Poučavanje programiranja

Tucak-Roguljić, Ivana

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Applied Mathematics and Informatics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet primijenjene matematike i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:586997

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni diplomski nastavnički studij matematike i informatike

Ivana Tucak-Roguljić

Poučavanje programiranja

Diplomski rad

Osijek, 2023.

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni diplomski nastavnički studij matematike i informatike

Ivana Tucak-Roguljić

Poučavanje programiranja

Diplomski rad

Mentorica: izv. prof. dr. sc. Darija Marković

Osijek, 2023.

Sadržaj

Uvod

U današnjem sve digitaliziranijem svijetu, programiranje je postalo ključna vještina koja omogućava razumijevanje i interakciju s tehnologijom koja nas okružuje. Stoga, sve veći naglasak se stavlja na podučavanje programiranja u obrazovnom sustavu kako bismo pripremili nove generacije učenika za izazove i mogućnosti koje donosi digitalno doba.

Programiranje više nije rezervirano samo za one koji žele postati programeri, već je postalo umijeće koju bi trebali usvojiti svi učenici. Učenje programiranja ima veliki značaj jer, uz to što nam omogućuje stvaranje tehnoloških rješenja, razvija i važnu kognitivnu sposobnost - računalno razmišljanje.

Prvi dio rada ističe razliku između programiranja i računalnog razmišljanja. Detaljno objašnjava kako je računalno razmišljanje nastalo, te povezuje njegov razvoj s drugim oblicima razmišljanja. Kako bi uspjeli razviti računalno razmišljanje kod učenika, ključno je da učenici steknu ispravan mentalni model za računalno razmišljanje, stoga se u radu opisuje jedan od tih modela koji je podijeljen na 6 aspekata. Uz to, u prvom dijelu rada su prikazani projekt CS Unplugged te natjecanja Klokan i Dabar, kao načini kojima se može potaknuti razvoj računalnog razmišljanja kod učenika, pri čemu se ne koriste standardni programerski zadaci.

U drugom dijelu rada fokus je na programskim jezicima za poučavanje, koji se mogu podijeliti u tri kategorije: vizualno-tekstualni, vizualno-blokovski i proceduralno-tekstualni programski jezici. Opisan je programski jezik Logo kao primjer vizualno-tekstualnog jezika. Također, kao predstavnik vizualno-blokovskog jezika, pojašnjen je Scratch, dok je za primjer proceduralno-tekstualni jezik obrađen Python. Za svaki od navedenih programskih jezika dan je primjer zadatka koji se pojavljuju u udžbeniku iz informatike ili natjecanju iz programiranja. Nakon toga, u radu su izneseni kriteriji koje možemo koristiti za procjenu prikladnosti programskog jezika za poučavanje. Pomoću tih kriterija uspoređena su dva konkretna programskog jezika - Python i Logo - s obzirom na njihovu primjerenost za poučavanje.

Posljednji dio ovog rada usredotočen je na miskoncepcije - to jest, pogrešne predodžbe određenih koncepata koje često proizlaze iz netočnih ili nepotpunih informacija. Miskoncepcije su prisutne u različitim područjima znanja, uključujući i programiranje. U radu su predstavljene miskoncepcije koje često imaju učenici pri njihovom prvom susretu s programiranjem.

1. Računalno razmišljanje

1.1 Što je računalno razmišljanje?

Programiranje često smatramo izazovnim za naučiti jer zahtijeva kombinaciju različitih vještina. Kako bismo uspješno programirali, potrebno je razviti sposobnost čitanja i pisanja u određenom programskom jeziku. Međutim, programiranje nije samo mehaničko tipkanje linija koda. Ono zahtijeva čovjekovu sposobnost formuliranja i rješavanja problema na način kako to čine računala, poznatije kao računalno razmišljanje (prema [18]). Kroz računalno razmišljanje, možemo efikasno pretvoriti realne probleme u formalne probleme koje uz pomoć računala znamo riješiti. Dakle, kako bismo uspješno programirali, trebamo kombinirati znanja određenog programskog jezika i računalnog razmišljanja (više u [3]).

Važno je razlikovati računalno razmišljanje od programiranja. To nisu isti pojmovi, ali su snažno povezani. Programiranje može pomoći u poticanju računalnog razmišljanje, ali se računalno razmišljanje također može primijeniti na različite vrste problema koji ne uključuju izravno programske zadatke (vidi [16]). Stoga, iako programiranje može biti koristan alat za razvoj računalnog razmišljanja, važno je shvatiti da računalno razmišljanje ima širu primjenu i može biti korisno u rješavanju različitih vrsta problema.

Ideju korištenja računalnog razmišljanja za rješavanje problema nevezanih za programiranje imao je i Seymour Papert[1], začetnik ideje o računalnom razmišljanju. U svojoj knjizi *Mindstorms: children, computers, and powerful ideas* iz 1980. godine istaknuo je vrijednost programiranja kao alata za razvoj vještina kod učenika koje će im koristiti i izvan nastave informatike (prema [3]).

Kao matematičar, Papert je bio osobito zainteresiran i za nastavu matematike te okruženja za učenje koji bi učenici mogli koristiti za istraživanje apstraktnih matematičkih koncepata na konkretniji način. Ova ga je ideja dovela do razvoja programskog jezika Logo. Papert je vjerovao da će učenici koji programiraju u Logu bolje razumjeti matematičke pojmove kroz proces otklanjanja pogrešaka u kodu [7]. Programski jezik Logo pokazao se kao koristan alat za poučavanje programiranja, te ga neki učitelji koriste i danas u svojoj nastavi. Logo je detaljnije opisan u 2. poglavlju.

Papert je uočio kako učenici matematičko razmišljanje mogu razvijati pomoću računalnog razmišljanja. Veza između njih je obostrana, tj. matematičko razmišljanje može pomoći u razvijanju računalnog razmišljanja. Ova tema bila je zanimljiva mnogim znanstvenicima, pa postoje istraživanja koja se bave proučavanjem sličnosti i razlika između računalnog i matematičkog razmišljanja. Navest ćemo zajedničke koncepte koje imaju matematičko i računalno razmišljanje prema [16]:

1. rješavanje problema
2. modeliranje
3. analiza i interpretacija podataka
4. statistika i vjerojatnost

Također, postoje istraživanja koja se bave povezanosti računalnog razmišljanja s drugim načinima razmišljanja kao što su inženjerski ili dizajnerski oblik razmišljanja [16].

[1] Seymour Aubrey Papert (29. veljače 1928. - 31. srpnja 2016.) bio je američki matematičar, informatičar i pedagog koji je većinu svoje karijere proveo predajući i istražujući na MIT-u.

1.2 Aspekti računalnog razmišljanja

Kako bi učenici uspješno savladali sposobnost računalnog razmišljanja, ključno je razvijanje pravilnog mentalnog modela tog koncepta. Mentalnim modelom raščlanjujemo računalno razmišljanje na koncepcijski bitne sastavnice, tj. aspekte. Aspekti računalnog razmišljanja nisu točno definirani, nego različiti autori nude različite modele (vidi [18]). Ipak postoje aspekti koji se pojavljuju u više modela računalnog razmišljanja, a to su: apstrakcija, dekompozicija, algoritmi i otklanjanje pogrešaka. Kako bismo bolje razumjeli navedene aspekte, u ovome radu ćemo prikazati jedan od modela računalnog razmišljanja preuzet iz [16].

Prema njemu, model računalnog razmišljanja obuhvaća 6 glavnih aspekata:

- dekompozicija - rastaviti složen problem na manje, lakše rješive dijelove. Podjela ne treba biti nasumična, nego funkcionalna i tako da svi dijelovi zajedno čine cjelinu.
- apstrakcija - pronaći bit problema. Ovu apstrakciju možemo podijeliti na 3 potkategorije:
  - prikupljanje i analiza podataka: prikupiti najbitnije informacije iz više izvora i razumjeti odnos među njima
  - prepoznavanje uzoraka: identificirati pravilnost ili ponavljanje u strukturi podataka, tj. informacijama
  - modeliranje: izraditi model ili simulaciju kako bi predstavili funkcioniranje sustava
- algoritmi - dizajniranje logičkih uputa za rješenja problema. Postoje četiri potkategorije:
  - dizajn algoritma: kreiranje niza poredanih koraka za rješavanje problema
  - paralelnost: izvođenje više koraka u isto vrijeme
  - učinkovitost: dizajniranje najmanjeg broja koraka za rješavanje problema, tj. uklanjanje suvišnih i nepotrebnih koraka
  - automatiziranje: automatiziranje rješenja za slične probleme
- otklanjanje pogrešaka - pronaći i popraviti pogreške u slučaju da rješenje problema ne funkcionira
- iteracija - ponavljanje procesa dok se ne postigne zadovoljavajuće rješenje
- generalizacija - prenošenje sposobnosti računalnog razmišljanja na različite situacije u cilju efikasnog rješavanja problema

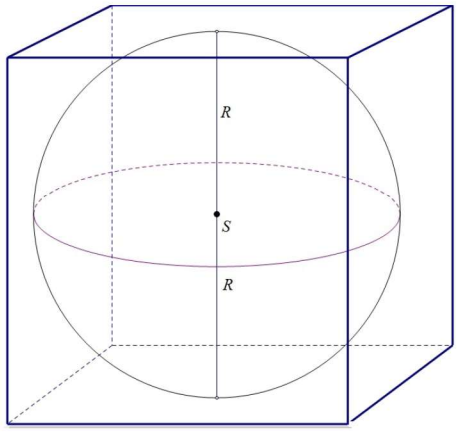
Prikažimo sada na primjeru kako problem iz svakodnevnog života možemo pokušati riješiti pomoću aspekata računalnog razmišljanja. Primjer je preuzet iz [17] gdje su autori provodili istraživanje o računalnom razmišljanju na učenicima srednjih škola. Njihov model aspekata računalnog razmišljanja razlikuje se od prethodno navedenih aspekata po tome što ne sadrži aspekte iteracije i generalizacije. Dakle, u ovom modelu imat ćemo aspekte: dekompozicija, apstrakcija, dizajn algoritma i otklanjanje pogrešaka.

