Kreiranje Pearsonovih slagalica za učenje programskog jezika Python

Perišin, Martina

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

Kreiranje Pearsonovih slagalica za učenje programskog jezika Python

Martina Perišin

Split, rujan 2024.

Temeljna dokumentacijska kartica

Diplomski rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Kreiranje Pearsonovih slagalica za učenje programskog jezika Python

Martina Perišin

SAŽETAK

Ovaj rad istražuje razvoj i primjenu različitih metoda poučavanja programiranja u obrazovnim ustanovama, s posebnim naglaskom na povijesne perspektive i inovativne pristupe koji koriste suvremene tehnologije i interaktivne tehnike. Kroz pregled povijesti računalnog programiranja u obrazovanju, identificirani su ključni momenti i promjene koje su oblikovale način na koji se programiranje danas podučava.

Tradicionalne metode, koje uključuju predavanja, demonstracije i vježbe, dugo su bile temelj obrazovnih kurikuluma. Iako su ove metode učinkovite u prenošenju osnovnih znanja, često se pokazale nedovoljno angažirajućima za učenike i neadekvatnima za razvoj kritičkog mišljenja i praktičnih vještina potrebnih u stvarnom svijetu.

S druge strane, inovativne metode poučavanja, poput preokrenute učionice, učenja temeljenog na problemima, učenja temeljenog na upitima, integracije gamifikacije, korištenja alata poput Parsonovih slagalica i virtualnih laboratorija, nude nove mogućnosti za poboljšanje iskustva učenja. Ove metode potiču aktivno sudjelovanje, kolaboraciju među učenicima, praktičnu primjenu znanja i razvoj dubokog razumijevanja programskih koncepata.

Parsonove slagalice, koje su detaljno obrađene u ovom radu, predstavljaju poseban oblik učenja programiranja koji kombinira elemente igre i izazova, čime se povećava angažman učenika.

Kroz različite varijacije, uključujući osnovne i napredne zadatke, one pomažu učenicima u savladavanju sintakse i logičkog toka programa na zanimljiv i učinkovit način.

Zaključno, evolucija metoda poučavanja programiranja odražava širu promjenu u obrazovanju prema interaktivnijem pristupu orijentiranom prema učeniku. Ove inovativne metode ne samo da poboljšavaju ishode učenja, već također čine proces učenja programiranja pristupačnijim i zanimljivijim za širi krug učenika. Stoga je važno nastaviti istraživanje i implementaciju ovih metoda kako bi se osigurala kvaliteta i relevantnost obrazovanja u dinamičnom polju računalnih znanosti.

Ključne riječi: učenje programiranja, metode podučavanja, inovativna pedagogija, Parsonove slagalice, informatika, kurikulum, tehnologije učenja, Python

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: [38] stranica, [27] grafičkih prikaza i [8] literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **prof. dr. sc. Ani Grubišić,** *Prirodoslovno-matematički fakultet u Splitu*

Ocjenjivači: **prof. dr. sc. Ani Grubišić,** *Prirodoslovno-matematički fakultet u Splitu*

prof. dr. sc. Branko Žitko, *Prirodoslovno-matematički fakultet u Splitu*

Ines Šarić-Grgić, dipl.ing., *Prirodoslovno-matematički fakultet u Splitu*

Rad prihvaćen: **rujan 2024.**

Basic documentation card

Graduation thesis

University of Split
Faculty of Science
Department of computer science
Ruđera Boškovića 33, 21000 Split, Croatia

Creating Pearson puzzles for learning Python programming language

Martina Perišin

ABSTRACT

This paper explores the development and application of various methods for teaching programming in educational institutions, with a particular emphasis on historical perspectives and innovative approaches that utilize modern technologies and interactive techniques. Through a review of the history of computer programming in education, key moments and changes that have shaped the way programming is taught today have been identified.

Traditional methods, including lectures, demonstrations, and exercises, have long been the cornerstone of educational curricula. While these methods are effective in conveying fundamental knowledge, they have often been found to be insufficiently engaging for students and inadequate for developing critical thinking and practical skills needed in the real world.

