima?

Za one koji žele saznati više

1. Provjerite načine spajanja vanjskih senzora na računalo na donjoj poveznici

   ☞ https://problemsolvingwithpython.com/11-Python-and-External-Hardware/11.04-Reading-a-Sensor-with-Python/

2. Proučite kako se koristiti vanjskim senzorom temperature iz Pythona.

   ☞ Https://usbtemp.com/

1. Uvod u programiranje

Nakon što ste u prvoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o praktičnoj primjeni objektno orijentiranog programiranja, moći ćete:

- objasniti kontekst programskih jezika
- razlikovati strojni jezik od programskih jezika više razine
- objasniti što su dijagrami tijeka i pseudokod te kako se rabe u izradi računalnih programa
- opisati povijest programskog jezika Python
- prepoznati osnovne faze ili korake u izradi računalnog programa: analizu problema, crtanje dijagrama tijeka, pisanje programskoga koda i unos programskoga koda u računalo te pokretanje programa
- obrazložiti što obuhvaća prvi korak u izradi računalnog programa: analizu potreba, definiranje s pomoću jednadžbi, uvjeta i pseudoalgoritama te kako se on radi kod jednostavnijih i složenijih programa
- razviti algoritamski način razmišljanja i primijeniti ga u rješavanju problema uporabom dijagrama tijeka i pseudokoda.

PowerPoint prezentacija

Snimka 1. radionice

Programiranje

Tehnike programiranja

odrednice programiranja

Programiranje je vještina koja korisniku omogućuje da stvori algoritme koristeći se određenim programskim jezicima kako bi izradio računalni program. Programiranje uključuje sastavne dijelove umjetnosti, znanosti, matematike i inženjerstva. Strojni jezik (strojni kod ili binarni kod) jedini je programski jezik koji računalo može izravno izvršavati.

povijesni razvoj računalnog pisma, programiranja

U početcima računalstva programeri su pisali u strojnom kodu što je bilo vrlo komplicirano i zamorno. Zatim je uslijedila upotreba simboličnih jezika, poznatih pod zajedničkim nazivom asembler, koji se sastoje od jednostavnih instrukcija koje se izravno i jednoznačno mogu prevesti u strojni kod. Iako je mnogo pogodnije od strojnog programiranja, asemblersko programiranje karakterizira velika količina posla koju programer mora obaviti zbog činjenice da su operacije i dalje elementarne.

Zbog toga su stručnjaci stvorili programske jezike više razine s pomoću kojih se piše izvorni kod koji se prevodi u strojni kod posredovanjem specijalnih programa – prevoditelja poput tumača i kompajlera.

koraci za izradu programa

Pri izradi svakog programa potrebno je proći četiri osnovne faze ili koraka: analizu problema, crtanje dijagrama tijeka, pisanje programskoga koda i unos programskoga koda u računalo te pokretanje programa.

analiza potreba

Prvi korak u izradi računalnog programa je analiza potreba koja obuhvaća razmatranje situacije i problema koji treba riješiti, definiranje preko jednadžbi, uvjeta i tzv. pseudoalgoritama. Kod jednostavnijih programa taj je korak u domeni usmene analize, a kod složenijih programa radi u pisanoj i simboličnoj formi uz uredno dokumentiranje svih promjena.

Algoritam

Algoritam je precizno zapisan niz postupaka, radnji ili naredaba koje služe tomu da obavimo neki posao uz potrebne resurse. Programiranjem razvijamo algoritamski način razmišljanja što znači da učimo rješavati probleme na strukturiran način. Dijagram tijeka grafički je prikaz algoritma koji olakšava razumijevanje postavljenog zadatka.

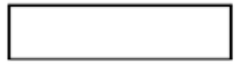
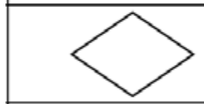
Dijagram tijeka

odrednice dijagrama tijeka u Pythonu

Dijagram tijeka u Pythonu se može napraviti s pomoću programskog jezika Python i njegove sastavnice. Dijagram tijeka u Pythonu prikazuje korake u programu u obliku grafičkog prikaza, a rabi se za vizualizaciju procesa izvođenja programa i olakšavanje njihova razumijevanja.

simboli dijagrama tijeka

Za crtanje dijagrama tijeka upotrebljavaju se sljedeći simboli

Simbol	Značenje
	simbol koji označava početak i kraj algoritma
	simbol za ulaz podataka
	simbol obrade podataka
	simbol odluke
	simbol za izlaz podataka
	simbol za nastavak dijagrama (spojna točka, poveznica)

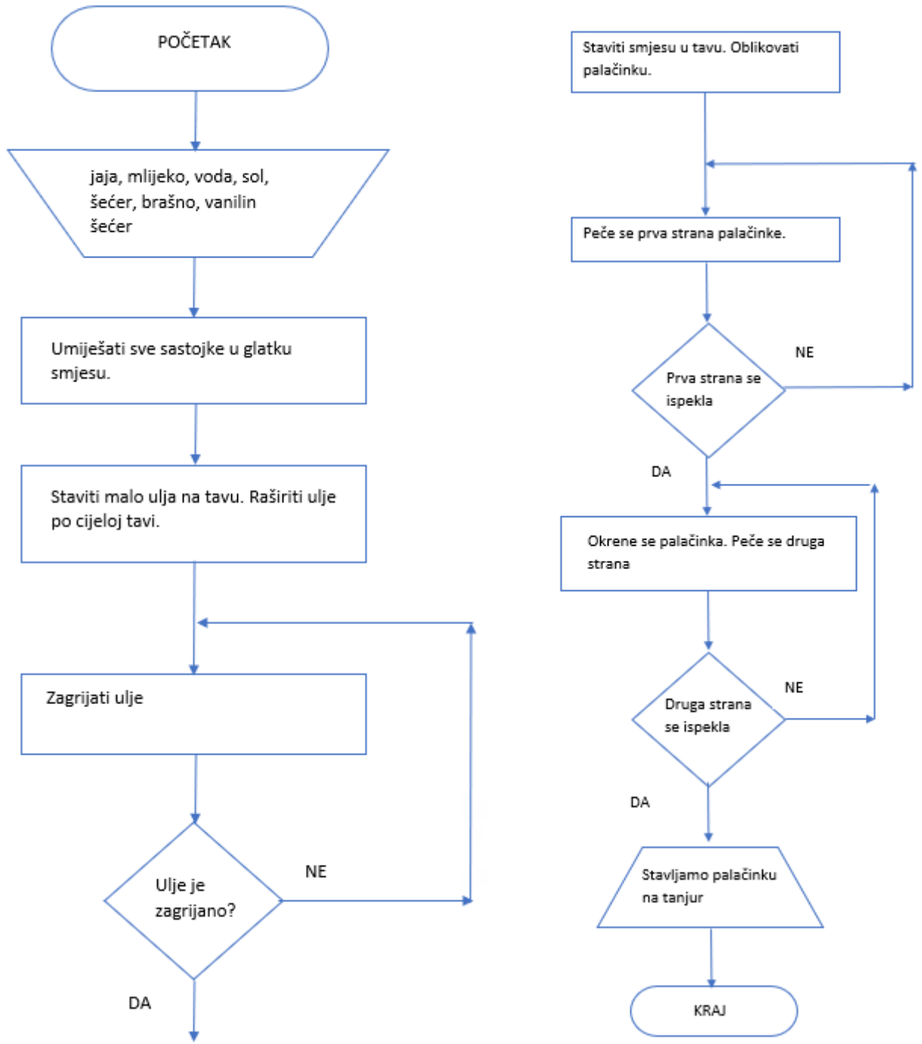
dijagrama tijeka u Pythonu na primjeru pečenja palačinki

Koraci dijagrama tijeka na primjeru pečenja palačinki:

- pokrenite program
- pripremite sastojke

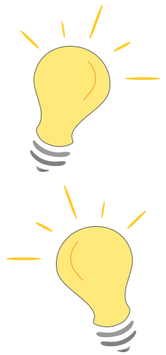
- pomiješajte sastojke u zdjeli (brašno, jaja, mlijeko, sol, šećer)
- zagrijte tavu na srednje jakoj vatri
- dodajte malo ulja u tavu
- malom kutlačom zagrabite smjesu i ulijte je u tavu
- pecite palačinku sve dok ne dobije smeđu boju s jedne strane
- okrenite palačinku i pecite je sve dok ne dobije smeđu boju s druge strane
- ponovite korake 6 – 9 sve dok ne potrošite svu smjesu
- isključite vatru i poslužite palačinke sa šećerom, medom ili drugim preljevom prema želji
- završite program.

Na idućoj stranici je prikazan dijagram tijeka u Pythonu koji prikazuje korake u programu za pečenje palačinki što olakšava razumijevanje i izvođenje tog procesa.


POČETAK
jaja, mlijeko, voda, sol, šećer, brašno, vanilin šećer
Umiješati sve sastojke u glatku smjesu.
Staviti malo ulja na tavu. Raširiti ulje po cijeloj tavi.
Zagrijati ulje
Ulje je zagrijano?
NE
DA
Staviti smjesu u tavu. Oblikovati palačinku.
Peče se prva strana palačinke.
Prva strana se ispekla
NE
DA
Okrene se palačinka. Peče se druga strana
Druga strana se ispekla
NE
DA
Stavljamo palačinku na tanjur
KRAJ

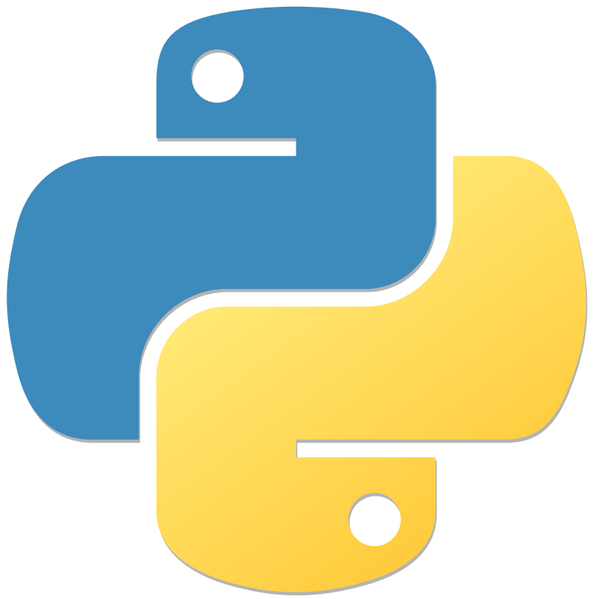
Provjerite i primijenite

1. Napravite algoritam koji će izračunati prosjek ocjena studenata iz tri predmeta.
2. Izradite program Python koji će omogućiti korisniku unos triju brojeva (ocjena) i izračunati prosjek tih brojeva te ispisati rezultat.
3. Izradite dijagram tijeka za program Pythonu koji će omogućiti korisniku unos riječi te će program ispisati broj samoglasnika i suglasnika u toj riječi.



Za one koji žele saznati više

Koristeći se literaturom s mreže, izradite dijagram tijeka za rješavanje složenijeg problema: kako organizirati svoj radni tjedan ako poslove obavljamo svakodnevno, a dostavu parnim danima poslijepodne, neparnim ujutro, a kupnju na tržnici petkom?



python™

2. Instaliranje Pythona i razvojnog okružja

Nakon što ste u drugoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o instaliranju Pythona i razvojnom okružju, moći ćete:

- razumjeti mogućnosti upotrebe programskog jezika Python kao što su lokalno instaliranje i korištenje mrežnim okružjem
- poznavati korake u lokalnom instaliranju programskog jezika Python
- razumjeti načine programiranja u Pythonu uključujući ljusku (engl. shell) i konzolu
- riješiti jednostavne primjere zadataka programiranja u Pythonu kao što su izračunavanje matematičkih operacija i ponavljanje riječi
- spremiti program Python i pokrenuti ga
- razumjeti koncept slijednog programiranja i koristiti se naredbom Print za prikaz teksta.

PowerPoint prezentacija

Snimka 2. radionice

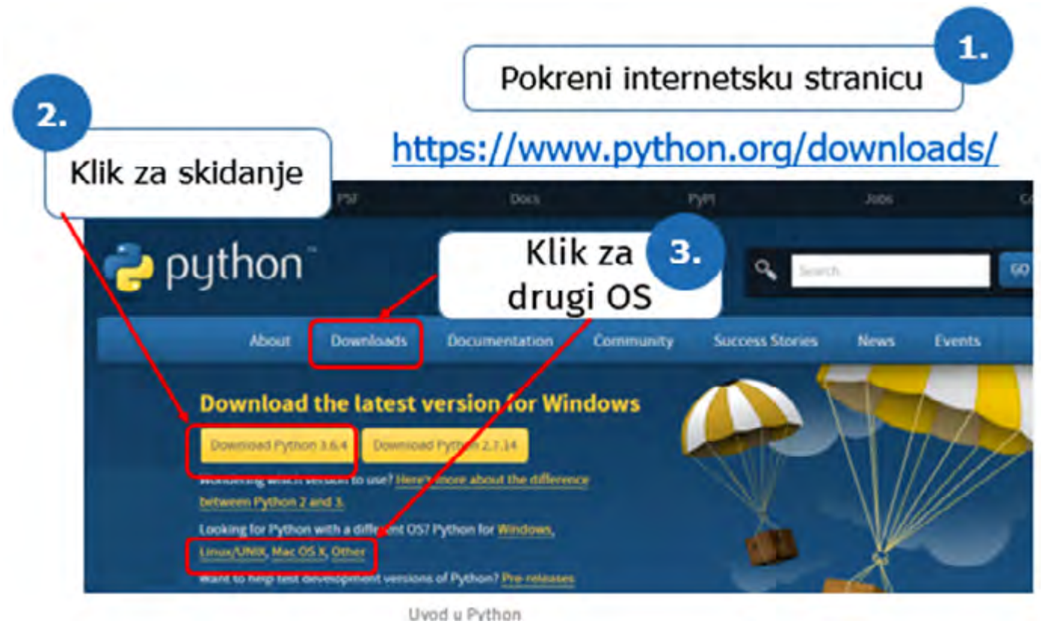
Mogućnosti upotrebe Pythona

Python se može upotrebljavati:

- lokalno instaliran na računalo
- alatom vizualizacije u mrežnom okružju.

Koraci u lokalnom instaliranju

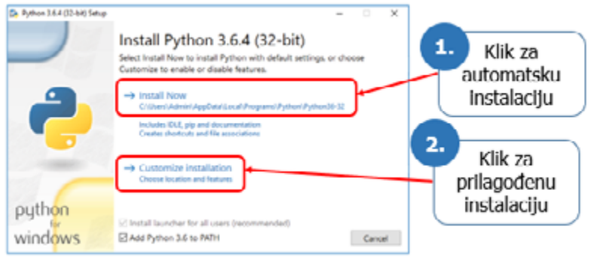
Python se najlakše upotrebljava kad se instalira lokalno. Potrebno je posjetiti mrežnu stranicu https://www.python.org/downloads



Uvod u Python

Nakon pokretanja instalacijske datoteke pojavljuje se dijaloški okvir čarobnjaka za instaliranje.