Primjer 1.1. *Pakiranje kutija za selidbu*

Osoba želi pospremiti sve predmete iz ormara i polica u kutije. Kako bi selidba bila što jeftinija, treba smjestiti sve predmete u što manji broj jednakih kutija.

Mogući odgovor učenika:

- *dekompozicija - kutije su jednake veličine i oblika, a predmeti su različitog oblika i veličine*
- *apstrakcija*
  - *prikupljanje i analiza podataka - predmeti mogu biti različitih oblika*
  - *prepoznavanje uzoraka - predmete možemo promatrati kao geometrijska tijela*
  - *modeliranje - volumen geometrijskog tijela kojim je opisan predmet pretvaramo u kocku ili kvadar kako bi vidjeli koliko prostora u kutiji će zauzeti, npr. lopta (kugla) postaje kocka s duljinom stranice promjera lopte. Navedeni primjer skiciran je na (slici 1)*[2]*.*



Slika 1: Kugla u kocki

- *dizajn algoritma*
  1. *razvrstaj predmete po veličini*
  2. *uzmi predmet najvećeg volumena*
  3. *ako predmet stanu u kutiju: stavi ga u kutiju, inače otvori sljedeću kutiju*
  4. *idi na korak 2.*
- *otklanjanje pogrešaka - testiranjem na primjeru vidi se da treba veći broj kutija od optimalnog broja kutija pa se rješenje mora prepraviti*

Očito je da nismo dobili najefikasnije rješenje zadatka. Bit primjera nije u rješenju zadatka, nego u načinu razmišljanja koje je učenik mogao imati u susretu s ovakvim zadatkom. Detaljan opis i rješenje ovog zadatka se nalazi u [19], 38. poglavlje *Bin Packing or „How Do I Get My Stuff into the Boxes?"*.

[2]Preuzeto s https://edutorij.e-skole.hr.

1.3 Razvoj računalnog razmišljanja

Razvoj računalnog razmišljanja kod učenika zahtijeva raznolike i kreativne pristupe. Samo fokusiranje na programerske zadatke može biti ograničeno i neće zadovoljiti potrebe svakog učenika. Srećom, postoje alternativni tipovi zadataka i aktivnosti koji mogu potaknuti razvoj računalnog razmišljanja na zabavan i interaktivan način. U ovom radu ćemo se fokusirati na tri primjera koji nude aktivnosti za razvoj računalnog razmišljanja kod učenika, a to su CS Unplugged projekt te natjecanja Dabar i Klokan.

1.3.1 CS Unplugged

CS Unplugged projekt razvijen je kao nastavno pomagalo za razvijanje računalnog razmišljanja bez korištenja računala. Cilj projekta je promovirati informatiku i računalstvo mladima kao zanimljivu, privlačnu i intelektualno poticajnu disciplinu. Projekt se sastoji od različitih aktivnosti koje uključuju igre, zagonetke i eksperimente. Sve informacije o projektu i aktivnosti dostupne su na mrežnoj stranci projekta (vidi [5]).

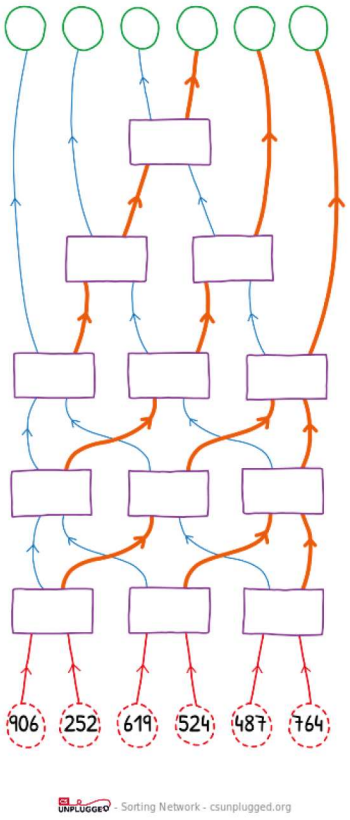
U današnje vrijeme postoji mnogo projekata koji se bave promicanjem programiranja i računalne znanosti. Navest ćemo karakteristike koje razlikuju CS Unplugged projekt od ostalih:

- Bez upotrebe računalne tehnologije: Za provođenje aktivnosti nisu potrebna računalna. Umjesto toga, koriste se igre, manipulativni materijali i druge interaktivne metode kako bi se djeca učila osnovnim konceptima računalne znanosti.
- Aktivno sudjelovanje: Aktivnosti potiču aktivno sudjelovanje učenika. Oni su aktivno uključeni u igre, rješavanje problema, suradnju s drugima i kritičko razmišljanje kako bi razvili svoje vještine računalnog razmišljanja.
- Konceptualno razumijevanje: Fokus aktivnosti je na razumijevanju osnovnih koncepata računalne znanosti. Cilj nije savladati vještinu programiranja nego poticati apstraktno razmišljanje i razumijevanje algoritama, logike, uzročno-posljedičnih veza i druge temeljne koncepte.
- Prilagodljivost i kreativnost: Aktivnosti su prilagodljive i potiču kreativnost. Nastavnici i učenici imaju fleksibilnost prilagoditi aktivnosti svojim potrebama, interesima i razini znanja. Samim time potiče se kreativnost u rješavanju problema i omogućava se individualizacija učenja.
- Suradnja i timski rad: Aktivnosti često potiču suradnju i timski rad. Učenici rade zajedno kako bi riješili probleme, razvijali strategije i dijelili ideje. Ovo promiče komunikaciju, razmjenu znanja i razvoj socijalnih vještina.
- Aktivnosti nisu povezane u cjelinu: Aktivnosti su osmišljene kao samostalni moduli koji se mogu koristiti neovisno jedan o drugome. Nastavnici imaju fleksibilnost odabrati određenu aktivnost koja odgovara njihovim potrebama ili kombinirati aktivnosti na različite načine. To omogućuje prilagodbu aktivnosti prema nastavnim planovima i programima, kao i mogućnost korištenja aktivnosti samostalno.

- Otpornost na pogreške: Aktivnosti su otporne na pogreške učenika. Dakle, manje pogreške neće spriječiti učenike u razumijevanju osnovnih načela. Upute su jasne i jednostavne, obično se sastoje od jednog ili dva pravila i cilja koji se mogu izraziti u jednoj rečenici. Ovo omogućuje učenicima fokus na koncepte i ideje, a ne na otklanjanje pogrešaka.

Primjer jedne aktivnosti CS Unplugged projekta je „Sorting Networks". Kroz aktivnost učenici istražuju kako računala mogu sortirati podatke pomoću mreža za sortiranje. Aktivnost je prilagođena za različite uzraste učenika. Za najmlađe učenike od 5 do 7 godina, učenici sortiraju jednoznamenkaste brojeve. Teže varijante zahtijevaju sortiranje višeznamenkastih brojeva, slova, riječi po abecedi... Vizualni prikaz mreže za sortiranje troznamenkastih brojeva nalazi se na slici 2 koja je preuzeta iz [5].

Aktivnost započinje podjelom učenika u grupe od šest učenika. U svakom trenutku mrežu koristi samo jedan tim. Trenutačni tim stoji na krugovima na početku mreže. Svakom od šest učenika daje se kartica s brojem koju će držati. Prva dva učenika slijede linije iz svojih krugova do pravokutnika, gdje se susreću i pozdrave se. Zatim usporede kartice kako bi odlučili tko ima manji, a tko veći broj. Učenik s manjim brojem slijedi liniju s lijeve strane, a učenik s većim brojem slijedi liniju s desne strane do sljedećeg pravokutnika. Parovi učenika ponavljaju ovaj postupak sve dok ne dođu do kraja mreže. Na kraju, učenici se okreću prema početnim krugovima i čitaju kartice, provjeravajući jesu li u pravilnom redoslijedu od najmanjeg do najvećeg (vidi [5]).



Slika 2: Mreža za sortiranje u aktivnosti „Sorting Networks"

Uz detaljan opis provedbe aktivnosti, nalaze se razrađeni ishodi učenja koje učenik treba usvojiti i aspekti računalnog razmišljanja koji se razvijaju tijekom provođenja aktivnosti. Ovo je izuzetno korisno za nastavnike jer im pomaže da jasno razumiju ciljeve i očekivanja od aktivnosti te da ih lakše integriraju u svoju nastavu.

1.3.2 Natjecanje Dabar i Klokan

Primjere zadataka koji potiču računalno razmišljanje možemo pronaći i na natjecanju Dabar. Službeno ime ovog natjecanja je Bebras koje potječe iz Litve i znači dabar. Ime Bebras koristi se i u drugim zemljama kao što su Velika Britanija, Australija, SAD... Natjecanje Dabar je pokrenuto 2003. godine u Litvi radi promocije informatičke edukacije u školama i upotrebe informacijsko-komunikacijskih tehnologija. Natjecanje je 2006. godine postalo međunarodno. Održava se svake godine s preko 60 zemalja i nekoliko milijuna natjecatelja (više u [6]). Organizator ovog natjecanja za Hrvatsku je udruga „Suradnici u učenju" uz podršku Hrvatskog saveza informatičara i Visokog učilišta Algebra, CARNeta te CROZ-a.

Drugi tjedan studenog proglašen je Svjetskim BEBRAS tjednom rješavanja zadataka. Neke su ga zemlje produljile na dva tjedna. Mnoge zemlje provode cjelogodišnje Bebras aktivnosti kao što su događaji dodjele nagrada sudionicima, drugi krug izazova, ljetni kampovi, radionice za nastavnike (vidi [1]).