On the other hand, innovative teaching methods, such as flipped classrooms, problem-based learning, inquiry-based learning, gamification integration, the use of tools like Parsons puzzles and virtual laboratories, offer new opportunities to enhance the learning experience. These methods encourage active participation, collaboration among students, practical application of knowledge, and the development of a deep understanding of programming concepts.

Parsons puzzles, which are extensively discussed in this paper, represent a special form of learning programming that combines elements of play and challenge, thereby increasing student engagement. Through various variations, including basic and advanced tasks, they help students master the syntax and logical flow of programs in an interesting and effective way.

In conclusion, the evolution of programming teaching methods reflects a broader shift in education towards a more interactive, student-centered approach. These innovative methods not only improve learning outcomes but also make the process of learning programming more accessible and engaging for a wider range of students. Therefore, it is important to continue researching and implementing these methods to ensure the quality and relevance of education in the dynamic field of computer science.

Key words: programming education, teaching methods, innovative pedagogy, parsons puzzles, computer science, curriculum, learning technologies, Python

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: [38] pages, [27] figures and [8] references

Original language: Croatian

Mentor: **Ani Grubišić, Ph.D.** *full professor at Faculty of Science, University of Split*
Reviewers: **Ani Grubišić, Ph.D.** *full professor at Faculty of Science, University of Split*
Branko Žitko, Ph.D. *full professor at Faculty of Science, University of Split*
Ines Šarić-Grgić, *assistent at Faculty of Science, University of Split*

Thesis accepted: **September, 2024**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam diplomski rad s naslovom „Kreiranje Pearsonovih slagalica za učenje programoskog jezika Python“ izradila samostalno pod voditeljstvom prof. dr. sc. Ani Grubišić. U radu sam primijenila metodologiju znanstvenoistraživačkog rada i koristila literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući navela u diplomskom radu na uobičajen, standardan način citirala sam i povezala s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Studentica

Martina Peršin

1 Sadržaj

Uvod

U brzo razvijajućem digitalnom dobu, sposobnost razumijevanja i pisanja računalnih programa postaje ključna vještina. Kako tehnologija nastavlja prožimati svaki aspekt svakodnevnog života, rano upoznavanje s konceptima programiranja može djeci pružiti vrijedne vještine koje nadilaze područje računalstva. Ovaj rad istražuje inovativne metode poučavanja programiranja, s posebnim naglaskom na Parsonove slagalice i programski jezik Python.

Parsonove slagalice, nazvane po računalnom znanstveniku Daleu Parsonsu, oblik su programskog zadatka u kojem se učenicima pružaju pomiješane linije koda koje moraju posložiti u ispravan redoslijed kako bi riješili zadani problem. Ovaj pristup naglašava logičko razmišljanje i vještine rješavanja problema, dok minimalizira početni strah koji može pratiti pisanje koda od nule (Ericson, Margulieux, & Rick, 2017). Kroz rad s Parsonovim slagalicama, učenici mogu steći temeljno razumijevanje programskih konstrukcija i principa na pristupačniji i manje frustrirajući način.

Posljednjih godina, računalno programiranje postalo je sastavni dio kurikuluma. Intenzivna istraživanja i rasprave unutar obrazovne zajednice računalnih znanosti usredotočena su na pronalaženje najbolje paradigme programiranja, jezika i filozofskog pristupa poučavanju programiranja (Fincher, 1999). Iako su ta pitanja ključna, učenje programiranja dijeli mnoge sličnosti s učenjem drugih disciplina. Na primjer, učenje programskog jezika je slično učenju novog govornog jezika, gdje je potrebno savladati sintaksu i semantiku (Ericson, Margulieux, & Rick, 2017). Također, podsjeća na rano učenje matematike, gdje se razlikuju osnovne strukture i razvoj vještina apstraktnog mišljenja.

Python, poznat po svojoj jednostavnosti i čitljivosti, idealan je programski jezik za početnike. Njegova jednostavna sintaksa i opsežne knjižnice čine ga izvrsnim izborom za obrazovne svrhe. Zbog svoje fleksibilnosti, Python je pogodan za različite razine znanja i stilove učenja, što ga čini izvrsnim alatom u kombinaciji s Parsonovim slagalicama, pružajući strukturirano, ali prilagodljivo okruženje koje djeci omogućuje postupan napredak u učenju programiranja.