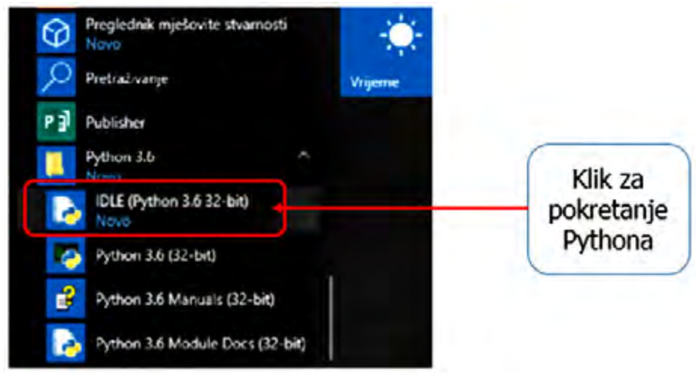
dijaloški okvir čarobnjaka za instaliranje



Nakon što se pokrene instaliranje klikom na **Install Now**, slijedi prikaz tijeka instaliranja.

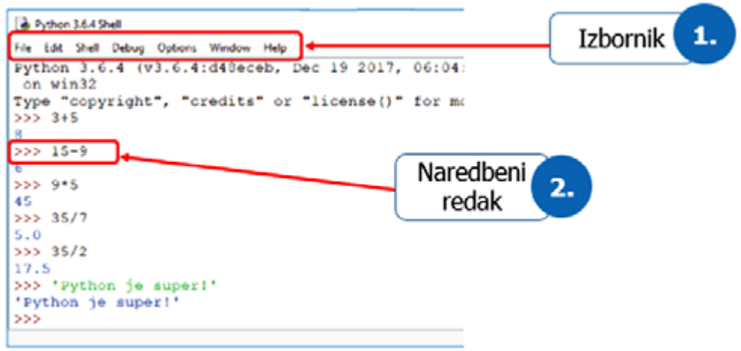
Python pokrećemo klikom na IDLE (Python GUI) s popisa programa.

pokretanje Pythona



Programiranje Pythona

Python se može programirati u tzv. ljusci (engl. *shell*) ili konzoli u kojoj se naredbe provode redak po redak kako piše. Sve što se programira u Pythonu odmah se i vidi nakon unesene **naredbe**.



primjer

Zadatak 1. Izračunajte zbroj, razliku, umnožak i količnik dvaju brojeva i ispišite rezultate.

Osnovne računske operacije u Pythonu:

Računska operacija	Znak
zbrajanje	+
oduzimanje	-
množenje	*
dijeljenje	/
djelomični količnik	//
ostatak pri dijeljenju	%

Radnja	Primjer	Opis
Djelomični količnik	`>>> 14//5` `2`	Za izračun koristimo oznaku // Rezultat dijeljenja brojeva 14 i 5 je 2.8 pri čemu je 2 djelomični količnik
Ostatak pri dijeljenju	`>>> 14%5` `4`	Za izračun koristimo oznaku %
Višečlani izraz	`>>> 20*5+45/9-55` `50.0`	Python poštuje prioritet računskih operacija. Rezultat je zbog operacije dijeljenja decimalan.

Multipliciranje teksta	`>>> 2*'abcd'`
	`'abcdabcd'`
	`>>> 3*'Python '`
	`'Python Python Python '`
		Tekst možemo multiplicirati pomoću znaka *.
		Za razdvajanje teksta koji se spaja, na kraju teksta, trebamo umetnuti jedan razmak.

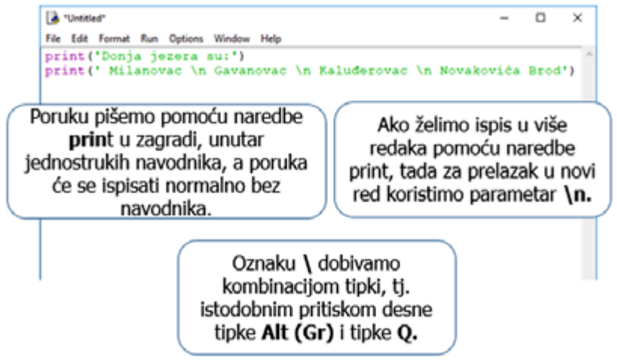
Zadatak 2. Izračunajte koliki je ostatak pri dijeljenju brojeva 1234567 i 25.

```
1234567%25
>>> 11
```

Zadatak 3. Napišite kod koji 10 puta ponavlja riječ „Ravni kotari”.

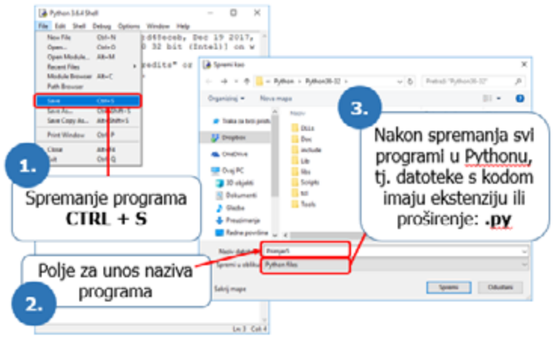
Postupci pri programiranju

pisanje programa



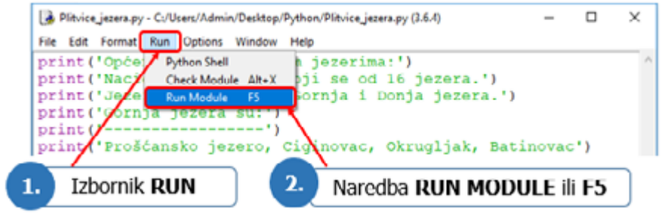
Prije pokretanja programa potrebno ga je spremiti. To se radi na sljedeći način.

spremanje programa



Nakon spremanja preostaje pokretanje programa klikom na *Run*, pa *Run Module* ili F5. Pokretanje programa jest postupak kojim počinje provođenje naredaba programskoga koda.

pokretanje programa



Programiranje u kojemu se program izvodi redoslijedom kojim je napisan, naredbu po naredbu, zovemo slijedno programiranje. Za prikaz teksta koristimo se naredbom Print.

slijedno programiranje

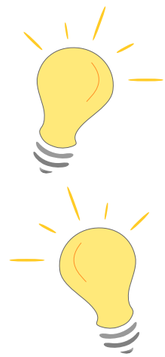
Zadatak 4. Izradite program koji zbraja i dijeli dva broja te na zaslonu ispisuje zbroj i ostatak pri dijeljenju dvaju brojeva.

```
print('Zbroj 40 + 50 = ', 40 + 50)
print('Ostatak pri dijeljenju 140 i 50 = ', 140%50)
```

Zadatak 5. Dodaj u prethodni zadatak množenje, dijeljenje i oduzimanje brojeva te djelomični količnik brojeva.

```
print ('Zbroj brojeva 50 i 40 je ', 50+40)
print ('Razlika brojeva 50 i 40 je ', 50-40)
print ('Umnožak brojeva 50 i 40 je ', 50*40)
print ('Količnik brojeva 50 i 40 je ', 50/40)
print ('Ostatak kod djeljenja brojeva 50 i 40 je', 50%40)
print ('Djelomični količnik brojeva 50 i 40 je', 50//40)
```

Provjerite i primijenite



Zadatak 1. Izradite program koji ispisuje „Danas je:” te navodi određeni datum i zbraja brojeve datuma (primjer 23. 4. 2019. = 23+4+2019).

Zadatak 2. Izradite program koji ispisuje raspored sati razreda za prva tri dana (ponedjeljak, utorak i srijedu).

```
print (35*'=')
print ('    Ponedjeljak|Utorak|Srijeda \n \t HRV\t ENG\t \n \t GK\t MAT\t PID\t\n \t PID\t MAT\t GEO\t')
print (35*'=')
```

Za one koji žele saznati više

Izradite program za rješavanje složenijeg problema – kako organizirati svoj radni tjedan ako je dostava svakog parnog dana poslijepodne, neparnog ujutro, a odlazak na tržnicu utorkom i petkom.

3. Osnovne i elementarne jedinice i pojmovi

Nakon što ste u trećoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o osnovnim i elementarnim jedinicama i pojmovima, moći ćete:

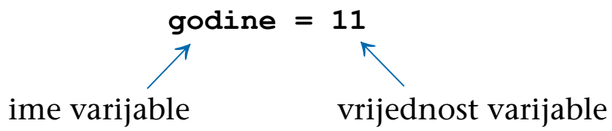
- razumjeti pojam varijable i njezinu ulogu u programiranju
- prepoznati ulazne i izlazne vrijednosti te objasniti njihovu ulogu u programiranju
- opisati osnovne korake u programiranju
- primijeniti pridruživanje varijablama i koristiti se njima za pohranu i upravljanje podatcima
- objasniti što je petlja
- razumjeti dvije vrste petlji
- primijeniti petlje bez logičkog uvjeta za iteraciju kroz nizove podataka i ponavljanje naredaba
- koristiti se brojačem petlje za praćenje njezina ponavljanja.

PowerPoint prezentacija

Snimka 3. radionice

Varijabla

Varijabla označava nešto promjenjivo. Ona je dio memorije u kojoj se pohranjuje neki izmjenjivi podatak.



Svake godine vrijednost će se promijeniti, stoga su godine varijabla, odnosno veličina koja poprima različite vrijednosti.

Svaki program radi s pomoću ulaznih i izlaznih vrijednosti.

vrijednosti varijable

ulazne vrijednosti	Ono su što korisnik unosi u računalo (tipkovnicom, mišem, prstom ili drugom ulaznom jedinicom).
izlazne vrijednosti	Rezultat su rada nekog programa (pomak pokazivača na ekranu, kretanje lika u računalnoj igri, prikaz slike na zaslonu itd.).

Rad svakog programa mogao bi se predočiti u tri osnovna koraka:

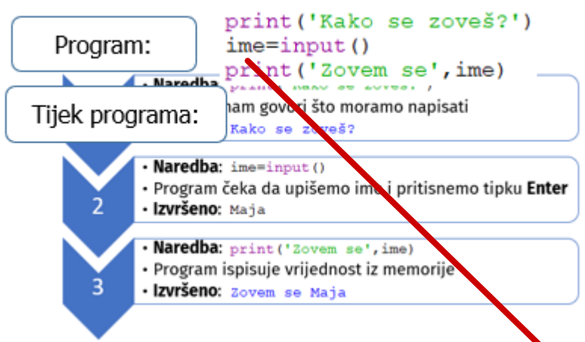
osnovni koraci u programiranju

ulaz	To su ulazne vrijednosti (zadaju se naredbom Pridruživanje).
obrada	Pridruživanje je ulaznih vrijednosti, a rezultat toga su izlazne vrijednosti.
izlaz	Izlazne su vrijednosti nastale kao rezultat obrade tih podataka (naredba Print).

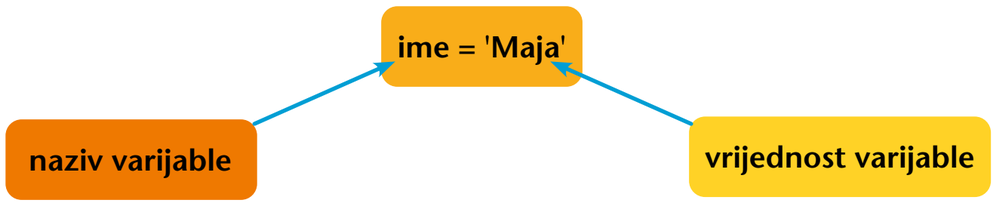
primjer

Zadatak 1. Izradite program koji unosi i ispisuje imena.

primjer uporabe varijable



Točno je da se u Pythonu ulazni podatci mogu unijeti naredbom Input() te da se pridružuju varijablama naredbom Pridruživanja kao što je prikazano u primjeru `ime=input()`. Varijabla je simboličan naziv za neku vrijednost koja se sprema i pamti u memoriji računala. Ime varijable je nepromjenjivo, a vrijednost varijable može se mijenjati tijekom izvođenja programa. Svaki put kad se varijabli pridružuje nova vrijednost, prethodna se vrijednost zamjenjuje novom vrijednošću. Varijabla je jedinstveno određena svojim nazivom i vrijednošću te se može upotrijebiti u programu za pohranjivanje, upravljanje i obradu podataka.



Broj varijabla u stvaranju programa je **neograničen**, ali svaka varijabla mora imati drukčiji naziv. Za naziv varijable ne smiju se upotrebljavati naredbe i funkcije Pythona (npr. Print, Input, int). Valja imati na umu da su „ime" i „Ime" različite varijable jer Python razlikuje velika i mala slova.

određivanje naziva varijable

Zadatak 2. Zadatak je izraditi tri varijable i program koji zbraja dva broja te ispisuje rezultat na zaslonu.

zadatak za uvježbavanje uporabe varijable

Program:

```
a=input('Upiši prvi broj:')
b=input('Upiši drugi broj:')
zbroj=a+b
print('Zbroj je:',zbroj)
```

Rezultat:

```
Upiši prvi broj:14
Upiši drugi broj:24
Zbroj je: 1424
```

- Program je umjesto zbroja izvršio spajanje brojeva.
- Python podatke doživljava kao riječi, a ne kao brojeve.

Ulazne podatke treba pretvoriti u brojeve, a potom ih zbrojiti. To se radi s pomoću funkcije `int`. Umjesto a+b, piše se int(a)+int(b):

obrada podataka

```
a=input('Upiši prvi broj:')
b=input('Upiši drugi broj:')
zbroj=int(a)+int(b)
print('Zbroj je:', zbroj)
```

Naredba **Input** sada se može zapisati i ovako: a=int(input('Upiši prvi broj')). Ovaj način zapisa zove se ugniježđena naredba. **Ugniježđena naredba** znači „naredba u naredbi". Pri njezinoj provedbi najprije će se provesti unutarnja naredba, a onda vanjska.

funkcija `int` i ugniježđena naredba

```
a=int(input('Upiši prvi broj:'))
b=int(input('Upiši drugi broj:'))
zbroj=a+b
print('Zbroj',a,'+',b,'=',zbroj)
```

Petlja

Petlja je dio programa koji se ponavlja. Postoje **dvije vrste** petlji:

petlja i vrste petlji

- petlje bez logičkog uvjeta
- petlje s logičkim uvjetom.

Petlje bez logičkog uvjeta su vrsta petlje u kojoj se unaprijed postavlja broj ponavljanja. One imaju **konačan broj ponavljanja** te uvijek počinju određenim naredbama kojima je definiran broj ponavljanja. U Pythonu se za petlju bez logičkog uvjeta upotrebljava naredba **For**.

petlje bez logičkog uvjeta

primjer

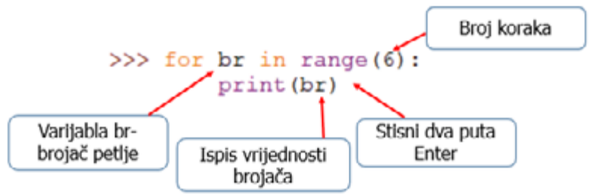
Zadatak 3. Ispišite brojeve od 0 do 10.

primjer petlje bez logičkog uvjeta

Program

```
print(0)
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

Kraći program



- Nakon dvotočke prelaskom u novi red sljedeći redak se uvukao.
- Da bi se u konzoli izvršila gornja naredba, nakon drugog retka treba dva puta stisnuti tipku Enter.

Da bi petlja za koju se upotrebljava naredba For u svakom trenutku znala na kojem je koraku ponavljanja, pomaže joj **kontrolna varijabla** petlje ili **brojač, npr. br** in range (6). Varijabla **br** je brojač petlje. Brojač se **upotrebljava tako** da se postavi **početna vrijednost** i **broj koraka** izvođenja petlje. Broj ponavljanja može se zadavati izravnim upisivanjem broja ponavljanja u naredbi petlje ili zadavanjem s pomoću varijable.

Program može sadržavati onoliko petlji koliko je potrebno uz napomenu da u radu s petljama treba paziti kako se ne bi stvorila **beskonačna petlja – program koji sam sebe pokreće i ne završava**. Beskonačna petlja je korisna u nekim sustavima koji svoj posao obavljaju bez stajanja.

Treba uvijek paziti na to da u većini programskih jezika, pa tako i u Pythonu, naredba For počne od broja 0.

Zadatak 4. Ispišite brojeve od 0 do 10.

zadatak za uvježbavanje petlje bez logičkog uvjeta

```
for br in range (11):
    print (br*2)
```

Broj ponavljanja u petlji može se dati izravno ili zadavanjem varijable unesene u program (tipkovnicom ili iz samog programa).

Zadatak 5. Izradite program koji računa zbroj prvih n brojeva.