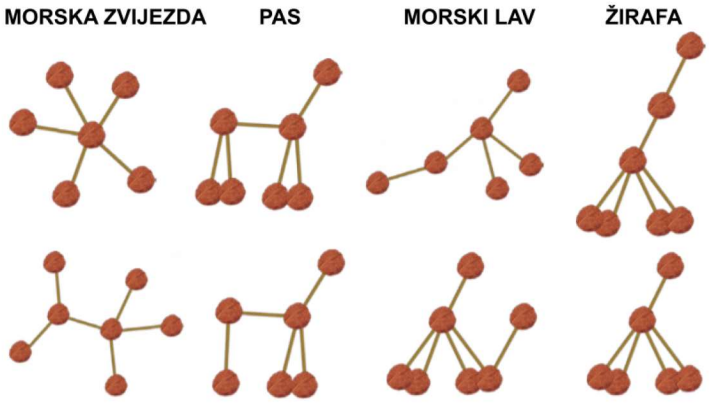
Cilj natjecanja Dabar je promovirati vještine rješavanja problema i računalnog razmišljanja. Kroz rješavanje zadataka, učenici razvijaju sposobnost razdvajanja složenih problema na jednostavnije komponente, razvijaju algoritamsko razmišljanje, prepoznaju uzorke, generaliziraju ih i apstrahiraju.

Dabar se provodi putem online platforme. U Hrvatskoj se pristupa natjecanju putem korisničkog identiteta AAI@skole.hr. Zadaci se rješavaju u prostoru škole uz obavezan nadzor nastavnika. U slučaju da učenik iz tehničkih razloga napusti učionicu tijekom rješavanja testa, bit će mu dopušteno ponovno se prijaviti i nastaviti s rješavanjem, pod uvjetom da nije premašio vremenski rok. Svaki zadatak nosi maksimalno 1 bod, a za netočne odgovore nema negativnih bodova. Upotreba kalkulatora i drugih digitalnih pomagala nije dozvoljena (prema [6]).

Zadaci na natjecanju Dabar su oblikovani tako da potiču učenike na razmišljanje izvan ustaljenih okvira te da ih potaknu na kritičko razmišljanje, analizu problema i pronalaženje efikasnih rješenja. Zadaci često simuliraju realne situacije s kojima se susreću stručnjaci iz područja informatike i računalnih znanosti. Predstavit ćemo jedan od zadataka koji se pojavio na natjecanju Dabar pod nazivom „Životinje od oraha", a preuzet je iz [9].

Primjer 1.2. *Životinje od oraha*

Dabar je bio inspiriran slikom životinje napravljene od orašastih plodova, te je od oraha, konca i ljepila sam napravio četiri životinje. Njegova sestra se igrala s tim životinjama i transformirala ih tako da je postalo teško prepoznati što je što. Spoji dabrove životinje iz prvog reda s transformiranim životinjama iz drugog reda na slici 3.



Slika 3: Zadatak s natjecanja Dabar „Životinje od oraha”

Zadatak od učenika zahtjeva prepoznavanje izomorfnih grafova. Ovaj zadatak provjerava usvojenost aspekta računalnog razmišljanja kojeg smo već spominjali - apstrakcije (više u [9]). Rješenje zadatka je sljedeće: u drugom redu su životinje poslagane redom - pas, morski lav, žirafa i morska zvijezda.

Po svom inicijalnom karakteru natjecanje Dabar je vrlo slično većem i starijem međunarodnom natjecanju Klokan. Natjecanje Klokan je nastalo iz originalne matematičke igre Klokan koju je 1980. godine osmislio profesor matematike Peter O'Holloran u Sydneyu, Australija. Ova matematička igra omogućila je tisućama učenika da istovremeno odgovaraju na zadane zadatke, a brzo je stekla veliku popularnost u Australiji. Godine 1991., dva francuska profesora prenijela su igru u Francusku i nazvala je „Klokan” u čast profesora O'Hollorana iz Australije. Kroz godine, Klokan je prerastao u međunarodno natjecanje pod nazivom „Klokan bez granica”. Ovo natjecanje provodi istoimena udruga koja djeluje pod okriljem UNICEF-a (prema [6]).

Cilj natjecanja Klokan je motivirati učenike i potaknuti njihov interes za matematiku izvan redovne nastave. Kroz sudjelovanje u natjecanju, učenici imaju priliku razviti matematičko i računalno razmišljanje, logiku te vještinu rješavanja problema. Natjecanje Klokan pruža izazovne zadatke i priliku za natjecanje, ali isto tako naglašava zabavni aspekt matematike kako bi potaklo učenike da razviju pozitivan stav prema ovom predmetu, ali i STEM-u općenito. Natjecanje se održava jednom godišnje među državama članicama, obično treći četvrtak u ožujku (vidi [10]).

Na natjecanju Klokan, učenici rješavaju 12 ili 24 zadatka koji imaju po pet ponuđenih odgovora, od kojih je samo jedan točan. Učenici drugog i trećeg razreda osnovne škole imaju 60 minuta za rješavanje 12 zadataka, dok učenici ostalih razreda osnovne i srednje škole imaju 75 minuta za rješavanje 24 zadatka. Zadaci su razvrstani u tri stupnja težine. Svaki zadatak, ovisno o težini, nosi 3, 4 ili 5 bodova. Svi učenici koji se prijave na natjecanje dobivaju poklon, što uključuje torticu, trokut i kemijsku olovku. Također, najuspješnijih 10 posto učenika dobiva i dodatne nagrade za svoj trud i postignuća (više u [10]).

2. Programski jezici za poučavanje

Programski jezici za poučavanje su jezici koji su posebno dizajnirani kako bi olakšali proces učenja programiranja. Oni pružaju jednostavnije sintakse, vizualne elemente i alate koji pomažu u razumijevanju osnovnih koncepata programiranja. Takvi jezici se nazivaju mini-jezicima zbog svoje manje kompleksne sintakse i ograničenog skupa naredbi (prema [13]).

Mini-jezici imaju osnovne konstrukcije kao što su petlje, grananja, funkcije i varijable, ali su namjerno ograničeni u svojim mogućnostima kako bi se olakšalo razumijevanje i učenje programiranja početnicima. Naglasak je stavljen na semantičku strukturu programa, tj. logiku i koncepte iza koda, umjesto na detalje sintakse samog jezika.

Upotreba mini-jezika u nastavi može pomoći učenicima steći temeljno razumijevanje programiranja i razvoj računalnog razmišljanja. Nakon toga se postepeno mogu uvoditi složeniji programski jezici i koncepti. Mini-jezici su moćan alat za upoznavanje učenika s osnovama programiranja na pristupačan i motivirajući način.

2.1 Vrste programskih jezika za poučavanje

S obzirom na sintaksu programskog jezika, programske jezike za poučavanje možemo podijeliti na tri vrste: vizualne-blokovske, vizualne-tekstualne i proceduralne-tekstualne programske jezike (vidi [13]). Svaka od ovih grupa jezika ima svoje prednosti i koristi se u različitim kontekstima ovisno o ciljevima nastave i sposobnostima učenika. Odabir odgovarajućeg programskog jezika treba ovisiti o dobi učenika, prethodnom iskustvu i željenim ciljevima učenja programiranja.

2.1.1 Vizualni-tekstualni programski jezici

Jedan od najpoznatijih primjera vizualno-tekstualnih programskih jezika je zasigurno Logo. Logo se smatra jednim od prvih mini-jezika koji je nastao 60-ih godina prošlog stoljeća. Primarno je osmišljen za uvođenje programiranja osnovnoškolskom uzrastu kroz geometriju. Osnovan je na konstrukcionističkom pristupu učenju. Prema [13], konstrukcionizam se temelji na dvije vrste „konstrukcije“:

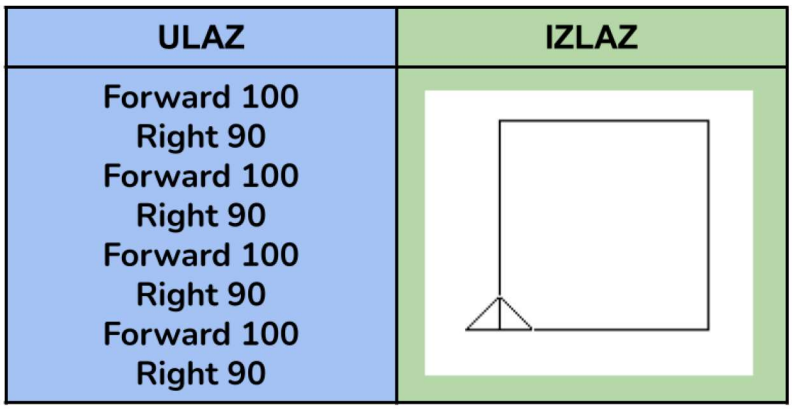
- učenje se smatra procesom u kojem učenik aktivno konstruira znanje iz vlastitog iskustva, tj. učenik gradi svoje razumijevanje i znanje putem interakcije s okolinom, eksperimentiranjem, istraživanjem i rješavanjem problema
- učenik stječe novo znanje s određenom učinkovitošću kada je uključen u konstruiranje smislenih produkata kao što je slaganje LEGO kockica, robota ili računalnih programa

Dakle, jedna od ključnih ideja Loga je aktivno sudjelovanje učenika u izradi nečega smislenog sebi i svijetu oko sebe. U Logu cilj nije samo napisati kod, nego stvoriti crteže i simbole koji imaju značenje.

Glavna značajka Loga je kornjačina grafika. Ona omogućava crtanje na zaslonu koristeći „kornjaču” koja se kreće na ekranu. „Kornjaču” na ekranu najčešće predstavlja trokut. Moguće je i promijeniti oblik lika u nešto drugo, npr. u robota. Učenici mogu upravljati kornjačom putem glavnih naredbi: forward (fd), right (rt), left (lt) i back (bk). U naredbu se treba upisati i za koliko se kornjača treba pomaknuti. Ako se radi o pomicanju naprijed

ili nazad, potrebno je upisati broj koraka za koji će se kornjača pomaknuti. Za naredbe lijevo i desno treba naglasiti za koliko se stupnjeva kornjača treba rotirati u lijevu ili desnu stranu. Uz navedene naredbe, postoji još naredbi koje se mogu koristiti u Logu, npr. vraćanje kornjače u početni položaj.