Sadržaj prvog tečaja programiranja nužno se mora u određenoj mjeri usredotočiti na usvajanje sintaktičkih pravila i osnovnih programskih struktura (Ericson, Margulieux, & Rick, 2017). Programi se ne mogu pisati bez dobrog razumijevanja sintaktičkih pravila jezika. Tradicionalne metode učenja, poput ponavljanja i vježbanja, koje se koriste u poučavanju matematike i stranih jezika, mogu se također primijeniti na učenje programiranja.

Međutim, vježbe ponavljanja mogu biti dosadne i neophodno je osigurati angažman učenika kako bi ove metode bile učinkovite. Također, izoliranje sintaktičkih jedinica iz logičkog konteksta može biti izazovno, što otežava učenje ispravnog korištenja sintakse u složenijim programskim zadacima.

Cilj ovog rada je istražiti učinkovitost Parsonovih slagalica kao pedagoškog alata u poučavanju programiranja početnika. Kroz kombinaciju pregleda literature i praktične primjene, rad će procijeniti kako ove slagalice mogu poboljšati razumijevanje, zadržavanje znanja i angažman učenika. Također, istraživat će se potencijalni izazovi i ograničenja ovog pristupa, nudeći uvide i preporuke za učitelje koji žele uvrstiti Parsonove slagalice u svoje obrazovne strategije.

Na kraju ovog istraživanja, očekuje se da će Parsonove slagalice biti prikazane kao vrijedno dopunsko sredstvo za poučavanje programiranja s krajnjim ciljem da proces učenja programiranja postane pristupačnije i ugodnije za nadolazeće generacije.

1. Pristupi učenju programiranja

Programiranje zahtijeva usvajanje niza različitih, ali nužnih vještina, što predstavlja izazove i za učenike i za učitelje. Učenici moraju usvojiti poznavanje sintakse i konstrukcije jezika, naučiti kako rastaviti probleme i dizajnirati algoritme, prepoznati i primijeniti korisne obrasce programiranja te razviti sposobnost čitanja i razumijevanja koda na različitim razinama složenosti. Tečajevi programiranja često očekuju da učenici nauče ove vještine pišući kod, ali složenost ovog zadatka može biti prezahtjevna za početnike. Učitelji i znanstvenici su stoga uložili značajne napore istražujući raznolike pedagoške pristupe i aktivnosti kako bi pomogli učenicima u razvoju ovih neophodnih vještina (Parsons & Haden, 2006).

Ključni trenuci u obrazovanju računalnog programiranja obuhvaćaju implementaciju novih nastavnih metoda, istraživačkih projekata te inicijativa koje su promicale inkluzivnost i dostupnost računalnog obrazovanja za sve učenike. Ovo razdoblje obilježeno je pomicanjem fokusa s pasivnog konzumiranja informacija na aktivno sudjelovanje u stvaranju digitalnih sadržaja te poticanje kritičkog razmišljanja i rješavanja problema.

Utjecaj računalnog programiranja na obrazovni sustav proteže se izvan klasičnih granica učionice. Uvođenje programiranja u nastavu potiče razvoj digitalnih vještina, kreativnosti i inovativnosti kod učenika, pripremajući ih za buduće izazove i mogućnosti koje nosi digitalno doba (Parsons & Haden, 2006). Također, računalno programiranje oblikuje buduće karijere i profesionalne putove mladih generacija, omogućujući im da postanu aktivni sudionici u digitalnom društvu.

1.1 Tradicionalne metode

Tradicionalne metode poučavanja programiranja oslanjaju se na klasične pristupe obrazovanju koje se već desetljećima koriste u raznim područjima. Ove metode uključuju (Parsons & Haden, 2006):

Predavanja

Jedna od najčešćih tradicionalnih metoda su predavanja. Učitelji drže predavanja pred cijelom grupom učenika, objašnjavajući teoretske koncepte i prikazujući primjere na ploči ili kroz prezentacije. Predavanja omogućavaju prenošenje velike količine informacija u kratkom vremenu, no često imaju ograničenu interakciju s učenicima. Učenici su pasivni primatelji informacija, što može rezultirati problemima s pažnjom i razumijevanjem gradiva.

