Predgovor

Na početku 21. stoljeća u razvijenijim se državama svijeta zbivaju značajne promjene. Tradicionalna industrijska gospodarstva zasnovana na materijalnim dobrima pretvaraju se u gospodarstva zasnovana na znanju ljudi. Takva gospodarska dinamika prisiljava države na uspostavu sustava obrazovanja koji će u uvjetima svjetske konkurencije omogućiti ljudima dostojan život. Svim svojim građanima država mora stoga osigurati obrazovanje koje će im omogućiti stjecanje kompetencija za složene poslove, za kreativno i inovativno djelovanje, za rješavanje složenih problema s kojima se još nisu susreli, za produktivnu suradnju s drugima i za preuzimanje vodeće uloge kada to zatreba.

Posebnu važnost u obrazovnom procesu ima obrazovanje iz područja matematike, prirodoslovlja, tehnike i informatike koje posljednjih godina nosi zbirni naziv STEM (od engl. *Science*, *Technology*, *Engineering*, *Mathematics*). Dakle pod nazivom informatika uobičajeno se u hrvatskom obrazovnom sustavu podrazumijeva:

- upoznavanje informacijske i komunikacijske tehnologije koja se koristi za uobličavanje multimedijskih sadržaja, njihovo pohranjivanje, pretraživanje i prijenos na udaljena mjesta (*informacijska i komunikacijska tehnologija*, *digitalna pismenost*)
- uporaba računala u obrazovnom procesu (*edukacijska tehnologija*, *e-učenje*)
- rješavanje problema računalom uporabom programskog jezika pri čemu su prepoznatljivi sljedeći koraci: specifikacija i raščlamba problema, analiza problema i odabir postupaka za njegovo rješavanje, priprema i izrada programa, ispitivanje programa i uporaba programa (*rješavanje problema*, *programiranje*).

Informacijsku i komunikacijsku tehnologiju učenici bi trebali upoznavati u kontekstu njezine stvarne primjene u realnim okolnostima. To znači da elemente digitalne pismenosti, osim u predmetu *Informatika*, treba stjecati i produbljivati u predmetima u kojima se digitalne tehnologije rabe. Osim savladavanja tehničkih vještina za uporabu različitih programskih pomagala, učenici moraju usvojiti i odgovarajuća društvena, komunikološka i etička načela uporabe te tehnologije i načine njezine sigurne i neškodljive primjene.

Sve mogućnosti uporabe *edukacijske tehnologije* (*e-učenja*) prvenstveno moraju usvojiti, istraživati i poticati učitelji i nastavnici u svojim obrazovnim područjima. Nužno je da i oni sudjeluju u pripremi digitalnih sadržaja te stvaranju digitalnih repozitorija.

Okosnica predmeta *Informatika* treba biti *rješavanje problema i programiranje* i tom području informatike treba posvetiti posebnu pažnju. Njime se potiče računalni način razmišljanja (engl. *computational thinking*) koje omogućuje razumijevanje, analizu i rješavanje problema odabirom odgovarajućih strategija i programskih rješenja. Takav način razmišljanja nadovezuje se na matematički način razmišljanja (engl. *mathematical thinking*) koji se sustavno mora razvijati u matematici. Preduvjet za uspješno djelovanje u području rješavanja problema programiranjem je ovladavanje nekim od računalnih programskih jezika kojima se osmišljena rješenja mogu relativno jednostavno provjeriti i ispitati.

Ovaj priručnik za učitelje i nastavnike podloga je za izvođenje nastave iz područja rješavanja problema i programiranja. Priručnik pokriva gradivo koje bi učenici trebali savladati od 5. do 8. razreda sadašnje osnovne škole, odnosno u 3. obrazovnom ciklusu eventualne nove organizacije obveznog školovanja. U dokumentu *Nacionalni okvirni kurikulum* (objavljenom u listopadu 2011. godine) u okviru Tehničkog i informatičkog područja predviđena je domena ***Rješavanje problema pomoću računala***, a u dokumentu *Nacionalni kurikulum nastavnog predmeta Informatika* prijedloga cjelovite

kurikularne reforme (objavljenom u lipnju 2016. godine) predviđena je domena ***Računalno razmišljanje i programiranje***. U oba se dokumenta specificiraju obrazovni ishodi i kompetencije koje bi učenici trebali steći u tom obrazovnom razdoblju.

Važno je utvrditi da današnje stanje razvoja programskih radnih okružja omogućuje da se za sve razine obrazovanja (od predškolskog odgoja i obrazovanja pa do završnih razreda srednjih škola) praktična nastava iz domene rješavanja problema i programiranja može obaviti na razumno skromnoj opremi, pri čemu čak i svaki učenik takva radna okružja može imati i na svojoj vlastitoj opremi. Prema tome, računalna oprema nije ograničavajući faktor za inoviranje nastave informatike. Za uspjeh procesa inoviranja nastave informatike mnogo veću, pa i odlučujuću, važnost ima pripremljenost učitelja i nastavnika za uvođenje promjena.

Ovaj bi priručnik trebao poslužiti kao potpora učiteljima i nastavnicima u tom inovacijskom procesu. Njegova svrha je višestruka.

Priručnik je prvenstveno namijenjen učiteljima i nastavnicima ***informatike*** kao potpora za usvajanje osnova računalnog razmišljanja i pripreme programskih rješavanja različitih problema. Nadalje, priručnik je namijenjen i svim učiteljima i nastavnicima predmeta STEM područja, prvenstveno ***matematike***, ***fizike*** i ***tehničke kulture***. Simulacijski računalni postupci i postupci vizualizacije olakšavaju stjecanje novih spoznaja iz tih predmeta. Osim toga, rješavanje problema računalnim razmišljanjem koje se nadograđuje na matematičko razmišljanje dobiva svoj puni smisao u sadržajnom kontekstu cijelog STEM područja.

Gradivo ovog priručnika trebalo bi biti uključeno u osnovno obrazovanje, ne samo učitelja i nastavnika informatike, već i ostalih učitelja i nastavnika STEM područja. Jednako tako, svi učitelji i nastavnici spomenutih predmeta koji se danas nalaze u obrazovnom sustavu morali bi gradivo ovog priručnika svladati u organiziranom postupku stručnog usavršavanja. Takvo je stručno usavršavanje nužni preduvjet za bilo kakve kurikularne zahvate u našem školskom sustavu.

Nadalje, ovaj priručnik može poslužiti kao referentni izvor za oblikovanje nastavnih tekstova namijenjenih učenicima.

Konačno, priručnik može poslužiti i kao podloga za programe cjeloživotnog obrazovanja i to za veliki broj zanimanja u Hrvatskom kvalifikacijskom okviru.

Autori zahvaljuju mnogim kolegicama i kolegama koji su ih poticali na pripremu ovog priručnika. Sadržaj priručnika odredili su i zaključci okruglih stolova koji su održavani tijekom zadnjih nekoliko godina (okrugli stol *Digitalna pismenost i računarstvo u predvisokoškolskom obrazovanju* održan na skupu MIPRO 2014, okrugli stol *Informatika u novom Okviru nacionalnog kurikuluma i kompetitivnost Hrvatske* održan na skupu HDPIO 2014, okrugli stol *Nastava informatike u hrvatskom obrazovnom sustavu* održan na FER-u 2015. godine, okrugli stol pod istim naslovom održan na PMF-u u Splitu 2015. godine). Rasprave i zaključci tih okruglih stolova u velikoj su mjeri odredili sadržaj ovog priručnika. Zbog toga svim sudionicima tih okruglih stolova također upućujemo veliku zahvalu.