Dakle, kako bi programirali u Logu potrebno je upisati naredbu. Kada se program koji smo napisali izvrši, na ekranu vidimo grafički prikaz koji je nastao kao rezultat izvršavanja zadanih naredbi. Zbog toga Logo pripada u skupinu vizualno-tekstualnih programskih jezika (više u [13]).



Slika 4: Primjer ulaznih i izlaznih podataka u Logu

Kada je riječ o vizualno-tekstualnim programskim jezicima poput Loga, kontekst programiranja često je geometrija. Pomoću njih, učenici imaju konkretno iskustvo programiranja lika koji se kreće po ekranu te crta različite geometrijske oblike. Izvršavanje programa u vizualnom obliku omogućuje učenicima iskustvo „od konkretnog prema apstraktnom", jer se tekstualne naredbe izvršavaju kroz crtanje, što učenici mogu vidjeti i doživjeti (prema [13]).

Logo je programski jezik koji je prvi implementirao koncept „kornjačine grafike". Ideja Logo jezika bila je izuzetno dobro prihvaćena, što je rezultiralo razvojem niza novih vizualnih programskih jezika. Čak i Python, koji je temeljen na tekstualnom programiranju, uključuje turtle biblioteku koja omogućuje korištenje kornjačine grafike na način sličan onome u Logu [13].

Dokaz tome koliko je Logo dobro prihvaćen u poučavanju programiranja je Hrvatska Logo Liga. S ciljem popularizacije programiranja među najmlađim učenicima, Hrvatski savez informatičara je 2016. godine pokrenuo program Hrvatska Logo Liga. Ovo natjecanje obuhvaća najviše šest kola tijekom jedne školske godine i otvoreno je za sve učenike osnovnih škola u Republici Hrvatskoj. Natjecanje je namijenjeno učenicima od 1. do 8. razreda, koji se mogu organizirati u ekipe svojih škola, informatičkih klubova, udruga ili neformalnih inicijativa (više u [8]).

Svako kolo natjecanja sastoji se od osam zadataka različite težine, s različitim brojem bodova. Za rješavanje zadataka koristi se programski jezik Logo putem FMSLogo okruženja. Natjecanje traje tri sata, tijekom kojih sudionici moraju razvijati i implementirati svoje ideje koristeći Logo jezik (prema [8]).

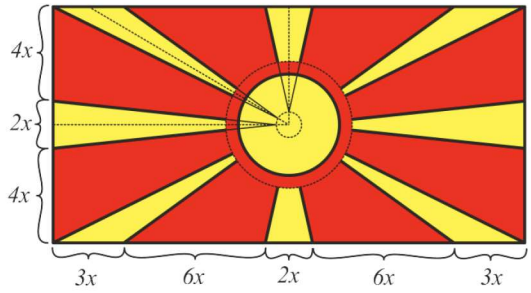
Na slici 5 nalazi se primjer jednog zadatka koji se pojavio na natjecanju Hrvatske Logo Lige. Zadatak spada među teže zadatke i zbog toga nosi puno bodova. Za uspješno rješavanje

zadatka učenici su trebali iskoristiti znanje programiranja, ali i matematike. Matematika, ili preciznije geometrija, im je bila potrebna prvenstveno za razumijevanje zadatka, a kasnije i za rješavanje. U ovom zadatku vidljiva je snažna veza između matematičkog i računalnog razmišljanja koju je istaknuo sam Seymour Papert prilikom razvoja programskog jezika Logo.

Hrvatska Logo Liga 2019./2020.
6. kolo, od 6. do 16. ožujka 2020.

Zadatak MAKEDONIJA
120 bodova

Početkom 2019. godine, dotadašnja Republika Makedonija promijenila je ime u Republika Sjeverna Makedonija, nakon pregovora s Grčkom. Još od proglašenja neovisnosti makedonske države, Grčka je imala problem s njenim imenom. Naime, pokrajina na sjeveru Grčke također se zove Makedonija, a Grci smatraju kako to ime spada u grčko povijesno nasljeđe.



Napišite proceduru `MAKEDONIJA :x :y` koja crta makedonsku zastavu po dimenzijama sa skice. Sunce i zrake sunca na zastavi potrebno je obojiti žutom bojom (`SETFC [249 214 22]`), a ostatak zastave crvenom bojom (`SETFC [206 32 40]`). Radijusi kružnica na skici iznose `:y`, `4*:y` i `5*:y`. Rubovi zraka uz sredinu zastave su kružni lukovi.

ULAZNI PODACI

Varijable `:x` i `:y` su prirodni brojevi takvi da je varijabla `:y` manja ili jednaka dvije trećine varijable `:x`.

PRIMJERI TEST PODATAKA

```
CS MAKEDONIJA 14 5
```

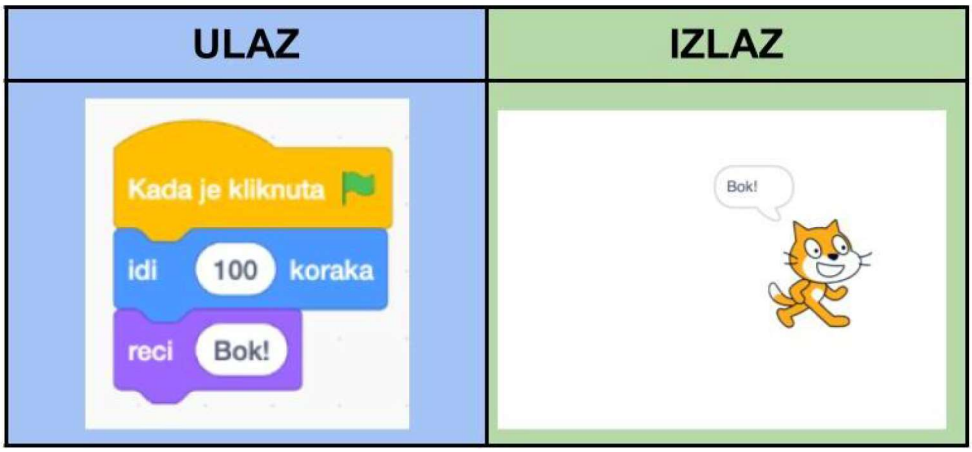
Slika 5: Primjer zadatka iz Logo Lige

2.1.2 Vizualni-blokovski programski jezici

Vizualno-blokovski programski jezici koriste grafički prikaz blokova koji se mogu povlačiti i spajati kako bi se stvorio programski kod. Ovi jezici su posebno popularni među početnicima u programiranju, posebno djecom, jer olakšavaju razumijevanje osnovnih programskih koncepata [12]. Kod vizualno-blokovskih programskih jezika su ulazne naredbe predstavljene blokovima kombiniranim u obliku slagalice, dok je izlaz grafički prikaz u obliku kretanja lika (prema [13]). Jedan od vizualno-blokovskih programskih jezika je Scratch. Primjer ulaznih i izlaznih vrijednosti u Scratchu nalaze se na slici 6.

Scratch je jedan od najpoznatijih vizualno-blokovskih programskih jezika. Razvijen je pod vodstvom Mitcha Resnika u sklopu MIT Media Laba i prvi put je predstavljen 2003. godine. Scratch omogućava korisnicima stvaranje interaktivnih priča, igri i animacija povlačenjem i spajanjem blokova. Ovaj jezik ima jednostavno i intuitivno sučelje koje omogućava djeci da se brzo upuste u stvaranje vlastitih projekata (vidi[12]).

Scratch podržava stvaranje interaktivnih priča, igara i animacija putem blokova naredbi koji su grupirani prema funkcionalnosti kao što su kretanje, izgled, zvuk, događaji, upravljanje, očitanja, operacije, varijable te omogućuje izradu vlastitih blokova. Ovi blokovi imaju različite boje što olakšava učenicima pronalaženje i povezivanje blokova (vidi [12]). Također, važno je napomenuti kako su naredbe prevedene na više od 50 jezika, uključujući i hrvatski jezik. Pomoću tih naredbi korisnik može kontrolirati likove koji su prikazani kao 2D grafički objekti na pozornici. Korisnik slaže odgovarajuće naredbe u skripte, a pokretanjem programa može vidjeti rezultat izvršavanja u obliku kretanja likova na pozornici. Ova interaktivna prikazna forma omogućuje korisniku da na konkretan način vidi kako se program izvršava (prema [13]).



