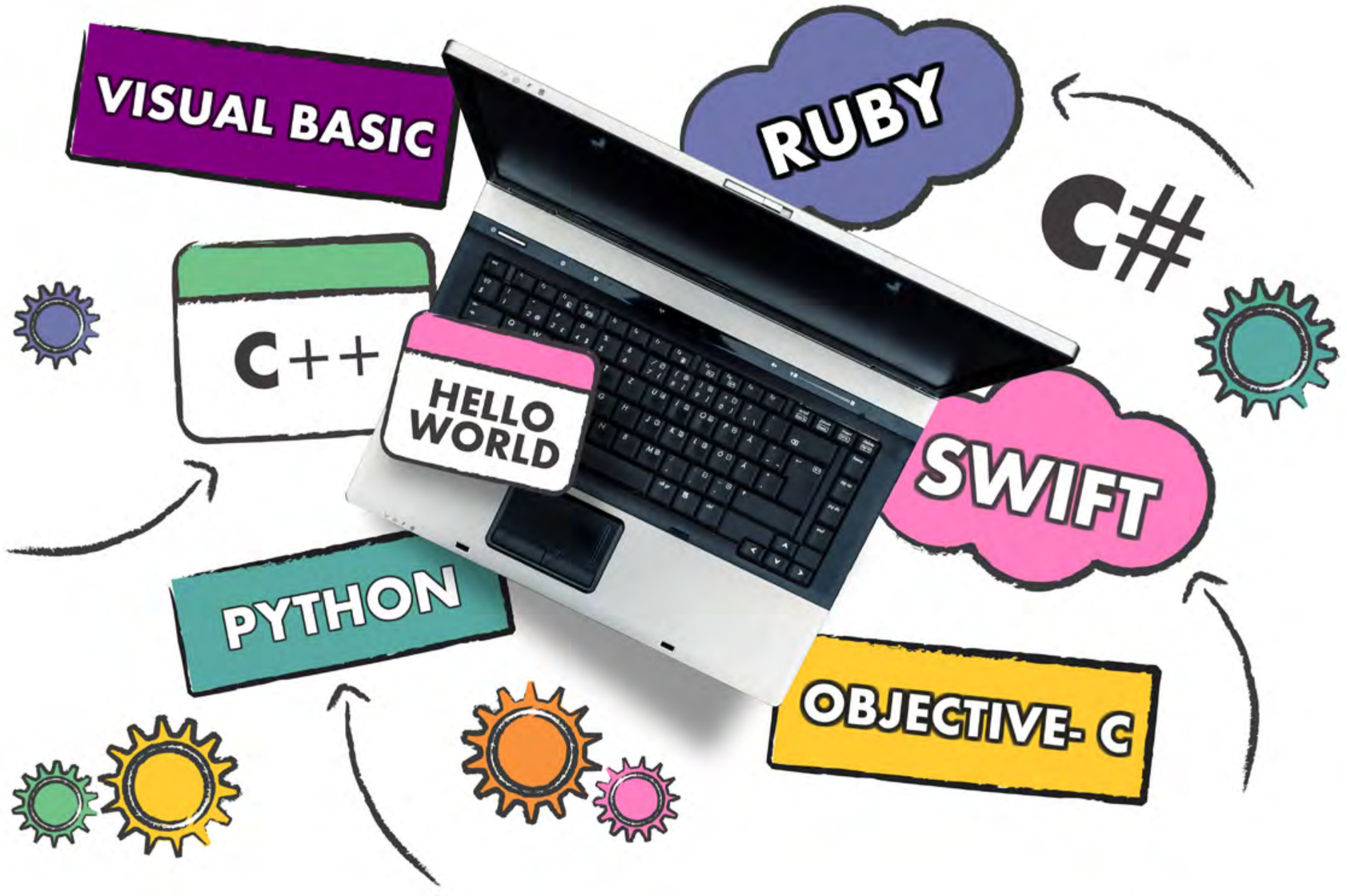
Ravni kotari

III.



Damir Jakšić
Boris Počuča

OSNOVE PROGRAMIRANJA

agrostem

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Sadržaj priručnika isključiva je odgovornost Udruge za ruralni razvoj Ravni kotari

Ravni kotari

Damir Jakšić & Boris Počuča

Osnove programiranja

Sadržaj:

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Uvod

Sadržaj edukativnog modula Osnove programiranja, jedna je od pet cjelina priručnika za AgroSTEM kojeg su izradili vanjski stručnjaci IT Praxisa u sklopu provedbe Ugovor o pružanju usluga organizacije radionica: Osnove programiranja, Radionica programiranja, Digitalizacija kao pokretač ruralnog razvoja, Razvoj pedagoških vještina s ciljem popularizacije STEM-a, Korištenje brzih terenskih metoda i inovativnih alata u poljoprivredi te izrade AgroSTEM priručnika za Udrugu za ruralni razvoj Ravni kotari. Objavljen je na internetu i dostupan svima na edukativnoj mrežnoj platformi Naručitelja.

Tekst o *Osnovama programiranja* nastao je na temelju provedene desetodnevne radionice (po četiri školska sata) u razdoblju od 29. kolovoza do 19. rujna 2022. Radionica je bila namijenjena članovima organizacija civilnog društva (OCD-a) uključenih u projekt AgroSTEM (UP.04.2.1.10.0021) i organizirana u tri modula: 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). Tako su polaznici stekli znanja i vještine za poučavanje polaznika za tri navedene različite skupine krajnjih korisnika. Polaznici su uz to upoznali osnove programskog jezika Pyhton i mogućnosti njegove primjene u okružju AgroSTEM-a.

Cijelina Osnove programiranja sastoji se od 10 sadržajno zaokruženih tematskih cjelina. To su:

Modul I.

1. Izrada dokumentacije za provođenje nastavnog procesa (ostvarivanje odgojno-obrazovnih ciljeva edukacije)
2. Metodološki pristup učenicima osnovne škole u poučavanju programiranja programom Python
3. Ulazne i izlazne vrijednosti programa
4. Uvjet u programiranju

Modul II.

5. Metodologija programiranja programskim jezikom Python i njegove funkcije za studente
6. Skladišta podataka – Varijable

7. Proces upravljanja memorijom računala – napredno korištenje Pythonom, Rječnici
8. Moduli i paketi

Modul III.

9. Objektno orijentirano programiranje
10. Interakcija s fizičkom okolinom (senzori).

Svaka nastavna tema počinje ishodima učenja koji se očekuju od polaznika i poveznicom za prezentaciju u PowerPointu i videozapis s 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) i 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).

1. Izrada dokumentacije za provođenje nastavnog procesa i programiranje

Nakon što ste u prvoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o izradi dokumentacije za provođenje nastavnog procesa i programiranju, moći ćete:

- istražiti i kritički vrednovati u digitalnom okružju i mrežnom poučavanju
- razumjeti svrhu izrade pripreme nastavnog sata
- opisati sastavnice pripreme sata
- objasniti ishode učenja
- izraditi pripremu za nastavni sat
- prikazati dijagram tjeka na primjeru iz svakidašnjeg života

PowerPoint prezentacija

Snimka 1. radionice

1.1. Izrada dokumentacije za provođenje nastavnog procesa

Osnove istraživanja

Današnje vrijeme karakterizira golema količina brzo dostupnih, najraznovrsnijih informacija do kojih možemo doći na različite načine. Upravo zbog te velike količine informacija od kojih su mnoge objavljene na internetu potrebno je naučiti učinkovito pretraživati i odabirati informacije, ali i kritički razlikovati kvalitetu dostupnih izvora. Svaki stručan, znanstvenoistraživački rad počinje pronalaženjem i prikupljanjem literature koja je važan izvor informacija.[1]

Standardni koraci u izradi i provođenju istraživanja:

koraci u izradi i provođenju istraživanja

- određivanje teme istraživanja / istraživačkog problema
- pregled literature
- utvrđivanje teorijskog okvira, definiranje pojmova
- formuliranje istraživačkih pitanja (i hipoteza)
- utvrđivanje odgovarajuće metodologije
- razrada metoda/postupaka/instrumenata (protokola, upitnika)
- prikupljanje i obrada podataka (kvantitativnih i/ili kvalitativnih)
- interpretacija podataka
- raspisivanje istraživanja
- etički aspekti istraživanja.

Jedan od najizazovnijih aspekata istraživačkog procesa je otkrivanje istraživačkog problema, određivanje teme.

istraživački problemi

[1] Čendo Metzinger, Tamara, Toth, Marko, *Metodologija istraživačkog rada za stručne studije*, Veleučilište Velika Gorica, 2020., str. 15.

Istraživački problemi su specifične teme zbog kojih se provodi istraživanje, npr. Primjena tehnologije u optimiranju procesa navodnjavanja. Istraživački se problemi najčešće formuliraju u uvodu istraživačkog izvještaja u obliku jedne rečenice ili nekoliko rečenica.

Tema istraživanja: širi pojam, odnosi se na opću temu, npr. rad u agronomiji. — tema istraživanja

Cilj istraživanja – glavna je svrha istraživanja, na primjer, otkrivanje i razumijevanje tehnologije koja utječe na optimiranje procesa navodnjavanja. — cilj istraživanja

Istraživačka pitanja – trebaju biti konkretna, npr. utječe li vlažnost zraka na proces navodnjavanja. — istraživačka pitanja

Svrha izrade pripreme nastavnog sata

Smisao nastavne pripreme nije opterećivanje učitelja i nastavnika dodatnim obvezama. Realno i smisleno planiranje izvođenja nastavnog procesa olakšava nastavu učitelju/nastavniku. Zahvaljujući kvalitetnoj pripremi, cilj i tijek nastavnog sata znatno su jasniji, a etape izvođenja nastavnog procesa mogu se izvesti.

Dakle, pripremu je nužno izraditi samostalno prije nastavnog sata, i to radi učitelja i učenika, a ne radi zadovoljavanja forme. Improvizacija na satu bez detaljnog planiranja ne može u učenika pobuditi zanimanje za nastavu jer oni sami uočavaju nespremnost učitelja/nastavnika za sat. Priprema može biti opširna ili kraća, što ovisi o učitelju/nastavniku, ali mora sadržavati obvezatne sastavnice.

Sastavnice pripreme sata

Priprema za svaki sat mora imati cilj, ishode, tip, nastavne oblike, nastavne metode, nastavna sredstva, pomagala i literaturu.