Posebno zahvaljujemo recenzentima prof. dr. sc. Marku Rosiću i doc. dr. sc. Leonardu Jelenkoviću. Konačno, zahvala pripada suradnicima izdavačke kuće Element i to posebice gospodinu Edi Kadiću koji je strpljivo uredio vrlo zahtjevni tekst te glavnoj urednici Sandri Gračan.

Autori

Rješenja zadataka za ponavljanje te programski kôdovi zadataka za vježbu dostupni su na: **http://erz.element.hr/**

Zahtjevniji su zadatci u knjizi dodatno označeni ikonom 

O sadržaju priručnika

Za poučavanje programiranja u višim razredima osnovne škole, zadnjih se nekoliko godina prikladnim pokazao programski jezik *Python*. On je posljednjih godina postao i najrašireniji jezik za početno učenje programiranja. Razlozi za to su višestruki:

- *Python* ima jasna pravila oblikovanja te se programi pisani u *Pythonu* lako čitaju i imaju veliku pedagošku vrijednost. Radi se o vrlo intuitivnom jeziku koji kod učenika razvija smisao za sistematičnost i preciznost.
- Programsko okružje potiče pisanje programa s jasno izraženom logičkom strukturom programa.
- Podjela problema na manje dijelove i njihovo međusobno povezivanje u cjelovito rješenje obavlja se vrlo jednostavno. Razrađena rješenja mogu se lako pohraniti i kasnije upotrebljavati pri rješavanju drugih problema.
- Programsko okružje *Pythona* omogućuje jednostavan i brz prijelaz iz faze pisanja u fazu ispitivanja programa i obrnuto.

Okružje za pripremu i izvođenje programa u *Pythonu* pripada skupini besplatnih programa i može se instalirati praktički na sve računalne platforme.

Uz sve spomenute prednosti za učenje programiranja, *Python* se koristi i za profesionalne primjene. Tako je *Python* jedan od triju službenih programskih jezika u tvrtki Google. Youtube je u velikom svom dijelu razvijen u *Pythonu*, a u svemirskoj agenciji NASA mnogi su primjenski programi pripremljeni također u *Pythonu*. Stoga se u ovom priručniku taj jezik koristi za ostvarenje računalnih programa.

Priručnik je podijeljen na četrnaest poglavlja.

U prvom se poglavlju obrazlaže da računalno razmišljanje i programiranje mora biti zasnovano na novim spoznajama kognitivnih znanosti i moderne znanosti o učenju. Utvrđuje se da programiranje podupire konstrukcijski oblik učenja, aktivno učenje i postupnu nadogradnju znanja učenika. Nadalje, ukazuje se na činjenicu da učenje programiranja može imati velikog učinka na izgradnju kognitivnih procesa potrebnih za učinkovito učenje u svim disciplinama, posebice u predmetima STEM područja.

Preostalih trinaest poglavlja mogu se podijeliti u dvije skupine. Prva se skupina sastoji od sedam poglavlja (2. do 8. poglavlje). U tim se poglavljima postupno uvode osnovni elementi programskog jezika koji se koriste kod oblikovanja računalnog programa: programske varijable, funkcije, naredbe pridruživanja, grananja i ponavljanja, tipovi podataka i slično. Objašnjena su i osnovna načela izrade urednih i čitkih programa, kao što je grupiranje naredbi u programske funkcije, izrada modula i komentiranje dijelova programa. Svi programski elementi uvode se kroz jednostavne primjere u trenutku kada oni postanu prikladni za rješavanje postavljenog problema.

Preostalih šest poglavlja (9. do 14. poglavlje) usmjereno je na primjenu programiranja za rješavanje problema iz područja matematike i fizike. Nakon što su usvojeni osnovni programski elementi, kroz složenije se primjere pokazuje kako se njihovom kombinacijom postižu složeni računalni algoritmi te kako analiza rezultata izvođenja tih algoritama povratno pomaže u razumijevanju matematičkih postupaka i fizičkih pojava.

U drugom poglavlju opisuje se radno okružje za pripremu i ispitivanje programa u *Pythonu*. Uvode se pojmovi programske varijable i osnovni tipovi podataka (cijeli brojevi i nizovi znakova) te se opisuje način definiranja programskih funkcija i preporučeni oblik strukture programa.

U trećem poglavlju uvode se osnovne funkcije programskog modula *Turtle* koji sadrži funkcije za crtanje zasnovane na kretanju virtualne kornjače koja je poznata još iz programskog jezika *Logo*. Na primjerima crtanja geometrijskih likova, objašnjena je jezična konstrukcija ponavljanja, odnosno programska petlja i objašnjena su načela pripreme vlastitih funkcija i oblikovanja programa.

Operacije s cijelim brojevima obrađene su u četvrtom poglavlju. U tom se poglavlju opisuje i način pripreme funkcija koje vraćaju rezultate izračunavanja. Obrađene su i funkcije za prevođenje znakovnih nizova u brojeve i brojeva u znakovne nizove te funkcija za generiranje slučajnih brojeva. Ukazuje se na važnost dobrog dokumentiranja pojedinih funkcija.

U petom poglavlju uvode se logičke varijable i osnovni logički operatori koji se koriste za potrebe naredbe grananja i uvjetovano ponavljanje izvođenja dijelova programa. Uz to se uvodi i formatizirano ispisivanje brojeva, teksta i logičkih vrijednosti.

Pri rješavanju različitih problema često je potrebno pohraniti više srodnih ili povezanih podataka. U šestom poglavlju uvode se dva načina pohranjivanja takvih zbirki podataka: liste podataka i *n*-torke podataka. Na nizu primjera pokazuje se način primjene tih dvaju tipova zbirki te funkcije za njihovu obradu. Posebno su obrađeni različiti načini konstruiranja lista.

U sedmom poglavlju pokazan je postupak izrade vlastitog *Python* modula. Na primjerima matematičkih operacija s prirodnim brojevima, pripremljena je zbirka funkcija u obliku vlastitog modula. Na tom je primjeru ilustrirano načelo rješavanja složenih problema kombinacijom rješenja za njegove manje dijelove. Razrađeni su algoritmi za pronalaženje prostih faktora, najvećeg zajedničkog nazivnika i najmanjeg zajedničkog višekratnika. Opisan je i euklidski algoritam i pripremljene su na njemu zasnovane funkcije koje su vremenski nadmoćne funkcijama zasnovanih na rastavljanju brojeva na proste faktore. Na kraju je pripremljen modul koji se sastoji od svih pripremljenih funkcija.

U osmom poglavlju ponovno se koristimo modulom *Turtle*, ali se njegovim funkcijama sada koristimo na napredniji način kojim su ilustrirane raznolike mogućnosti vizualizacije složenih oblika. Detaljno su opisane sve funkcije modula kornjačine grafike. Ilustrirana je i mogućnost ispisivanja tekstova u grafičkom kornjačinom prostoru. Pokazane su i mogućnosti vizualizacije komutativnosti i asocijativnosti zbrajanja, vizualizacije komutativnosti množenja te distributivnosti množenja prema zbrajanju.