Udžbenici i pisani materijali

Udžbenici i pisani materijali također su važan dio tradicionalnog poučavanja. Učenici koriste udžbenike, skripte i druge pisane materijale za učenje programskih jezika i koncepata. Ovi materijali osiguravaju samostalno učenje i nude strukturirani sadržaj. Međutim, nedostatak interaktivnosti i ograničena mogućnost primjene teorije u praksu predstavljaju značajne nedostatke ovog pristupa.

Vježbe na papiru

Još jedna tradicionalna metoda su vježbe na papiru. Učenici rješavaju zadatke i probleme na papiru, uključujući pisanje koda ručno. Ova metoda pomaže u razumijevanju sintakse i logičkog razmišljanja, ali je nepraktična u odnosu na stvarno programiranje na računalu i nema trenutne povratne informacije.

Laboratorijske vježbe

Laboratorijske vježbe su ključne za stjecanje praktičnog iskustva. Učenici provode vježbe u računalnim laboratorijima, pišući i izvršavajući kod na računalima. Ovaj pristup omogućava praktično iskustvo s programiranjem i primjenu teorije u praksu, no često je ograničen nadzor učitelja i mogući su tehnički problemi.

Testovi i ispiti

Na kraju, testovi i ispiti su standardna metoda procjene znanja. Učenici se ocjenjuju putem testova i ispita koji procjenjuju njihovo znanje sintakse i teorije programiranja. Ova metoda omogućava kvantitativnu evaluaciju znanja, ali može izazvati stres kod učenika i ne reflektira uvijek praktične vještine programiranja.

1.2 Inovativne metode podučavanja

Inovativne metode poučavanja programiranja uključuju suvremene pristupe koji koriste tehnologiju i interaktivne tehnike kako bi poboljšali učenje. Tradicionalni pristupi često se oslanjaju na pasivno slušanje predavanja i individualno rješavanje zadataka, što može biti nedovoljno za razvoj dubljeg razumijevanja programskih koncepata i praktičnih vještina. S druge strane, inovativne metode naglašavaju aktivno sudjelovanje studenata, kolaboraciju, kritičko razmišljanje i primjenu znanja u stvarnim situacijama (Parsons & Haden, 2006). Ove

metode ciljaju na povećanje motivacije, angažiranosti i uspješnosti studenata u učenju programiranja.

Igrifikacija

Igrifikacija (engl. gamification) koristi elemente igara, poput bodova, znački i rang-lista, kako bi povećala motivaciju i angažman učenika. Ova metoda može učiniti učenje programiranja zabavnijim i izazovnijim, potičući učenike da aktivno sudjeluju i natječu se međusobno. Igrifikacija je osobito korisna u održavanju interesa i motivacije kod mlađih učenika.

Interaktivni udžbenici

Interaktivni udžbenici kombiniraju tekst, multimedijalne elemente i interaktivne kodne zadatke. Učenici mogu čitati o konceptima programiranja, gledati videozapise s objašnjenjima i odmah rješavati kodne zadatke unutar istog okruženja. Ova metoda omogućava učenicima da uče vlastitim tempom i odmah primijene naučeno.

Online tečajevi i platforme za učenje

Online tečajevi i platforme kao što su Coursera, edX i Codecademy pružaju pristup širokom spektru programskih tečajeva. Učenici mogu učiti od vrhunskih učitelja iz cijelog svijeta, rješavati zadatke i sudjelovati u diskusijama s vršnjacima. Ova metoda omogućava fleksibilnost u učenju i pristup različitim resursima.

Učenje uz pomoć projekata

Učenje uz pomoć projekata uključuje učenike u stvaranje stvarnih programskih projekata kao dijela kurikuluma. Učenici rade na problemima iz stvarnog svijeta, što im pomaže da razviju praktične vještine i iskustvo. Ova metoda potiče kreativnost, kritičko razmišljanje i sposobnost rješavanja problema.