Slika 6: Primjer ulaznih i izlaznih podataka u Scratchu

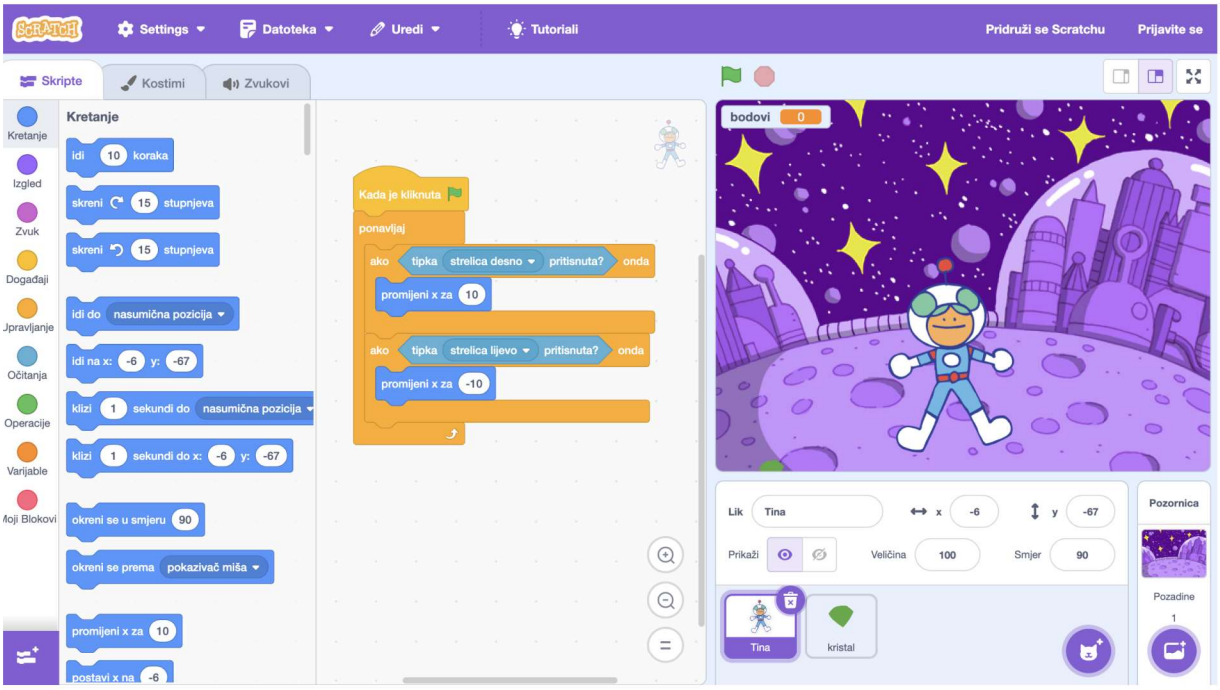
Programiranje u vizualno-blokovskim programskim jezicima omogućuje jednostavno uvođenje osnovnih programerskih pojmova putem programiranja igara, što omogućuje učenicima da se upoznaju sa svim ključnim konceptima programiranja već u prvim susretima. Takav pristup učenju programiranja, koristeći kontekst igara, priča, animacija i sličnih projekata u blokovskim programskim jezicima, potiče učenje otkrivanjem.

Učenje otkrivanjem se odvija kada učenik koristi vlastito iskustvo i prethodno znanje kako bi riješio nove probleme. Kroz programiranje igara, učenici moraju postaviti scenarij igre u obliku algoritma kako bi je programirali. Također, moraju manipulirati naredbama u obliku slagalica kako bi konstruirali svoj program. Ako program ne radi ispravno, učenik mora istražiti program kako bi pronašao i ispravio greške. Tijekom ovog procesa, učenici aktivno primjenjuju koncepte programiranja, često nesvjesni da to čine. Uloga učitelja je otkriti korištene pozadinske pojmove programiranja učenicima tijekom nastavnog procesa (vidi [13]).

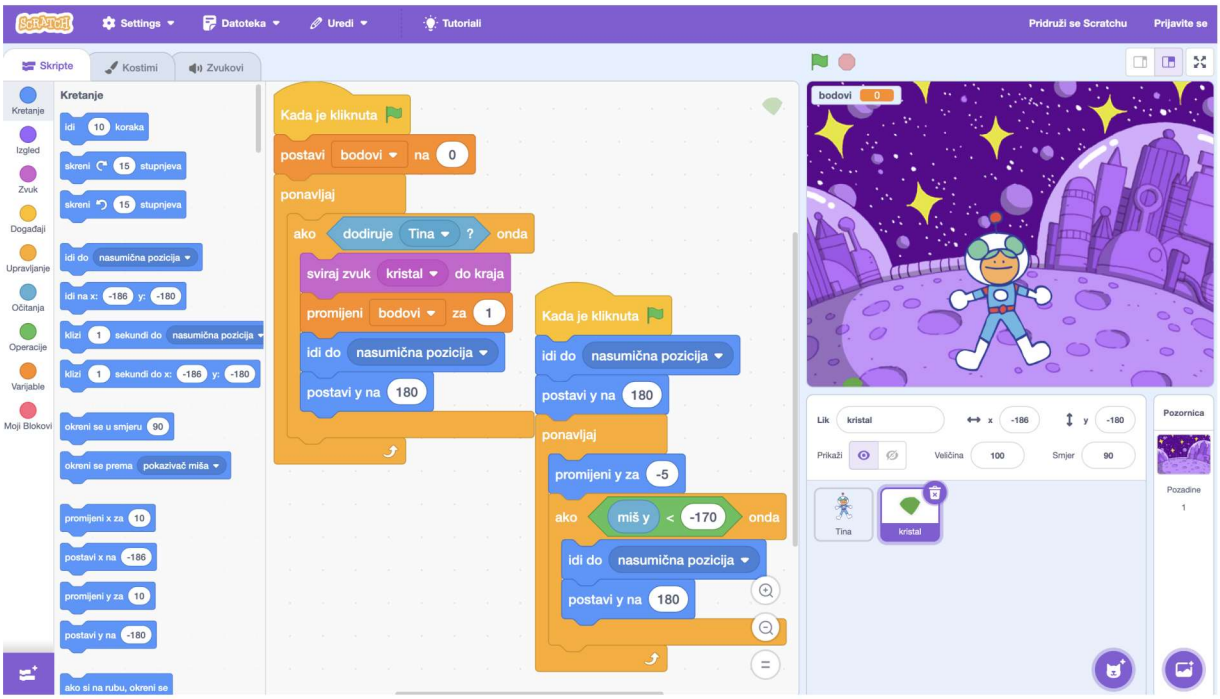
Pokazat ćemo na primjeru kako učenici mogu napraviti računalnu igru, bez puno znanja o programiranju. Zadatak se pojavljuje u udžbeniku za četvrti razred osnovne škole, a vezan je za programiranje u Scratchu. Zadatak je preuzet iz [2]. U udžbeniku je dano objašnjenje i rješenje, ali je zadatak pogodan i za provedbu putem učenja otkrivanjem. Učenje otkrivanjem je u ovom zadatku moguće jer se učenici mogu koristiti svojim znanjem kako bi riješili zadatak. Zadaća učitelja je voditi učenike kroz taj cjelokupan proces.

Cilj zadatka je napraviti igru u kojoj lik Tina skuplja zelene kristale. Kristali se pojave na vrhu pozornice i lagano padaju prema dnu. Tinin zadatak je pokupiti kristal prije nego

što padne na dno. Tina se može pomicati lijevo ili desno po pozornici pomoću strelica na tipkovnici. Prije programiranja, učenici su trebali dodati likove Tinu i kristal te postaviti pozornicu koja im je prikazana u udžbeniku. Nakon toga, potrebno je napisati programe za Tinino kretanje, kristal i skupljanje bodova. Program za kretanje Tine prikazan je na slici 7, a program za kristal i bodovanje na slici 8.



Slika 7: Program za kretanje Tine



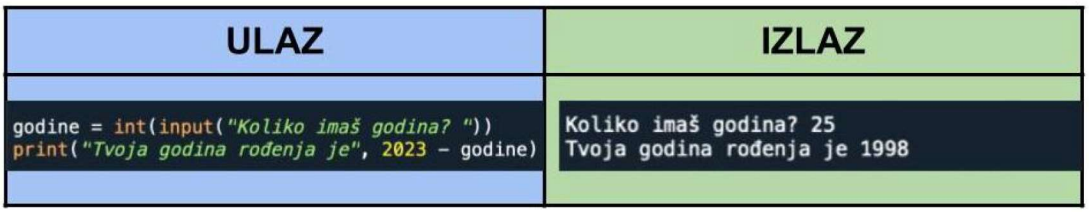
Slika 8: Program za bodovanje i kristal

Za razliku od programskog jezika Logo, Scratch ne zahtijeva poznavanje matematike kao preduvjet za programiranje. Naglasak u Scratchu nije isključivo na rješavanju matematičkih problema, nego učenici mogu koristiti Scratch za stvaranje priča i drugih kreativnih projekata. Zbog toga je programiranje u Scratchu prikladno čak i za učenike koji nemaju prethodno znanje iz matematike. Ovo je posebno važno za mlađe učenike, uključujući i one vrtićke dobi. Scratch omogućuje djeci da se upuste u svijet programiranja i stvaranja interaktivnih projekata, bez potrebe za dubokim razumijevanjem matematičkih koncepata (više u [12]).

2.1.3 Proceduralni-tekstualni programski jezici

Proceduralno-tekstualni programski jezici koriste tekstualne naredbe za unos podataka te izvršavanje programa rezultira podacima u obliku teksta. U sustavu obrazovanja, posebno u osnovnim školama, fokus programiranja u tekstualnim programskim jezicima još uvijek većinom ostaje na rješavanju matematičkih problema. S obzirom na to da su ulaz i izlaz izvršavanja programa u obliku teksta, apstraktno razmišljanje postaje ključna vještina za razumijevanje i usvajanje koncepta programiranja tijekom nastave. Unatoč jednostavnoj sintaksi, tekstualni programski jezici možda nisu najprikladniji za mlađe razrede osnovne škole (prema [13]).

Primjeri proceduralno-tekstualnih programskih jezika su: Python, Pascal, BASIC, C++. Python je najpoznatiji primjer proceduralno-tekstualnog jezika. Na slici 9 vidimo primjer ulaznih i izlaznih podataka u Pythonu. Nastao je 1991. godine, a njegov tvorac je Nizozemac Guido van Rossum. Glavna ideja ovog programskog jezika je olakšati učenje programiranja (više u [12]).



Slika 9: Primjer ulaznih i izlaznih podataka u Pythonu

U slučaju Pythona, čitljivost je jedno od glavnih načela dizajna. Python je čitljiviji od većine drugih programskih jezika jer koristi uvlake umjesto vitičastih zagrada za odvajanje dijelova koda. Ova sintaksa omogućava programerima da izraze svoje namjere u manjem broju linija koda. Zato Python ima kraću i jednostavniju sintaksu u usporedbi s programskim jezicima poput C++ (vidi [15]). Naredbe su intuitivne i jednostavne, bez potrebe za dodatnim znakovima poput vitičastih zagrada ili begin-end blokova. To znači da početnici u programiranju mogu lakše usmjeriti svoju pažnju na sam algoritam rješavanja problema, umjesto da se previše brinu o sintaksi jezika (više u [15]).