Cilj treba biti jednoznačno definiran s pomoću utvrđenih aktivnosti, a iz njega proizlaze ishodi. Ostvarivanjem ishoda dolazimo do cilja, konačnog rezultata poučavanja. — cilj nastavnog sata

Ishodi

Aktivnosti prikazuju razine složenosti ishoda učenja. U osnovnoj se školi primjenjuju aktivnosti vezane za sljedeće razine. Te su razine: pamćenje, razumijevanje i primjena a te aktivnosti mogu se izraziti sljedećim glagolima.

razine složenosti ishoda

pamćenje	pamćenje i dosjećanje informacija, prisjećanje: identificirati, imenovati, iskazati/izreći (definiciju/pravilo/zakon), ispisati, ispričati, izdvojiti, izvijestiti, nabrojiti, navesti, opisati, označiti, ponoviti, prepoznati/odabrati, prisjetiti se, poredati, sastaviti popis, sjetiti se (ne: definirati, zapamtiti)
razumijevanje	shvaćanje, sposobnost organiziranja i uređivanja, razumijevanje onoga što je pročitano, slušano – dati primjer, diskutirati, svrstati, utvrditi, izdvojiti, izračunati, izraziti (svojim riječima), izvijestiti, klasificirati, objasniti (glavnu ideju), opisati, pokazati, predvidjeti, preoblikovati, prepoznati, raspraviti, razlikovati, razmotriti, sažeti, smjestiti, svrstati, usporediti
primjena	upotrebljavanje općeg koncepta za rješenje problema: demonstrirati, ilustrirati, interpretirati, intervjuirati, planirati, istražiti, izabrati, izložiti, izračunati, izvesti, koristiti se, odabrati, otkriti, pokazati, povezati, predvidjeti, prevesti, prikazati, prikupiti, prilagoditi, primijeniti (pravilo/zakon...), provesti, protumačiti, rasporediti, riješiti, rukovati, skicirati, upotrijebiti (ne: vježbati, navesti primjer)

Potrebno je izbjegavati riječi kojima se ne mogu izraziti ishodi učenja: biti osposobljen, biti sposoban, imati osnovna znanja, imati snažan smisao za, naučiti, ovladati, osposobljavati se za, shvatiti, postići, poznavati, primjenjivati znanje, razumjeti, razviti potrebe, upoznati, usvojiti, shvatiti da se isti događaj i pojave mogu različito tumačiti, spoznati osnovna načela, steći znanja/sposobnost/uvjerenje, zapamtiti, znati…

Kompetencije i ishodi

- Kompetencije (engl. *Competences*) označavaju skup znanja i vještina te pripadajuću samostalnost i odgovornost.
- Ishodi učenja su znanje, vještine, samostalnost i odgovornost.
- Znanje (engl. *Knowledge*) označava skup stečenih i povezanih informacija.

znanje

- Znanja se odnose na činjenična i teorijska, odnosno na stečene informacije te njihovo povezivanje.
- Stečene informacije mogu biti pojmovi, njihove definicije te zasebna druga činjenična znanja koja sama po sebi ne otvaraju jednoznačnu mogućnost stvaranja novih informacija na temelju ograničenog broja postojećih informacija.

vještine

Vještine (engl. *Skills*) označavaju skup primjene znanja i upotrebe unaprijed poznatih načina rada u rješavanju zadaća i problema. Vještine se dijele na spoznajne, psihomotoričke i socijalne.

spoznajne vještine razmišljanja (engl. *cognitive skills*)	Označavaju skup stečenih logičkih i kreativnih razmišljanja.
psihomotoričke vještine (engl. *practical skills*)	Označavaju stečenu spretnost te upotrebu unaprijed poznatih metoda, instrumenata, alata i materijala.
socijalne vještine (engl. *social skills*)	Označavaju skup stečenih vještina koje su potrebne za stvaranje i razvijanje međuljudskih odnosa.

samostalnost i odgovornost

Samostalnost i odgovornost (engl. *Autonomy and responsibility competence*) označavaju postignutu primjenu konkretnih znanja i vještina u skladu s danim standardima.

samostalnost (engl. *autonomy*)	Označava pravo na vlastito upravljanje, a temelj je za određivanje nečije odgovornosti.
odgovornost (engl. *responsibility*)	Označava preuzimanje obveze da se riješe preuzete zadaće, a u skladu je sa samostalnošću rješavanja i upravljanja.

Ishodi učenja na primjeru radionice iz nastavnog predmeta Informatika

primjer ishoda učenja

Znanje	
razumjeti i definirati	odnosi se na pojmove: osnovni oblici, kontura i ispunjavanje oblika
objasniti	odnosi se na postupke umetanja, uređivanja i svrstavanja osnovnih oblika
navesti	odnosi se na primjene programa za crtanje u svakidašnjem životu
Vještine	
spoznajne vještine	odabiranje i svrhovito korištenje gotovim grafičkim oblicima u programu za obradu teksta, uređivanje gotovih grafičkih oblika odabiranjem različitih boja i debljina linija te ispunjavanjem objekta bojom
psihomotoričke vještine	mijenjanje redoslijeda objekta, svrstavanje i razvrstavanje objekata, dodavanje sjene i 3D efekata
socijalne vještine	primjenjivanje uspješne suradnje i komunikacije u interakciji s pojedincima
Samostalnost i odgovornost	
samostalnost	samostalno crtanje s pomoću programa za obradu teksta, poticanje zanimanja za programiranje, uviđanje svrhe uporabe programa za programiranje
odgovornost	prilagođavanje vlastita ponašanja unutar zadanih smjernica, aktivno sudjelovanje u radu te preuzimanje odgovornosti za rješavanje zadataka

Tip nastavnog sata

Može biti informativni i formativni.

informativni nastavni sat	Cilj je sata usvajanje činjenica, znanja.
formativni nastavni sat	Cilj je razvoj sposobnosti, a znanje je sredstvo.

Nastavni oblici

Mogu biti: frontalni, skupni, rad u paru, individualni. O nastavnim oblicima mogli ste više pročitati u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a (str. 23 – 33.)

Nastavne metode

Mogu se podijeliti na: demonstracije, praktičan rad, istraživačku metodu, metodu pokušaja i pogrešaka, metodu usmenog izlaganja, metodu razgovora, metodu čitanja, metodu pisanja i metodu crtanja.

metoda demonstracije	To je detaljnije opisano i objašnjeno s pomoću primjera u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu vizualnih metoda (str. 27).
praktičan rad	To je detaljnije opisano u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu praktične metode (str. 27).
istraživačka metoda	To je detaljnije opisano i objašnjeno s pomoću primjera u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu aktivnog učenja (str. 25).
metoda pokušaja i pogrešaka	Općenita metoda rješavanja problema prema kojoj se s nekoliko mogućih rješenja pokušava riješiti glavni problem, samostalno učenje onoga što funkcionira i onoga što ne funkcionira.
metoda usmenog izlaganja	To je detaljnije opisano i objašnjeno s pomoću primjera u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu verbalnih metoda (str. 27).
metoda razgovora	To je detaljnije opisano i objašnjeno s pomoću primjera u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu verbalnih metoda (str. 27).
metoda čitanja	To je detaljnije opisano i objašnjeno s pomoću primjera u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu verbalnih metoda (str. 27).

metoda pisanja	To je detaljnije opisano i objašnjeno s pomoću primjera u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu verbalnih metoda (str. 27).
metoda crtanja	To je detaljnije opisano i objašnjeno s pomoću primjera u 1. cjelini Razvoj pedagoških vještina radi popularizacije STEM-a u sklopu vizualnih metoda (str. 27).

Nastavna sredstva i pomagala

Obvezatni su dio pripreme za nastavni sat.

nastavna sredstva	Didaktički su oblikovana izvorna stvarnost koja omogućuje pristupačnije učenikovo spoznavanje tijekom nastavnog procesa (slike, crteži, sheme, modeli, udžbenici, priručnici, radne bilježnice, radni materijali).
nastavna pomagala	Pomažu u primjeni i predstavljanju nastavnih sredstava u nastavnom procesu (pribor za pisanje, instrumenti, aparati, demonstracijska pomagala, pano, računalo).

Upute za pisanje priprema

- Priprema za izvođenje nastave treba biti cjelovita i sažeta.
- Sadržajno treba odgovarati odabranoj temi, a intenzitetom i dubinom sadržaja dobi učenika.
- Metodički treba biti korektna što podrazumijeva pravilnu uporabu metodičke terminologije.

primjer

Primjer nastavne pripreme i tijeka izvođenja nastave

Za one koji žele saznati više. Primjer nastavne pripreme i tijeka izvođenja nastave.

Više o pisanju priprema saznajte na sljedećoj poveznici:

☞ https://www.azoo.hr/app/uploads/uvezeno/datoteke/STRUCNI2020/Priprema_2019_upute,_L._Mileti%C4%87.pdf

1.2. Programiranje

Računalni program je smisleni skup naredaba koje govore računalu što i kako treba napraviti. **Programiranje** je postupak pisanja programa. Stručnjaci koji se bave programiranjem zovu se **programeri**.

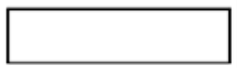
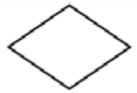
Algoritam i dijagram tijeka

Algoritam je precizno zapisan niz postupaka, radnji ili naredaba u točno određenom redoslijedu koje, uz potrebne izvore, služe da se obavi neki posao. Programiranjem se razvija algoritamski način razmišljanja, tj. Uči se rješavati probleme.

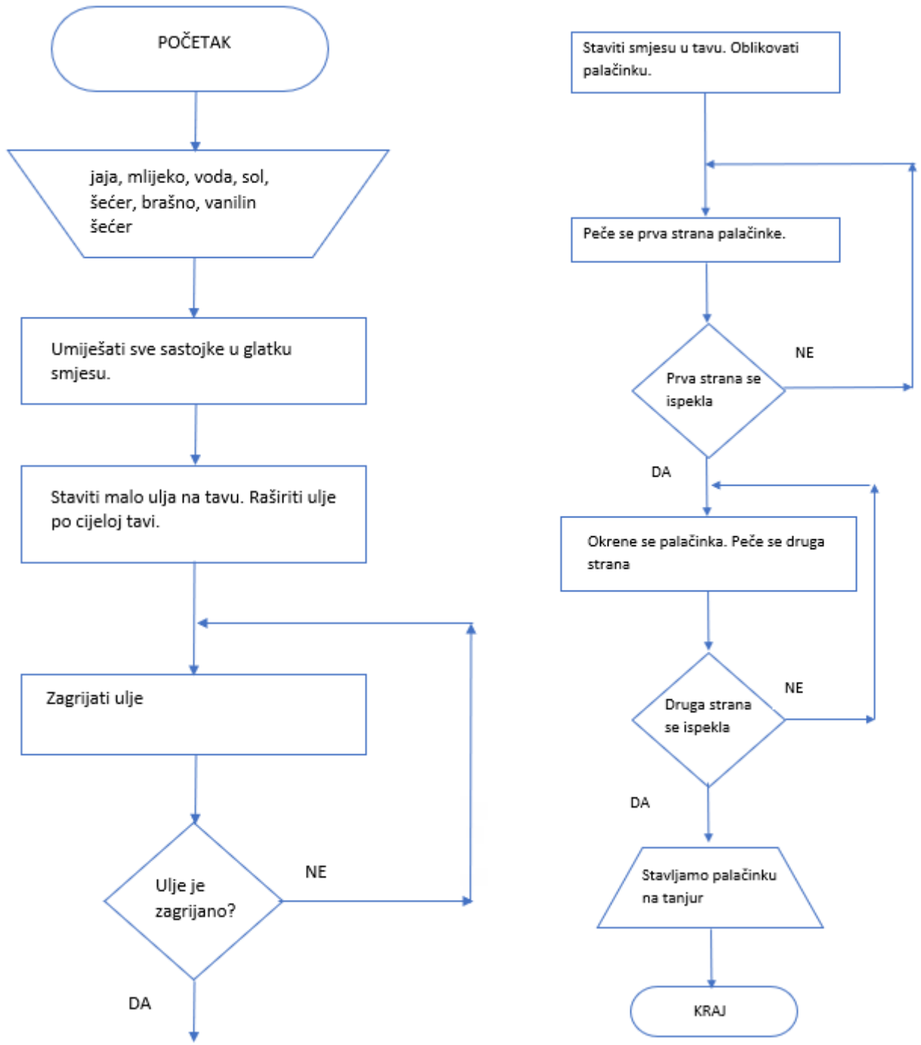
algoritam

Dijagram tijeka je grafički prikaz algoritma.

dijagram 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)

Dijagram tijeka za rješavanje stvarnog problema. Kako ispeći palačinke.



dijagram tijeka na primjeru pečenja palačinki

2. Metodološki pristup učenicima osnovne škole u poučavanju korištenja Pythonom

Nakon što ste u drugoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o metodološkom pristupu učenicima osnovne škole u poučavanju korištenja Pythonom, moći ćete:

- navesti načine pokretanja programa Python.
- upoznati sućelje te osnovne dijelove Pythona.
- složiti jednostavan niz uputa koristeći se osnovnim naredbama.
- povezati nekoliko naredbi u cjelinu.
- samostalno razviti rješenje nekog problema koristeći se ulaznim i izlaznim podatcima.