Mogućnosti prikaza i provođenja operacija s racionalnim brojevima obrađene su u devetom poglavlju. Detaljno su opisana dva ugrađena modula koje nudi programsko okružje *Pythona*. Modul `fractions` omogućuje prikaz razlomaka i provođenje osnovnih aritmetičkih operacije s razlomcima, dok modul `decimal` pruža mogućnost prikaza decimalnih brojeva i obavljanja operacija s decimalnim brojevima na način kako to činimo na papiru. Prikazana je i pretvorba razlomaka u njihov decimalni prikaz i obrnuto. U tom se poglavlju također opisuje i strojni način pohranjivanja brojeva s pomičnom decimalnom točkom (tipa `float`). Ukazuje se na to da se tim strojnim oblikom zapisa ne postiže apsolutna, već samo približna točnost brojčanih vrijednosti, ali s vrlo malim pogreškama koje se u većini slučajeva mogu potpuno zanemariti. Detaljno je opisan postupak zaokruživanja brojeva na zadani broj decimalnih mjesta i način formatiranja ispisa zaokruženih brojeva.

U desetom poglavlju pripremljen je modul za prikaz dvodimenzionalnog koordinatnog sustava i za crtanje točaka, dužina, pravaca i kružnica u tom koordinatnom sustavu. Sve su funkcije detaljno razrađene i mogu poslužiti kao model za pripremu sličnih funkcija. S druge strane, cijeli se modul može koristiti tako da se njegove funkcije rabe za vizualizaciju rješenja različitih geometrijskih zadataka u ravnini. Na nizu primjera pokazana je uporabljivost pripremljenih funkcija.

Rješavanje linearnih jednadžbi je tema jedanaestom poglavlja. Ukazuje se na to da su u matematici razrađeni postupci za rješavanje problema koji se mogu formalno prikazati jednadžbama. Ti se postupci mogu napisati kao algoritmi i zatim kao funkcije koje će jednostavno vraćati rješenja (ako ona postoje). Prema tome, nakon što smo neki problem zadan riječima opisali jednadžbama, postupak pronalaženja rješenja događa se potpuno automatski. Pripremljene su funkcije za rješavanje dvije jednadžbe s dvjema nepoznanicama i za rješavanje tri jednadžbe s trima nepoznanicama.

U dvanaestom poglavlju obrađuju se postupci za rješavanje problema koji se mogu prikazati proporcijama kao i problemi u kojima se pojavljuju postotci. U poglavlju se također razmatraju koeficijenti proporcionalnosti s mjerom (brzina, potrošnja energije, tečaj) koji se pojavljuju pri modeliranju nekih fizičkih pojava. Prikazana su dva načina vizualizacije postotka te postotnog povećanja i postotnog smanjenja.

Primjena računala za postupke prikupljanja, obrade i analize podataka ima veliku važnost u svim granama ljudske djelatnosti. Metode za provođenje tih postupaka razvijene u matematičkoj statistici dolaze do punog izražaja tek uporabom računala. Posljednjih se godina ta sprega računarstva i matematike počela nazivati znanošću o podatcima (engl. *Data Science*). U trinaestom poglavlju se obrađuju neki postupci obrade i analize podataka i to na razini koja je primjerena elementima matematičke statistike koja se poučava u okviru nastavnog programa matematike za osnovnu školu. U tom se poglavlju opisuje zbirka podataka tipa `dict`, koja je prikladna za pripremu algoritama statističke obrade. Taj tip podataka nazivamo rječnikom (engl. *dictionary*). Isto tako, opisuju se načini trajnog pohranjivanja podataka u znakovne datoteke. Pripremljen je i modul s funkcijama za pohranjivanje lista i rječnika u znakovne datoteke i funkcije za obnovu njihova sadržaja iz znakovnih datoteka. Pripremljene su i funkcije za vizualizaciju podataka.

U završnom četrnaestom poglavlju opisuju se ugrađeni modul `time` čije funkcije služe se mjerenje vremenskih intervala ili za odgađanje izvođenja dijelova programa. Opisane su i funkcije kojima se očitavaju kalendarski podatci i protjecanje vremena unutar jednog dana. Pokazano je kako se funkcije tog modula mogu rabiti za precizno mjerenje trajanja izvođenja programa.

www.element.hr

1. Računalno programiranje kao potpora konstrukcijskom obliku učenja

Suvremena nastava informatike u području rješavanja problema programiranjem mora biti zasnovana na novim spoznajama kognitivnih znanosti i moderne znanosti o učenju. Tradicionalno su se metode poučavanja temeljile na tzv. instrukcijskom obliku učenja koji se primjenjivao u gotovo svim školskim predmetima. Međutim, nove znanstvene spoznaje ukazuju na to da je za predmete STEM područja, a naročito informatike, prikladniji tzv. konstrukcijski oblik učenja.

1.1. Instrukcijski oblik poučavanja i konstrukcijski oblik učenja

1.1.1. Instrukcijski oblik poučavanja

Sve su napredne zemlje tijekom 19. i 20. stoljeća razvile školske sustave koji nude formalno obrazovanje zasnovano na sljedećim općeprihvaćenim pretpostavkama:

- Znanje je zbroj činjenica o svijetu oko nas i **postupaka** za rješavanje problema (postupci se sastoje od razrađenih koraka koje treba sustavno provoditi).
- Učitelji i nastavnici znaju te činjenice i postupke te je njihov posao prenijeti ih učenicima.
- Cilj je školovanja pohranjivanje tih činjenica i postupaka u pamćenje učenika. Ljudi su se smatrali obrazovanima kada su posjedovali veliki broj zapamćenih činjenica i postupaka.
- Učenje započinje savladavanjem jednostavnih činjenica i postupaka i nastavlja se sve složenijim. Svrstavanje činjenica i postupaka po složenosti obavljaju učitelji i autori udžbenika i znanstvenici i to bez punog saznanja o tome kako djeca uče.
- Uspješnost školovanja ocjenjuje se ispitima na kojima se provjerava koliko su činjenica i postupaka učenici usvojili.

Taj tradicionalni sustav obrazovanja može se nazvati **instrukcijskim oblikom obrazovanja** (engl. *instructionism*) ili **instrukcionizmom**. On je pripremljen za razdoblje industrijskog gospodarstva ranog 20. stoljeća, no zadržao se u većini školskih sustava sve do današnjih dana.

1.1.2. Konstrukcijski oblik učenja

Međutim, svijet je danas razvojem znanosti i tehnologije postao mnogo složeniji. Gospodarstva su postala složenija i njihov se razvitak zasniva na stalnim inovacijama. U takvom inovativnom gospodarstvu ne mogu uspješno djelovati pojedinci koji su tijekom školovanja samo zapamtili neke činjenice i naučili provoditi određene postupke. Od njih se zahtijeva mnogo više. Oni moraju steći dublje razumijevanje složenih koncepata, biti ih u stanju strukturno i uzročno-posljedično analizirati te ih kreativno rabiti za generiranje novih ideja, novih artefakata i novog znanja. Oni moraju biti osposobljeni za kritičku ocjenu onoga što čuju i čitaju te za smisleno usmeno i pismeno izražavanje svojih stavova. Posebice, moraju savladati koncepte na kojima je sazdana priroda i tehnika te matematički i računalni način razmišljanja. Formalnim obrazovanjem mora se steći navika i sposobnost preuzimanja odgovornosti za cjeloživotno učenje koje je preduvjet za trajnije uspješno djelovanje u suvremenom gospodarstvu i uspješan suživot u demokratskom društvu te ispunjenje vlastitih životnih očekivanja.