U interaktivnom sučelju Pythona, učenici mogu unositi naredbe i odmah vidjeti što se točno događa nakon svake naredbe. To im omogućava da izvode jednostavne naredbe i odmah vide rezultate njihovog izvršavanja, slično kao u programskom jeziku Logo. Kasnije te naredbe mogu postati dio kompleksnijeg programa.

Python je programski jezik koji nema potrebu za eksplicitnom deklaracijom varijabli. Ovo svojstvo olakšava programiranje učenicima jer se ne moraju brinuti o različitim tipovima podataka i manipulaciji s njima. Nedostatak deklaracija varijabli često predstavlja

apstraktan koncept za početnike u programiranju, ali s vremenom postaje neophodan kako bi se rješavali složeniji problemi. U Pythonu, varijable se automatski dodjeljuju prema vrsti podataka koju sadrže, što olakšava rad s podacima i smanjuje mogućnost pogrešaka uzrokovanih neusklađenošću tipova podataka [15].

Zbog navedenih prednosti, Python je postao jedan od najkorištenijih jezik za poučavanje programiranja na svim razinama obrazovnog sustava. Python je svestrani jezik koji podržava razvoj programa za različite razine vještine i vrste programiranja. Nudi interaktivno sučelje koje omogućuje učenicima i studentima istraživanje funkcionalnog, proceduralnog i objektno orijentiranog načina rješavanja problema.

Python je izuzetno dobro prihvaćen u obrazovnom sustavu. Autori udžbenika za srednje škole prepoznaju prednosti Pythona te ga često uključuju u svoje udžbenike. Kroz udžbenik za četvrti razred srednje škole [4], može se primijetiti značajna prisutnost Pythona kao programskog jezika na kojem se temelje cjeline. U udžbeniku [4], neke od cjelina u kojima se koristi Python su: osnove objektno orijentiranog programiranja, osnove mrežnog programiranja, paralelno programiranje...

Pogledajmo primjer kako Python može biti korišten za poučavanje osnova objektno orijentiranog programiranja. Primjer je preuzet iz [4]. Za uspješno rješavanje primjera potrebno je primijeniti znanje iz matematike.

Primjer 2.1. *Odredimo neke osnovne atribute i metode za kvadrat te kreirajmo klasu Kvadrat s pripadnim atributima i metodama.*

```
class Kvadrat:
    def __init__(self, a=0):
        self.stranica = a

    def opseg(self):
        return 4 * self.stranica

    def povrsina(self):
        return pow(self.stranica,2)
```

```
In [49]: k=Kvadrat(7)

In [50]: k.opseg()
Out[50]: 28

In [51]: k.povrsina()
Out[51]: 49
```

Slika 10: Primjer zadatka u Pythonu

Klasa je temeljni koncept u objektno orijentiranom programiranju koji predstavlja skup objekata s istim obilježjima. Definicija klase uključuje definicije njenih metoda i atributa. Atributi opisuju objekt pomoću određenih podataka, dok se ponašanje objekta određuje metodama koje djeluju na te atribute. Jedna od tih metoda ima važno značenje i naziva se konstruktor. Konstruktor ima predefinirano ime i oblik `__init__(self, parametri)` u Pythonu. Ova metoda se izvodi prilikom stvaranja svakog objekta iz klase (više u [4]).

U primjeru je definirana klasa Kvadrat kod koje je prilikom konstruiranja objekta potrebno navesti duljinu stranice. Ako duljina stranice nije navedena, postavit će se na 0. Atribut ove klase je duljina stranice, a metode su opseg i površina. Kako bi provjerili je li

program dobro napisan, testirali smo na primjeru. U interaktivnom sučelju definirali smo kvadrat duljine stranice 7, te mu izračunali opseg i površinu.

2.2 Poželjne karakteristike programskih jezika za poučavanje

Tvorci programskih jezika za poučavanje programiranja, kao što su Seymour Papert (tvorac Loga) ili Guido van Rossum (tvorac Pythona), predložili su kriterije koje bi programski jezici za poučavanje programiranja trebali zadovoljavati. Prema [11], ukupno je 17 takvih kriterija koji su raspoređeni u 4 smislene skupine: učenje, dizajn i okruženje, podrška i dostupnost te programiranje na višim razinama.

U skupinu učenje svrstani su sljedeći kriteriji za programske jezike:

- pogodan je za poučavanje: Kako bi zadovoljio ovaj kriterij, svrha programskog jezika treba biti obrazovanje. Jezik treba imati jednostavnu sintaksu i prirodnu semantiku te izbjegavati upotrebu skraćenica i drugih simbola koji mogu biti višeznačni ili zbunjujući. Pripadajući alati i okruženje trebaju biti jednostavni za korištenje kako bi podržali proces poučavanja i olakšali rad s jezikom.

- može se primijeniti fizička analogija: Programski jezik trebao bi omogućiti učenicima prepoznavanje utjecaja programskog koda na stvarne situacije. Potrebno je uložiti napor kako bi se učenici doveli do razine na kojoj mogu iskoristiti taj potencijal i primijeniti svoje znanje u različitim okruženjima. Na primjer, programiranje fizičkih objekata je jedan način na koji učenici primjenjuju programski jezik za fizičke analogije i odmah uočavaju rezultate svog rada.

- daje općenitu sliku o programiranju: Kroz programiranje učenici trebaju razumjeti osnove i temeljna načela programiranja, koje će kasnije poslužiti kao izvrsna podloga za napredno programiranje u drugim programskim jezicima. Osnovno razumijevanje principa programiranja omogućuje lakši prijelaz i usvajanje naprednijih koncepta u drugim jezicima.

- promovira novi pristup poučavanju: Programski jezik ne smije se ograničiti samo na implementaciju, već bi trebao obuhvatiti i druge aspekte procesa razvoja softvera. Trebao bi biti metodološki osmišljen kao niz procesa potrebnih za izgradnju softvera. Dakle, jezik bi predstavljao cjelovitu metodologiju za izgradnju softvera temeljena na programskom jeziku i skupu principa, alata i knjižnica.

Pomoću navedenih kriterija u [11], uspoređeno je 11 različitih programskih jezika za poučavanje. Među njima se nalaze i dva jezika koja smo opisali u ovom radu, a to su Logo i Python. Navest ćemo koje kriterije su Logo i Python ispunili. Oba jezika dijele zajedničke karakteristike koje ih čine pogodnima za poučavanje, kao i mogućnost primjene fizičkih analogija. Za Logo ne vrijedi da daje općenitu sliku o programiranju, dok Python ne promovira novi pristup poučavanju.

Dizajn i okruženje su jedan od faktora za poželjne karakteristike programskih jezika namijenjenih poučavanju. Kriteriji koji se nalaze u ovoj skupini su sljedeći:

- ima interaktivno i jednostavno okruženje: Programski jezik treba omogućiti interakciju i podržavati brz razvoj koda. Time bi mlađi učenici imali priliku biti kreativni i pisati programe čak i ako nemaju duboko znanje jezika. Stoga, programski jezik za učenje programiranja treba poticati učenike da napišu svoje prve programe čak i ako nisu u

potpunosti upoznati s jezikom. Ovo se može postići kroz interaktivno, jednostavno i intuitivno okruženje za pisanje programa.

- potiče pisanje ispravnih programa: Kako bi se osiguralo da je kod koji učenici pišu ispravan i bez grešaka, važno je pružiti im smjernice i alate koji će im u tome pomoći. Učenicima treba pokazati strategije za provjeru i testiranje njihovog koda kako bi identificirali i ispravili eventualne greške kao što je postavljanje uvjeta na izvršavanje koda. Cilj ovog kriterija je odmaknuti se od pristupa „pokušaja i pogrešaka" koje se često primjenjuje.

- omogućuje podjelu koda na manje dijelove: Kada učenici pišu svoje prve programe, ključno je da razumiju svaki dio napisanog koda i shvate zašto su ti dijelovi neophodni kako bi program bio ispravan. Stoga je važno da programski jezik podržava dekompoziciju koda kroz korištenje funkcija, procedura i sličnih konstrukata.

- pruža cjelovito razvojno okruženje: Tijekom učenja programiranja, važno je razumjeti postupak kojim se izvorni kod pretvara u izvršnu datoteku programa. Neka integrirana razvojna okruženja mogu sakriti te detalje, pojednostavljujući proces radi veće jednostavnosti i brzine, što može biti korisno iskusnim programerima, ali manje korisno za početnike. Druga okruženja mogu pomoći u prevladavanju razlika između dizajna i implementacije, na primjer, omogućujući pretvaranje dijagrama toka u izvorni kod ili čak obrnuti ovaj proces, pretvarajući kod u vizualni prikaz.

Prema [11], u ovoj kategoriji nalazi se samo jedan kriterij koji zadovoljavaju Python i Logo. To je kriterij koji omogućava podjelu koda na manje dijelove. Python djelomično ispunjava i kriterij poticanja pisanja ispravnih programa ako tijekom programiranja u Pythonu koristimo jedinično testiranje. Jedinično testiranje je proces testiranja u kojem je fokus na testiranju pojedinačnih komponenti programa.

Sljedeća skupina kriterija za poučavanje programskih jezika, prema [11], je podrška i dostupnost. Kriteriji koji pripadaju cjelini podrška i dostupnost su da programski jezik:

- ima podrška korisnicima: U situacijama kada su resursi i podrška ograničeni, to može predstavljati izazov za učitelje i učenike. Kako bi se zadovoljio ovaj kriterij, važno je osigurati adekvatnu podršku za studente, nastavno osoblje i ostale zainteresirane osobe u učenju i korištenju jezika. Ta podrška može biti pružena na različite načine, kao što su mrežne stranice, tečajevi, knjige, tutoriali, vježbe, dokumentacija i mailing liste. Ovi resursi mogu pružiti korisne informacije, upute i primjere koji pomažu učenicima i nastavnom osoblju da se bolje upoznaju s jezikom i njegovom primjenom.