```
print('Koliko brojeva želiš zbrojiti?')
n=int(input('Upiši broj:'))
zbroj=0
for br in range(1, n+1):
    zbroj=zbroj+br
print('Zbroj prvih',n,'prirodnih brojeva je:', zbroj)
```

Ako na memorijskoj lokaciji zbroj možda već postoji neka vrijednost, program bi tu vrijednost zbrojio s vrijednošću varijable **br**, pa upisivanjem:

zbroj = 0 *osiguravamo da se to ne dogodi.*

U rasponu **range (1, n+1)** završna vrijednost **n** treba biti povećana za 1 kako bi se dobio točan raspon jer Python završnu vrijednost uvijek smanji za 1.

Zadatak 6. Izradite program koji ispisuje prvih n neparnih brojeva.

```
n=int(input('Upiši broj n:'))
for br in range (1,2*n,2):
    print(br)
```

```
for br in range(a,b,k):
```

Varijabla br – brojač petlje

Početna vrijednost

Završna vrijednost + 1

Korak brojača petlje

n=3	n=5	n=10
1,3,5	1,3,5,7,9	1,3,5,7,9,11,13,15,17,19

```
n=int(input('Koliko puta želiš ponoviti "HOP"?'))
for br in range(0,n):
    print('HOP')
```

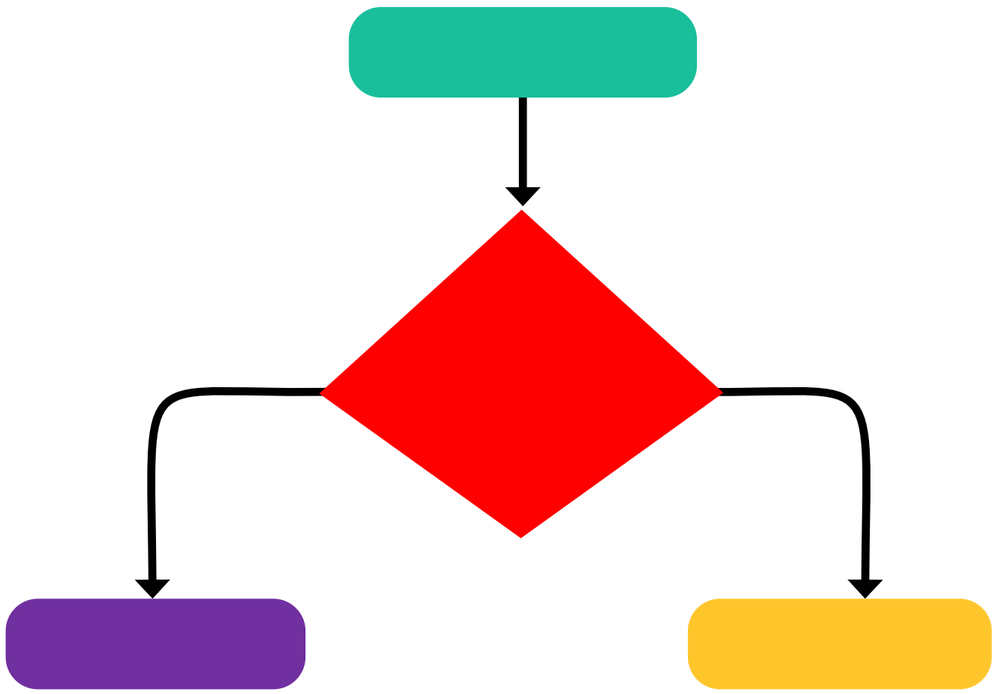
Provjerite i primijenite

Zadatak 1. Izradite program koji ispisuje prvih n parnih brojeva.

Zadatak 2. Izradite program koji ispisuje brojeve od 9 do 0.

Za one koji žele saznati više

Zadatak: Izradite program u kojemu korisnika pitamo koliki je broj ponavljanja uz uvjet pogađanja i nakon toga program toliko puta ponovi poruku „POGODIO ” uz provjeru rezultata.



4. Osnovne sastavnice kontrole tijeka

Nakon što ste u četvrtoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o osnovnim sastavnicama kontrole tijeka, moći ćete:

- razumjeti što je naredba If
- objasniti ulogu naredbe If u programiranju
- prepoznati logičke uvjete
- koristiti se operatorima usporedbe za postavljanje uvjeta u naredbi If
- razumjeti kako se istodobno provjeravaju dva uvjeta korištenjem operatora AND i OR u naredbi If
- objasniti kako se koristiti naredbom If-else za donošenje odluke kad postoji više ishoda ovisno o uvjetu
- primijeniti naredbu If-else za provjeru uvjeta i izvođenje odgovarajućeg niza naredaba
- kombinirati nekoliko uvjeta u naredbi If-else
- objasniti njihovo provođenje.

PowerPoint prezentacija

Snimka 4. radionice

Naredba If

If je složena naredba koja ovisi o logičkom uvjetu koji postavljamo. **Logički uvjet** je pitanje koje se postavlja uporabom operatora usporedbe i provjerava se istinitost izraza. Ako je logički uvjet **istinit**, provodi se niz naredaba koji se postavlja nakon uvjeta. Ako **nije istinit**, zaobilazi se grananje i nastavlja izvođenje programa.

U Pythonu a=0 znači „varijabli a pridruži vrijednost nula”, a a==0 znači „usporedi vrijednost varijable s nulom”.

Operator	Značenje
=	jednako
≠	različito (nije jednako)
<	manje
≤	manje ili jednako
>	veće
≥	veće ili jednako

Dva se uvjeta mogu provjeriti istodobno s pomoću jednog bloka **naredbe If** tako da se u logički uvjet postavi operator **and** (I) ili **or** (ILI).

operator ***and***	provjerava dva uvjeta istodobno kad su oba ispunjena, provodi naredbe unutar uvjeta
operator ***or***	provjerava dva uvjeta istodobno i, kad je barem jedan ispunjen, provodi naredbe unutar uvjeta

Grananje If-else

Logička naredba grananja **If-else** složena je naredba koja služi za donošenje odluke kad postoje **dva ili više ishoda**, ovisno o odgovoru na pitanje postavljeno u uvjetu.

Za razliku od naredbe If, ako uvjet nije ispunjen, dio skripta unutar odluke neće se zaobići, nego će se izvesti neki drugi niz naredaba.


```
If uvjet:
        naredba1
        naredba2
        ...
else:
    naredba1
    naredba2
    ...
```

primjeri upotrebe naredbe If-else

primjer

Zadatak 1. Izradite program koji provjerava je li korisnik dobro zbrojio brojeve.

```
odg=int(input('Koliko je 9+11?'))
if odg==20:
    print('Bravo! Točan odgovor!')
else:
    print('Žao mi je, netočan odgovor!')
```

1. Postaviti pitanje i unos odgovora:

```
odg=int(input('Koliko je 9+11?'))
```

2. Postaviti uvjet:

```
if odg==20:
```

3. Ako je uvjet ispunjen, tj. odgovor točan, ispiši poruku "Bravo! Točan odgovor!":

```
if odg==20:
    print('Bravo! Točan odgovor!')
```

4. Ako uvjet nije ispunjen, tj. odgovor je netočan, ispiši poruku "Žao mi je, netočan odgovor!":

```
else:
    print('Žao mi je, netočan odgovor!')
```

Taj primjer detaljno pokazuje korisnost naredbe If-else. Korak po korak u dijagramu tijekamože se vidjeti kako svaki red naredbe izvodi i provjerava uvjete postavljene u naredbi. Umjesto dviju naredaba If, upotrebljava se jedna naredba If-else.

Je li moguće postaviti više naredaba If-else te na taj način riješiti više slučajeva?

zadatci za uvježbavanje upotrebe naredbe If-else

Zadatak 2. Izradite program koji od korisnika zahtijeva da pritisne tipku A za nastavak. Ako se klikne ili učini bilo što drugo, lik ispisuje tekst: Pritisni tipku A.

```
slovo=str(input('Pritisni tipku a:'))
if slovo=='a':
    print ('Bravo! Pritisnuo si tipku a.')
else:
    print ('Šteta, pogriješio si. Nisi pritisnuo tipku a.')
```

Zadatak 3. Izradite program koji unosi broj te provjerava je li on paran.

```
a=int(input('Upiši broj:'))
ostatak=a%2
if ostatak==0:
    print(a, 'je prani broj')
else:
    print(a, 'je neparni broj')
```

prednosti uporabe naredaba If-else

Korištenjem naredaba If-else može se kombinirati nekoliko uvjeta. Provode se samo naredbe koje slijede prvi uvjet za koji se ustanovi istinitost, a sve se druge preskaču. Naredbe se nakon konačne naredbe If-else provode ako nijedan od uvjeta nije istinit.

Provjerite i primijenite

Zadatak 1. Provjerite kod programa koji unosi dva broja te provjerava njihovu djeljivost.

```
a=int(input('Upiši broj a:'))
b=int(input('Upiši broj b:'))
ostatak=a%b
if ostatak==0
    print(a,'je djeljiv s', b)
else:
    print(a,'nije djeljiv s', b)
```

Zadatak 2. Provjerite kod programa koji unosi broj te provjerava je li djeljiv sa sedam.

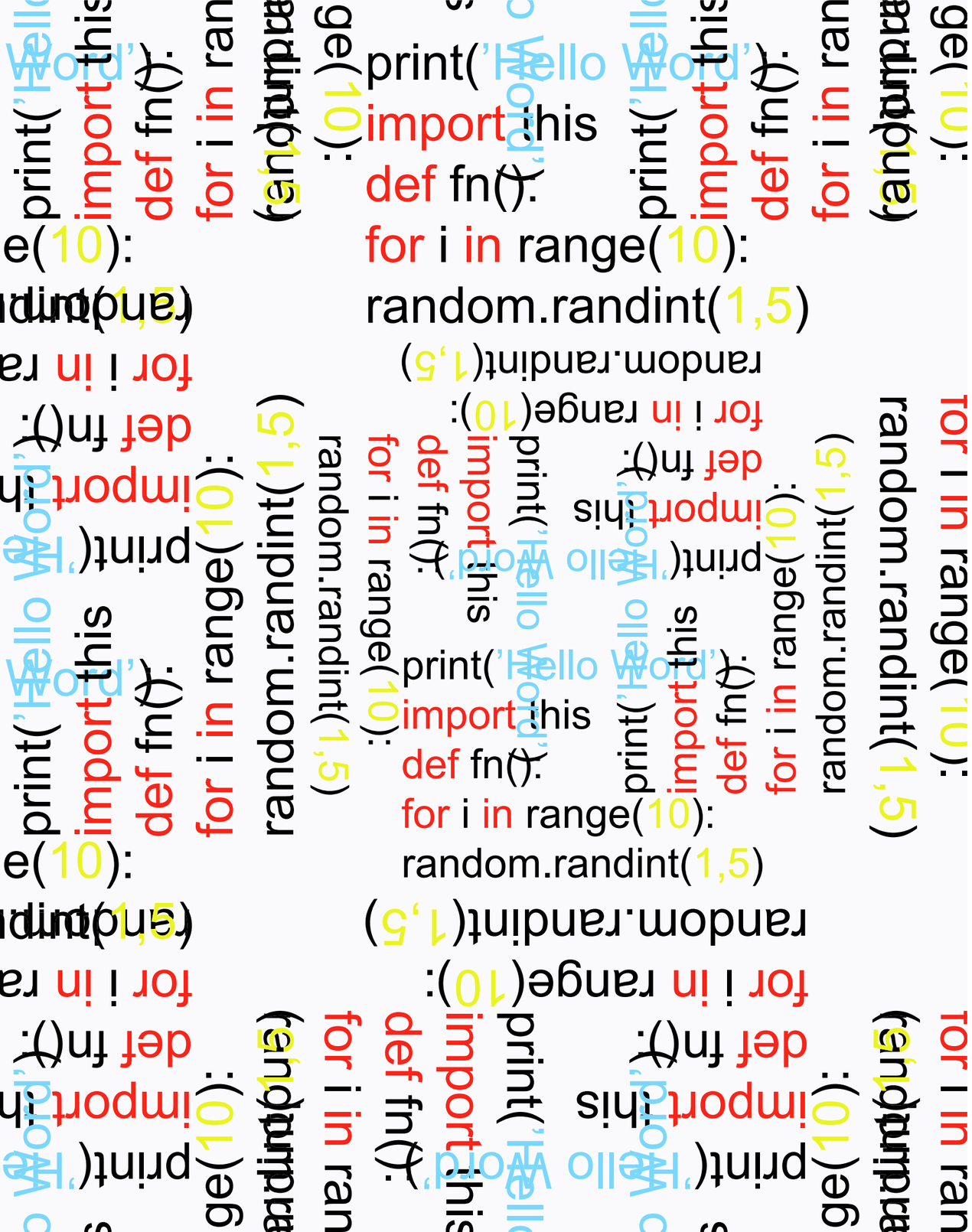
```
a=int(input('Unesi broj'))
if a%7==0
    print('Broj',a, 'je djeljiv sa sedam.')
else:
    print('Broj',a, 'nije djeljiv sa sedam')
```

Zadatak 4. Izradite program koji unosi broj te provjerava je li broj pozitivan.

Za one koji žele saznati više

Zadatak 1. Izradite program koji unosi duljine dviju stranica. Ako su unesene duljine jednake, izračunat će površinu kvadrata, a ako su različite izračunat će površinu pravokutnika.

Zadatak 2. Izradite program koji unosi dva broja, zbraja ih te provjerava je li zbroj veći od 20.



5. Funkcije

Nakon što ste u petoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o funkcijama, moći ćete:

- definirati funkciju i njezin naziv
- razumjeti različite vrste parametara u funkcijama (obvezatne parametre, opcionalne parametre i parametre s pretpostavljenom vrijednošću)
- koristiti se različitim vrstama parametara u funkcijama (obvezatnim parametrima, opcionalnim parametrima i parametrima s pretpostavljenom vrijednošću)
- razumjeti različite vrste povratnih vrijednosti u funkcijama (jednu vrijednost, više vrijednosti i nula vrijednosti)
- koristiti se različitim vrstama povratnih vrijednosti u funkcijama (jednom vrijednošću, s više vrijednosti i nula vrijednosti)
- pozivati funkciju i prosljeđivati argumente
- razumjeti što su globalne i lokalne varijable i kako se njima koristiti u funkcijama
- koristiti se funkcijama za strukturiranje koda i izbjegavanje ponavljanja koda
- razumjeti što su rekurzivne funkcije i kako se njima koristiti za rješavanje problema
- razumjeti koncept modula i kako se koristiti funkcijama iz drugih modula u Pythonu
- koristiti se ugrađenim funkcijama u Pythonu i razumjeti njihove funkcionalnosti.

PowerPoint prezentacija

Snimka 5. radionice

Odrednice funkcije

svrha funkcije

Funkcija je dio programskoga kôda koji je organiziran prema određenoj logičkoj cjelini. Cilj je korištenja funkcijom da se dijelovi programske logike koji se ponavljaju u programskom kôdu napišu samo jednom, i to unutar funkcije. Uporabom funkcija jedan te isti kôd može se iskoristiti neograničen broj puta jednostavnim pozivom funkcije koja je u nekom trenutku potrebna. Takav pristup omogućuje veću preglednost i organizaciju programskoga kôda. Jednako tako znatno je lakše nakon pisanja pročitati programski kod koji je zapisan koristeći se funkcijama. Funkcijama se koristimo kao osnovnim građevnim sastavnicama te se na temelju njih jednostavnije mogu slagati složeniji blokovi koda a da se pritom zadrži čitkost.

Dakle, može se reći da funkcija služi:

- minimiziranju koda
- postizanju veće razine apstrakcije/čitkosti koda
- ponovnoj uporabi koda / optimiranju.

savjet za kreiranje funkcije

Glavna je misao vodilja pri kreiranju funkcija da odsječak koda unutar funkcije bude što jednostavniji, tj. podijeljen na što manje logičke cjeline. Takva organizacija omogućuje veću ponovnu iskoristivost programskoga kôda.

unaprijed napisane funkcije

Funkcije mogu biti napisane unaprijed. Ako su funkcije napisane unaprijed, to su funkcije koje su ugrađene u sâm programski jezik ili su dio standardnog ili uključenog paketa. Pojam paketa detaljnije je obrađen na 8. radionici *Moduli i paketi*.

obvezatne sastavnice funkcije

Funkcija nužno mora imati naziv, nula ili više ulaznih parametara te nula ili više izlaznih vrijednosti.