Moderna edukacijska znanost (edukologija) oblikovana je osamdesetih godina prošlog stoljeća i ustoličena 1991. godine na međunarodnoj konferenciji, a tim je povodom počeo izlaziti časopis *Journal of the Learning Sciences* [http://www.tandfonline.com/loi/hlns20]. Ona interdisciplinarno obuhvaća kognitivnu znanost, edukacijsku psihologiju, računarsku znanost, informacijsku znanost, antropologiju, psihologiju, neuroznanost i druga znanstvena područja. Istraživanjima u edukologiji došlo se do spoznaje da instrukcijsko školovanje nije prikladno za novo osposobljavanje učenika[1] i da ga nužno treba nadograditi ili čak potpuno preobličiti u novi **konstrukcijski sustav učenja** (engl. *constructivism*) ili **konstruktivizam**.

Konstrukcijski sustav zasnovan je na sljedećim osnovnim postavkama.

Važnost dubljeg razumijevanja koncepata, duboko učenje

Istraživanja ukazuju da učenje zasnovano na prikupljanju činjenica i postupaka ne priprema osobu za kreativno rješavanja problema. Te su činjenice i postupci korisni samo onda kada ta osoba nauči kako ih primijeniti u određenoj situaciji i kako ih prilagoditi za neku novu situaciju. Rezultati instrukcijskog učenja vrlo su teško primjenljivi izvan razreda. Zbog toga učenici pri savladavanju činjenica i postupaka moraju razviti i dublje razumijevanja njihova konteksta.

Aktivno učenje

Učenici ne mogu steći dublje razumijevanje koncepata samo na temelju poboljšanja instrukcijskog načina poučavanja. Oni moraju aktivno sudjelovati u svom vlastitom učenju. Nova edukologija fokusira se stoga podjednako na unapređenje tehnike poučavanja i na postupak učenja.

1 R. Keith Sawyer, ed., *The Cambridge Handbook of the Learnning Sciencies*, second edition, Cambidge University Press, New York, 2014.

Nadogradnja na prethodno znanje učenika

U trenutku učenja, učenici već imaju neka prethodna znanja o svijetu koji ih okružuje. Neka od tih saznanja su korektna, no neka od njih mogu biti i potpuno pogrešna. Najbolji je način učenja kada se nova saznanja izgrađuju na temelju učenikovih prethodnih znanja. Ako se u tom postupku ne polazi od prethodnih znanja, često se događa da učenici prihvate nove informacije tek toliko da prođu testove znanja i nakon toga se vraćaju na svoje prethodne, ponekad čak i krive, spoznaje.

Promišljanje i utvrđivanje novih spoznaja

Učenici uče bolje kada aktivno razmišljaju o novim znanjima koja upravo savladavaju. To mogu činiti u raspravama s drugim učenicima, pripremom pisanih izvještaja, grafičkih prikaza ili drugih artefakata. Pritom imaju mogućnost analiziranja novih saznanja i ocjene svog vlastitog znanja.

1.2. Računalno programiranje kao potpora konstrukcijskom učenju u području STEM

1.2.1. Uporaba računala u obrazovanju

Uporaba računala u obrazovanju počela je šezdesetih godina 20. stoljeća, odmah nakon što su računala postala dostupna obrazovnim institucijama. Tada su se počeli razvijati programski sustavi za poučavanje (engl. *Computer Assisted Instruction – CAI*), preteče današnjih sustava e-učenja, koji rabe kombinaciju teksta, grafičkih prikaza, zvuka i videozapisa za predstavljanje nastavnog gradiva i interaktivno nadziru tijek učenja.

Devedesetih je godina 20. stoljeća u mnogim zemljama prevladalo mišljenje da škole treba opremiti računalima i omogućiti učenicima slobodan pristup računalnim mrežama. Međutim, pokazalo se da razmjerno velike investicije u opremu nisu urodile plodom. Nije primijećen mjerljivi napredak u učeničkim postignućima. Istraživanja su pokazala[2] da je osnovni razlog tome što su učitelji i nastavnici računala upotrebljavali samo kao dodatno sredstvo u svom instrukcijskom obrazovnom postupku.

Tek se zadnjih desetak godina shvatilo da računala mogu unaprijediti obrazovni proces ako potpomognu kreacijski pristup obrazovanju. Pokazalo se da računala opremljena prikladnom programskom podrškom mogu podržati najvažnije komponente učenja: dublje razumijevanje koncepata, aktivno učenje i utvrđivanje stečenih saznanja.

Računalna snaga današnjih računala s odgovarajućom programskom potporom može uspješno podržati "duboko" učenje. Međutim, potrebno je tu potporu dobro osmisliti i prilagoditi obrazov-

2 L. Cuban, *Oversold & Underused Computer sin the Classroom*, Harvard University Press, Cambrige, MA, 2001.

nom procesu i to u suradnji s učiteljima i nastavnicima. Preobrazba primjene računala za sustave e-učenja te za općenitu djelotvornu uporabu informacijske i komunikacijske tehnologije u procesu učenja mora se prilagoditi specifičnostima pojedinih nastavnih disciplina. To je posebna tema kojom se ovaj priručnik dalje ne bavi, već je usmjeren na mogućnosti uporabe računala za rješavanje problema programiranjem i to posebice u STEM području.

1.2.2. Programski jezici i računalno programiranje

Računala su elektroničke naprave koje mogu obavljati raznovrsne poslove. Ona su revolucionarno promijenila način kako ljudi razmjenjuju informacije, kako rade, kako se druže i kako se zabavljaju i igraju. Računala su omogućila veliki napredak u medicini, gospodarstvu, bankarstvu i drugim djelatnostima. Sav taj napredak omogućen je i razvojem novih "jezika" putem kojih ljudi komuniciraju s računalima – računalnih programskih jezika ili kraće programskim jezicima.

Ti se jezici razlikuju od jezika kojima se ljudi služe u međusobnoj komunikaciji. Programski jezici se sastoje od **skupa naredbi** ili **instrukcija** kojima ljudi određuju koje operacije mora obaviti računalo. Računalo potom slijedi naredbe programskog jezika i izvodi radnje. Računalo se povratno ne obraća ljudima istim takvim jezikom. Odgovarajućim naredbama programskog jezika određuje se kako će izgledati povratna komunikacija računala prema ljudima.

Niz naredbi kojima naređujemo obavljanje nekog složenijeg posla nazivamo **računalnim programom**, a postupak pripreme računalnog programa zovemo **programiranjem**.