- kod je javno dostupan: Kod otvorenog tipa ima važnost jer smanjuje troškove i omogućuje kontinuirano poboljšavanje programskog jezika kako bi bio što bolji za poučavanje. Dostupnost koda omogućuje zajednici da sudjeluje, dijeli ideje i iskustva te doprinosi razvoju jezika. To stvara priliku za suradnju i inovaciju te osigurava da jezik bude prilagođen potrebama korisnika i bolje podržava proces učenja. Kako bi zadovoljio ovaj kriterij, jezik trebao biti rezultat zajednice koja nije usmjerena na stvaranje komercijalnog proizvoda i u kojem svatko može sudjelovati ako to želi.

- podržava različite platforme: Važno je pružiti učenicima programske jezike koji su dostupni i kompatibilni na različitim platformama. Na taj način, učenici im mogu pristupiti bez obzira na postavke sustava uređaja kojim se koriste.

- besplatan i lako dostupan: Učenici koji su tek započeli s programiranjem vjerojatno neće biti u mogućnosti uložiti velike iznose u skupi jezik ili integrirano razvojno okruženje. Stoga je poželjno da jezik bude besplatan i ne zahtijeva pretplatu ili obvezu. Također je važno da jezik bude dostupan širom svijeta, bez geografskih ograničenja, kako bi se omogućio pristup svim studentima bez obzira na njihovu lokaciju.

- ima literaturu: Jezik treba biti podržan kvalitetnim nastavnim materijalom koji je dostupan nastavnicima i učenicima. Materijali pružaju vrijedne perspektive i predlažu prikladne nastavne planove i programe za određeni programski jezik. Važno je da postoje relevantni udžbenici i drugi materijali koji su pristupačni za upotrebu u učionicama.

Python je primjer programskog jezika koji zadovoljava sve navedene kriterije za podršku i dostupnost. Za razliku od Pythona, Logo je programski jezik koji može biti ograničen u pogledu dostupnosti literature i podrške korisnicima (vidi [11]).

Sljedeći kriteriji ispituju koliko je programski jezik za poučavanje prilagođen za više od samog programiranja za početnike. Ovi kriteriji posebno su važni za učenike koji žele znati više ili učiti bržim tempom. Kriteriji koji pripadaju pod cjelinu programiranje na višim razinama su:

- jezik se ne koristi samo za obrazovanje: Učenici mogu biti više motivirani jezikom koji se ne koristi samo za poučavanje, nego ima primjenu u industriji, npr. za razvoj velikih i poznatih aplikacija.

- može se upotrijebiti za različite svrhe: Programiranje ne mora koristiti samo na satima informatike i za rješavanje računalnih problema. Kada je jezik primjenjiv u drugim kontekstima, to omogućuje učenicima stvaranje programa koji rješavaju konkretne probleme, npr. učenik može napisati program za rješavanje kvadratne jednadžbe koji će mu pomoći pri učenju matematike.

- efikasan za kompliciranije programe: Unatoč tome što učenici rijetko iskorištavaju potencijal brzine jezika, akademsko okruženje pruža mogućnost istraživanja tih granica. Iako iz pedagoškog aspekta ovaj kriterij može imati niži prioritet u odnosu na druge kriterije, jezik treba biti koristan za razvoj aplikacija visoke brzine.

- jezik nije primjer za QWERTY fenomen[3]: U kontekstu programskih jezika, QWERTY fenomen odnosi se na pojavu u kojoj prvi dostupni jezik postaje dominantan i ostaje u upotrebi unatoč postojanju naprednijih i boljih alternativa. Ova tendencija može rezultirati ograničenjem inovacija i napretka u području programiranja. Kako bi jezik zadovoljio ovaj kriterij, treba pokazati svoju korisnost sada i u budućnosti, prepoznajući i nadilazeći svoju primjenjivost u prošlosti.

Prema [11], Python je izuzetno pogodan za programiranje na višim razinama jer ispunjava sva četiri navedena kriterija. S druge strane, Logo nije najbolji primjer za programiranje na višim razinama jer zadovoljava samo jedan kriterij, a to je da nije primjer QWERTY fenomena.

S obzirom na sve kriterije, programski jezik Python se ističe kao jedan od najprikladnijih jezika za poučavanje programiranja jer zadovoljava čak 15 od 17 navedenih kriterija.

[3]QWERTY fenomen opisuje tendenciju da se prvi upotrebljivi, ali primitivni proizvod nove tehnologije ukorijeni i ostane prisutan unatoč postojanju naprednijih i boljih alternativa. Najpoznatiji primjer QWERTY fenomena je QWERTY raspored tipki na standardnoj tipkovnici, koji je razvijen kako bi se smanjila vjerojatnost zapinjanja tipki na pisaćim strojevima. Unatoč tome što danas postoje učinkovitiji rasporedi tipki kao što je Dvorak, QWERTY raspored se i dalje koristi kao dominantan.

3. Miskoncepcije u programiranju

Kako bi razumjeli pojam miskoncepcije, potrebno je upoznati se s pojmom koncepta. Koncept definiramo kao utemeljenu znanstvenu spoznaju (vidi [20]). Učenici ulaze u sustav obrazovanja s određenim prethodnim znanjem, koje može biti usklađeno ili neusklađeno sa znanstvenim spoznajama. Prethodno znanje koje učenici posjeduju naziva se predkoncepcija te je rezultat njihovog osobnog iskustva ili prethodnog učenja. Tijekom školovanja često se ne pridaje dovoljno pažnje predkoncepcijama učenika. Ispravne predkoncepcije mogu doprinijeti daljnjem razvoju znanja, dok pogrešne predkoncepcije mogu dovesti do razvoja miskoncepcija, odnosno učeničkih koncepata koji nisu u skladu sa znanstvenim spoznajama. Miskoncepcije često nastaju kod učenika jer, iako netočne, pružaju jednostavnija objašnjenja koja učenici brže i lakše prihvaćaju (više u [20]).

U [14], autor istražuje različite vrste miskoncepcija koje učenici čine neovisno o specifičnim naredbama ili programskim jezicima. Ove miskoncepcije nisu povezane s dizajnom programskih jezika, već s davanjem uputa računalu tijekom procesa učenja. Navest ćemo i opisati miskoncepcije iz [14].

Paralelizam je jedna od konceptualnih pogrešaka koja se pojavljuje u raznim kontekstima, a u osnovi se temelji na pretpostavci da različite linije programskog koda mogu biti aktivne istovremeno, odnosno paralelno. Razlikujemo dvije glavne kategorije u kojima je pogrešno korištenje paralelizma uobičajeno.

Jedan od primjera miskoncepcije paralelizma je kada se uvjet pojavljuje izvan grananja, npr. prvo u programu imamo:

```
if size==10:
    print("hello");
```

a kasnije u programu slijedi petlja

```
for size in range(1,10):
   size = size+1;
```

Učenicima koji razumiju program, jasno je kako se prvo provjerava uvjet, tj. ako je varijabla size jednaka 10, ispisuje se *hello*, nakon čega program prelazi na sljedeću liniju koda. Ako size nije jednak 10, ništa se neće ispisati i programa će se nastaviti dalje izvršavati. Dakle, kada je grananje izvršeno, postaje neaktivno i nevažno za ostatak programa jer se kontrola više ne vraća unatrag.

Međutim, više od polovine učenika u istraživanju iz [14] smatra kako će se u nastavku programa, kada varijabla size unutar for petlje postane jednaka 10, ipak ispisati *hello*. Objašnjenje koje ispitanici navode za takvo razmišljanje glasi: uvjet iz grananja čeka sve dok varijabla size unutar petlje ne postane jednaka 10 kako bi tada mogla obaviti zadani ispis. Međutim, ovo je pogrešno shvaćanje jer u stvarnosti program izvršava liniju po liniju i ne vraća se unatrag na prethodne naredbe.

Još jedan primjer paralelizma se javlja u programu kada se naredbe pridruživanja varijablama pojavljuju nakon naredbe koje se pozivaju na te varijable, primjerice u programu:

```
height = 0
width = 0
area = height * width
height= input("height:")
width = input("width:")
print(area)
```

Učenici očekuju kako će program ispisati umnožak vrijednosti visine i širine koje korisnik unese. Međutim, kada se izvrši treći red u kojem se area definira kao umnožak visine i širine, tada program još nije primio ulazne vrijednosti visine i širine, već ih tretira kao 0. Stoga je rezultat umnoška 0 * 0 = 0 i to će biti ispisano.

Navedeni primjer je miskoncepcija paralelizma jer se linije koda izvršavaju redom po kojem su napisane, a ne istovremeno. Učenici koji ne razumiju redoslijed izvršavanja programa mogu pomisliti da će vrijednost varijable biti ažurirana prije nego se koristi u drugim izjavama, što dovodi do pogrešnih rezultata.

U prirodnom jeziku, osim proceduralnih uputa poput recepata ili građevinskih planova, često nam redoslijed nije bitan. Međutim, pri programiranju, učenik mora razmišljati: „Koje su vrijednosti ulaznih podataka važeće prilikom izvršavanja ove linije?". U prirodnom jeziku rijetko narušavamo značenje teksta čitanjem dijelova izvan redoslijeda, što nije slučaj u programiranju (vidi [14]).