Računalno potpomognuto učenje

Računalno potpomognuto učenje koristi softverske alate i aplikacije za podršku učenju programiranja. To može uključivati simulatore, razvojne okoline i automatizirane sustave za ocjenjivanje. Ovi alati omogućuju učenicima da eksperimentiraju i dobiju povratne informacije u stvarnom vremenu.

Kolaborativno učenje

Kolaborativno učenje uključuje rad u grupama gdje učenici surađuju na rješavanju programskih zadataka. Ova metoda potiče timski rad, komunikacijske vještine i dijeljenje znanja među

učenicima. Kolaborativno učenje može uključivati programiranje u parovima, rad na zajedničkim projektima ili sudjelovanje u online forumima.

Mikroučenje

Mikroučenje se fokusira na pružanje malih, lako probavljivih jedinica sadržaja koje učenici mogu konzumirati u kratkom vremenskom razdoblju. Ova metoda omogućava fleksibilno učenje koje se može uklopiti u užurbane rasporede. Mikroučenje može uključivati kratke video lekcije, kvizove i interaktivne zadatke.

Upotreba virtualne i proširene stvarnosti

Virtualna i proširena stvarnost koriste se za stvaranje interaktivnih i imerzivnih okruženja za učenje programiranja. Učenici mogu programirati unutar virtualnih svjetova, što može učiniti učenje intuitivnijim i vizualno privlačnijim. Ova metoda također može omogućiti simulaciju složenih scenarija koji bi bili teško izvedivi u stvarnom svijetu.

Adaptivno učenje

Adaptivno učenje koristi tehnologiju za prilagodbu obrazovnog sadržaja potrebama pojedinog učenika. Sustavi za adaptivno učenje mogu analizirati performanse studenata i prilagoditi zadatke, pružiti dodatne resurse ili promijeniti razinu težine kako bi se optimalno podržao napredak studenata.

Ove inovativne metode pružaju raznolike pristupe poučavanju programiranja, prilagođavajući se različitim stilovima učenja i potrebama studenata. Kombinacija ovih metoda može značajno poboljšati učinkovitost i angažman u učenju programiranja.

2. Pearsonove slagalice

Parsonove slagalice, poznate i kao Parsonovi problemi, predstavljaju specifičan oblik obrazovnog zadatka koji se koristi u poučavanju računalnog programiranja. Osmišljene su kako bi pomogle učenicima u razumijevanju i savladavanju sintakse i logičkog toka programskog jezika. Ovi zadaci uključuju slaganje ili preuređivanje dijelova koda u ispravan redoslijed kako bi se postigao funkcionalan program. Svaka slagalica obično se sastoji od fragmenta koda koji su ispravno napisani, ali su pomiješani i potrebno ih je rasporediti na pravi način (Ericson, Margulieux, & Rick, 2017).

Parsonove slagalice često započinju s jasnim opisom problema koji učenici trebaju riješiti. Opis uključuje ciljeve koje treba postići i uvjete koje ispravno rješenje mora zadovoljiti. Fragmenti koda, koji se učenicima daju na raspolaganje, mogu sadržavati potpune linije koda, dijelove linija ili čak dijelove logičkih struktura poput petlji i uvjetnih iskaza. Zadatak studenata je prepoznati ispravne dijelove, ignorirati eventualne distraktore (dijelove koji ne pripadaju rješenju) i rasporediti kod u ispravnom redoslijedu (Shah, 2020).

Jedna od ključnih karakteristika Parsonovih slagalica je da omogućavaju učenicima interaktivno i angažirano učenje. Ove slagalice mogu biti prezentirane u različitim formatima, uključujući digitalne platforme gdje učenici koriste funkciju "drag-and-drop" za preuređivanje koda. Ovakav interaktivan pristup pomaže u smanjenju kognitivnog opterećenja jer učenici mogu fokusirati svoju pažnju na razumijevanje logike i strukture koda, umjesto na memoriranje sintakse (Shah, 2020).

Parsonove slagalice također omogućavaju pružanje neposredne povratne informacije. Kada učenici pogrešno slože kod, sustavi za učenje često pružaju sugestije ili ističu greške, što im omogućuje da odmah korigiraju svoje rješenje i uče iz svojih pogrešaka. Ova vrsta povratne informacije može biti bazirana na izvršavanju koda (gdje sustav pokušava pokrenuti složeni program) ili može biti linijski orijentirana, gdje se specifične greške u fragmentima koda označavaju kako bi ih učenici lakše uočili (Ihantola & Karavirta, 2011).