1.2.3. Računalno programiranje kao konstrukcijski proces

Edukacijska znanost[3] je utvrdila da učenici uče mnogo učinkovitije ako artikuliraju i eksternaliziraju svoje znanje tijekom savladavanja novih znanja. Artikuliranje i javno izražavanje naučenog idu ruku pod ruku u pozitivnoj povratnoj vezi sa savladavanjem novih znanja.

Tradicionalni instrukcijski pristup u poučavanju programiranja polazio je uobičajeno od opisa svojstava računalnog sustava, upoznavanja internog prikaza načina pohranjivanja osnovnih tipova podataka i detaljnog upoznavanja sintakse programskog jezika. Jednostavni programi kojima se rješavaju neki manji problemi služili su prvenstveno kao ilustracija svojstava pojedinih elemenata programskog jezika.

Iako se taj instrukcijski pristup ne može potpuno izbjeći, moderni programski jezici i njihovo radno okružje omogućuju da svaki učenik postupno izgrađuje vlastiti model računalnog sustava u kojem na konstrukcijskim načelima rješava jednostavnije pa redom sve složenije probleme. Pritom se mentalni model računalnog načina razmišljanja u glavi svakog učenika prilagođuje razini složenosti problema.

Međutim, konstrukcijski pristup učenju ne bismo trebali shvatiti kao proces u kojem svaki učenik sam otkriva svoj vlastiti pogled na svijet. Na temelju artikulacije učenikovih trenutačnih znanja, učitelj mora prepoznati kakvu mu potporu mora pružiti i na koji ga način mora usmjeravati kako bi se postigli određeni obrazovni ciljevi. Djelotvorno radno okružje i interaktivno sučelje

3 *Innovating to Learn, Learning to Innovate*, Centre for Educational Research and Innovation, OECD, 2008.

za pripremu programskih rješenja olakšava ulogu učitelja i nastavnika kao podupiratelja konstrukcije znanja svakog učenika. Pritom je bitno da učenik sam metodom pokušaja i pogrešaka aktivno djeluje u stjecanju novih saznanja. Taj proces učitelj treba samo podupirati – on mora, u skladu s modernom teorijom učenja, postavljati tzv. skelu (engl. *scaffolding*) s pomoću koje učenik sam izgrađuje svoje znanje. Taj se postupak uspoređuje s izgradnjom građevine. Kada radnici dosegnu neku razinu gradnje, izgrađuje se skela koja pomaže izgradnju viših katova građevine. Nakon što je građevina završena, cijela se skela može odstraniti.

1.3. Učenje programiranja potiče razvoj poželjnih načina razmišljanja

Pokazalo se da učenjem programiranja, uporabom prikladnog programskog jezika i radnog okružja za pripremu programskih rješenja učenici razvijaju načine razmišljanja kompatibilne s općim osnovnim postavkama konstrukcijskog modela učenja primjenljivog u ostalim problemskim domenama.

Već prije više od trideset godina pokazalo se da aktivno učenje programiranja potiče[4]:

- temeljito razmišljanje, precizno izražavanje i formalni opis problema jer se programi zasnivaju na dobro razrađenim algoritmima
- razumijevanje osnovnih koncepata kao što su: formalne procedure, funkcije i varijable, koji se temelje na konceptima izrade računalnih programa
- heurističke pristupe rješavanju problema kao što su: izrada plana rješavanja, prepoznavanje sličnosti s nekim već riješenim problemima, dekompozicija složenih problema na manje dijelove
- postupno pronalaženje rješenja problema metodom pokušaja i pogrešaka. U početnim fazama učenja upravo je to jedna od metoda koja se pokazuje učinkovitom jer se postupnim uočavanjem i otklanjanjem pogrešaka napreduje od početne ideje i njene realizacije do zadovoljavajućeg rješenja. U kasnijoj "zrelijoj" fazi rješavanja problema ova metoda će imati sve manji udio
- pronalaženje rješenja problema na način da se međusobno povezuju prethodno razrađeni i provjereni manji dijelovi rješenja tj. realizacija rješenja na principu povezivanja već provjerenih funkcija
- prepoznavanje mogućnosti da se neki problem može riješiti na više načina te da se odabir najboljeg rješenja može obaviti temeljem odgovarajućih kriterija usporedbe u odnosu na zauzeće memorije, duljinu trajanja izvođenja itd.

U skladu s tim može se zaključiti da učenje programiranja može imati veliki učinak na izgradnju kognitivnih procesa potrebnih za učinkovito učenje u svim disciplinama, posebice u predmetima STEM područja. U dobro osmišljenom edukacijskom procesu, kognitivna aktivnost rješavanja problema programiranjem i aktivnost usvajanja novih spoznaja u ciljnoj disciplini (matematika, fizika, kemija itd.) međusobno se skladno nadopunjuju. Primjerice, da bi učenik mogao izraditi računalni program koji rješava postavljeni zadatak iz matematike ili fizike, on mora biti svjestan postupka za rješavanje tog zadatka koji prethodno treba naučiti u ciljnoj disciplini.

4 Roy D. Pea, D. Midian Kurland. *On the cognitive effects of learning computer programming*. New Ideas in Psychology, Elsevier, 1984, 2(2), pp.137-168

Nakon toga svoje znanje temeljeno na pojedinoj disciplini koristit će pri izradi računalnog programa koji će postupak rješavanja zadanog problema ponavljati na većem broju različitih ulaznih podataka. Pri tome će učenik uočiti da se postupak izračuna za različite ulazne podatke izvodi brže i bez pogrešaka u odnosu na jednake postupke koji se mogu izvoditi primjenom klasičnih metoda izračuna.

Ta činjenica učenicima otvara dodatni prostor za eksperimentiranje s različitim vrijednostima ulaznih podataka, praćenje i analizu ovisnosti rezultata o promjenama ulaznih podataka, rubnim uvjetima, graničnim vrijednostima i slično. U konačnici učenici samostalno otkrivaju zanimljive pojave do kojih bi samostalno teško mogli doći ručnim rješavanjem istog problema na malom broju primjera. Otvaraju im se nova pitanja i novi pogledi na problematiku ciljne discipline, a njihovo razjašnjavanje kroz nastavu ciljnog predmeta dovodi do dubljeg razumijevanja tematike.

2. Upoznavanje programskog jezika *Python*

U ovom ćemo poglavlju upoznati:

- osnovna svojstva programskog jezika *Python*
- rad u interaktivnom sučelju *Pythona*
- uporabu interaktivnog sučelja kao kalkulatora
- izračunavanje aritmetičkih izraza
- način pohranjivanja podataka u pretince spremnika i programske varijable
- pojam programske naredbe
- naredbu pridruživanja `"="`
- tipove podataka `int` za cijele brojeve i `str` za nizove znakova
- metodu `format()`
- programske funkcije **`int`**`()` i **`str`**`()`
- programske funkcije **`input`**`()` i **`print`**`()`
- način pisanja programa i njegovo pohranjivanje u datoteku
- postupak pokretanja pohranjenih programa
- preporučenu strukturu programa.