Sljedeća miskoncepcija je intencionalizam. Intencionalizam čine učenici kada programu dodjeljuju namjenu koja prelazi informacije dane u linijama programskog koda. Oni pretpostavljaju da će program izvršiti nešto što nije eksplicitno navedeno i pripisuju mu ljudske karakteristike. Navest ćemo jedan primjer intencionalizma koji je detaljnije istražen i opisan u [14]. Autori su zamolili učenike da predvide ponašanje sljedećeg rekurzivnog programa u programskom jeziku Logo:

```
to shape :side
if :side = 10 [stop]
repeat 4 [fd :side rt 90]
shape :side/2
end
```

pri pozivu

```
shape 40
```

Neki učenici misle kako će navedeni program nacrtati kvadrat koji ima duljinu stranice 10 nakon što ga pokrenu. Njihov zaključak proizlazi iz načina na koji pristupaju programskom kodu. Kada dođu do linije koda koja sadrži grananje, učenici unaprijed pogledaju sljedeću liniju koda kako bi vidjeli što ona radi. U ovom slučaju, to je crtanje kvadrata. Zatim se vraćaju na grananje i tumače je tako da će program nacrtati kvadrat koji ima duljinu stranice 10 jer „program želi nacrtati kvadrat". S druge strane, drugi dio učenika ipak je prepoznao točnu vrijednost varijable side u grananju koja je na početku bila 40, pa 20 i na kraju 10 kada se program zaustavio.

U ovoj miskoncepciji, kao i u slučaju paralelizma, učenici programu pripisuju osobine živih bića koja posjeduju namjere, ciljeve te sposobnost znanja i unutarnjeg viđenja budućih događaja. Oni pretpostavljaju kako program ima svijest o svojim koracima te da će svjesno izvršiti određene postupke kako bi postigao željeni rezultat.

Miskoncepcija egocentrizma je suprotna strana intencionalizma i odnosi se na pisanje programa iz vlastite perspektive, pretpostavljajući da računalo razumije našu namjeru i način razmišljanja. Umjesto da se usredotočimo na objektivne zahtjeve i logiku programa, egocentrični pristup se temelji na vlastitim očekivanjima i razumijevanju.

Na primjer, učenici izostavljaju linije koda ili određene podatke jer pretpostavljaju da računalo zna što treba napraviti. Učenici ne tvrde doslovno da program zna što treba učiniti. Pogreške koje stvara ova miskoncepcija gotovo su perceptivne prirode - učenikove trenutne koncepcije ne usmjeravaju njegovu pozornost na te probleme kao relevantne razloge zašto njegov program ne radi kako je planirano. Uobičajeni problem ove vrste je izostavljanje interpunkcijskih znakova ili izostavljanje vrijednosti za varijable.

Autori u [14] htjeli su potvrditi postojanje ove miskoncepcije pa su zadali učenicima zadatak kojim bi to mogli provjeriti. Zadatak učenika je bio napisati program u Logu koji crta kvadrat s duljinom stranice 30. Učenici su ponudili rješenje u kojem su iskoristili naredbu za pomicanje kornjače naprijed, ali su zanemarili naredbu za okretanje koja je potrebna za stvaranje pravih kutova u kvadratu:

```
repeat 4[fd 30]
```

Učenici nisu svjesni da računalo nema sposobnost popunjavanja ili razumijevanja nedostajućih dijelova koda na način na koji oni to misle. Oni su dali samo osnovnu strukturu programa, pretpostavljajući da će računalo na neki način interpretirati njihove namjere i ispuniti preostale dijelove koda.

Naveli smo neke od miskoncepcija koji se pojavljuju na početku učenja programiranja. Postoje i druge miskoncepcije u programiranju koje vežemo za određene pojmove poput miskoncepcije o funkcijama ili miskoncepcije za objektno orijentirano programiranje (više u [20]).

Literatura

[1] *Bebras (Dabar)*, URL: https://www.bebras.org/.

[2] J. Blagus, N. Ljubić Klemše, I. Ružić, M. Stančić, *e-SVIJET 4 - radni udžbenik informatike s dodatnim digitalnim sadržajima u četvrtom razredu osnovne škole*, Školska knjiga, Zagreb, 2020.

[3] N. Bubica, I. Boljat, *Assessment of Computational Thinking – A Croatian Evidence-Centered Design model*, Informatics in education, **21**(2022), 425-463.

[4] L. Budin, P. Brodanac, Z. Markučić, S. Perić, E. Wendling,*Informatika 4, udžbenik za 4. razred gimnazija (2 ili 3 sata nastave tjedno)*, Element, Zagreb, 2021.

[5] *CS Unplugged*, URL: https://www.csunplugged.org/en/.

[6] N. Hajdin, *Logičke igre u razrednoj nastavi*, Sveučilište u Rijeci, Učiteljski fakultet, Diplomski rad, 2020.

[7] D. Hickmott, E. Prieto-Rodriguez, K. Holmes, *A Scoping Review of Studies on Computational Thinking in K–12 Mathematics Classrooms*, Digital Experiences in Mathematics Education, **4**(2018), 48–69.

[8] *Hrvatska Logo Liga*, URL: https://logoliga.hsin.hr/.

[9] C. Izu, C. Mirolo, A. Settle, L. Mannila, G. Stupurienė, , *Exploring Bebras Tasks Content and Performance: A Multinational Study*, Informatics in Education, **16**(2017), 39-59.

[10] *Klokan*, URL: https://matematika.hr/klokan.

[11] L. Mannila, M. de Raadt, *An objective comparison of languages for teaching introductory programming*, Baltic Sea '06: Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006, Association for Computing Machinery, 2016, 32-37.

[12] A. Milošević, *Analiza učenja osnovnih koncepata programiranja*, Sveučilište u Splitu, Prirodoslovno-matematički fakultet, Diplomski rad, 2021.

[13] M. Mladenović, *Poučavanje početnog programiranja oblikovanjem računalnih igara*, Sveučilište u Splitu, Prirodoslovno-matematički fakultet, Disertacija, 2019.

[14] R. Pea, *Language-independent conceptual "bugs" in novice programming*, Journal educational computing research, **2**(1986), 25-36.

[15] M. Radošević, *Poželjne karakteristike jezika za poučavanje programiranja u osnovnoj školi*, Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet, Diplomski rad, 2017.

[16] V. J. Shute, C. Sun, J. Asbell-Clarke, *Demystifying computational thinking*, Educational Research Review, **22**(2017), 142-158.

[17] B. Standl, *Solving everyday challenges in a computational way of thinking*, Informatics in Schools: Focus on Learning Programming, Springer, 2017, 180–191.

[18] K. Tomljenović, *Računalno razmišljanje i uloga učenja pomoću igre na njegov razvoj*, Učiteljski fakultet Sveučilišta u Zagrebu, Kvalifikacijski rad, 2018.

[19] B. Vöcking, H. Alt, M. Dietzfelbinger, R. Reischuk, C. Scheideler, H. Vollmer, D. Wagner, *Algorithms Unplugged*, Springer Berlin, Heidelberg, 2011.

[20] Ž. Žanko, *Miskoncepcije u uvodnoj nastavi programiranja*, Sveučilište u Splitu, Prirodoslovno-matematički fakultet, Seminar, 2018.

Sažetak

Cilj ovog rada je detaljno prikazati što je sve potrebno za uspješno poučavanje programiranja. Programiranje uključuje razvoj računalnog razmišljanja i vještinu pisanja koda u odabranom programskom jeziku. U radu su izloženi aspekti računalnog razmišljanja, te su dani konkretni primjeri za poticanje razvoja računalnog razmišljanja kod učenika, izbjegavajući standardne zadatke iz udžbenika. Predstavljeni su neki od programskih jezika namijenjeni poučavanju programiranja - Scratch, Python i Logo. Navedeni su i objašnjeni kriteriji za procjenu prikladnosti programskog jezika za poučavanje programiranja. Kriteriji omogućavaju usporedbu programskih jezika i donošenje zaključaka o njihovoj prikladnosti za poučavanje. Važno je naglasiti da se tijekom procesa poučavanja programiranja često javljaju pogrešna shvaćanja određenih koncepata, poznata kao miskoncepcije. Zbog toga su u radu iznesene i opisane česte miskoncepcije koje se javljaju kod učenika prilikom prvog susreta s programiranjem.

Ključne riječi: programiranje, računalno razmišljanje, programski jezici, miskoncepcije

Teaching programming

Summary

The aim of this paper is to present in detail what is necessary for successful teaching of programming. Programming involves the development of computational thinking and the skill of writing code in a chosen programming language. Aspects of computational thinking are presented in the paper, and concrete examples are given for encouraging the development of computational thinking in students, avoiding standard tasks from textbooks. Some of the programming languages intended for teaching programming are presented - Scratch, Python and Logo. The criteria for evaluating the suitability of a programming language for teaching programming are listed and explained. The criteria make it possible to compare programming languages and draw conclusions about their suitability for teaching. It is important to emphasize that during the process of teaching programming, misunderstandings of certain concepts, known as misconceptions, often occur. For this reason, the paper presents and describes common misconceptions that occur among students during their first encounter with programming.

Keywords: programming, computational thinking, programming languages, misconceptions

Životopis

Ivana Tucak-Roguljić rođena je 21.veljače 2000. godine u Osijeku. Pohađala je Prirodoslovno-matematičku gimnaziju u istom gradu. Nakon završetka srednje škole, nastavlja svoje obrazovanje na Odjelu za matematiku u Osijeku, koji danas nosi naziv Fakultet za primijenjenu matematiku i informatiku. Godine 2021. stječe zvanje prvostupnice matematike. Trenutno pohađa diplomski nastavnički studij matematike i informatike. Tijekom studiranja, sudjelovala je u projektu „Kreativna STEM revolucija u Slavoniji" kao članica tima Odjela za matematiku.

Bila je članica Studentskog zbora Odjela za matematiku i Studentskog zbora Sveučilišta J.J. Strossmayera u Osijeku te je za vrijeme članstva imala priliku osmisliti i provesti nekoliko projekata koji su bili usmjereni na podršku i edukaciju studenata. Među njima se ističu Mathos AI Hackathon i studentska grupa podrške „iSTRESi se".

U slobodno vrijeme, volontirala je u udruzi Klikeraj, koja se bavi radom s darovitom djecom. Kasnije postaje članicom iste i sudjeluje u projektu „Razvoj djece i mladih kroz STEM područje".