```
def ime_funkcije (popis parametara)
        blok_naredbi
        return vrijednosti
```

Funkcija se sastoji od zaglavlja funkcije i tijela funkcije. U nastavku slijedi primjer definicije funkcije.

sastavnice i primjer funkcije

primjer

```
def sumaBrojeva():
    # tijelo funkcije
```

Svaka funkcija koja se želi primijeniti počinje ključnom riječju ***def*** nakon čega slijedi ime funkcije. U prethodnom slučaju ime funkcije je `sumaBrojeva()`. Ime funkcije može biti proizvoljno, no treba pripaziti na to da se funkcija ne zove poput neke od ključnih riječi u *Pythonu* ili pak imenom neke druge funkcije. Nakon imena funkcije obvezatne su zagrade i nakon zagrada dvotočka.

opis funkcije s pomoću prethodnog primjera

Tijelo funkcije mora biti uvučeno – u suprotnom Pythonov tumač javlja pogrešku pri pokretanju.

savjet za pisanje funkcije

Za razliku od drugih programskih jezika koji se koriste vitičastim zagradama ili ključnim riječima za razlikovanje programskih blokova, *Python* se koristi uvlačenjem. U prethodnom slučaju tijelo funkcije je programski blok koji je potrebno na neki način omeđiti kako bi se pri provođenju znalo koje sve linije programskoga kôda potpadaju pod funkciju.

razlikovanje programskih blokova uvlačenjem

Python se koristi uvlačenjem kao načinom razlikovanja programskih blokova. Povećanje uvlačenja znači da dolazi novi, ugniježđeni blok, a smanjenje označava kraj trenutačnog bloka programskoga kôda.

Nakon što je funkcija definirana i upisana u tumač te sadržava ispravno napisane naredbe (logički i sintaksom), može se pozivati tako da se napiše njezino ime te u oblim zagradama navedu odgovarajući argumenti (vrijednosti parametara koje su ulaz u funkciju). Argumenti funkcije bit će obrađeni u nastavku.

Sintaksa poziva funkcije bez parametara i povratnih tipova prikazana je u sljedećem primjeru:

primjer poziva funkcije

primjer

```
>>> def moja_funkcija():
...     print("Pozdrav iz funkcije")
...
...
>>> moja_funkcija()
Pozdrav iz funkcije
>>>
```

Kao što se može vidjeti u primjeru, najprije je napisana funkcija (zaglavlje i tijelo funkcije) te je tek nakon toga izveden sâm poziv funkcije `moja_funkcija()`. Bitno je primijetiti to da definicija funkcije mora biti napisana ispred samog poziva funkcije. Ako je poziv funkcije u programskom kôdu ispred definicije funkcije, kod pokretanja programa dogodit će se pogreška. U primjeru koji slijedi može se vidjeti pozivanje funkcije prije njezine definicije.

Korišten je IDLE u interaktivnom načinu rada, no ista bi se pogreška pojavila u pokretanju iz konzolnog načina rada.

primjer

```
>>> moja_funkcija()
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    moja_funkcija()
NameError: name 'moja_funkcija' is not defined
>>>
>>> def moja_funkcija():
...     print("Pozdrav iz funkcije")
...
...
>>>
```

primjer pisanja poziva funkcije prije njezine definicije

Za davanje naziva funkcijama preporučuju se sljedeća pravila:

pravila za davanje naziva funkcijama

- sve treba pisati malim slovima
- ključne riječi i razmaci nisu dopušteni
- umjesto razmaka stavlja se donja crta (engl. *underscore*).

Parametri funkcije

Da bismo funkcijama mogli slati različite vrijednosti na temelju kojih će funkcije provesti neku obradu, u zaglavlju funkcije u oble zagrade upisuju se parametri koje funkcija prima.

Formalni parametri funkcije mogu se mijenjati ovisno o potrebi. U istom programu može se nekoliko puta pozvati jedna te ista funkcija, primjerice funkciju `kvadrat()`.

formalni parametri

primjer

Funkcija kvadrat() prima samo jedan parametar i, ovisno o vrijednosti tog parametra, ona izračunava kvadrat unesenog broja. Prvi put se poziva funkcija kvadrat(3), a drugi put kvadrat(5). Primjećuje se da rezultat funkcije pri prvom pozivu mora biti 9, a rezultat funkcije pri drugom pozivu mora biti 25, tj. da rezultat ovisi o vrijednostima koje je funkcija primila.

primjer funkcije kvadrata

Pozicija u popisu varijabla važna je za pozivanje. Tip se implicitno dinamički zaključuje.

„a”,2,

„a”,”b”

Primjer

primjer

```
>>> def zbroji(a,b):
...     return a+b
...
>>> zbroji(3,2)
5
```

primjer sljedivosti

Formalni parametri su parametri, tj. varijable koje se nalaze u definiciji funkcije.

formalni parametri

```
def kvadrat(x):
```

Varijabla x je formalni parametar. Ovdje je prikazan primjer definicije funkcije `kvadrat()`. Vidljivo je da funkcija prima jedan parametar po imenu x. U tijelu funkcije na temelju vrijednosti u varijabli x izračunava se i ispisuje kvadrat prenesene vrijednosti.

```
def kvadrat(x):
    print(x*x)
```

Funkcije mogu primati i više parametara. Na primjer, ako funkcija mora vratiti zbroj tri broja, a sve su tri vrijednosti predane funkciji kao parametri, definicija funkcije izgleda ovako:

```
def suma(var1, var2, var3):
    print(var1 + var2 + var3)
```

poziv funkcije s formalnim parametrima

U donjem se primjeru vidi definicija funkcije `suma()` s dva formalna parametra var1 i var2. Nakon same funkcije (zaglavlja i tijela funkcije) nalazi se poziv funkcije `suma(10, 20)`. U pozivu se prenose u funkciju vrijednosti 10 i 20. Vrijednost 10 sprema se u varijablu var1, a vrijednost 20 u varijablu var2. Pri pozivu funkcije formalni parametri zamjenjuju se stvarnim argumentima, tj. konkretnom vrijednošću.

```
def suma(var1, var2):
    print(var1 + var2)

suma(10, 20)
```

```
Izlaz:
30
```

argument

U gornjem dijelu objašnjenja uveden je novi pojam – argument. Riječi parametar i argument često se miješaju iako je u biti riječ o jednom te istom pojmu koji se gleda iz različitih perspektiva.

parametri funkcije

Parametri funkcije – to su varijable koje su napisane i koje se rabe u samoj funkciji (zaglavlje funkcije i tijelo funkcije). Parametri funkcije mogu se smatrati običnim varijablama.

argumenti funkcije

Argumenti funkcije – to su vrijednosti koje se koriste pri pozivanju funkcije.

unaprijed zadane vrijednosti parametara

Python omogućuje to da se unaprijed zadaju predefinirane vrijednosti parametara funkcije. To je omogućeno jer broj parametara funkcije ne mora uvijek odgovarati broju argumenata koji se šalju pri pozivu funkcije.

U nastavku je prikazan način s pomoću kojega se parametru funkcije postavlja unaprijed zadana vrijednost. U donjem slučaju ta unaprijed zadana vrijednost pridružena je varijabli (formalnom parametru funkcije) var3 na

vrijednost 0. Varijabla var3 poprimit će vrijednost 0 samo ako se pri pozivu funkcije prenesu dva argumenta, a ne sva tri koliko ih maksimalno može biti. Ako se prenesu tri argumenta, varijabla var3 poprima vrijednost trećega prenesenog argumenta.

Prikazan je način s pomoću kojega se parametru funkcije postavlja unaprijed zadana vrijednost.

primjer

```
def suma(var1, var2, var3=0):
    print(var1 + var2 + var3)

suma(10, 20)
```

```
Izlaz:
      30
```

primjer načina s pomoću kojega se parametru funkcije postavlja unaprijed zadana vrijednost

U gornjem primjeru pri pozivu funkcije `suma()` prenesene su vrijednosti 10 i 20, tj. prenesena su samo dva argumenta. U ovom slučaju varijabla var3 poprima predefiniranu vrijednost, a to je vrijednost 0.

Varijable var1 i var2 poprimaju vrijednost iz poziva funkcije – 10 i 20.

primjer

```
def suma(var1, var2, var3=0):
    print(var1 + var2 + var3)

suma(10, 20, 500)
```

```
Izlaz:
      530
```

U ovom primjeru prenose se tri argumenta, tj. vrijednosti. Kako funkcija `suma()` može poprimiti najviše tri parametra, pri pozivu funkcije `suma()` prenijeli smo tri argumenta u funkciju. Varijabla var3 neće poprimiti predefiniranu vrijednost 0, već će poprimiti vrijednost trećega prenesenog argumenta iz poziva funkcije `suma()`, tj. vrijednost 500.

Nekoliko se funkcija može pozivati iz tijela jedne funkcije te se tako postiže ulančavanje i bitno čitkiji kod cijelog programa. Princip s tim primjerom prikazan je na donjoj slici.

ulančavanje poziva više funkcija

```
>>> def vece(broj):
...     print(f"{broj} je veći")
...
...
>>> def manje(broj):
...     print(f"{broj} je manji")
...
```

Najprije se definiraju funkcije na „najnižoj razini”, drugim riječima one funkcije koje sadržavaju najmanje funkcionalnosti i na kojima se grade druge funkcije.

Nakon toga tako definirane i isprobane funkcije mogu se pozivati iz drugih funkcija kao što je prikazano na donjem primjeru.

primjer

primjer načina pozivanja iz drugih funkcija

```
>>> def usporedi(a,b):
...     if a>b:
...         vece(a)
...         manje(b)
...     else:
...         vece(b)
...         manje(a)
...
...
>>> usporedi (3,10)
10 je veći
3 je manji
```

Provjerite i primijenite

1. Funkcije možemo podijeliti na dva dijela. Koji su to dijelovi?
2. Ako funkcija prima pet parametara, je li pri pozivu funkcije potrebno navesti svih pet argumenata ili postoji način da se neki argumenti izostave, a ako postoji,koji bi to način bio?
3. Napišite funkciju koja prima dva parametra. Rezultat izračuna funkcije ovaj se put ne ispisuje izravno u funkciji, nego u glavnom dijelu programa. Funkcija mora izračunati rezultat formule:

   (a*a) + (b*b)

   Rezultat spremite u varijablu koja se nalazi u dijelu programskoga kôda u kojem se funkcija poziva te ispišite tu varijablu.

4. Pozovite funkciju koja ne prima nijedan parametar, ali mora izračunati i ispisati zbroj dvaju brojeva. Brojevi neka se dohvate iz glavnog dijela programa preko globalnih varijabli.
5. Prethodni zadatak napišite tako da se ne koristite globalnim varijablama (korištenjem parametara funkcije).
6. ** Izradite program koji će inicijalizirati u varijable n i m dva cijela broja proizvoljnih vrijednosti. Provjerite zadovoljavaju li inicijalizirane vrijednosti uvjet: `0 ≤ n ≤ m`. Ako uvjet nije zadovoljen, tada ispišite poruku: „Nedopuštene vrijednosti!” Ako je uvjet zadovoljen, izračunajte i ispišite binomni koeficijent „m povrh n” pri čemu se koristite sljedećim izrazom:

$$\binom{m}{n} = \frac{m!}{[n! * (m-n)!]}$$

Primjer: 5! izračunava se na način 5*4*3*2.

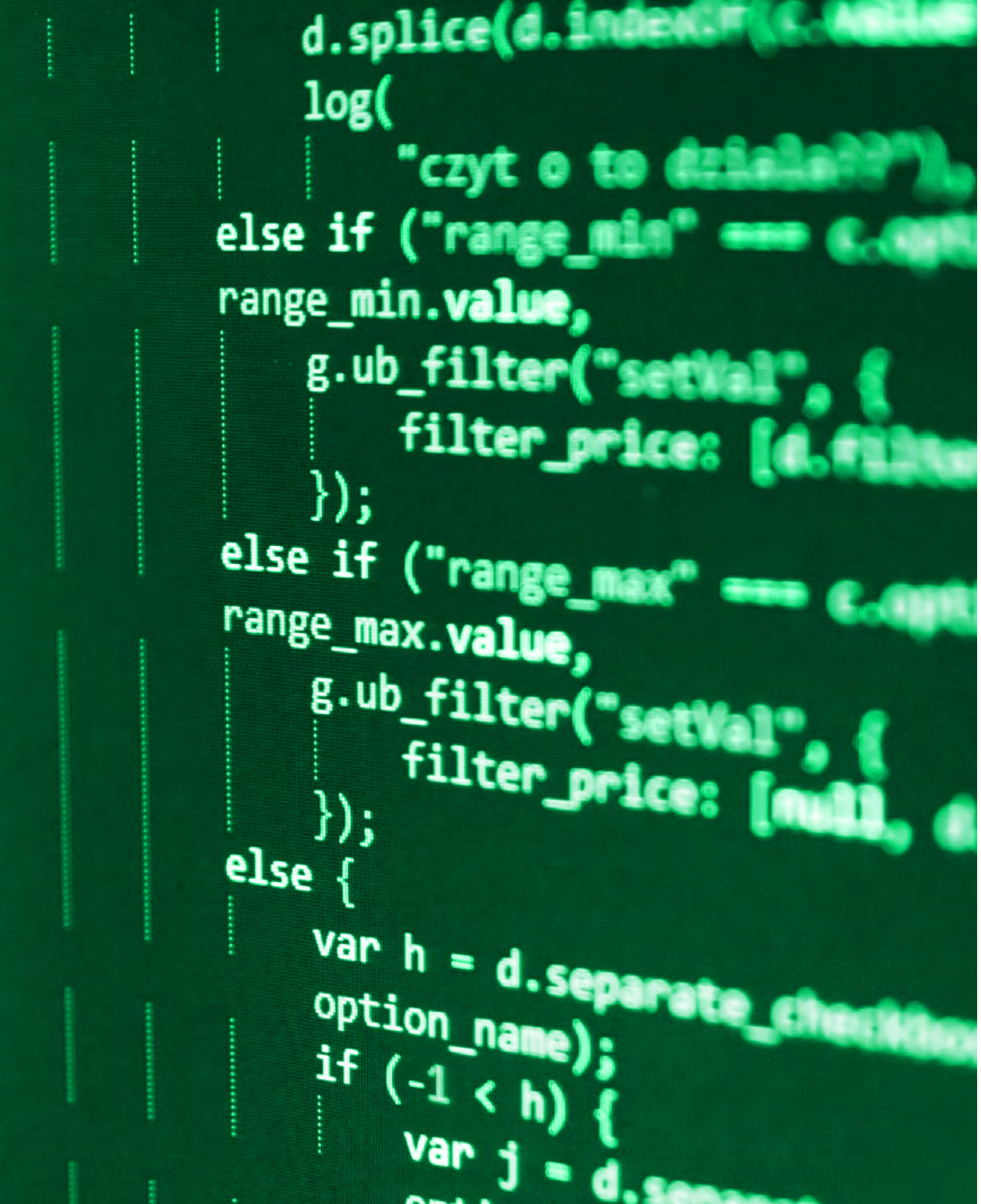
(Napomena: Primijenite funkciju `fakt()`. Tako primijenjena funkcija izračunava faktorijele, na primjer za poziv funkcije fakt(5) povratna vrijednost je 120. Funkcija se mora pozivati iz glavnog programa za sve tri vrijednosti: m!, n!, (m-n)!).

Za one koji žele saznati više

Nekoliko tema preporučuje se kao samostalno istraživanje za polaznike koji su s lakoćom svladali dosadašnje gradivo *dubljom obradom funkcija*.

Promjena redoslijeda parametara **Ugnježđivanje funkcija**

Rekurzija



6. Skladišta podataka – varijable

Nakon što ste u šestoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o varijabli, moći ćete:

- objasniti pojam vidljivosti varijable u programiranju
- razlikovati lokalne i globalne varijable
- opisati životni vijek varijable
- napraviti dijagram odnosa između globalne i lokalne varijable
- objasniti vidljivost lokalnih varijabla i njihov životni vijek
- objasniti vidljivost globalnih varijabla i njihov životni vijek
- definirati predugrađene varijable
- objasniti preklapanje vidljivosti i nadjačavanje
- primijeniti ključnu riječ global za pristup globalnim varijablama.

PowerPoint prezentacija

Snimka 6. radionice

Vidljivost varijable

Vidljivost varijable je pojam koji određuje iz kojih je dijelova programa neka varijabla dohvatljiva, tj. dostupna. Varijable mogu biti lokalne ili globalne.

Životni vijek varijable je pojam koji je određen trenutkom u kojemu je varijabla nastala u memoriji i trenutkom u kojemu se ta ista varijabla briše iz memorije. Životni vijek varijable ograničen je završetkom bloka programskoga kôda u kojemu je ta varijabla nastala, na primjer funkcija.

životni vijek varijable

Ugrađene varijable

U širem kontekstu riječ je o varijablama dostupnima unutar razine jedne instancije tumača (engl. *shell*) ili varijablama koje su dostupne u svim instancijama tumača ili konzolnog načina rada.

Prikazan je dijagram odnosa između ovih pojmova.

dijagram odnosa između globalne i lokalne varijable

Lokalne varijable

Lokalne varijable su varijable koje su korištene unutar tijela funkcija, a u tu se kategoriju mogu svrstati i parametri funkcija. Načelno se može smatrati da se parametri funkcija ponašaju identično kao i obične varijable unutar tijela funkcije.

Lokalne su varijable vidljive samo unutar funkcije u kojoj su prvi put upotrijebljene. Prvom upotrebom može se smatrati izraz pridruživanja vrijednosti varijabla, na primjer var=10.

vidljivost lokalnih varijabla

Ako je varijabla definirana unutar funkcije, nakon što izvođenje funkcije završi (bilo s pomoću naredbe Return ili pak završetkom tijela funkcije), varijabla se trajno uništava. To znači da sljedeći poziv iste funkcije više ne može vidjeti vrijednost varijable iz prethodnog poziva (jer je varijabla pri izlasku iz funkcije trajno uništena).

životni vijek varijable

primjer programa

primjer