PowerPoint prezentacija

Snimka 2. radionice

Uvod u Python

Python kao programski jezik

Python je programski jezik koji je zbog svoje jednostavnosti postao jedan od najprihvaćenijih programskih jezika tijekom posljednjeg desetljeća. Danas se može pronaći svuda. Njegova je primjena svestrana, pa je izrazito prihvaćen u raznim industrijama i obrazovanju.

Zanimljivo je da omogućuje „razgovor" s mikrokontrolerima ili „razgovor" senzora, mikrokontrolera i izvršnih uređaja s pomoću jednostavnog serijskog sučelja koje se može svesti pod zajednički naziv internet stvari.

odrednice programskog jezika Python

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

- jednostavnih programa
- programa s grafičkim sučeljem (GUI)
- računalnih igara
- integraciju u mrežne stranice.

područja primjene programskog jezika Python

Mnogo je primjera primjene tog programskog jezika od kojih će neki biti obrađeni unutar programa, novih rješenja koja se primjenjuju u različitim sektorima poljoprivrede i koja pokazuju obećavajuće rezultate jer donose nove mogućnosti njihove kontrole, praćenja i pružanja naprednih usluga.

mogućnosti upotrebe Pythona

Python se može upotrebljavati:

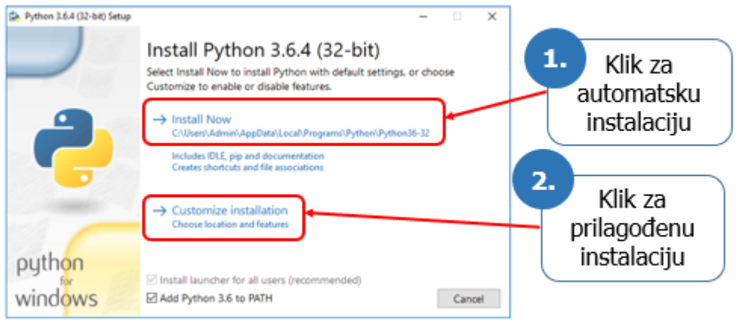
- lokalno instaliran na računalu
- alatom vizualizacije u mrežnom okružju.

koraci u lokalnom instaliranju

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

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

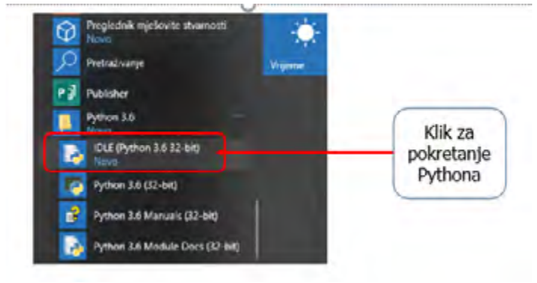
dijaloški okvir čarobnjaka za instaliranje



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

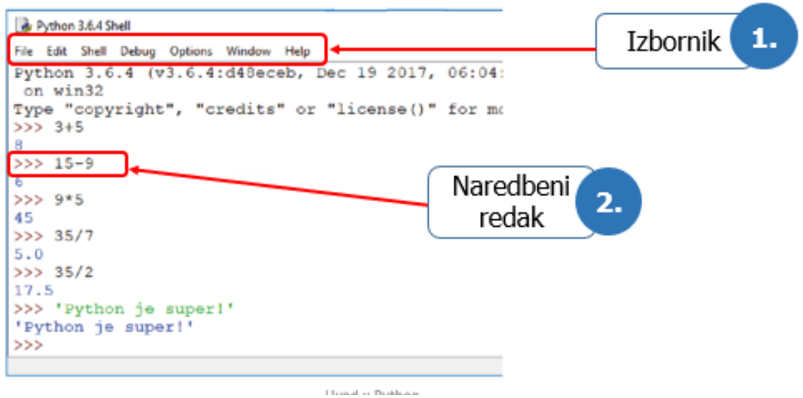
Python se pokreće klikom na IDLE (Python GUI) iz popisa programa.

pokretanje Pythona

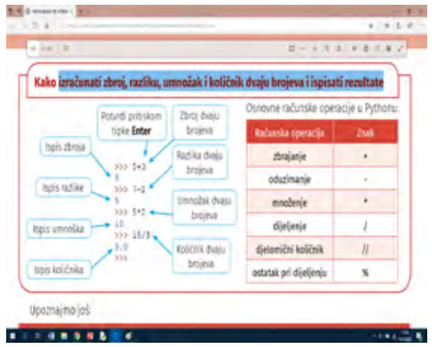


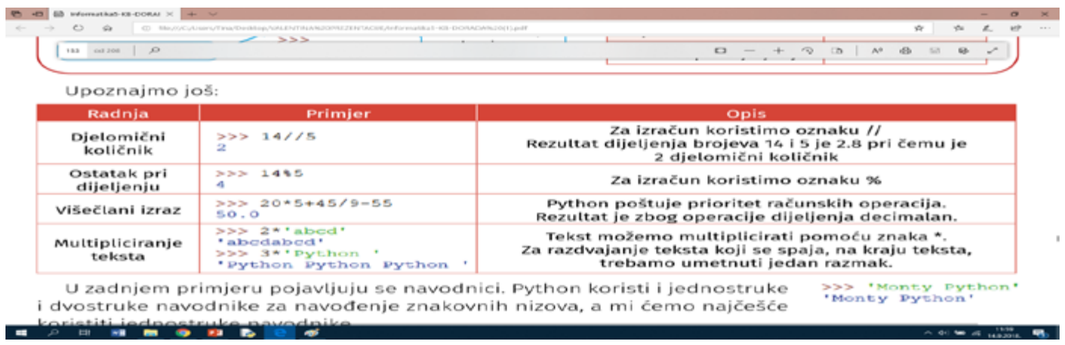
Python se može programirati u tzv. ljusci (engl. *shell*) ili konzoli u kojoj se naredbe rješavaju redak po redak kako piše. Sve što se programira u Pythonu odmah se i vidi nakon unesene **naredbe**.

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.

>>> 'Monty Python'
'Monty Python'

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

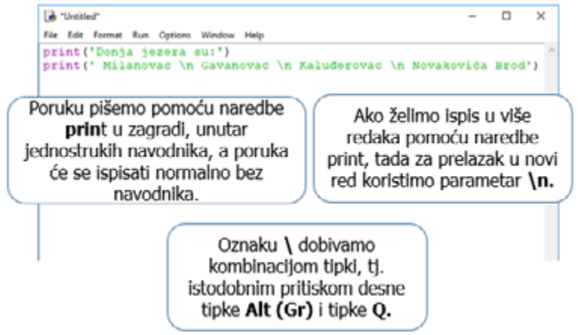
```
234561%25

>>> 11
```

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

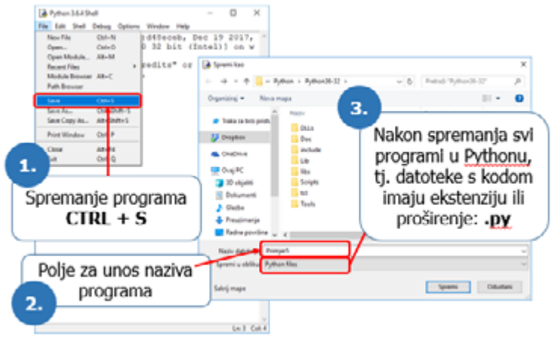
```
10* 'Hop'
```

Pisanje programa



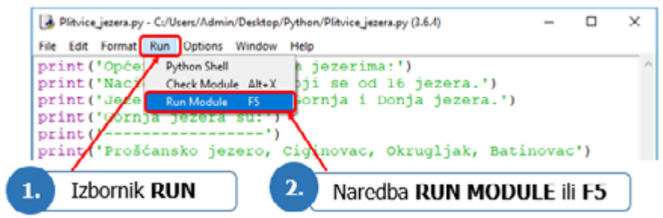
Prije pokretanja programa potrebno ga je spremiti. To se radi na sljedeći način.

spremanje programa



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

pokretanje programa



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

slijedno programiranje

Zadatak 4. Napiši program koji zbraja i dijeli 2 broja te na zaslonu ispisuje zbroj i ostatak pri dijeljenju 2 broja.

zadatci programiranja

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

Zadatak 5. Dodaj u prethodni zadatak množenje, dijeljenje, oduzimanje brojeva te djelomičan 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. Napiši program koji ispisuje „Danas je divan dan” te napiši današnji datum i zbroji 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. Napravi 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 se želi otvoriti prethodno snimljeni program, u izborniku *File* odabere se *Recent Files*, a zatim željeni program s popisa.

otvaranje prethodno snimljenog programa

Jeste li znali ...

Provjerite i primijenite

Zadatak 1:

Zbrojite dva decimalna broja i rezultat ispišite kao decimalni broj.

Zadatak 2:

Unesite troznamenkasti broj.

- Trebate ispisati:
- broj koji unosite
- broj u binarnom brojevnom sustavu **(baza 2)**.

Za one koji žele saznati više o tipovima podataka

3. Ulazne i izlazne vrijednosti programa

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

- prepoznati osnovne segmente izrade programa: ulaz – obrada – izlaz
- analizirati zadani problem
- odabrati ili predložiti niz naredbi kao moguće rješenje
- prepoznati ulazne vrijednosti i algoritamske strukture koje se upotrebljavaju za rješavanje problema
- samostalno planirati
- složiti niz uputa (naredbi) kao rješenje problema primjenom algoritamskih struktura slijeda, grananja i ponavljanja
- analizirati problem
- odabrati strategiju rješavanja
- rješenje realizirati u obliku programa s odgovarajućim tipovima podataka.

PowerPoint prezentacija

Snimka 3. radionice

Varijabla

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

varijabla

```
godine = 11
```

ime varijable — vrijednost varijable

Svake godine vrijednost će se promijeniti i zato 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:

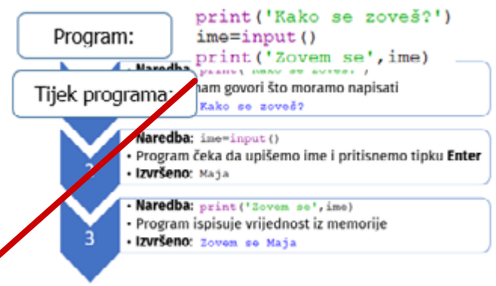
osnovni koraci programiranja

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

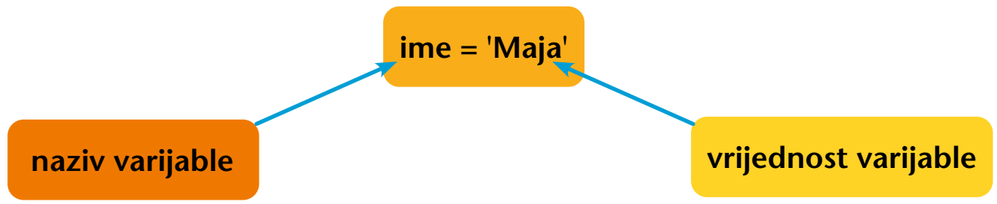
Zadatak 1. Napravi program koji unosi i ispisuje imena.

primjer uporabe varijable



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.



ime je **naziv varijable**, a *Maja* je **vrijednost 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.