Osim toga, Parsonove slagalice se mogu prilagoditi različitim razinama težine i kompleksnosti, što ih čini prikladnim alatom za učenje od početničke do napredne razine. Mogu se koristiti za uvodno učenje programiranja, ali i za kompleksnije zadatke koji zahtijevaju dubinsko razumijevanje algoritama i strukture podataka. Primjena ovih zadataka može se proširiti i na različite programske jezike, omogućujući učenicima prilagodbu različitim sintaksama i stilovima programiranja.

2.1 Podrijetlo Pearsonovih slagalica

Podrijetlo Parsonovih slagalica, kako su danas poznate, može se pratiti do rada iz 2006. godine (Parsons & Haden, 2006) (Shah, 2020). Rad pod naslovom "Parsonove programerske slagalice: Zabavan i učinkovit alat za učenje u početnim tečajevima programiranja" predstavljen je na Australasijskoj konferenciji o obrazovanju računalstva (ACE) te godine u Hobartu, Australija. U ovom pionirskom radu, Parsons i Haden su naglasili važnost prakse za savladavanje sintakse i semantike programiranja, ali su primijetili dva problema s tipičnim vježbama. Prvi problem bio je da su takve vježbe često dosadne, što je otežavalo učenicima ustrajanje u praksi. Drugi problem bio je izazov izoliranja ključnih aspekata sintaktičke prakse od koncepcijskih elemenata rješavanja problema u programiranju. Oni su predložili ideju Parsonovih slagalica kako bi pomogli učenicima zapamtiti sintaktičke konstrukte dok uče o algoritmima i logičkom slijedu, te kako bi vježbe bile zanimljive i motivirajuće za učenike. Dizajn ovih slagalica, kako su ga opisali Parsons i Haden, slijedio je pet principa: maksimiziranje angažmana, ograničavanje logike, dopuštanje uobičajenih grešaka, modeliranje dobrog koda i pružanje trenutnih povratnih informacija.

2.2 Karakteristike Pearsonovih slagalica

U nastavku će se definirati standardne karakteristike Pearsonovih slagalica. Ove karakteristike mogu se koristiti za opisivanje, ako ne i definiranje, tipičnih primjera Parsonovih slagalica te za razlikovanje od drugih pristupa programiranju (Shah, 2020).

Svrha i uporaba

Parsonove slagalice koriste se u nastavi i imaju pedagošku svrhu. Često su dio kurikuluma tečajeva i mogu se koristiti za podržavanje učenja (što je češće) i za ocjenjivanje (što je rjeđe).

Opis problema

Parsonove slagalice sadrže opis problema koji treba riješiti. Uvijek postoji cilj za učenike, a postizanje tog cilja je često provjerljivo. Izjave problema su usredotočene i imaju eksplicitne ciljeve (tj. Parsonove slagalice nisu otvorenog tipa).

Atomarnost

Nudi se ograničen broj fragmenata. Svaki fragment obično predstavlja jednu liniju koda, ali fragment može sadržavati više linija. Iako rjeđe, fragmenti mogu biti i dijelovi linije koda.

Prostor problema

Prostor problema ograničen je brojem dostupnih fragmenata i obično ne dopušta njihovo ponovno korištenje. Međutim, Parsonove slagalice mogu sadržavati ometajuće fragmente (engl. distractors) koje ne treba koristiti, kao i mogućnosti dovršavanja fragmenata (npr. unosom vrijednosti u varijable ili upisivanjem dijelova fragmenta).

Konstrukcija rješenja

Parsonove slagalice obično započinju s praznim prostorom za rješenje u koji se fragmenti postavljaju redoslijedom. Neke Parsonove slagalice koriste samo jedno područje u kojem je cilj preurediti fragmente unutar tog područja.