```
13. def test():
14. var=5
15.
16. test()
    print(var)
```

```
Izlaz:
     NameError: name 'var' is not defined
```

U prethodnom programu u glavnom dijelu programa pozvana je funkcija `test()`, a unutar funkcije `test()` definirana je varijabla var s vrijednošću 5. Nakon što se izađe iz funkcije (funkcija ništa ne ispisuje, već samo postavlja vrijednost varijable var), u glavnom se programu pokušava ispisati vrijednost varijable var, no nakon izlaska iz funkcije, varijable var više nema.

Nakon izlaska iz funkcije životni je vijek varijable završio, pa te varijable više nema ni u memoriji, a usto je varijabla var lokalna varijabla te joj se ne može pristupiti iz glavnog dijela programa jer je vidljiva samo unutar funkcije.

primjer sistemske pogreške na upravitelju zadataka unutar operativnog sustava

Globalne varijable

Uz lokalne postoje i globalne varijable. Globalne varijable su varijable koje su kreirane i korištene u glavnom dijelu programa i njihov životni vijek traje dokle god se program izvršava. Globalne varijable zovu se tako jer osim što im se može pristupati iz glavnog dijela programa, može im se pristupati i iz funkcija.

primjer

```
def test():
    print(var)

var=5
test()
```

```
Izlaz:
      5
```

primjer programa

U prethodnom programu kreirana je varijabla var i pridružena joj je vrijednost 5. Nakon toga u sljedećoj liniji je pozvana funkcija `test()`. Funkcija `test()` ispisuje sadržaj varijable var, no primjećuje se da funkcija nema ni formalne parametre ni varijable koje bi bile definirane u njoj, ali da svejedno uspijeva ispisati vrijednost varijable var. To znači da je varijabla var globalna varijabla koja je vidljiva iz svih dijelova programa (što uključuje i funkcije).

Deklaracija globalnih varijabla može biti napravljena izravno u glavnom tijelu programa, bez enkapsulacije u funkcije/metode kao u donjem primjeru.

deklaracija globalnih varijabla

primjer

```
>>> globalna_varijabla=100
>>>
>>> print(globalna_varijabla)
100
```

Alternativno se isto može učiniti i unutar bloka funkcije kao što je prikazano u sljedećem primjeru.

primjer

```
>>> def misli_globalno():
...     global pero
...     pero=250
...
...
>>> pero
Traceback (most recent call last):
  File "<pyshell#83>", line 1, in <module>
    pero
NameError: name 'pero' is not defined
>>> misli_globalno()
>>> pero
250
```

Predugrađene varijable

To su varijable koje su u istom obliku dostupne za čitanje iz bilo koje konzole ili tumača.

Primjer takve varijable je „credits” koja ispisuje niz string s osnovnim informacijama o organizacijama koje podržavaju rad Pythonove zajednice.

primjer

```
>>> credits
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast of thousands
    for supporting Python development.  See www.python.org for more information.
```

primjer predugrađene varijable

Ovo je popis svih predugrađenih varijabla i imena:

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning',
 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning',
'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError',
 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError',
'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',
'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError',
'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError',
'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError',
'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError',
'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning',
 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '__build_class__',
 '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__',
 '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray',
'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright',
 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec',
'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr',
'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter',
'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range',
'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod',
'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

Preklapanje vidljivosti

Ako se u različitim dosezima koristimo istim nazivom za varijablu, vrijednost varijable šireg dosega prekriva se varijablom užeg dosega.

Ako globalna i lokalna varijabla imaju isti naziv, događa se nadjačavanje, tj. preklapanje vidljivosti u kojoj ime koje je deklarirano „bliže" lokaciji poziva ima veću važnost te se stoga varijabla s manjom važnošću u tom trenutku ne vidi preko poziva istog imena.

Kao što je vidljivo u sljedećem primjeru, definiranjem globalne varijable ona postaje dostupna u kompletnom tumaču.

primjer

```
>>> globalna_varijabla=100
>>>
>>> globalna_varijabla
100
```

primjer definiranja globalne varijable

Kada se globalnom varijablom koristimo unutar funkcije, ona također preko imena ima istu vrijednost kao i pozivom izvan funkcije.

primjer

```
>>> def pisi():
...     print (globalna_varijabla)
...
...
>>> pisi()
100
```

U situaciji kad se unutar tijela funkcije definira globalna varijanta s istim imenom kao i globalna varijabla, lokalna varijabla će isključivo u toj funkciji nadjačati poziv i dodjelu vrijednosti kao što je prikazano u sljedećem primjeru.

primjer

```
>>> def pisi2():
...     globalna_varijabla=200
...     print(globalna_varijabla)
...
...
>>> pisi()
100
>>> pisi2()
200
>>> pisi()
100
```

primjer definiranja globalne varijante s istim imenom kao i globalna varijabla

Kako bi se eksplicitno odredilo da se iz tijela funkcije želi djelovati nad globalnom varijablom ili joj pristupiti, potrebno je koristiti se ključnom riječju global.

primjer

```
>>> def pisi2_bez_preklapanja():
...     global globalna_varijabla
...     globalna_varijabla=200
...     print(globalna_varijabla)
...
...
>>>
```

primjer korištenja s ključnom riječju „global”

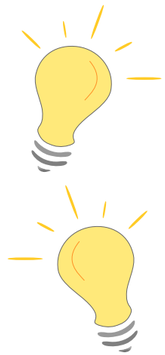
Sukladno navedenu primjeru, kad se pozove ova funkcija, ona iz svojeg tijela promijeni vrijednost globalne varijable.

```
>>> pisi()
100
>>> pisi2_bez_preklapanja()
200
>>> pisi()
200
```

izlazak izvan okvira dosega funkcije

Provjerite i primijenite

1. Kako Python razlikuje globalne i lokalne varijable?
2. Koje su prednosti i mane korištenja globalnim varijablama u Pythonu?
3. Može li se lokalna varijabla iz jedne funkcije upotrijebiti u drugoj funkciji? Zašto?
4. Kako se može spriječiti preklapanje vidljivosti u Pythonu?



Za one koji žele znati više

Polaznicima koji žele proširiti svoje znanje s područja skladišta podataka – varijabla preporučuje se sljedeća tema kao samostalno istraživanje unutar Pythonova razvojnog okružja.

- Koncept *Namespace*
- Ugnježđivanje i enkapsulacija
- Metoda O*verride* – kod objektnog programiranja

7. Proces upravljanja memorijom računala – napredno korištenje Pythonom

Nakon što ste u sedmoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o procesu upravljanja memorijom računala – naprednim korištenjem programskim jezikom Python, moći ćete:

- razumjeti kategorije memorije dostupne na računalu
- razumjeti razlike između fizičke radne memorije i radne memorije operativnog sustava
- razumjeti važnost upravljanja potrošnjom memorije u programiranju
- razumjeti automatski mehanizam za upravljanje potrošnjom memorije u modernim programskim jezicima
- razumjeti životnog ciklus varijabla/objekata u programskom jeziku *Python*
- razumjeti globalne varijable i njihovo zauzimanje memorije u shellu
- poznavati naredbu Del za ručno uklanjanje varijabla
- razumjeti podjele objekata na tri klase (generacije) u automatskom mehanizmu za upravljanje memorijom u Pythonu.

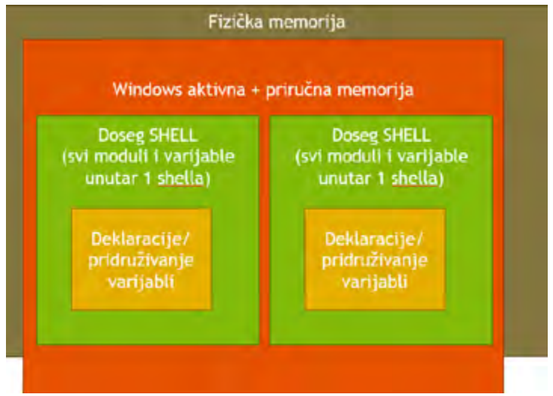
PowerPoint prezentacija

Snimka 7. radionice

Upravljanje memorijom računala

Sva memorija koja je dostupna na računalu dijeli se na nekoliko kategorija što je prikazano na sljedećem dijagramu.

kategorije memorije



Fizička radna memorija konačnog je dosega, no radna memorija kojom se koristi operativni sustav (u ovom slučaju Windows) zbirno je većeg iznosa nego isključivo fizička radna memorija zbog uporabe stranične datoteke.

razlika između fizičke radne memorije i radne memorije operativnog sustava

Unutar memorije kojom se koristi operativni sustav nalaze se sve aplikacije uključujući i tumač (engl. shell) u kojemu se nalaze sve varijable kojima se program koristi.

primjer

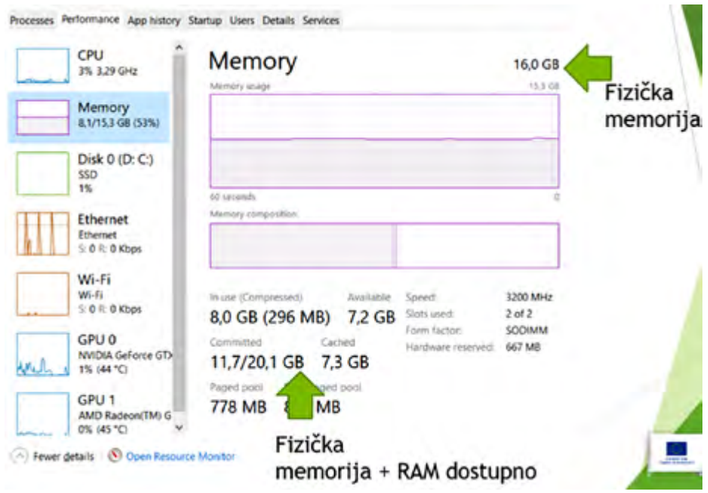
Ako se uporabom potroši više memorije od fizički dostupne i veličine definirane u postavkama za straničnu datoteku, a proces za uklanjanje memorije kojom se ne koristi, događaju se razne sistemske pogreške. Primjer je prikazan na slici.

primjer sistemske pogreške

primjer

Jednako je vidljivo i na upravitelju zadataka unutar operativnog sustava.

primjer sistemske pogreške na upravitelju zadataka unutar operativnog sustava



Kako je već pokazano u prijašnjim dijelovima ovog poglavlja, ako se potrošnjom memorije ne upravlja na kvalitetan način, posljedice za rad našeg programa i cijelog računala koje ga izvodi mogu biti izrazito negativne. Zbog toga je u današnjim modernim programskim jezicima upravljanje potrošnjom memorije automatsko. U prvim programskim jezicima upravljanje potrošnjom memorije bilo je ručno jer nisu postojali automatski mehanizmi.

upravljanje potrošnjom memorije

Ručno upravljanje potrošnjom memorije često bi prouzročilo sljedeće situacije:

nedostatci ručnog upravljanja potrošnjom memorije

- vrlo česte pogreške sa zaboravljanjem oslobađanja memorije
- vrlo česte pogreške s preranim oslobađanjem memorije.

Kako je spomenuto, moderni programski jezici (uključujući Python) već imaju ugrađeno upravljanje potrošnjom memorije te se ti mehanizmi neovisno o korisniku brinu o:

prednosti programskog upravljanja potrošnjom memorije

- potrebama kada treba ukloniti neku varijablu koja se više ne upotrebljava
- alociranju sistemske memorije da bude dovoljno memorije za objekte/podatke u memoriji tumača (engl. *shell*).

Skupni naziv za sve mehanizme za upravljanje potrošnjom memorije engleski je termin *Garbage collection*, tj. uklanjanje nekorištenih objekata iz memorije.

uklanjanje nekorištenih objekata iz memorije

Automatski mehanizam za upravljanje potrošnjom memorije koji uklanja nekorištene objekte iz memorije može taj zadatak raditi s nekoliko razina agresivnosti uklanjanja objekata koje variraju od minimalne do maksimalne. Svaki od tih pristupa ima svoje pozitivne i negativne strane.

automatski mehanizam za upravljanje potrošnjom memorije

preagresivno (maksimalno) uklanjanje objekata	Vodi do sporijeg izvršavanja zbog ponovnog učitavanja ako je objekt ponovno potreban, ali je zahvaljujući njemu gotovo uvijek dostupna dovoljna količina memorije.
konzervativno uklanjanje objekata (minimalno)	Vodi do velike uporabe resursa računala, ali će zahvaljujući njemu jednom stvorene varijable uvijek biti dostupne.

Automatski mehanizam za upravljanje potrošnjom memorije događa se prema unaprijed postavljenim postavkama umjerenim tempom u pozadini rada i izvršavanja programa. U situacijama u kojima programer unaprijed zna da će biti velika potrošnja memorije zbog jednokratnog korištenja nekih objekata/varijabli može se ručno ubrzati oslobađanje memorije. Alati koji služe za praćenje potrošnje memorije / brzine izvršavanja programa zovu se na engleskom jeziku profileri.

djelovanje automatskog mehanizma za upravljanje potrošnjom memorije

Životni vijek varijabla/objekata u Pythonu

Svaki objekt koji se rabi u programskom jeziku Python prolazi tri faze u svojemu životnom ciklusu: fazu stvaranja, fazu korištenja i fazu uništenja.

Stvaranje (engl. *instantiation*) je prva faza u životnom ciklusu varijable, tj. objekta u programskom jeziku Python koja počinje čim se u programskom kodu prvi put navede ime te varijable. Prvi korak u toj fazi kroz koju prolaze svi objekti je konstrukcija. U primjeni to znači da se poziva funkcija, tj. metoda zvana konstruktor. Uobičajen sadržaj rada metode konstruktora je:

stvaranje objekata

- **prvi korak** – rezerviranje memorije
- **drugi korak** – dodjela vrijednosti.

Nakon ovoga slijedi aktivno korištenje varijablom, tj. objektom. Tijekom faze korištenja svaki objekt ima jednu ili nekoliko referencija, tj. poziva iz drugih odsječaka programskoga koda.

korištenje objektom

Na kraju životnog ciklusa svake varijable, tj. objekta u programskom jeziku Python dolazi faza koja se zove destrukcija.

uništavanje objekata

Slično kao kod metode, tj. funkcije koja se poziva pri stvaranju objekta, i pri uništavanju objekta poziva se metoda koja se zove destruktor, a oslobađa memoriju koju je varijabla ili objekt zauzeo tijekom svojega životnog ciklusa. Destruktor se poziva ili ručno ili mehanizmom automatskog upravljanja memorijom nakon što u određenu razdoblju nije bilo aktivne uporabe te varijable ili objekta.

Mehanizam automatskog upravljanja memorijom u programskom jeziku Python

Deklaracijom globalne varijable zauzima se dio memorije *shella*.

stanje mehanizma za upravljanje memorijom