Zadatak 2. Zadatak je izraditi tri varijable te napraviti program koji zbraja dva broja i 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

```
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 njezinoj provedbi najprije će se provesti unutarnja, a onda vanjska naredba.

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. Ispiši brojeve od 0 do 10

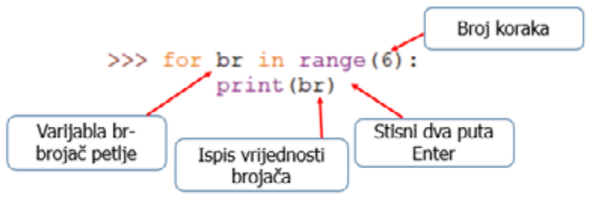
primjer petlje bez logičkog uvjeta

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

```
>>> for br in range(6):
        print(br)
```



- 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 kojemu 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.

Program može sadržavati onoliko petlja 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. Ispiši 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: Napiši 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. Napiši 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

Zadatak 7. Napiši 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. Napiši program koji ispisuje brojeve od 9 do 0.

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

Zadatak 9. Napravi program u kojemu se korisnika pita 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')
```

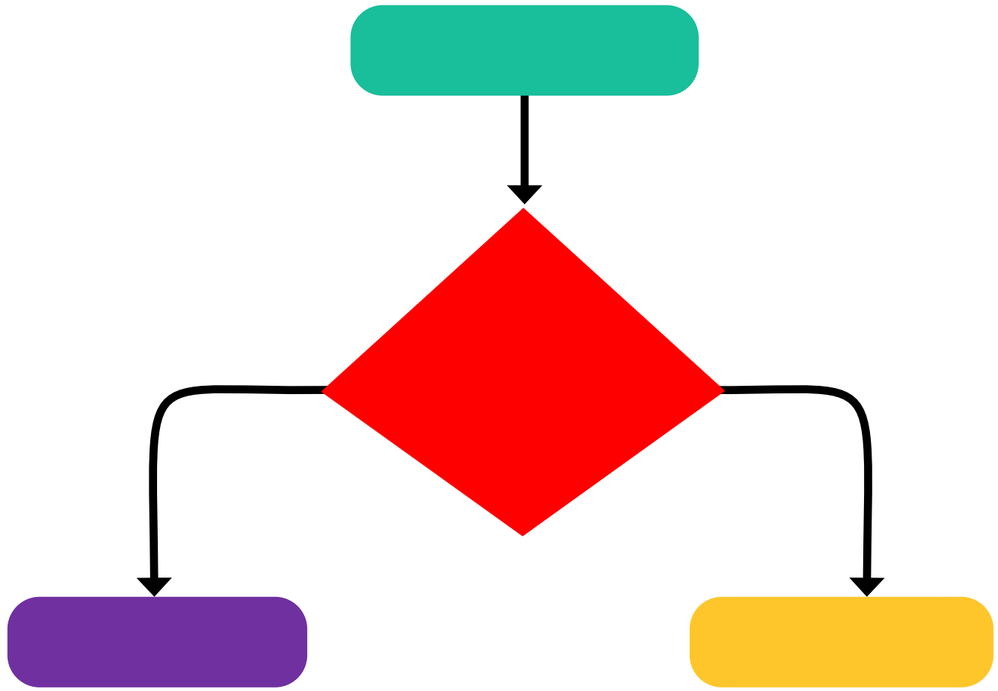
Provjerite i primijenite

Zadatak 1:

Unesite dva broja i izračunajte njihov **sumu**, **razliku**, **količnik**, **umnožak**.

- Sumu ispišite u binarnom sustavu.
- Razliku ispišite u heksadekadskom sustavu.
- Umnožak ispišite u oktalnom sustavu.
- Količnik ispišite u dekadskom sustavu.

Za one koji žele saznati više o mrežnom vizualizatoru programskih jezika



4. Uvjet u programiranju

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

- opisati odabrani problem
- prikazati osnovne korake koje predlažete za rješavanje problema
- prepoznati potrebu za uporabom nekoga složenog tipa podataka
- koristiti se funkcijama za rad sa složenim tipom podataka
- prepoznati u problemu potprobleme.
- analizirati problem te povezivati module program odgovarajućim parametrima
- navesti koji su ulazni podatci te koje rezultate treba dobiti.
- provjeriti ispravnost algoritamskog rješenja
- algoritamsko rješenja po potrebi predurediti

PowerPoint prezentacija

Snimka 4. radionice

Naredba `If`

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

naredba 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
    ...
```

primjer

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

primjer upotrebe naredbe If-else

```
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 dvije naredbe 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. Izradi program koji od korisnika zahtijeva da pritisne tipku „a“ za nastavak. 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. Napiši program koji unosi broj te provjerava je li taj 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. Napiši program koji unosi 2 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. Napiši program koji unosi broj te provjerava je li taj broj djeljiv sa 7.

```
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. Napiši program koji unosi broj te provjerava je li taj broj pozitivan.

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

Zadatak 7. Napiši program koji unosi dva broja, zbraja ih te provjerava je li njihov 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. Napiši program koji unosi duljine dvije stranice. Ako su unesene duljine jednake, izračunat će površinu kvadrata, a 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)
```

Primjenom naredaba If-else može se kombinirati nekoliko uvjeta. Izvode se samo naredbe koje slijede prvi uvjet za koji se ustanovi istinitost, a sve druge se preskaču. Naredbe If-else izvode se ako nijedan od uvjeta nije istinit.

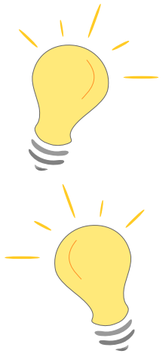
prednosti uporabe naredaba If-else

Provjerite i primijenite

Zadatak 1:

Postupak zadatka 3. provjerite u vizualizatoru rješenja na sljedećoj poveznici:

☞ https://pythontutor.com/python-debugger.html#mode=edit



Za one koji žele saznati više o operatorima koji se upotrebljavaju pri naredbi If.

5. Metodologija programiranja Pythonom i njegove funkcije za studente

Nakon što ste u petoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o metodologiji programiranja programskim jezikom Python i njegovim funkcijama za studentsku populaciju, moći ćete:

- razumjeti značenje pojmova poučavanje i učenje
- poticati studente na slobodno iznošenje vlastita i argumentirana mišljenja, na aktivno sudjelovanje i samostalan rad
- motivirati studente za aktivno učenje, rješavanje problemskih zadataka i praktično djelovanje
- odrediti ishode učenja radionica Programiranja
- objasniti pojam funkcije njezine sastavnice
- pokazati postupak poziva funkcije

PowerPoint prezentacija

Snimka 5. radionice

5.1. Uvod

pojmovno određenje poučavanja i učenje

Koncept programiranja Pythona vrlo je složen, ali nužan, pa ga treba učiti već od prvog ciklusa uz objašnjenje jednostavnih primjera i u skladu s dobi studenata. Poučavanje i učenje podrazumijeva stjecanje potrebnih znanja i vještina kao preduvjeta za razvoj kritičkoga promišljanja radi osposobljavanja za rješavanje problema te pisanje programskoga koda. Studente treba poticati na slobodno iznošenje vlastita i argumentirana mišljenja što znači da je zadaća nastavnika stvoriti ugodnu sredinu punu povjerenja i razumijevanja smisla programiranja. Način poučavanja treba osigurati ostvarivanje odgojno-obrazovnih ciljeva.

problemski usmjereno poučavanje

Problemski usmjereno poučavanje može se temeljiti na situacijama iz predložene arhitekture pametnog vrta ili na pretpostavljenim primjerima ili stvarnim životnim situacijama. Aktivnosti koje se provode trebaju pridonositi razvoju odgovornosti za vlastite postupke. Preporučene metode za ostvarivanje navedenih ciljeva jesu metode u kojima je student aktivan sudionik.

očekivanja od polaznika

Od polaznika radionica očekuje se da budu aktivni sudionici procesa učenja i poučavanja. Očekuje se da polaznici iznose svoje ideje, predlažu i smišljaju teme koje ih zanimaju, pronalaze i predlažu idejna rješenja čija je svrha održivost i opća dobrobit.

Na početku edukacije student je usmjeren na promatranje, uspoređivanje i analiziranje i postupno se kreće prema samostalnom radu u programskom okružju. U idućim radionicama odgojno-obrazovna očekivanja postaju složenija i podrazumijevaju samostalan rad. Pretpostavi li se da svaki student uči i prerađuje iskustva učenja na sebi svojstven način, potrebno je da studenti samostalno izrade svoje zadatke te ih potom izlože skupini čime će spoznati svoje sposobnosti. Aktivacijom izvođenja programa provjerit će se stupanj ostvarenosti cilja.

uloga nastavnika

Nastavnik usmjerava učenje i poučavanje osluškujući što studente zanima i utvrđujući njihova prethodna znanja i iskustva. Njegova je uloga motivirati studente za aktivno učenje, rješavanje problemskih zadataka i praktično djelovanje.

Odgojno-obrazovni ciljevi i ishodi radionica *Programiranja*

odgojno-obrazovni ciljevi

Odgojno-obrazovni ciljevi radionica namijenjenih studentima su:

- stjecanje znanja i vještina potrebnih za analizu i rješavanje problema primjenom sastavnih dijelova računalnog razmišljanja te na temelju tako dobivenih rezultata analize problema izraditi dijagram tijeka, odnosno algoritam, kao konačan rezultat cijelog procesa
- stjecanje znanja i vještina potrebnih za izradu jednostavnih računalnih programa u programskom jeziku Python; izradom jednostavnih programskih rješenja, primjenjivih u praksi, polaznici će se upoznati s osnovnim sastavnicama svojstvenim svakom programskom jeziku, pa tako i programskom jeziku Python
- stjecanje znanja i vještina potrebnih za izradu složenih računalnih programa i aplikacija u programskom jeziku Python primjenjivih u okružju interneta stvari; izradom složenih programskih rješenja, primjenjivih u praksi, polaznici će se upoznati s naprednim sastavnicama svojstvenim svakom programskom jeziku, pa tako i programskom jeziku Python
- istraživanje i kritičko vrednovanje u digitalnom okružju i mrežnom poučavanju
- poticanje razmišljanja usmjerena prema budućnosti.

ishodi učenja

Ishodi učenja radionica su sljedeći:

- razložiti kompleksne probleme na više manjih samostalnih logičkih cjelina
- primjenjivati tehnike programiranja za rješavanje problema
- primijeniti rješenje problema u programskom jeziku Python
- riješiti individualne zadatke i timske projekte
- aktivno sudjelovati u timu u izradi projekta.