Ispravnost i povratne informacije

Parsonove slagalice obično imaju ispravno rješenje definirano jasnim kriterijima. Često se mogu automatski ocjenjivati. Neki sustavi zahtijevaju da učenik riješi problem u određenom broju poteza ili pokušaja da bi rješenje bilo ispravno. Parsonove slagalice koje se koriste za vježbanje obično pružaju trenutne povratne informacije o učenikovom rješenju. Postoje dvije vrste povratnih informacija: temeljene na izvršavanju koda i temeljene na linijama. Povratne informacije temeljene na izvršavanju daju se izvođenjem koda, dok se povratne informacije temeljene na linijama obično daju označavanjem jednog ili više fragmenata kako bi se naznačilo da su netočni ili na pogrešnom mjestu.

Način rada i korisničko sučelje

Parsonove slagalice se obično rade u interaktivnom okruženju s funkcijom povuci i ispusti. To omogućuje njihovu upotrebu i u preglednicima i u mobilnim okruženjima. Fragmenti u Parsonovim slagalicama često se postavljaju na mjesto kada se pomaknu, ne dopuštajući praznine između fragmenata. Fragmenti mogu sadržavati mjesta u koja se može unositi tekst (npr. vrijednosti varijabli). Neka okruženja izvršavaju konstruirane programe u tekstualnom

okruženju i prikazuju izlaz liniju po liniju, dok druga okruženja imaju grafički prikaz (npr. Turtle graphics).

Sintaksa

Kada su Parsonove slagalice povezane s programskim jezikom, obično koriste sintaksu tog programskog jezika. Program prikazan pomoću Parsonovih slagalica može biti sintaktički neispravan. Sintaktički neispravna rješenja pojavljuju se na dva načina: (1) fragmenti koji su poredani tako da rezultirajuća sintaksa nije ispravna, i (2) ometajući fragmenti koji su sintaktički neispravni.

Podrška u učenju

Parsonove slagalice predstavljaju oblik podrške u učenju programiranja (engl. *scaffolding*). Mogu uključivati dodatnu podršku (npr. na platformi Runestone ebook, kada učenici zatraže pomoć na Parsonovom problemu, sustav može ukloniti ometajući fragment ili kombinirati fragmente).

Prikladnost i očekivano vrijeme rada

S pedagoškog stajališta, Parsonove slagalice su namijenjene da budu korisne za učenje učenika. To se odražava u dizajnu zadataka. Prezentirani zadaci trebaju biti prikladni za trenutnu razinu učenika; rješavanje zadatka obično zahtijeva od jedne minute do manje od deset minuta, ovisno o broju fragmenata i složenosti problema.

2.3 Vrste Pearsonovih slagalica

Različite vrste Parsonovih slagalica mogu biti prilagođene različitim obrazovnim ciljevima, od uvodnih vježbi programiranja do složenijih zadataka rješavanja problema, čineći ih svestranim alatima u obrazovanju računalnih znanosti.

U nastavku su navedene neke od najpoznatijih vrsta Pearsonovih slagalica uključujući i primarnu svrhu zbog kojih se koriste.

Osnovno preuređivanje linija (engl. *Basic Line Reordering*)

Opis: Učenicima se daje skup linija koda u nasumičnom poretku te ih moraju preurediti kako bi formirali ispravan i funkcionalan program (Parsons & Haden, 2006).

Svrha: Pomaže učenicima razumjeti logički tijek i strukturu programa.

Primjer: Postavljen je jednostavan zadatak gdje treba izračunati zbroj dva broja i ispisati rezultat. Program je već napisan, ali linije su u pogrešnom redoslijedu (Slika 1). Zadatak je preurediti linije kako bi program ispravno funkcionirao (Slika 2).

```
y = int(input('Unesi drugi broj: '))
zbroj = x + y
print('Zbroj dva broja je: ',zbroj)
x = int(input('Unesi prvi broj: '))
```

Slika 1 Početni popis naredbi

```
x = int(input('Unesi prvi broj: '))
y = int(input('Unesi drugi broj: '))
zbroj = x + y
print('Zbroj dva broja je: ',zbroj)
```

Slika 2 Točno rješenje

Preuređivanje ugniježđenih struktura (engl. *Nested Structure Reordering*)

Opis: Slično osnovnom preuređivanju linija, ali uključuje gniježđene strukture poput petlji i uvjetnih naredbi. Učenici moraju ispravno poredati gniježđene blokove koda (Parsons & Haden, 2006).