```
>>> globalna_varijabla=100
```

Moduli sys i GC sadržavaju metode koje daju uvid u rad sistemskog alata za oslobađanje memorije te rad sustava s varijablama. Na sljedećoj slici prikazan je način kako dobiti uvid u stanje automatskog upravljanja memorijom za aktualne referencije i za već „potrošene" referencije, tj. one koje se tijekom izvođenja programa više neće ponavljati.

```
>>> import sys
>>> import gc
>>>
>>> sys.getrefcount(globalna_varijabla)
11
>>>
>>> len(gc.get_referrers(globalna_varijabla))
2
```

Sve privremene reference (već „potrošene" i aktualne")

Aktualne reference

Osim s pomoću ugrađenog mehanizma za upravljanje memorijom, varijable se mogu ukloniti i ručno – naredbom Del.

ručno uklanjanje varijable naredbom Del

```
>>> del globalna_varijabla
>>>
>>> globalna_varijabla
Traceback (most recent call last):
  File "<pyshell#170>", line 1, in <module>
    globalna_varijabla
NameError: name 'globalna_varijabla' is not defined
>>>
```

Ručno ”brisanje”

Važno je napomenuti da se atomarni tipovi (npr. samo varijable koje sadržavaju jednu vrijednost) i objekti koji zauzimaju malo memorije obično ne prate s pomoću ugrađenog mehanizma za upravljanje memorijom. Prikazana metoda daje nam uvid u to prati li se pojedini objekt, tj. varijabla, ili ne.

način rada ugrađenog mehanizma za upravljanje memorijom

```
>>> gc.is_tracked(globalna_varijabla)
False
```

Ugrađeni mehanizam za upravljanje memorijom radi na principu podjele svih objekata koje prati u tri klase, tj. generacije. Prva generacija sadržava objekte koji će posljednji biti uklonjeni, a treća generacija objekte koji će prvi biti uklonjeni pri sljedećem radu mehanizma.

djelovanje ugrađenog mehanizma za upravljanje memorijom

Provjera koliko je objekata trenutačno spremno za uklanjanje prema „generacijama” moguća je donjom metodom – svaki prolazak mehanizma uklanja posljednju generaciju objekata (krajnje desnu) i priprema sljedeću generaciju za uklanjanje ako nema referencija za nju.

```
>>> gc.get_count()
(29, 7, 3)
```

Automatsko pozivanje mehanizma za upravljanje memorijom ili ručno pozivanje s pomoću naredbe gc.collect() uklanja objekte koji nemaju referencije.

uklanjanje objekata koji nemaju referencije

```
>>> gc.collect()
66
>>> gc.get_count()
(46, 0, 0)
```

Provjerite i primijenite

1. Koje su kategorije memorije dostupne na računalu i kako se dijele?
2. Koja je razlika između fizičke radne memorije i radne memorije operativnog sustava?
3. Koji su primjeri sistemske pogreške koja se može pojaviti zbog nedovoljnog upravljanja potrošnjom memorije?
4. Kako moderni programski jezici automatski upravljaju potrošnjom memorije?
5. Koje su tri faze u životnom ciklusu varijable/objekta u programskom jeziku Python i što se događa u svakoj fazi?

Za one koji žele znati više

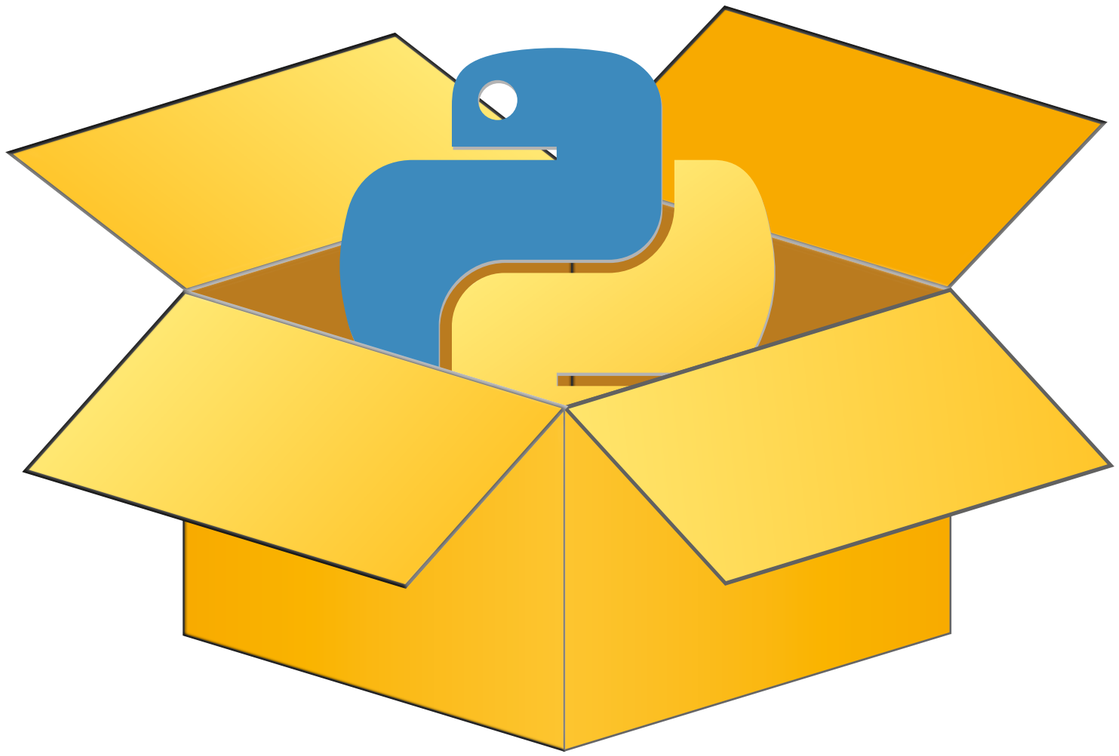
Autor preporučuje nekoliko tema kao samostalno istraživanje za polaznike koji su s lakoćom svladali dosadašnje gradivo. Postavke mehanizma za automatsko upravljanje memorijom

- Postavke mehanizma za automatsko upravljanje memorijom
- Aktivno praćenje zauzimanja memorije u sesiji/sustavu
- Pravila za uklanjanje objekata.

Detalji o mehanizmu dostupni su na internetskoj adresi

☞ https://docs.python.org/3/library/gc.html

8. Moduli i paketi



Nakon što ste u osmoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o modulima i paketima, moći ćete:

- razlikovati module i pakete
- objasniti ulogu modula i paketa u programiranju
- objasniti koncept standardne biblioteke
- navesti primjere modula koji dolaze sa standardnom bibliotekom
- koristiti se ključnim riječima import i from za uporabu funkcija, konstanta i razreda koji se nalaze unutar modula
- napisati primjere korištenja funkcijama, konstantama i razredima iz modula
- objasniti problem koji se može javiti pri uključivanju svih funkcija iz modula
- objasniti koncept kreiranja vlastitih modula te ga demonstrirati na primjeru
- objasniti postupak instaliranja modula trećih strana s pomoću upravljača paketa pip
- pretraživati i odabrati odgovarajuće module trećih strana na platformi pypi.org

PowerPoint prezentacija

Snimka 8. radionice

Moduli i paketi omogućuju lakši i brži razvoj programskoga kôda, a u konačnici i programa. Napisano je puno modula, tj. paketa za Python koji ubrzavaju razvoj programskoga kôda jednostavnim pozivanjem unaprijed napisanih funkcija, razreda, konstanta. Moduli, tj. paketi organizirani su u smislene cjeline radi što lakšeg snalaženja ne samo pri korištenju već i radi što lakšeg pretraživanja dokumentacije.

Module i pakete koji dolaze sa standardnom instalacijom *Pythona* nazivamo standardnom bibliotekom.

standardna biblioteka

Katkad su programerima potrebne funkcionalnosti koje ne dolaze s modulima, tj. paketima iz *standardne biblioteke*. Za takve funkcionalnosti potrebno je poslužiti se internetom i potražiti odgovarajuće pakete te ih instalirati na računalo na kojemu će se program izvoditi. U ovoj edukaciji obrađeni su samo moduli koji dolaze sa standardnom instalacijom Pythona.

Potrebno je razlikovati **module** i **pakete**. Moduli su sastavni dijelovi programskoga kôda unutar kojih je primijenjena neka specifična funkcionalnost. Za primjer se navodi nekoliko modula koji dolaze sa standardnom bibliotekom: math, cmath, random, datetime.

razlika između modula i paketa

Paketi su cjeline koje uključuju druge module i omogućuju njihovo hijerarhijsko grupiranje. Kao primjer navodimo paket koji dolazi sa standardnom bibliotekom, paket urllib, koji sadržava nekoliko modula:

primjer

- `urllib.request`
- `urllib.error`
- `urllib.parse`
- `urllib.robotparser.`

Rad s modulima i paketima

Funkcijom, konstantom i razredom koji se nalaze unutar nekog modula možemo se koristiti na dva načina:

dva načina korištenja funkcijom, konstantom i razredom

- najavljivanjem korištenja
- uključivanjem u program.

Najavljivanje korištenja

Najavljivanje korištenja modulom ostvaruje se s pomoću ključne riječi import. Sintaksa:

```
import <modul>
```

primjer najavljivanje korištenja modulom

primjer

```
import math

print(math.sin(55))
print(math.tan(55))
```

```
Izlaz:
      -0.9997551733586199
      0.022126756261955736
```

U prethodnom primjeru prikazan je način korištenja naredbom Import. Pri takvom načinu uporabe naredbe `Import` najavljuje se korištenje **svim** funkcijama, razredima i konstantama (u daljnjem tekstu spominjat će se samo funkcije, no podrazumijeva se da se u nekom modulu mogu nalaziti i razredi i konstante) iz nekog modula.

Ako se korištenje funkcijama samo najavi kao u navedenu primjeru, tada u svakom dijelu programa u kojemu se želi iskoristiti funkcija iz najavljenog modula mora pisati ime modula i funkcije odvojene točkom. Sintaksa takvog načina primjene najavljenih funkcija jest:

```
modul.funkcija()
```

Uključivanje u program

korištenje ključnim riječima from i import

Ako želimo izbjeći gornju sintaksu i koristiti se samo imenom funkcije kao pozivom, a ne da se ispred imena funkcije piše ime modula u kojemu se funkcija nalazi, tada je potrebno željene funkcije uključiti u program. Funkcije se uključuju u program korištenjem ključnih riječi from i import.

Sintaksa takvog načina korištenja funkcijama jest:

```
from <modul> import <funkcija>, <funkcija>, ...
```

Ključna riječ **from** govori kojim ćemo se modulom koristiti, a ključna riječ **import** pokazuje koje će se sve funkcije iz modula uključiti u naš program. Pri takvom načinu uključivanja funkcija u program, za razliku od prijašnjeg primjera, više nije potrebno pisati u kojem se modulu nalazi korištena funkcija.

značenje ključnih riječi from i import

primjer

```
from math import sin, cos

print(sin(55))
print(cos(55))
print(math.tan(55))
```

```
Izlaz:
      -0.9997551733586199
      0.022126756261955736
      Traceback (most recent call last):
        File "test.py", line 4, in <module>
          print(math.tan(55))
      NameError: name 'math' is not defined
```

primjer sintakse korištenja funkcijama tako da se one uključuju u programski kôd

U prethodnom primjeru prikazana je sintaksa korištenja funkcijama tako da se one uključuju u programski kôd. U program su uključene samo one funkcije koje će se upotrijebiti u programskom kôdu, a to su sin() i cos(). Primjećuje se da pri pozivu funkcija više nije potrebno ispred funkcije pisati ime modula u kojemu se ta funkcija nalazi.

U gornjem primjeru namjerno je izazvana pogreška. Funkcija `tan()` iz modula math nije uključena u programski kôd, a nije najavljeno ni korištenje njome u programskom kôdu. Za demonstraciju ona nije pozvana tako da se napiše samo ime funkcije, već je pozvana na način math.tan(), no svejedno se dogodila pogreška. Pogreška je nastala zbog toga što u navedenom načinu uključivanja funkcija iz modula math u program nije najavljeno i korištenje ostalim funkcijama iz tog paketa.

namjerno izazvana pogreška

Ako se želi ispraviti ta pogreška, to se može učiniti na dva načina:

ispravljanje namjerno izazvane pogreške

- dodavanjem funkcije `tan()` u uključene funkcije:

```
from math import sin, cos, tan
```

- najavljivanjem uporabe svih funkcija iz modula math: `import math`

Svejedno je koji ćemo način odabrati.

Ako pri uporabi (pozivu) funkcija iz modula ne želimo pisati iz kojeg modula dolazi funkcija, a ne želimo ni pri uključivanju modula sastavljati popis svih funkcija koje će se upotrebljavati, to se može napraviti tako da u program uključimo sve funkcije iz nekog modula.

razlog uključivanja svih funkcija iz nekog modula i mogući problem

```
from math import *
```

Problem koji se može pojaviti pri takvom načinu korištenja modulom jest to da se pri uključivanju više različitih modula može dogoditi kolizija ako se u dva i više modula nalazi isto ime funkcije.

Konstante

U modulima se mogu nalaziti i razne konstante. Tako se u modulu math nalazi nekoliko matematičkih konstanta: *Pi, e, Tau, INF, nan*. U nastavku slijede primjeri programskoga kôda unutar kojih se dohvaća i ispisuje vrijednost konstante *Pi*.

primjeri programskoga kôda unutar kojih se dohvaća i ispisuje vrijednost konstante *Pi*.

primjer

Najavljivanje korištenja svim funkcijama i konstantama modula math:

```
import math print(math.pi)
```

```
Izlaz:
     3.141592653589793
```

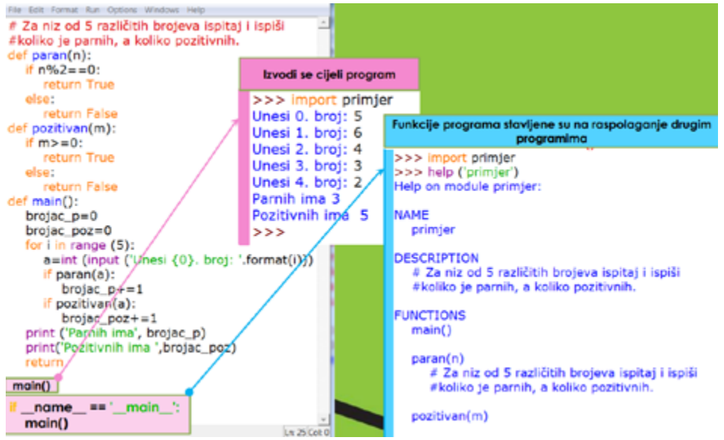
Uključivanje u program konstante *Pi* iz modula math:

```
from math import pi
print(pi)
```

```
Izlaz:
     3.141592653589793
```

Kreiranje vlastitih modula

Na sljedećoj slici prikazan je pristup kako napraviti vlastiti modul koji se nakon toga može uključiti u kod i upotrijebiti.

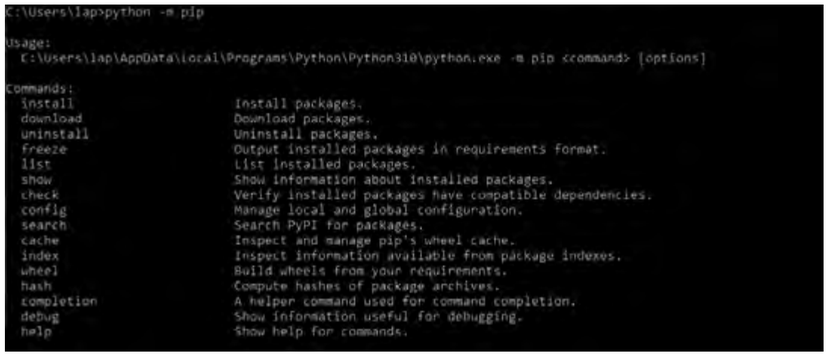


Instaliranje modula trećih strana

Postoji „upravljač paketa" – *packet manager za Python* – koji služi ažuriranju i instaliranju modula trećih strana.

Pokretanje iz komandnog retka

postupak pokretanja iz komandnog retka



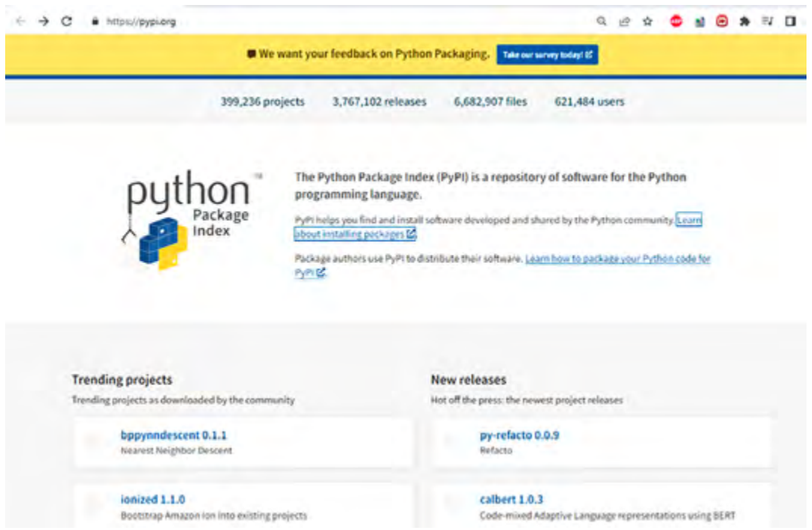
- Ako se već zna naziv modula i pip je instaliran

```
C:\Users\lap>python -m pip install IME
```

- Ako nema pip-instaliranja, najprije treba postaviti pip

```
C:\Users\lap>python -m ensurepip --default-pip
```

- Prikaz registra svih modula trećih strana s pomoću alata i kataloga pypi.org
  ☞ https://pypi.org/



Vježba korištenja *Pythonovim* modulima i paketima

1. Izradite program koji najavljuje korištenje funkcijama iz modula `math` te izračunajte i ispišite rezultat:

$$\sin(2) + \cos(1)$$

2. Pronađite na internetu u službenoj dokumentaciji *Python 3* kako se zove funkcija koja se nalazi u modulu `math` i primjenjuje za korjenovanje. S pomoću tipkovnice unesite broj i ispišite njegov korijen.

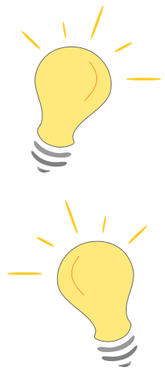
Službenu dokumentaciju *Python 3* možete pronaći na URL-u: https://docs.python.org/3/library/math.html

3. Primjenom konstanta koje su pohranjene u modulu math ispišite vrijednost konstante *Pi*.

4. Koristeći se konstantom *Pi* izračunajte i ispišite rezultat: sin(2*Pi*) + cos(*Pi*)

5. * U službenoj dokumentaciji *Python 3* pronađite ime funkcije za potenciranje (zamjena za aritmetički operator **). S tipkovnice učitajte cjelobrojne vrijednosti i spremite ih u varijable *a* i *b*. Nije potrebno provjeravati ispravnost učitanih brojeva. Na temelju tih vrijednosti izračunajte kvadrat zbroja $(a + b)^2$. Formula za kvadrat zbroja je $(a + b)^2 = a^2 + 2ab + b^2$. Funkcije kojima ćete se koristiti u ovom zadatku uključite u program.

Provjerite i primijenite



1. Napišite zašto se koristite modulima te kako biste ih upotrijebili u svojem radu.

2. Provjerite koji sve sastavni dijelovi mogu biti uključeni u module.

3. Provjerite mogu li se modulima koristiti drugi moduli.

4. Provjerite mogu li se nazivi između nekoliko modula preklapati i koje pravilo vrijedi ako se preklapaju.

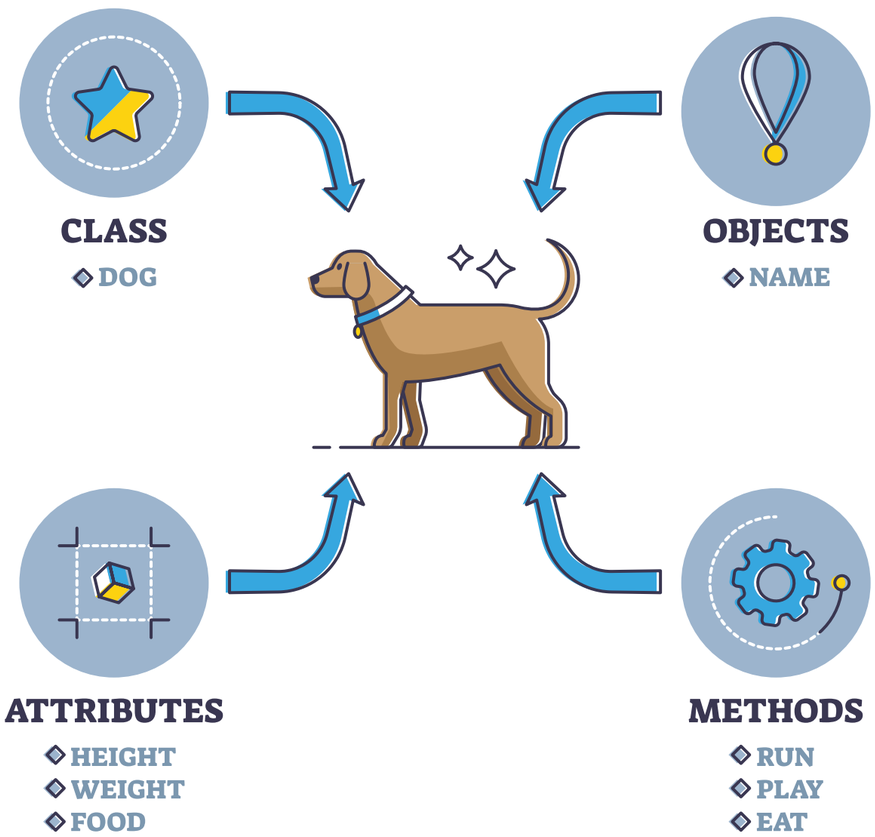
Za one koji žele znati više

1. Provjerite definiciju modula na službenoj dokumentaciji Pythona

   ☞ https://docs.python.org/3/tutorial/modules.html

2. Riješite interaktivne zadatke o modulima dostupne na sljedećoj internetskoj poveznici

   ☞ https://www.w3schools.com/python/python_modules.asp



9. Objektno orijentirano programiranje

Nakon što ste u devetoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o objektno orijentiranom programiranju, moći ćete:

- navesti prednosti objektno orijentiranog programiranja u odnosu prema proceduralnom programiranju
- objasniti osnovne koncepte objektno orijentiranog programiranja kao što su razred, objekt, atribut i metoda
- definirati razred i objekt
- objasniti međusobnu povezanost razreda i objekta
- objasniti kako se pozivaju metode na objektima i kako se instancira razred
- napisati jednostavne primjere koda koji se koristi objektno orijentiranim programiranjem
- prikazati osnovne koncepte kao što su definiranje razreda, instanciranje objekata i pozivanje metoda.

PowerPoint prezentacija

Snimka 9. radionice

Uvod u objektno orijentirano programiranje

proceduralno programiranje

Način razvoja programskoga kôda koji je prikazan u dosadašnjem dijelu priručnika naziva se proceduralno programiranje. To je način programiranja čija se logika zasniva na temelju funkcija, tj. blokova programskoga kôda.

Izrada programskoga kôda koja će biti objašnjena u ovom poglavlju zove se objektno orijentirano programiranje (kraće OOP). Većina programa može se kvalitetno napisati s pomoću proceduralnog načina razvoja programskoga kôda, no kod velikih projekata preporučljivo je koristiti se objektno orijentiranim programiranjem. Objektno orijentirano programiranje služi smanjenju kompleksnosti razvoja i održavanja programskih rješenja. OOP ne sprečava programere da pišu loš programski kôd, već ih usmjerava da razvijaju programski kôd u okvirima standarda ove paradigme.

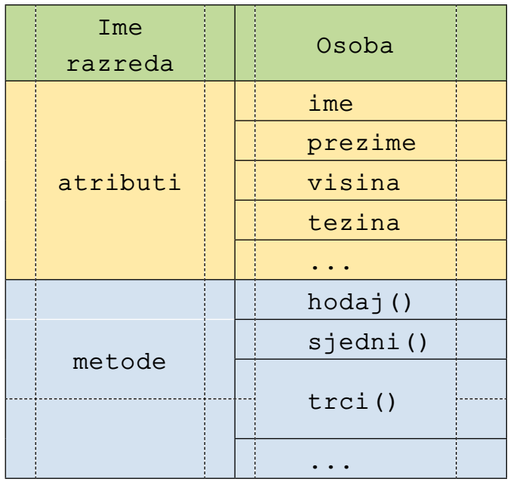
prednosti objektno orijentiranog programiranja

Kod objektno orijentiranog programiranja nakana je da problem koji se rješava bude razdijeljen na što manje logičke cjeline.

primjer objektno orijentiranog programiranja

primjer

Na primjer, ako program treba obrađivati podatke o osobama, tada je osnovni tip objekta u tom programu **razred** Osoba. Svakoj osobi možemo pridružiti razne parametre ili pak u duhu objektno orijentiranog pristupa atribute. Atributi mogu biti, na primjer, ime, prezime, visina, težina, OIB. Jednako tako svakom razredu (instancija razreda naziva se objekt) pridružene su i pripadajuće akcije, tj. metode. Na primjer, osoba može hodati, sjesti, trčati itd., pa tako imamo metode: hodaj(), sjedni(), trci() koje objektu Osoba daju život. Nudimo ilustrativan prikaz razreda Osoba.



Definiranje objekta/klase

U nastavku slijedi tablica sinonima koji se često pojavljuju pri čitanju literature, a imaju isto značenje.

tablica istoznačnica ili sinonima

Hrvatski nazivi (sinonimi)			Engleski nazivi
razred	klasa	–	engl. (*class*)
atribut	svojstvo	–	engl. (*attribute*)
objekt	instancija	jedinka	engl. (*instance*)

Razred je osnovna programska cjelina u objektno orijentiranom načinu programiranja. Unutar razreda zapisan je shematski plan (engl. *blueprint*) te se na temelju njega kreiraju **objekti**. Razred sadržava **atribute** i **metode**. Atributi opisuju objekt raznim podatcima, a svaki objekt ima kopiju svih atributa. Metode su zapravo funkcije, no one se pozivaju nad objektom kojemu je neka metoda pridružena.

osnovna programska cjelina u objektno orijentiranom načinu programiranja

U nastavku se nalazi osnovni primjer kreiranja razreda.

primjer

```
#deklaracija klase/nema atributa u tijelu klase
#nije potrebno unositi atribut u klasu Posuda()
class Posuda:
#poziv funkcije unosa biskvita
    def insertBiskvit(self, biskvit):
        self.biskvit = biskvit
#poziv funkcije ispisa biskvita
    def getBiskvit(self):
        print (self.biskvit)
```

primjer kreiranja razreda

Pozivanje metoda

Svaki razred počinje ključnom riječju class, a nakon ključne riječi navodi se proizvoljno ime razreda. Prema *Pythonovoj* preporuci riječi naziva razreda pišu se velikim početnim slovom, na primjer: Osoba, KoordinatniSustav i slično. Unutar tako kreiranog razreda (klase) pišu se:

kreiranje razreda

- **atributi** – svojstva razreda
- **metode** – funkcionalnosti razreda.

U nastavku je prikazan programski kôd koji ostvaruje neke funkcionalnosti samih ljudi. Unutar razreda imena Osoba primijenjene su tri metode `hodaj()`, `sjedni()`, `trci()`. U glavnom programu kreiran je jedan objekt (instancija) razreda Osoba imena o. U sljedećem primjeru može se vidjeti da se objekt nekoga razreda može kreirati na sljedeći način:

programski kôd

imeObjekta = ImeRazreda()

Metode nekog objekta pozivaju se s pomoću sljedeće sintakse:

imeObjekta.imeMetode()

Napomena: unutar donjeg primjera vidljivo je da se rabi parametar self koji označava varijable i funkcije jedne instancije objekta.

primjer