5.2. Odrednice, poziv i parametri funkcije

Odrednice funkcije

svrha funkcije

Funkcija je dio programskoga kôda koji je organiziran prema određenoj logičkoj cjelini. Svrha je korištenja funkcijom da se dijelovi programske logike koji se ponavljaju u programskom kôdu napišu samo jednom, i to unutar funkcije. U primjeni funkcija jedan te isti kôd može se iskoristiti neograničen broj puta jednostavnim pozivom funkcije koja je u određenom trenutku potrebna. Takav prist up omogućuje veću preglednost i organizaciju programskoga kôda. Jadnako tako, koristeći se funkcijama, znatno je lakše nakon pisanja pročitati programski kod koji je zapisan. Funkcije se upotrebljavaju kao osnovne građevne sastavnice 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 unaprijed napisane. Ako su funkcije unaprijed napisane, to su funkcije koje su ugrađene u sâm programski jezik ili su dio standardnog ili uključenog paketa. Pojam paket 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 primjeru 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 poput neke druge funkcije. Nakon naziva funkcije obvezatne su zagrade, (), i dvotočka, :, nakon zagrada.

opis funkcije s pomoću navedena 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 navedenu primjeru tijelo funkcije je programski blok koji je potrebno na neki način omeđiti kako bi se u 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 logički ispravno napisane naredbe u skladu sa sintaksom programskog jezika, može se pozivati tako da se napiše njezino ime te se 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 poziva funkcije

primjer

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

Kao što se može vidjeti iz navedena primjera, najprije je napisana funkcija (zaglavlje i tijelo funkcije) te je tek nakon toga izveden sâm poziv funkcije **moja_funkcija()**. Bitno je 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 idućem primjeru može se vidjeti poziv funkcije prije njezine definicije.

Korišten je IDLE u interaktivnom načinu rada, no ista pogreška pojavila bi se 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 donja crta (engl. *Underscore*)

Parametri funkcije

Da bi se funkcijama mogle slati različite vrijednosti na temelju kojih će one 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 funkcija **kvadrat()**. Funkcija kvadrat() prima samo jedan parametar i, ovisno o vrijednosti tog parametra, ona izračunava kvadrat unesena broja. Prvi put se poziva funkcija kvadrat(3), a drugi put kvadrat(5). Rezultat funkcije pri prvom pozivu mora biti 9, a rezultat funkcije pri drugom pozivu mora biti 25, tj. rezultat treba ovisiti o vrijednostima koje je funkcija primila.

formalni parametri

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

primjer

„a”,2,

„a”,”b”

primjer sljedivosti

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

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

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

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

primjer

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

primjer definicije funkcije kvadrat()

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:

primjer

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

primjer def suma

U donjem se primjeru vidi definicija funkcije suma() s dva formalna parametra var1 i var2. Iza same funkcije (zaglavlja i tijela funkcije) nalazi se poziv funkcije suma(10, 20). U pozivu se u funkciju prenose 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

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

suma(10, 20)
```

```
Izlaz:
      30
```

primjer poziva funkcije s formalnim parametrima

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 var3poprimit će vrijednost 0 samo ako se pri pozivu funkcije prenesu 2 argumenta, a ne

sva 3 koliko ih najviše može biti. Ako se prenesu 3 argumenta, tada varijabla var3 poprima vrijednost trećega prenesenog argumenta

primjer

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

suma(10, 20)
```

```
Izlaz:
      30
```

primjer način 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 2 argumenta. U ovom slučaju varijabla var 3 poprima predefiniranu vrijednost, a to je vrijednost 0.

primjer

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

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

suma(10, 20, 500)
```

```
Izlaz:
530
```

U tom primjeru prenose se 3 argumenta, tj. vrijednosti. Kako funkcija suma() može poprimiti najviše 3 parametra, pri pozivu funkcije suma() prenesena su 3 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 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.

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.

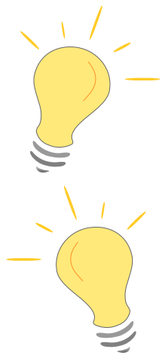
Nakon toga tako definirane i isprobane funkcije mogu se pozivati iz drugih funkcija kao što je prikazano na sljedećem primjeru.

primjer

```
>>> 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
```

primjer načina pozivanja iz drugih funkcija

Provjerite i primijenite



Pitanja za ponavljanje o funkcijama u programu

1. Funkcije se mogu podijeliti na dva dijela. Koji su to dijelovi?
2. Zaglavlje funkcije sastoji se od tri sastavnice. Koje su to sastavnice?
3. Ako funkcija prima pet parametara, je li pri pozivu funkcije potrebno navesti svih pet argumenata ili postoji način da se neki argumenti izostave?
4. Kako se zove naredba s pomoću koje funkcija vraća u pozivajući dio programa izračunanu vrijednost?
5. Kako se zove tip varijable koji je vidljiv samo u funkciji te se nakon završetka funkcije takva varijabla „uništava”?
6. Jesu li globalne varijable iz glavnog dijela programa vidljive u funkcijama?

Zadatci za uvježbavanje pisanja funkcija

1. Napišite funkciju koja ispisuje proizvoljan niz znakova, na primjer „Hello World!”. U glavnom dijelu programa pozovite napisanu funkciju.

2. Napišite funkciju koja prima 4 parametra. Funkcija mora ispisati rezultat matematičke formule:

   ((a*a) + (b*c) – d) / 2

   U glavnom dijelu programa pozovite napisanu funkciju.

3. Napišite funkciju mnozenje() koja može primiti dvije vrijednosti od kojih je druga unaprijed zadana, sami odredite broj koji ćete pridružiti unaprijed zadanoj vrijednosti. Funkcija mora izračunati i ispisati rezultat množenja.

   U glavnom dijelu programa pozovite funkciju dvaput. Pri prvom pozivu funkcije kao argumente funkcije navedite dvije vrijednosti (vrijednost drugog argumenta neka bude drukčija od unaprijed zadane vrijednosti). Drugi put funkciju pozovite samo jednim argumentom.

4. Napišite funkciju koja prima dva parametra. Rezultat izračuna funkcije ovaj put se 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 kojemu se funkcija poziva te ispišite tu varijablu.

5. 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 varijabla.

6. Prethodni zadatak napišite tako da se ne koristite globalnim varijablama (korištenjem parametara funkcije).

7. ** Napišite 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, ispišite poruku: „Nedopuštene vrijednosti!”. Ako je uvjet zadovoljen, izračunajte i ispišite binomni koeficijent „m povrh n”, pri čemu se koristite sljedećim izrazom:

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

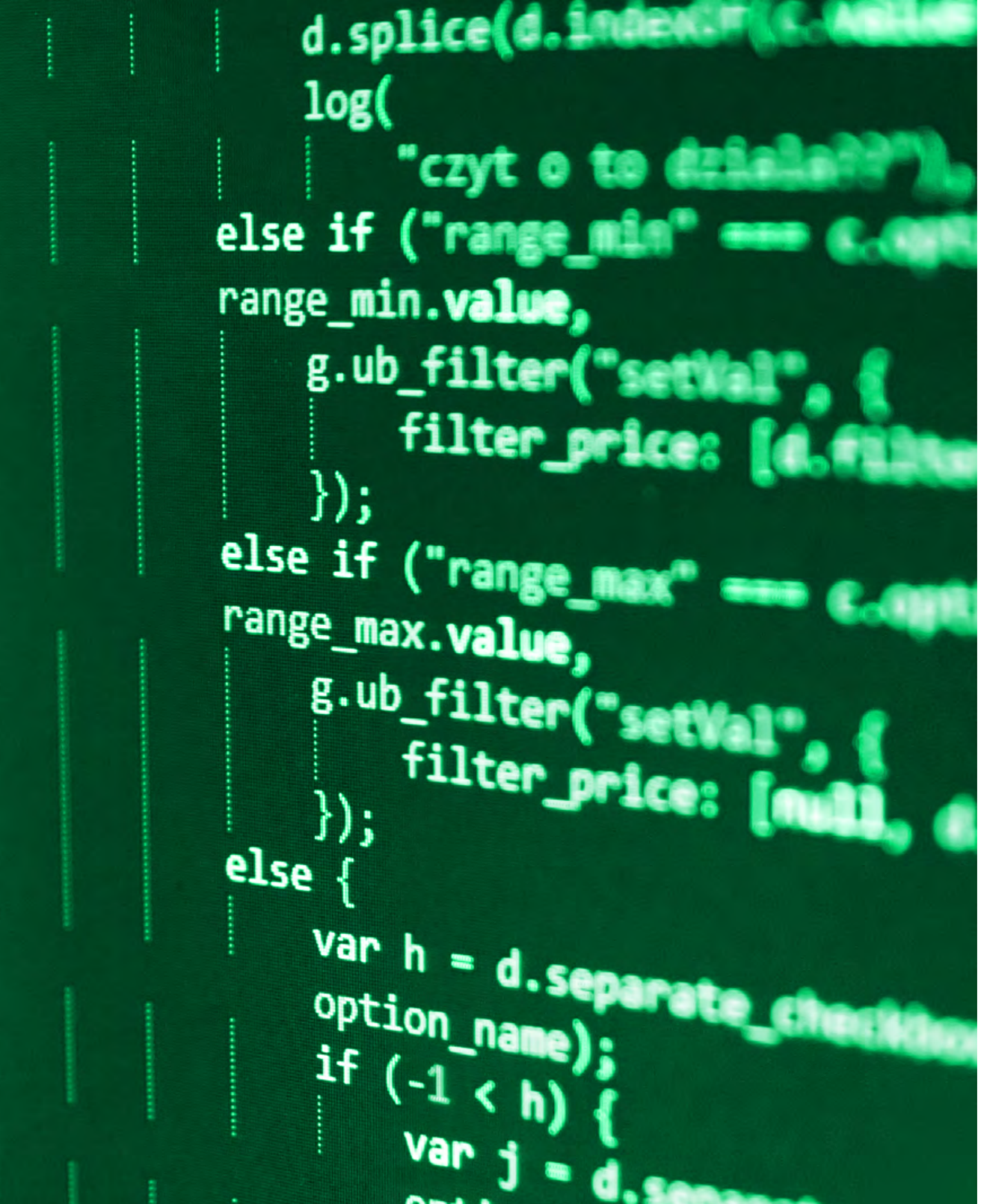
Primjer: 5! izračunava se na način 5*4*3*2.

(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 skladištima podataka – varijablama, moći ćete:

- razumjeti pojam vidljivosti varijable
- razlikovati lokalne i globalne varijable
- objasniti životni vijek varijable i kako se on ograničava na završetak bloka programskog koda u kojem je varijabla nastala
- razumjeti koncept ugrađenih varijabli koje su dostupne unutar razine jedne instancije interpretera ili u svim instancijama interpretera
- razumjeti preklapanje vidljivosti te objasniti situaciju nadjačavanja kada se globalna i lokalna varijabla imaju isti naziv.
- pokazati kako se deklarira globalna varijabla te kako se koristi ključnom riječi “global” kako bi se eksplicitno odredilo da se želi djelovati nad ili pristupiti globalnoj varijabli.
- identificirati pred-ugrađene varijable i objasniti kako se njima koristi

PowerPoint prezentacija

Snimka 6. radionice

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

vidljivost varijable

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 ovdje.

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 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

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

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

primjer programa

U navedenu programu u glavnom dijelu 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 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.

Globalne varijable

Uz lokalne varijable postoje i globalne. Globalne varijable su varijable koje su kreirane i korištene u glavnom dijelu programa i njihov životni vijek traje dokle god se provodi program. 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 globalne varijable

U gornjem programu kreirana je varijabla var i pridružena joj je vrijednost 5. Nakon toga u sljedećoj liniji pozvana je funkcija `test()`. Funkcija `test()` ispisuje sadržaj varijable var, no vidljivo je 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 može biti napravljena izravno u glavnom tijelu programa, bez stvaranja modularnog i organiziranog koda, u funkcijama/metodama kao u sljedećem primjeru.

deklaracija globalnih varijabla

primjer

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

primjer deklaracije globalnih varijabla

To je moguće i unutar bloka funkcije kao što je prikazano u primjeru koji slijedi.

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”, varijabla koja ispisuje niz (engl. *string*) s osnovnim informacijama o organizacijama koje podržavaju rad zajednice Python.

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.
```