2.1. Upoznavanje s radnim okružjem *Pythona*

Prilikom uključivanja računala pokrenut će se operacijski sustav – glavni program za nadzor i obavljanje svih poslova u računalu. Na zaslonu računala pojavit će se slika koju zovemo **grafičkim korisničkim sučeljem** operacijskog sustava (engl. *Graphical User Interface – GUI*). Posredstvom tog sučelja možemo pokrenuti ostale programe koji su pohranjeni u datotekama na čvrstom disku računala. To uobičajeno činimo dvostrukim klikom miša na imena tih datoteka ili na sličice (ikone) koje ih **simboliziraju**. Pokretanjem nekog programa najčešće se otvara posebni **prozor** s grafičkim sučeljem koji korisniku omogućuje rad s programom (pisanje teksta, crtanje crteža, pokretanje filmova, igranje igara i sl.).

Prilikom pripreme programa u *Pythonu* i prilikom njihova izvođenja susrest ćemo se s tri vrste takvih prozora:

- Prozor **interaktivnog sučelja** poslužit će nam za prikaz podataka koje tipkovnicom unosimo u računalo i prikaz izlaznih podataka iz računala – rezultata. Interaktivno sučelje služit će nam i za izvođenje pojedinačnih naredbi.
- Prozor za pisanje programa po svojoj ulozi podsjeća na prozor bilo kojeg programa za pisanje i uređivanje (editiranje) tekstova tako da se naziva i **editorom**. Sadržaj tog prozora moći ćemo pohraniti kao datoteku i trajno sačuvati na tvrdom disku računala.
- Prozor za slikovni (grafički) prikaz kraće nazivamo **grafički prozor**.

Osnove uporabe prvih dviju vrsta prozora upoznat ćemo u ovom poglavlju, a prozor za slikovni prikaz u 3. poglavlju.

2.2. Interaktivno sučelje *Pythona*

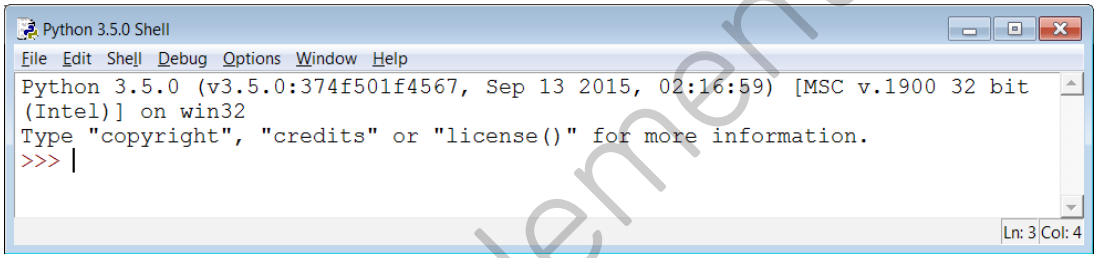
2.2.1. Instalacija interaktivnog sučelja

Ako program nije instaliran na računalu, datoteke za instalaciju moguće je preuzeti s *web*-adrese https://www.python.org/downloads/ i slijediti upute za instalaciju. Prilikom preuzimanja instalacijskih datoteka preporučamo instalirati najnoviju inačicu *Pythona* označenu s *Python 3.x.y*, umjesto starije inačice koja je označena s *Python 2.x.y*.

2.2.2. Pokretanje interaktivnog sučelja

Prilikom instaliranja programskog paketa za jezik *Python* sve njegove programske datoteke i programska pomagala bit će smješteni u mapu `Python 3.x.y` do koje možemo doći preko izbornika programa `All programs`. Klikom miša na `Python 3.x.y` pokazat će se podmapa, a unutar nje datoteka `IDLE(Python GUI)`. Klikom miša na ime te datoteke pojavit će se na zaslonu

računala prozor koji čini interaktivno sučelje s *Pythonom*. Naslov tog prozora je `Python 3.x.y Shell`, a njegov izgled prikazuje slika 2.1. U naslovu prozora može se pojaviti i naziv `Python 3.5.0 Shell` ili neki drugi, što ovisi o inačici programa koja je instalirana na našem računalu.



Slika 2.1. *Interaktivno sučelje Pythona*

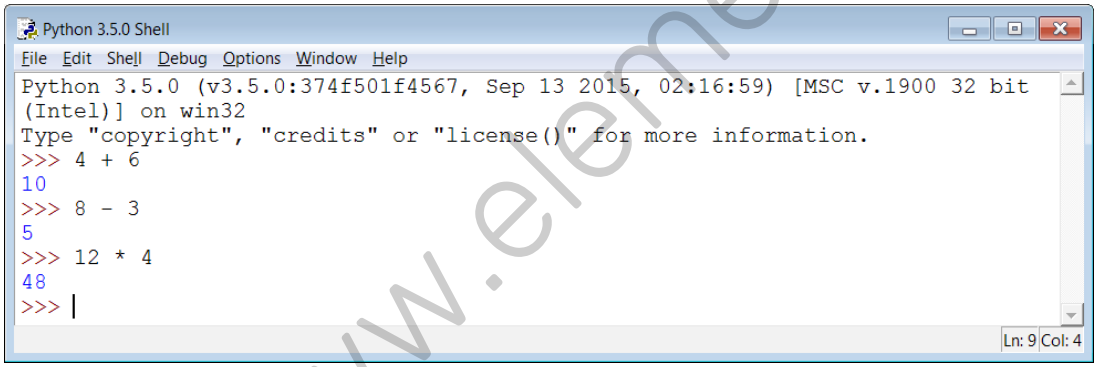
U prvim redovima tog prozora vidimo osnovne podatke o inačici programa koja je instalirana na našem računalu. Nakon toga vidimo red koji na početku ima tri znaka **>>>** i treptajuću oznaku "|". To je oznaka koja pokazuje da je program spreman prihvatiti znakove koje ćemo utipkavati s tipkovnice. Uobičajeni naziv za oznaku **>>>** je *prompt.* Naziv *prompt* preuzet je iz engleskoga jezika.

Taj će nam prozor biti osnovna veza s računalom prilikom pripremanja i izvođenja programa. Posredstvom tog prozora pokretat ćemo prozor za pisanje programa, preko njega ćemo unositi ulazne podatke u program i u njemu će programi ispisivati rezultate svojeg izvođenja.

Dakle, posredstvom tog prozora mi prenosimo sadržaje računalu i računalo ispisuje povratne informacije. Kažemo da smo preko tog prozora u stalnoj interakciji s računalom pa ga zbog toga i zovemo **interaktivnim sučeljem**.

2.2.3. Interaktivno sučelje kao kalkulator

Interaktivni prozor, slika 2.2, može poslužiti kao veliki kalkulator kao što pokazuju sljedeći primjeri.



Slika 2.2. *Interaktivno sučelje Pythona kao kalkulator*

U daljnjim primjerima nećemo više rabiti slike cijelog prozora već ćemo samo prikazivati sadržaj prozora.

Pogledajmo kako uporaba interaktivnog sučelja može biti korisna. Provjerimo pravila izračunavanja aritmetičkih izraza:

```
>>> 3 + 8                                    #1
11
>>> 8 + 3                                    #2
11
>>> 3 + (8 + 5)                              #3
16
>>> (3 + 8) + 5                              #4
16
>>> 3 * (8 + 5)                              #5
39
>>> 3 * 8 + 3 * 5                            #6
39
```

U ovim se primjerima pojavljuje znak **#** kojim se u *Pythonu* koristimo za pisanje komentara, odnosno bilješki koje pojašnjavaju ulogu pojedinih naredbi programa. Sve što piše iza tog znaka do kraja reda *Pythonov* prevoditelj zanemaruje, odnosno taj dio teksta nema utjecaja na funkcionalnost programa. Tako možemo objasniti da redovi (**#1**) i (**#2**) pokazuju da pri zbrajanju vrijedi zakon komutativnosti pribrojnika te da redovi (**#3**) i (**#4**) ilustriraju zakon asocijativnosti zbrajanja, a redovi (**#5**) i (**#6**) zakon distributivnosti množenja prema zbrajanju.