```
class Osoba:

    def hodaj(self):

        print(„Hodaj!”)


    def sjedni(self):

        print(„Sjedni!”)


    def trci(self):

        print(„Trci!”)
```

```
   o = Osoba()


   o.hodaj() o.sjedni
() o.trci() Izlaz:

   Hodaj!
   Sjedni!
   Trci!
```

Ako je potrebno kreirati „beskoristan” razred koji ne sadržava programski kôd, to se može postići tako da se u tijelo kreiranog razreda napiše naredba Pass.

U donjem primjeru vidljivo je da je razred imena Osoba razred bez atributa i metoda. Unutar glavnog programa kreirana su tri objekta (instancije) razreda Osoba, a to su: o1, o2, o3. Na temelju ispisa može se zaključiti da sva tri objekta imaju različitu memorijsku lokaciju što znači da su sva tri objekta međusobno neovisna.

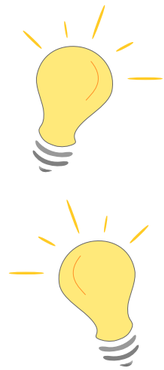
Na primjeru programskoga koda s početka na ovaj se način deklarira instancija klase, tj. objekt, te kako se poziva.

primjer

primjer deklariranja instancije klase

```
#pozivi klase i funkcija
mojaPosuda=Posuda()
mojaPosuda.insertBiskvit('mojBiskvit')
mojaPosuda.getBiskvit()
```

Provjerite i primijenite



1. Objasnite na svojem primjeru kako biste se koristili razredom i objektom.

2. Objasnite čemu služe objekti i razredi.

3. Napravite model klase koja opisuje ponašanje i atribute vozila te njegovu interakciju s garažom.

Za one koji žele znati više

1. Istražite službenu Pythonovu dokumentaciju o klasama i objektima

☞ https://docs.python.org/3/tutorial/classes.html

2. Riješite interaktivne vježbe dostupne na poveznici

☞ https://www.w3schools.com/python/python_classes.asp

10. Praktične primjene objektno orijentiranog programiranja

Nakon što ste u devetoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o praktičnoj primjeni objektno orijentiranog programiranja, moći ćete:

- razumjeti koncept objektno orijentiranog programiranja
- primijeniti koncept objektno orijentiranog programiranja u praksi
- razumjeti programske knjižnice za interakciju sa senzorima
- primijeniti programske knjižnice za interakciju sa senzorima
- koristiti se senzorima za detekciju i prikupljanje informacija, za interakciju sa zaslonom, sa zvučnim aktuatorima
- primijeniti objektno orijentirano programiranje u području poljoprivrede i poljoprivredne proizvodnje
- razumjeti koncepte programske podrške za uređaje poput mikrofona, kamere, tipkovnice i miša
- primijeniti koncepte programske podrške za uređaje poput mikrofona, kamere, tipkovnice i miša
- razumjeti koncepte regulacije visine i trajanja zvuka te reprodukcije zvuka iz datoteke
- primijeniti koncepte regulacije visine i trajanja zvuka te reprodukcije zvuka iz datoteke.

PowerPoint prezentacija

Snimka 10. radionice

Internet stvari (IoT) tehnologija je u kojoj se različiti fizički i virtualni uređaji (stvari) povezuju u mrežu kako bi međusobno komunicirali i razmjenjivali podatke bez potrebe za ljudskom intervencijom. U tom smislu IoT pruža mogućnost da se dobiju vrlo precizni podatci o raznim aspektima okoline i sustava što ima velik potencijal za primjenu u mnogim područjima uključujući i poljoprivredu.

Python je programski jezik koji ima važnu ulogu u razvoju tehnologije IoT jer se može primijeniti za programiranje uređaja povezanih s mrežom IoT. Python je vrlo popularan među programerima, a nudi i mnoge korisne biblioteke za rad s uređajima IoT, poput biblioteke Adafruit CircuitPython. U području poljoprivrede, primjena tehnologije IoT može biti različita, poput praćenja stanja usjeva, praćenja stanja tla i klimatskih uvjeta, praćenja kretanja životinja i tako dalje. Python omogućuje programiranje različitih senzora, kamere, zaslona i drugih uređaja kako bi se prikupili i obrađivali podatci te kako bi se upravljalo uređajima. Jednako tako Python je vrlo jednostavan za učenje, pa je idealan za programiranje uređaja IoT čak i za početnike.

Programske knjižnice za pristup ugrađenim senzorima

odrednice senzora

Senzor je pretvornik ili mjerno osjetilo i dio je mjernoga sustava koji je u izravnom dodiru s mjerenom veličinom i daje izlazni signal ovisan o njezinu iznosu. Zbog prevladavajuće primjene električnih i elektroničkih sustava, većina mjernih osjetila pretvara mjerenu veličinu u električki mjerljiv signal.

ugrađeni senzori i njihova primjena u poljoprivredi

Mikrofoni, kamere i tastature ulazni su uređaji koji se mogu upotrebljavatiu poljoprivredi u kombinaciji s programiranjem u Pythonu kako bi se poboljšala učinkovitost i produktivnost poljoprivrednih aktivnosti.

Mikrofoni se, primjerice mogu upotrebljavati za

primjer

snimanje zvukova u poljoprivrednim područjima kako bi se utvrdila razina buke, što može utjecati na rast usjeva ili zdravlje životinja. Korištenjem Pythona podatci se mogu analizirati kako bi se utvrdili uzroci buke i razvile strategije za smanjenje njezina utjecaja.

Kamere se mogu upotrebljavati za

primjer

nadzor usjeva, otkrivanje štetočinja i bolesti te za praćenje rasta biljaka. S pomoću Pythona mogu se razviti algoritmi za prepoznavanje slika koji mogu otkriti bolesti i štetočinje na usjevima, a to može pomoći poljoprivrednicima da brže reagiraju i poduzmu odgovarajuće mjere.

Tastatura se može upotrebljavati za

primjer

upravljanje poljoprivrednom opremom kao što su strojevi za obradu zemlje ili navodnjavanje. Korištenjem Pythona mogu se razviti programi za automatizaciju tih procesa što može pomoći da se uštedi vrijeme i poveća učinkovitost u poljoprivrednom radu.

Ukratko, korištenje mikrofonom, kamerom i tastaturom u kombinaciji s programiranjem u Pythonu može pomoći poljoprivrednicima u poboljšanju produktivnosti, smanjenju troškova i povećanju profitabilnosti.

doprinos Pythona prinosima u poljoprivredi

Pristup uređajima – rezolucija zaslona

Ugrađeni uređaji za izlaz informacija su, na primjer, zaslon i zvučnici.

ugrađeni uređaji za izlaz informacija

praktična primjena zaslona u poljoprivredi	prikaz i nadzor nad stanjem nasada, prikaz upozorenja na štetne događaje ili događaje u kojima treba primijeniti akciju kako bi se izbjegao npr. razvoj bolesti u nasadu
praktična primjena zvučnika u poljoprivredi	služe kao upozorenje na pojedine važne događaje

primjena zaslona i zvučnika u poljoprivredi

```
D:\py>pip install screeninfo
Collecting screeninfo
  Downloading screeninfo-0.8.1-py3-none-any.whl (12 kB)
Installing collected packages: screeninfo
Successfully installed screeninfo-0.8.1
```

```
D:\py>python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from screeninfo import get_monitors
>>> for m in get_monitors():
...     print(str(m))
...
Monitor(x=0, y=0, width=1920, height=1080, width_mm=344, height_mm=194, name='\\\\.\\DISPLAY1', is_primary=True)
Monitor(x=-1920, y=-80, width=1920, height=1080, width_mm=477, height_mm=268, name='\\\\.\\DISPLAY4', is_primary=False)
>>>
```

Interakcija sa zaslonom i zvučnim aktuatorima

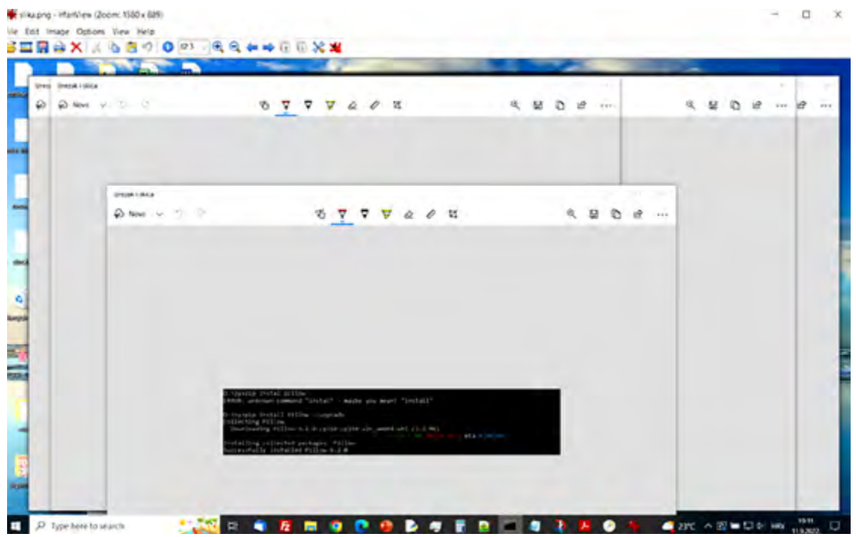
- Programske knjižnice za interakciju s ugrađenim zaslonom

```
D:\py>pip install pyautogui
Collecting pyautogui
  Downloading PyAutoGUI-0.9.53.tar.gz (59 kB)
     ---------------------------------------- 59.0/59.0 kB 774.5 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting pymsgbox
  Downloading PyMsgBox-1.0.9.tar.gz (18 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting PyTweening>=1.0.1
  Downloading pytweening-1.0.4.tar.gz (14 kB)
  Preparing metadata (setup.py) ... done
Collecting pyscreeze>=0.1.21
  Downloading PyScreeze-0.1.28.tar.gz (25 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting pygetwindow>=0.0.5
  Downloading PyGetWindow-0.0.9.tar.gz (9.7 kB)
  Preparing metadata (setup.py) ... done
Collecting mouseinfo
  Downloading MouseInfo-0.1.3.tar.gz (10 kB)
  Preparing metadata (setup.py) ... done
Collecting pyrect
  Downloading PyRect-0.2.0.tar.gz (17 kB)
  Preparing metadata (setup.py) ... done
Collecting pyperclip
  Downloading pyperclip-1.8.2.tar.gz (20 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: pyautogui, pygetwindow, pyscreeze, PyTweening, mouseinfo, pymsgbox, pyp
yrect
  Building wheel for pyautogui (setup.py) ... done
  Created wheel for pyautogui: filename=PyAutoGUI-0.9.53-py3-none-any.whl size=36614 sha256=e39fe81c4db557a453e
5d732780eca579efcffe0179d1bb7ff6f3d3
```

```
D:\py>pip instal pillow
ERROR: unknown command "instal" - maybe you meant "install"

D:\py>pip install Pillow --upgrade
Collecting Pillow
  Downloading Pillow-9.2.0-cp310-cp310-win_amd64.whl (3.3 MB)
     ---------------------------------------- 3.3/3.3 MB 885.6 kB/s eta 0:00:00
Installing collected packages: Pillow
Successfully installed Pillow-9.2.0
```

```
>>> import pyautogui
>>> slika = pyautogui.screenshot()
>>> slika.save(r'D:\py\slika.png')
```



- Programske knjižnice za interakciju sa zvučnikom računala

  Regulacija visine i trajanja zvuka

```
>>> import winsound
>>> winsound.Beep(440, 500)
```

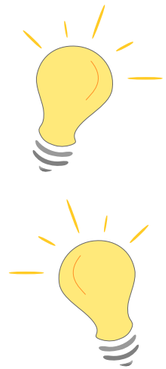
Reprodukcija sistemskih zvukova

```
>>> winsound.PlaySound("SystemExclamation", winsound.SND_ALIAS)
```

Reprodukcija zvuka iz datoteke

```
>>> winsound.PlaySound("beep.wav", winsound.SND_FILENAME)
```

Provjerite i primijenite



1. Čemu služe senzori i aktuatori?
2. Kako se koristiti senzorima i aktuatorima iz Pythonova programskoga koda?
3. Koji su ugrađeni senzori i aktuatori kojima se može upravljati iz Pythona?
4. Koja je vrsta Pythonova modula potrebna za rad s ugrađenim senzorima i aktuatorima?

Za one koji žele znati više

1. Kako se primjenjuje koncept objektno orijentiranog programiranja u području poljoprivrede?
2. Koje su najčešće primjene mikrofona kao senzora u poljoprivredi i kako se on upotrebljava za otkrivanje štetnih događaja?
3. Kako se kamere rabe za kontrolu kvalitete usjeva u poljoprivredi i praćenje životinjskog svijeta?
4. Koje su najčešće primjene tipkovnice i miša u poljoprivredi i kako se koriste za pokretanje određenih akcija?
5. Koje su programske knjižnice za interakciju sa zaslonom i zvučnim aktuatorima te kako se mogu primijeniti u poljoprivredi?

Literatura

1. https://www.python.org/about/gettingstarted/
2. Rihter, Rade, Toić Dlačić, Topić, Novaković, Bujadinović, Pandurić, *Like IT 5*, udžbenik informatike za peti razred osnovne škole, Alfa, 2019.
3. Gregurić, Hajdinjak, Jakšić, Počuča, Rakić, Svetličić, Šokac, Vlajinić, *Informatika 5*, udžbenik informatike za peti razred osnovne škole, Profil Klett, 2019.
4. Drezgić, Pavić, Trucek, *#MOJPORTAL5*, udžbenik informatike za peti razred osnovne škole, Školska knjiga, 2021.
5. Rihter, Rade, Toić Dlačić, Topić, Novaković, Bujadinović, Pandurić, Draganjac, *Like IT 6*, udžbenik informatike za šesti razred osnovne škole, Alfa, 2020.
6. Gregurić, Hajdinjak, Jakšić, Počuča, Rakić, Svetličić, Šokac, Vlajinić, *Informatika 6*, udžbenik informatike za šesti razred osnovne škole, Profil Klett, 2020.
7. Drezgić, Pavić, Trucek, *#MOJPORTAL6*, udžbenik informatike za šesti razred osnovne škole, Školska knjiga, 2021.
8. *Python zadatci za vježbanje i ponavljanje Ponavljanje 1* (slidetodoc.com), Carnet
9. Hruška, Marko, Domšić, Josip, Kurtović, Gorana, Kožul, Mia, *Osnove programiranja (Python): priručnik za polaznike*, Srce

Udruga za ruralni razvoj Ravni kotari

Vukovarska 3d, 23 000 Zadar

IT PRAXIS d.o.o. za savjetovanje i izdavaštvo

Matka Laginje 1b, 47000 Karlovac

Više o EU fondovima na:

☞ www.strukturnifondovi.hr

☞ http://www.esf.hr/