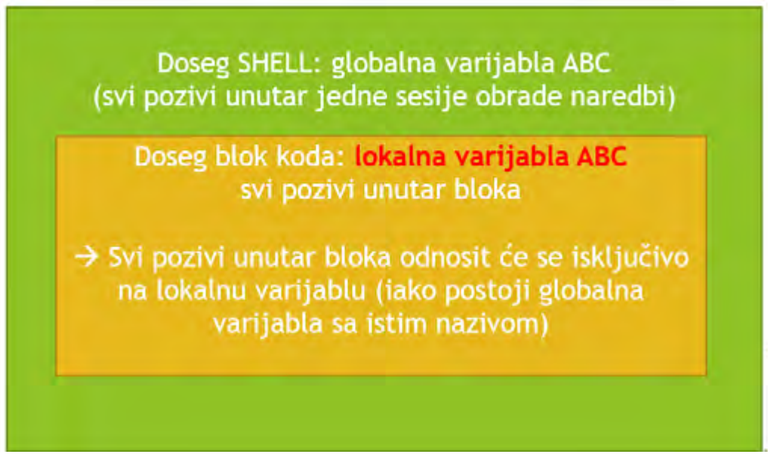
primjer predugrađene varijable

Popis svih predugrađenih varijabla i naziva:

```
['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, prekriva se vrijednost varijable šireg dosega varijablom užeg dosega.



Ako globalna i lokalna varijabla imaju isti naziv, preklapa se vidljivost 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

Kad se globalna varijabla rabi unutar funkcije, ona preko naziva ima istu vrijednost kao i pozivom izvan funkcije.

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

U situaciji kad se unutar tijela funkcije definira globalna varijanta s istim nazivom 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 nazivom 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)
...
...
>>>
```

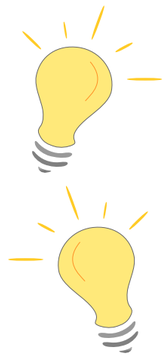
primjer korištenja ključnom riječju global

Nadovezujući se na prethodni primjer, kad se pozove ta 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. Napišite programski kod u Pythonu koji pokazuje kako kreiranje i mijenjanje vrijednosti lokalne varijable istog naziva kao i globalna varijabla ne utječe na vrijednost globalne varijable.
2. Objasnite zašto su korisne lokalne varijable, a kad mogu izazvati problem.
3. Pokažite na primjeru kakve su to predugrađene varijable te kako se rabe.
4. Napišite programski kod u Pythonu koji se koristi globalnim, lokalnim i predugrađenim varijablama.
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 na računalu te razlike između fizičke radne memorije i radne memorije operativnog sustava
- riješiti sistemske pogreške uzrokovanu prekoračenjem dostupne memorije
- razumjeti važnosti i načela upravljanja potrošnjom memorije u modernim programskim jezicima te razlika između ručnog i automatskog upravljanja potrošnjom memorije
- razumjet životni ciklus varijabli/objekata u programskom jeziku Python te faza stvaranja, korištenja i uništavanja
- razumjet mehanizam za automatsko upravljanje memorijom u programskom jeziku Python
- koristiti se ugrađenim metodama za praćenje i upravljanje memorijom

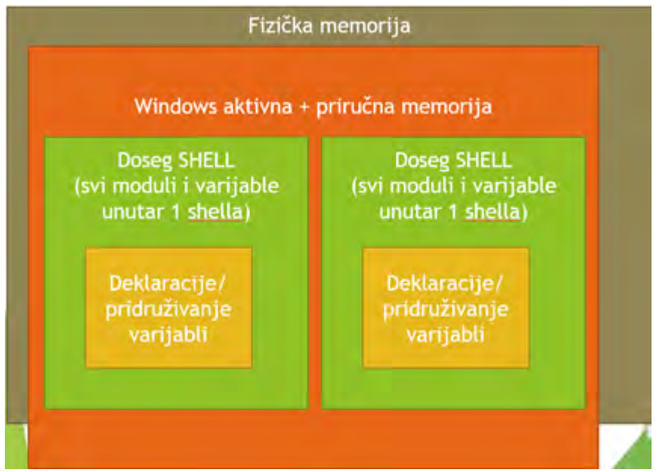
PowerPoint prezentacija

Snimka 7. radionice

Upravljanje memorijom računala

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

kategorije memorije



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

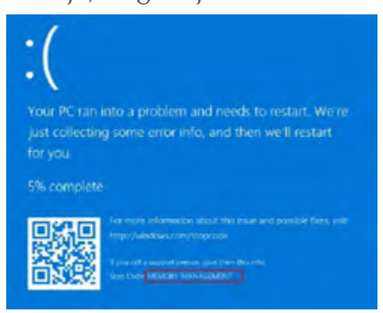
razlika između fizičke radne memorije i radne memorije operativnog sustava

Unutar memorije kojom se koristi operativni sustav nalaze se sve aplikacije uključujući i *shell* tumač 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 ne primjenjuje se proces za uklanjanje memorije, događaju se razne sistemske pogreške – primjer je prikazan na idućoj slici.

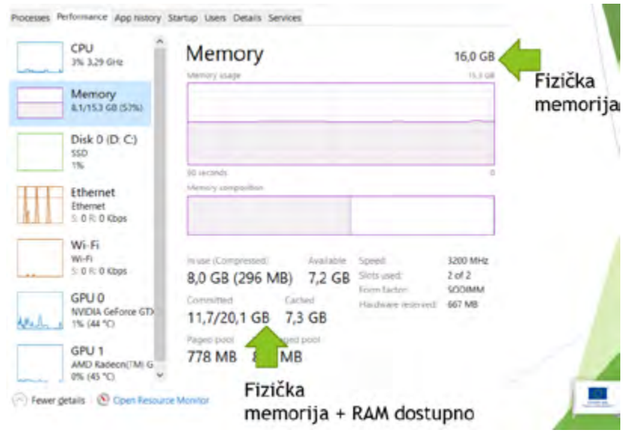
primjer sistemske pogreške



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, to izaziva izrazito negativne posljedice za rad našeg programa, ali i cijelog računala. Zbog toga je u današnjim, suvremenim 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 popraćeno sljedećim problemima:

nedostatci ručnog upravljanja potrošnjom memorije

- vrlo je česta pogreška zaboravljanje oslobađanja memorije
- vrlo je česta pogreška prerano oslobađanje memorije.

Kako je spomenuto, suvremeni programski jezici (uključujući Python) već imaju ugrađeno upravljanje potrošnje memorije i ti se 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 kako bi bilo dovoljno memorije za objekte/podatke u memoriji tumača (*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 na 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 pokreće 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/varijabla moguće je ručno ubrzati oslobođenje memorije. Alat koji služi za praćenje potrošnje memorije / brzine izvršavanja programa na engleskom se jeziku naziva profiler.

djelovanje automatskog mehanizma za upravljanje potrošnjom memorije

Životni vijek varijabla/objekata u Pythonu

Svaki objekt koji se primjenjuje u programskom jeziku Python prolazi kroz tri faze u svojemu 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 naziv te varijable. Prvi korak u toj fazi kroz koju prolaze svi objekti jest konstrukcija. U primjeni to znači da se poziva funkcija, tj. metoda zvana konstruktor. Uobičajen način rada metode konstruktor 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

Životni ciklus svake varijable, tj. objekta, u programskom jeziku Python završava fazom koja se zove destrukcija.

uništavanje objekata

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

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 i rada sustava s varijablama.

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

način dobivanja uvida u stanje automatskog upravljanja memorijom

```
>>> 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 i ručno ukloniti 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. Donja metoda daje 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 prema načelu 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 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.

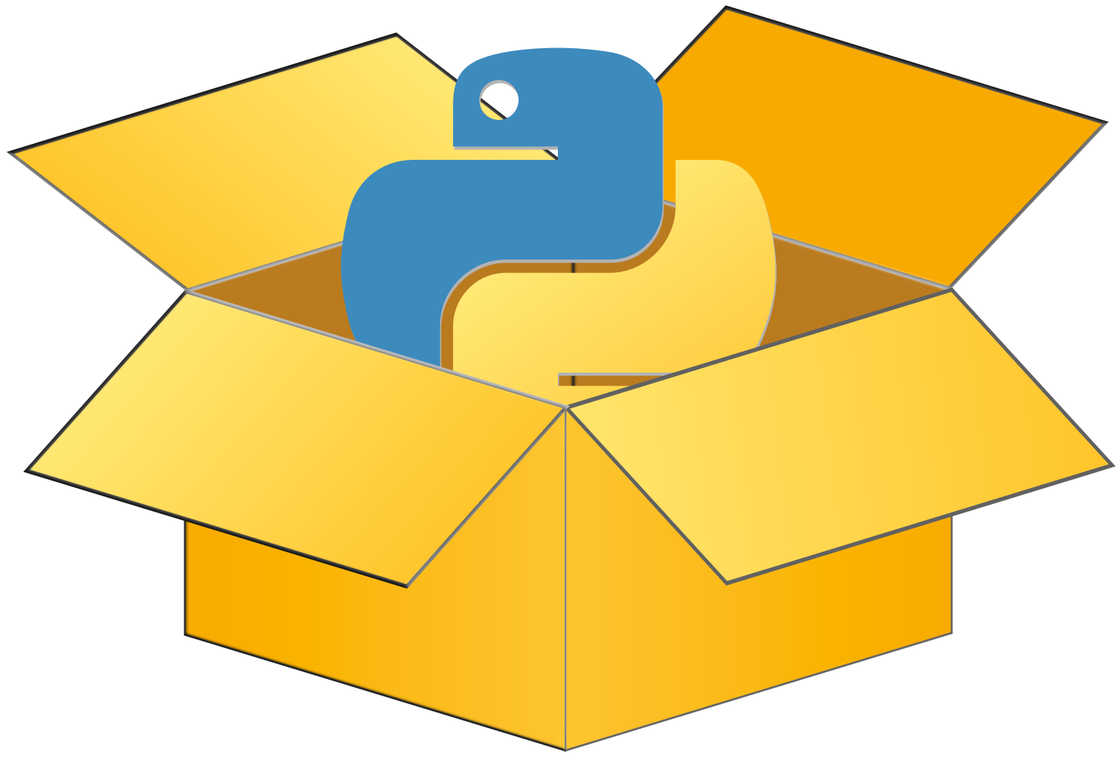
uklanjanje objekata koji nemaju referencije

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

> 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:

- razumjeti doprinos modula i paketa razvoju programskog koda te organizaciju modula i paketa radi lakšeg snalaženja i traženja po dokumentaciji
- prepoznati razliku između modula i paketa, odnosno što su moduli i kako su organizirani u pakete
- koristiti se funkcijama, konstantama i razredima koji se nalaze unutar modula na dva načina: najavljivanjem korištenja i uključivanjem u program
- prepoznati razlike između najavljivanja korištenja i uključivanja u program
- razumjeti značenje ključnih riječi “import” i “from” u kontekstu korištenja modula i paketa u programskom kodu.
- koristiti se konstantama koje su u modulima
- razumjeti način pristupa konstantama koje su u modulima
- instalirati module trećih strana korištenjem pip-a
- prepoznati katalog modula trećih strana na pypi.org.

PowerPoint prezentacija

Snimka 8. radionice

doprinos modula i paketa razvoju programskoga kôda

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

standardna biblioteka

Moduli i paketi sa standardnom instalacijom *Pythona* nazivaju se standardnom bibliotekom. Katkad su programerima potrebne funkcionalnosti koje ne dolaze s modulima i 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 provoditi. U ovoj edukaciji obrađeni su samo moduli sa standardnom instalacijom *Pythona*.

razlika između modula i paketa

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

Paketi su cjeline koje uključuju druge module i omogućuju njihovo hijerarhijsko svrstavanje. Kao primjer može poslužiti paket sa standardnom bibliotekom, paket urllib, koji sadržava nekoliko modula:

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

Rad s modulima i paketima

dva načina korištenja funkcijom, konstantom i razredom

Funkcijom, konstantom i razredom unutar nekog modula može se koristiti na dva načina:

- 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 najavljivanja korištenja modulom

U gornjem primjeru prikazan je način primjene naredbe 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 da se u nekom modulu mogu nalaziti i razredi i konstante) iz nekog modula.

Ako se primjena funkcija samo najavi kao u gornjem primjeru, tada u svakom dijelu programa u kojemu se želi iskoristiti funkcija iz najavljenog modula treba pisati naziv modula i funkcije odvojene točkom. Sintaksa takvog načina primjene najavljenih funkcija jest:

```
modul.funkcija()
```

Uključivanje u program

Ako se želi izbjeći prethodna sintaksa i primjenjivati se samo naziv funkcije kao poziv, a ne da se ispred naziva funkcije piše ime modula u kojemu se funkcija nalazi, tada je potrebno uključiti željene funkcije u program. Za uključivanje funkcija u program koriste se ključne riječi from i import.

korištenje ključnim riječima 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 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 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 primjenjivati u programskom kôdu, a to su sin() i cos(). Primjećuje se da pri pozivu funkcija više nije potrebno ispred funkcije pisati naziv modula u kojemu se ta funkcija nalazi.

U prethodnom 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 naziv funkcije, već je pozvana na način math.tan(), no svejedno se dogodila pogreška. A ta je pogreška nastala zbog toga što se u navedenom načinu uključivanja funkcija iz modula math u program nije najavilo i korištenje ostalim funkcijama iz tog paketa.

namjerno izazvana pogreška

Želi li se 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 primjene svih funkcija iz modula math:

```
import math.
```

Svejedno je koji će se način odabrati.

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

razlog uključivanja svih funkcija iz nekog modula i mogući problem

```
from math import *
```

Pri takvom načinu korištenja modulom može se pojaviti problem. Naime, pri uključivanju više različitih modula može se dogoditi kolizija ako se u dva ili više modula nalazi isti naziv 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
```

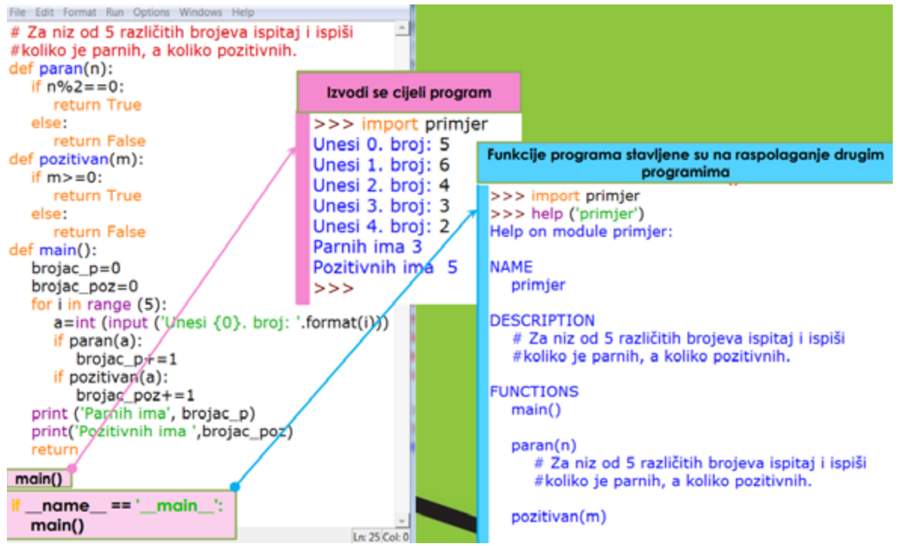
Uključivanje u program konstante Pi iz modula math:

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

```
Izlaz:
     3.141592653589793
```

Kreiranje vlastitih modula

Na sljedećoj slici prikazano je kako možemo napraviti vlastiti modul koji nakon toga možemo uključiti u kod i koristiti se njime.

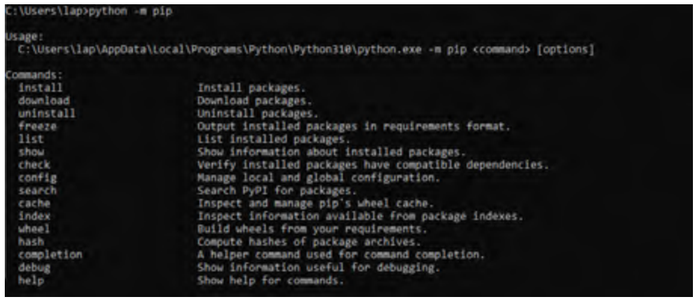


Instaliranje modula trećih strana

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

Pokretanje iz komandnog retka

postupak pokretanja iz komandnog retka



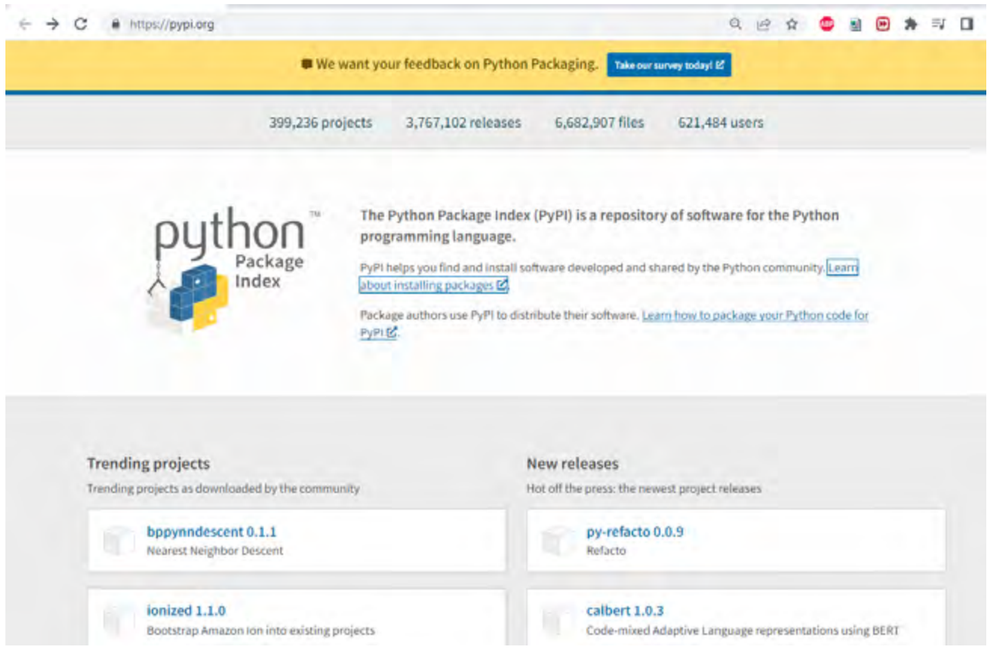
- Ako se već zna naziv modula i pip je instaliran

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

- Ako nema pip-instalacije, najprije treba postaviti pip

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

- Registar svih modula trećih strana prikazuje se s pomoću alata i kataloga pypi.org
  ☞ https://pypi.org/



Vježba korištenja modulima i paketima za *Python*

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

```
sin(2) + cos(1).
```

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

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

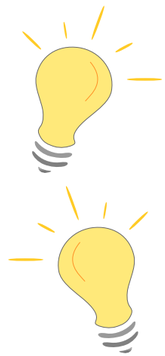
3. Koristeći se konstantama koje su pohranjene u modul math, ispišite vrijednost konstante *Pi*.

4. Koristeći se konstantom *Pi*, izračunajte i ispišite rezultat: sin(2 *Pi*) + cos(*Pi*).

5. * U službenoj dokumentaciji za *Python 3* pronađite naziv 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 $(a + b)2 = a^2 + 2\ ab + b^2$. Funkcije kojima ćete se koristiti u ovom zadatku uključite u program.

6. * U službenoj dokumentaciji za *Python 3* pronađite naziv funkcije koja služi za zaokruživanje na prvi viši cijeli broj i na prvi niži cijeli broj. Broj nad kojim se želi napraviti zaokruživanje učitajte s tipkovnice. Tako učitani broj neka bude decimalni broj. Na primjer, za učitani broj 5587 ispis na ekranu mora imati ovakav sadržaj: “5, 6”. U ovom zadatku najavite primjenu funkcija, no nemojte ih uključiti.

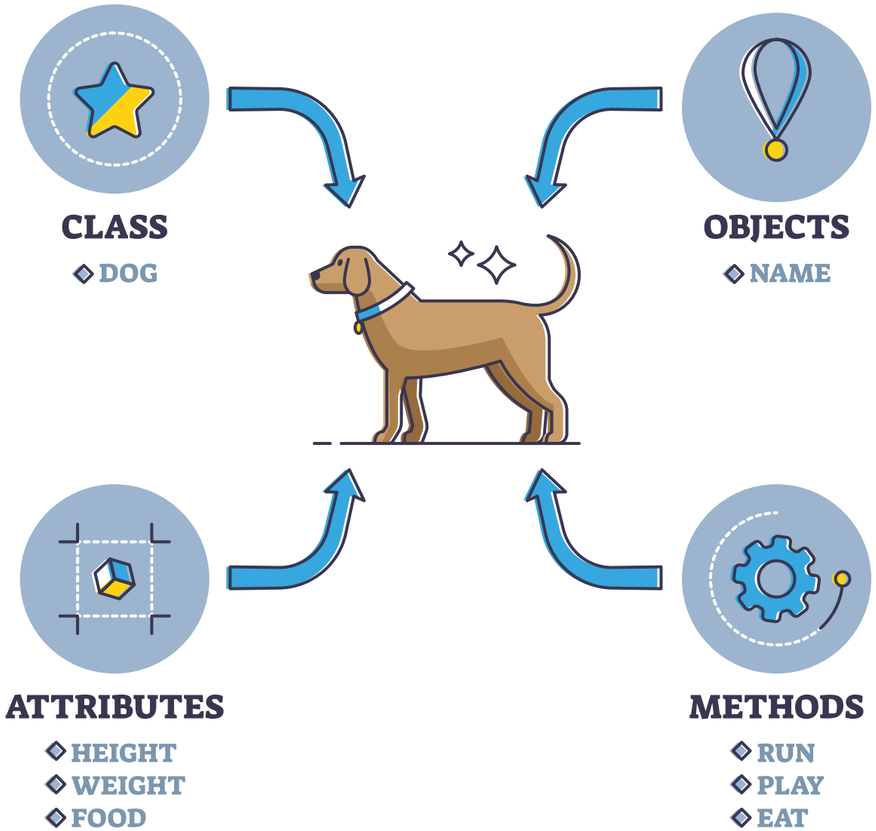
Provjerite i primijenite



1. Napišite zašto se upotrebljavaju 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 preklapaju.

Za one koji žele saznati više

1. Provjerite definiciju modula u 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 orjentiranom programiranju, moći ćete:

- razumjeti osnove objektno orijentiranog programiranja, uključujući razlike između objekata i funkcija te koncepte klasa, instancija, atributa i metoda
- odrediti klase, atribute i metode u Pythonu
- implementirati klase, atribute i metode u Pythonu
- razumjeti prednosti i mane objektno orijentiranog programiranja u odnosu na proceduralno programiranje
- primijeniti koncepte objektno orijentiranog programiranja za rješavanje problema i izradu programskih rješenja
- povezati objektno orijentirano programiranje s kulturom STEAM-a i primijeniti ga u različitim područjima.