Prilikom provjere primjera u interaktivnom sučelju nije potrebno upisivati oznake komentara i same komentare.

U *Pythonu* se aritmetički izrazi pišu po pravilima koja su vrlo slična onima koje poznajemo iz matematike. **Operatori** (simboli matematičkih operacija) za zbrajanje (+) i oduzimanje (–) jednaki su onima u matematici. Operator za množenje je u *Pythonu* zvjezdica (*) dok u matematici rabimo točku na sredini.

Za dijeljenje prirodnih brojeva u *Pythonu* postoje dva operatora. To su:

- operator za računanje količnika `//` (uzastopno napišemo znak `/`)
- operator za računanje ostatka pri dijeljenju `%`.

Djelovanje tih dvaju operatora možemo pogledati rješavanjem sljedećeg primjera.

Primjer 2.1.

Maja ima 16 bombona koje želi podijeliti svojim prijateljima tako da svaki od njih pet dobije jednak broj bombona. Maji je naravno stalo da njeni prijatelji dobiju najviše bombona što mogu dobiti, ali tako da svi dobiju jednako. Ako ostane bombona Maja će biti vesela da ih i ona dobije.

Rješenje:

Uočimo da imamo 16 bombona i 5 prijatelja. Kako bismo riješili ovaj problem važno je vidjeti koliko će najviše bombona dobiti svaki od prijatelja te koliko će bombona ostati Maji. Ako cjelobrojno podijelimo broj bombona s brojem prijatelja, dobit ćemo broj bombona

koji će dobiti svaki Majin prijatelj. Ako želimo izračunati koliko će bombona ostati Maji, potrebno je vidjeti koliki je ostatak cjelobrojnog dijeljenja broja bombona s brojem prijatelja.

To možemo izračunati sljedećim nizom naredbi:

```
>>> 16 // 5          #cjelobrojno dijelimo broj bombona s brojem prijatelja
3                    #svaki prijatelj će dobiti 3 bombona
>>> 16 % 5           #ostatak pri dijeljenju daje koliko bombona ostaje Maji
1                    #Maji će ostati 1 bombon
>>> 3 * 5 + 1        #provjera algoritma
16
```

Promotrimo djelovanje navedenih operatora na još jednom primjeru:

```
>>> 9 // 9                #cjelobrojno podijeljeno s 9 je 1
1
>>> 9 % 9                 #ostatak cjelobrojnog dijeljenja 9 s 9 je 0
0
>>> 9 * 1 + 0             #provjera prethodnog izračuna
9
```

2.2.4. Imena pretinaca, programske varijable

U većini slučajeva korisno bi bilo brojeve s kojima računamo, a i same rezultate računanja pohraniti u spremnik računala što bi nam omogućilo njihovo ponovno korištenje. Pritom svaka vrijednost koju trebamo pohraniti mora dobiti svoj pretinac u memoriji računala. Pretince ćemo imenovati tako da ih možemo razlikovati, primjerice: `pretinac_a`, `pretinac_b`, `pretinac_c`.

U pretinac ćemo pohraniti sadržaj (vrijednost) **naredbom pridruživanja**. U toj naredbi s lijeve strane znaka pridruživanja (=) treba napisati naziv pretinca, a s desne strane znaka pridruživanja vrijednost koju se u pretinac želi pohraniti. Pogledajmo primjer:

```
>>> pretinac_a = 3                                    #1
>>> pretinac_b = 8                                    #2
>>> pretinac_c = pretinac_a + pretinac_b              #3
>>> pretinac_a                                        #4
3
>>> pretinac_b                                        #5
8
>>> pretinac_c                                        #6
11
```

Naredbama pridruživanja (#1) i (#2) pohranili smo vrijednosti 3, odnosno 8, u pretince `pretinac_a` i `pretinac_b`. U retku s oznakom (#3) naznačeno je da se vrijednosti pretinaca `pretinac_a` i `pretinac_b` zbroje te tako dobiveni rezultat pohrani u pretinac `pretinac_c`.

Sadržaj pojedinih pretinaca može se pogledati tako da se napiše ime pretinca i nakon toga pritisne tipka *(Unos)*. Izvođenjem naredbi (#4), (#5) i (#6) vidimo sadržaje svih triju pretinaca.

Imena pretinaca moraju se pisati u skladu s određenim pravilima. Imena se mogu sastojati od proizvoljnog broja znakova koji mogu biti:

- velika i mala slova (možemo se koristiti i slovima hrvatske abecede s dijakritičkim znakovima: č, ć, đ, š i ž)
- znamenke dekadskog brojevnog sustava (0, 1, 2, 3, 4, 5, 6, 7, 8 i 9)
- znak donje crtice (_)

s time da ime ne smije početi znamenkom. Kod korištenja slova hrvatske abecede s dijakritičkim znakovima mogu se pojaviti neki problemi ako program izvodimo na različitim računalima.

Pogledajmo neke primjere:

```
jedan2tri          #ispravno ime
jedan_2_tri        #ispravno ime
jedan-2-tri        #neispravno ime (crtica nije dozvoljeni znak)
1dvatri            #neispravno ime (prvi znak mora biti slovo)
a_0                #ispravno ime
a.1                #neispravno ime (točka nije dozvoljeni znak)
a 3                #neispravno ime (praznina nije dozvoljeni znak)
a3                 #ispravno ime
```

Za pisanje imena u pravilu se nećemo koristiti velikim slovima. Velikim ćemo se slovima koristiti u posebnim slučajevima. O tim će slučajevima biti riječi u narednim poglavljima.

U *Pythonu* postoje tzv. rezervirane riječi koje imaju posebna značenja i ne smiju se rabiti kao imena.

Imena pretinaca treba birati tako da ne budu predugačka, ali da iz njih možemo zaključiti što znače vrijednosti na koje pokazuju. U našem prethodnom primjeru umjesto dugačkih imena možemo jednostavno pisati:

```
>>> a = 3                                                    #7
>>> b = 8                                                    #8
>>> c = a + b                                                #9
>>> c                                                        #10
11
```

U ovom slučaju dovoljna su nam jednostavna imena (#7), (#8) i (#9) jer vrijednostima nismo pripisali neka posebna značenja. Sljedeći primjer pokazuje da je korisno razlikovati imena po njihovu značenju.

Primjer 2.2.

Ana, Marko i Pero sakupljaju novac za kupovinu lopte čija je cijena 47 kuna. Ana može dati 13 kuna, a Marko 21 kunu. Koliko bi kuna morao dati Pero?