PowerPoint prezentacija

Snimka 9. radionice

namjena radionica

Radionice su namijenjene početnicima u programiranju, ali i onima koji su već programirali, a žele naučiti programski jezik Python kao jedan od najpopularnijih i najkorištenijih programskih jezika današnjice kojemu se predviđa još svjetlija budućnost.

sadržaj radionica

Nakon uvodnih predavanja u kojima je objašnjen način instalacije te osnove naredaba, u radionici će biti razrađen i drukčiji pristup programiranju, ponajprije namijenjen korisnicima koji prednosti Pythona vide u konkretnim situacijama u svakodnevnom životu.

pretpostavke doprinosa programiranju

Za navedene činjenice polazi se od sljedećih pretpostavki.

- Python je objektno orijentiran programski jezik. Za razliku od programiranja usmjerena na postupak, u kojemu je naglasak na funkcijama, objektno orijentirano programiranje naglašava objekte.
- Objekt je jednostavno zbirka podataka (varijabla) i metoda (funkcija) koje djeluju na te podatke. Slično tome, klasa je nacrt za taj objekt.
- O klasi se može razmišljati kao o skici (prototipu) kuće. Sadržava sve detalje o podovima, vratima, prozorima itd. Na temelju tih opisa gradi se kuća. Kuća je objekt.
- Kako se mnoge kuće mogu izgraditi prema nacrtu, i mnoge predmete možemo stvoriti iz klase. Objekt se naziva i instancijom klase, a postupak stvaranja tog objekta naziva se **instancijom**.

Python kao dio kulture STEM-a

Razumijevanje programskog jezika u kontekstu objektnog programiranja kao dio kulture STEM-a, jer on više nije samo pokret, najjednostavnije je objasnio jedan naš polaznik, a njegov osvrt na Python prenesen je u cijelosti.

„Ne namjeravam učiti kako bih zaradio od programiranja – Python uvježbavam isključivo zato što ga volim. Najveći mi je problem naći slobodno vrijeme za Python, pa sam zato sve ostalo stavio na stranu.

Želim za sebe praviti jednostavne programe, poput skripta koje bi povlačile podatke s interneta i redovito me obavještavale o promjenama. Znam da takve stvari već postoje, ali ja želim svoje programe, prilagođene točno

onome što treba meni. Pokušao sam napraviti nešto slično unutar *google sheets*, ali je on previše ograničen za ozbiljno baratanje podatcima. Jednako tako želim raditi s računalom *Raspberry* Pi koje planiram uskoro naručiti te iskoristiti Python u 3ds maxu.

Python sam odabrao nakon istraživanja na internetu kao program koji je jednostavan za učenje i za koji postoji dokumentacija na internetu. Naime, postoje primjeri baš za sve! Pisanje u Pythonu također je jednostavno – u usporedbi s drugim jezicima."

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 temelji na funkcijama, tj. blokovima 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.

prednosti objektno orijentiranog programiranja

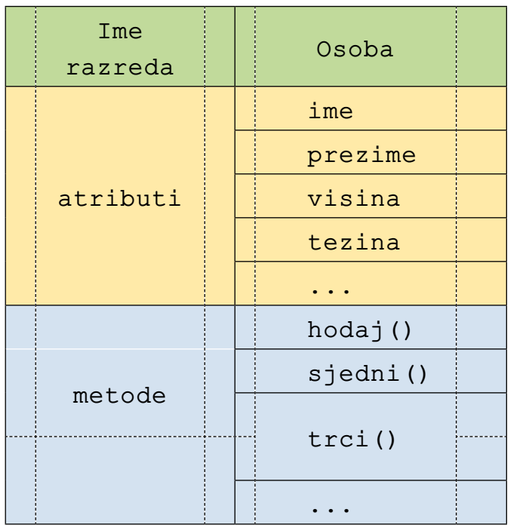
Objektno orijentirano programiranje (OOP) služi smanjenju kompleksnosti razvoja i održavanja programskih rješenja. OOP ne sprečava programere da pišu loš programski kôd, već ih usmjerava da razvijaju programski kôd u okvirima standarda te paradigme.

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 se osobi mogu pridružiti razni parametri ili pak u duhu objektno orijentiranog pristupa **atributi**. 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. Ovdje je ilustrativan prikaz razreda Osoba.



Definiranje objekta/klase

U nastavku slijedi tablica istoznačnica ili sinonima, riječi koje imaju isto značenje, a često se pojavljuju u literaturi.

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**.

osnovna programska cjelina kod objektno orijentiranog načina programiranja

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.

U nastavku se nalazi osnovni primjer kreiranja razreda.

primjer

primjer kreiranja razreda

```
#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)
```

Pozivanje metoda

kreiranje razreda

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

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 sljedećem 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 rabi parametar self koji označava varijable i funkcije jedne instancije objekta.

primjer

primjer upotrebe parametra self

```
    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.

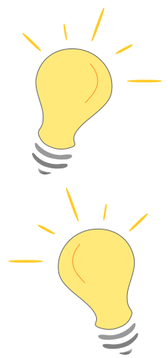
Na primjeru programskoga koda s početka na ovaj se način deklarira instancija klase, tj. objekt, te kako se poziva.

primjer

primjer poziva klase i funkcije

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

Provjerite i primijenite



1. Objasnite na svojem primjeru kako biste 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. Interakcija s fizičkom okolinom (senzori)

Nakon što ste u desetoj radionici, a potom i u ovom priručniku naučili sve što trebate znati o interakciji s fizičkom okolinom, moći ćete:

- razumjeti koncept ugrađenih senzora i njihovu ulogu u mjernim sustavima
- poznavati praktične primjene ugrađenih senzora u poljoprivredi
- upoznati se s programskim knjižnicama za pristup ugrađenim senzorima u programskom jeziku Python
- razumjeti kako se koristi ugrađenim senzorima u kombinaciji s internetom stvari (IoT) tehnologijom
- razumjeti ulogu ugrađenih uređaja za izlaz informacija, kao što su zasloni i zvučnici, te njihovu praktičnu primjenu u poljoprivredi
- upoznati se s programskim knjižnicama za interakciju sa ugrađenim zaslonima i zvučnicima u programskom jeziku Python
- implementirati jedan od ponuđenih zadataka koji se koristi ugrađenim senzorima i uređaje za izlaz informacija, te interakciju s njima u programskom jeziku Python.

PowerPoint prezentacija

Snimka 10. radionice

Jedna od novih, snažno rastućih tehnologija je i internet stvari (IoT) koji sadržava mrežnu infrastrukturu u kojoj fizičke i virtualne stvari svih vrsta komuniciraju i nevidljivo su integrirane.

internet stvari

Programski jezik Python omogućuje nam „razgovor", odnosno komunikaciju upravo tih „stvari" koristeći se sveprisutnom mrežom, odnosno internetom. Posebno će biti obrađeni primjeri korištenja programskim jezikom u različitim uređajima i sklopovima u području poljoprivrede i poljoprivredne proizvodnje koji pokazuju obećavajuće rezultate.

doprinos programskog jezika Python IoT-u

Programske knjižnice za pristup ugrađenim senzorima

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 električno mjerljiv signal.

odrednice senzora

Ugrađeni senzori su sljedeći.

ugrađeni senzori i njihova primjena u poljoprivredi

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: **pr.** npr., nadzor nad uzgojem životinja, otkrivanje štetnih događaja poput pada objekata na plastenik/staklenik itd.
kamera	Vizualni je senzor koji uzima 25 – 30 slika u sekundi i prikazuje izlaz ili kao sliku ili kao video. Praktična primjena u poljoprivredi: **pr.** udaljeni nadzor nad nasadima vinograda, maslinicima itd.

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

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

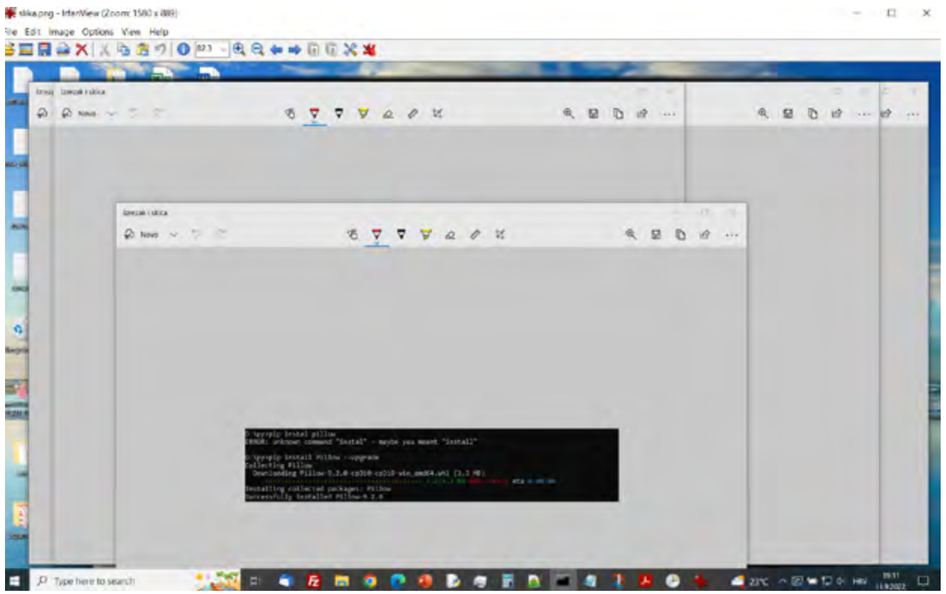
- Programske knjižnice za interakciju s ugrađenim zaslonom

```
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

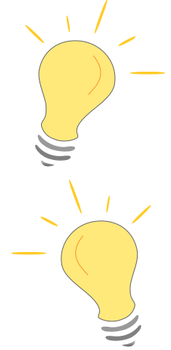
```
>>> winsound.PlaySound("beep.wav", winsound.SND_FILENAME)
```

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 programskog 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 navedenoj 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/

Literatura

https://www.python.org/about/gettingstarted/

Kurikulum nastavnog predmeta Informatike za osnovne i srednje škole, https://narodne-novine.nn.hr/clanci/sluzbeni/2018_03_ 22_ 436.html, MZO, 2018.

Rihter, Rade, Toić Dlačić, Topić, Novaković, Bujadinović, Pandurić, Like IT 5, udžbenik informatike za peti razred osnovne škole, Alfa, 2019.

Gregurić, Hajdinjak, Jakšić, Počuča, Rakić, Svetličić, Šokac, Vlajinić, Informatika 5,udžbenik informatike za peti razred osnovne škole, Profil Klett, 2019.

Drezgić, Pavić, Trucek, #MOJPORTAL5, udžbenik informatike za peti razred osnovne škole, Školska knjiga, 2021.

Rihter, Rade, Toić Dlačić, Topić, Novaković, Bujadinović, Pandurić, Draganjac, Like IT 6, udžbenik informatike za šesti razred osnovne škole, Alfa, 2020.

Gregurić, Hajdinjak, Jakšić, Počuća, Rakić, Svetličić, Šokac, Vlajinić, Informatika 6, udžbenik informatike za šesti razred osnovne škole, Profil Klett, 2020.

Drezgić, Pavić, Trucek, #MOJPORTAL6, udžbenik informatike za šesti razred osnovne škole, Školska knjiga

Python zadatci za vježbanje i ponavljanje Ponavljanje 1 (slidetodoc.com), Carnet

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/