Rješenje:

U pretinac `lopta` *pohranit ćemo cijenu lopte (#11). U pretince* `ana` *i* `marko` *pohranit ćemo iznose s kojima raspolažu Ana i Marko (#12) i (#13). Iznos koji će trebati dodati Pero jednak je cijeni lopte umanjenoj za iznos koji će zajedno dati Ana i Marko (#14).*

Primjer **2.2.**

U interaktivnom bi sučelju to izgledalo ovako:

```
>>> lopta = 47                                    #11
>>> ana = 13                                      #12
>>> marko = 21                                    #13
>>> pero = lopta - (ana + marko)                  #14
>>> pero
13
```

Jednostavnija imena olakšala bi nam pisanje. Pogledajmo:

```
>>> a = 47
>>> b = 13
>>> c = 21
>>> d = a - (b + c)
>>> d
13
```

No, iz ovog niza naredbi kasnije ćemo teže razumjeti značenja tih imena, kao i ulogu cjelokupnog programa koji se sastoji od pet naredbi. Razumljivija imena naročito će nam biti važna kada ćemo u našim izračunima činiti neke promjene.

Primjer **2.3.**

Pretpostavimo da su Ana, Marko i Pero odlučili kupiti skuplju loptu koja stoji 51 kunu. Ana i Marko imaju jednake iznose kao i u primjeru 2.1.

Rješenje:

Pokazat ćemo kako u interaktivnom sučelju možemo jednostavno promijeniti naredbe koje su ranije napisane (za rješavanje primjera 2.1.).

Za rješavanje novog problema trebamo promijeniti samo cijenu lopte. Kako ne bismo prepisivali već napisane naredbe, strelicama ćemo se pomaknuti u redak koji želimo kopirati (`lopta = 47`) i pritisnuti (Unos) (#11). Naredba će se kopirati na kraj našeg bloka naredbi. U kopiranoj naredbi moguće je učiniti i potrebne izmjene. U našem ćemo slučaju umjesto 47 upisati 51 i pritisnuti (Unos) (#15).

```
>>> lopta = 47                          #15 – umjesto 47 upisat ćemo 51
```

Kako bismo dobili ponovni izračun, potrebno je ponovno upisati (kopirati) i naredbu u kojoj imamo izračun koliko Pero treba dati novaca za kupnju lopte (#14). To ćemo napraviti na jednaki način kao i za kopiranje naredbe lopta. U ovom primjeru ostale naredbe nije potrebno kopirati. Kopiranjem retka u kojem je navedeno ime pretinca `pero` i pritiskom na tipku (Unos) u interaktivnom će nam se sučelju ispisati pohranjena vrijednost, odnosno traženi rezultat.

```
>>> lopta = 51
>>> pero = lopta - (ana + marko)
>>> pero
17
```

Postupak kopiranja i korigiranja pojedinih naredbi u interaktivnom sučelju često će nam olakšati posao.

U programskim jezicima, pa tako i u *Pythonu*, za pretince se rabi naziv **varijabla**. Tako su `lopta`, `ana`, `marko` i `pero` imena varijabli.

Danas se u programskim jezicima često pojmom varijabla koristimo i za vrijednosti koje se tijekom izvođenja programa mijenjaju, ali i za one koje se tijekom izvođenja programa ne mijenjaju, odnosno imaju stalnu ili konstantnu vrijednost. Tako se i u *Pythonu* naziv varijabla rabi i za promjenljive i konstantne vrijednosti. Vidimo da u primjerima 2.1 i 2.2 varijable `ana` i `marko` ne mijenjaju vrijednosti (možemo reći da su to konstante), dok je vrijednost varijable `pero` promjenljiva i ovisi o promjenljivoj vrijednosti varijable `lopta`.

Odsad pa nadalje ćemo pretince nazivati *varijablama* kao što je to i uobičajeno u svim programskim jezicima.

2.2.5. Naredba pridruživanja

U programskom jeziku *Python* se znakom "=" označava pridruživanje vrijednosti s desne strane znaka "=" varijabli koja se nalazi s lijeve strane znaka "=". Ovdje treba napomenuti da znak "="nema isto značenje kao znak "=" u matematici. Uočimo da u tom smislu naredbu `c = a + b` tumačimo: prvo se određuje vrijednost izraza koji stoji desno od znaka pridruživanja "=" te se ta vrijednost pridružuje varijabli `c` čije ime stoji lijevo od znaka pridruživanja.

Primjer 2.4.

U programiranju ćemo vrlo često trebati prebrojiti koliko se puta ponovio neki događaj. Kako bismo riješili taj problem, potrebno je napisati niz naredbi. Pogledajmo primjer:

```
>>> brojilo = 0                          #1
>>> brojilo = brojilo + 1                #2
>>> brojilo
1
>>> brojilo = brojilo + 1                #3
>>> brojilo
2
>>> brojilo += 1                         #4
>>> brojilo += 1                         #5
>>> brojilo
4
```

Ako želimo prebrojavati, zbrajati ili množiti određene vrijednosti, važno je, prije samog postupka, varijabli u koju spremamo rezultat pridružiti početnu vrijednost. U pravilu je to neutralni element za pojedinu operaciju, za zbrajanje to je `0`, a za množenje `1`. No, ovisno o zahtjevima zadatka možemo pridružiti i neku drugu početnu vrijednost.

Opišimo naredbe u primjeru 2.4.

#1	Varijabli `brojilo` pridružujemo nulu.

#2	Postojeću vrijednost varijable `brojilo` uvećavamo za jedan te dobivenu vrijednost ponovno pohranjujemo u varijablu `brojilo`. Vidimo da je sada ta vrijednost jednaka `1`.
#3	U varijabli `brojilo` pohranit će se vrijednost `2`. Varijabla se ponaša kao brojilo pa smo je zato tako i nazvali.

Često u programima treba povećavati sadržaj neke varijable za određenu vrijednost, u našem slučaju za 1, pa je u tu svrhu uveden i posebni znak pridruživanja kao što se vidi u naredbama (#4) i (#5). Te naredbe imaju jednako djelovanje kao i duže naredbe (#2) i (#3) i tumačimo ih ovako: povećaj sadržaj varijable za jedan.

Naredba pridruživanja može se primijeniti tako da se ista vrijednost istovremeno pohrani u više varijabli pa se može pisati:

```
>>> a = b = c = 1                                    #6
>>> a, b, c                                          #7
(1, 1, 1)
>>> a += 1                                           #8
>>> b += 2                                           #9
>>> c += 3                                           #10
>>> a                                                #11
2
>>> b                                                #12
3
>>> c                                                #13
4
>>> a, b, c                                          #14
(2, 3, 4)
```

Opišimo prethodne naredbe:

#6	Varijablama `a`, `b` i `c` istovremeno se pridružuje vrijednost `1`.
#7	Istovremeno ispisujemo sadržaje varijabli `a`, `b` i `c` koji će biti prikazan kao *trojka* vrijednosti napisana unutar okruglih zagrada. Iako su vrije-dnosti varijabli bile određene jednom naredbom, one se nakon toga mogu ne-zavisno mijenjati.
#8, #9 i #10	Vrijednosti varijabli uvećavaju se za različite vrijednosti.
#11, #12 i #13	Provjeravamo vrijednosti varijabli `a`, `b`, `c`.
#14	Istovremeno ispisujemo vrijednosti varijabli kao trojku.