Svrha: Poučava učenike o uvlačenju, opsegu i hijerarhijskoj strukturi koda.

Primjer: Potrebno provjeriti je li broj pozitivan i je li paran. Program uključuje uvjetne naredbe koje su ugniježđene, ali linije su poredane pogrešno (Slika 3). Potrebno je preurediti te linije kako bi program ispravno provjerio oba uvjeta i ispisao odgovarajuće poruke (Slika 4).

```
broj = int(input("Unesi broj: "))
else:
if broj % 2 == 0:
else:
print("Broj nije pozitivan.")
if broj > 0:
print("Broj je pozitivan, ali nije paran.")
print("Broj je pozitivan i paran.")
```

Slika 3 Početni popis naredbi

```
broj = int(input("Unesi broj: "))
if broj > 0:
    if broj % 2 == 0:
        print("Broj je pozitivan i paran.")
    else:
        print("Broj je pozitivan, ali nije paran.")
else:
    print("Broj nije pozitivan.")
```

Slika 4 Točno rješenje

Dopunjavanje fragmenata (engl. *Fragment Completion*)

Opis: Neki fragmenti koda su potpuni, dok kod drugih nedostaju ključni dijelovi (npr. nazivi varijabli, operatori). Učenici moraju dovršiti fragmente i staviti ih u ispravan redoslijed (Ericson, Margulieux, & Rick, 2017).

Svrha: Potiče učenike da obrate pažnju na sintaksu i korištenje varijabli unutar konteksta većeg programa.

Primjer: Postavljen je program koji računa prosjek dvaju brojeva, ali neki dijelovi koda su nepotpuni. Nedostaju ključni elementi, poput varijabli i operatora. Potrebno je popuniti te praznine i preurediti kod kako bi program ispravno izračunao i ispisao prosjek.

Uključivanje ometača (engl. *Distractor Inclusion*)

Opis: Uključuje dodatne fragmente koji ne pripadaju ispravnom rješenju. Učenici moraju identificirati i isključiti ove ometače dok poredaju ispravne fragmente.

Svrha: Povećava sposobnost kritičkog razmišljanja i otkrivanja pogrešaka (Ericson, Margulieux, & Rick, 2017).

Primjer: Treba ispisati je li broj paran ili neparan, ali uključene su linije koje nisu relevantne za rješenje (Slika 5). Potrebno je identificirati i ukloniti te ometajuće linije te preurediti preostale dijelove koda kako bi program ispravno radio (Slika 6).

```
for j in range(1, N)          DISTRAKTOR
if broj % 2 == 0:
print("Broj je neparan.")
else:
print("Broj je paran.")
broj = int(input("Unesi broj: "))
```

Slika 5 Početni popis naredbi

```
broj = int(input("Unesi broj: "))
if broj % 2 == 0:
    print("Broj je paran.")
else:
    print("Broj je neparan.")
```

Slika 6 Točno rješenje

Preuređivanje u jednom području (engl. *Single Area Reordering*)

Opis: Svi fragmenti su inicijalno smješteni u jednom području. Učenici moraju poredati fragmente unutar tog područja kako bi formirali ispravan program (Parsons & Haden, 2006).

Svrha: Pojednostavljuje sučelje i fokusira se na preuređivanje logike bez premještanja fragmenata između različitih područja.

Problemi s više koraka (engl. *Multi-Step Problems*)

Opis: Uključuje rješavanje više povezanih Parsonovih slagalica u sekvenci gdje svaki korak nadograđuje prethodni (Parsons & Haden, 2006).

Svrha: Ojačava proces razvoja i razumijevje evolucije i dorade programa.

Tematske ili kontekstualne slagalice (engl. *Thematic or Contextual Puzzles*)

Opis: Slagalice su dizajnirane oko određene teme ili stvarnog konteksta, poput rješavanja određenog problema u određenom području (npr. računanje ocjena, upravljanje inventarom) (Ihantola & Karavirta, 2011).

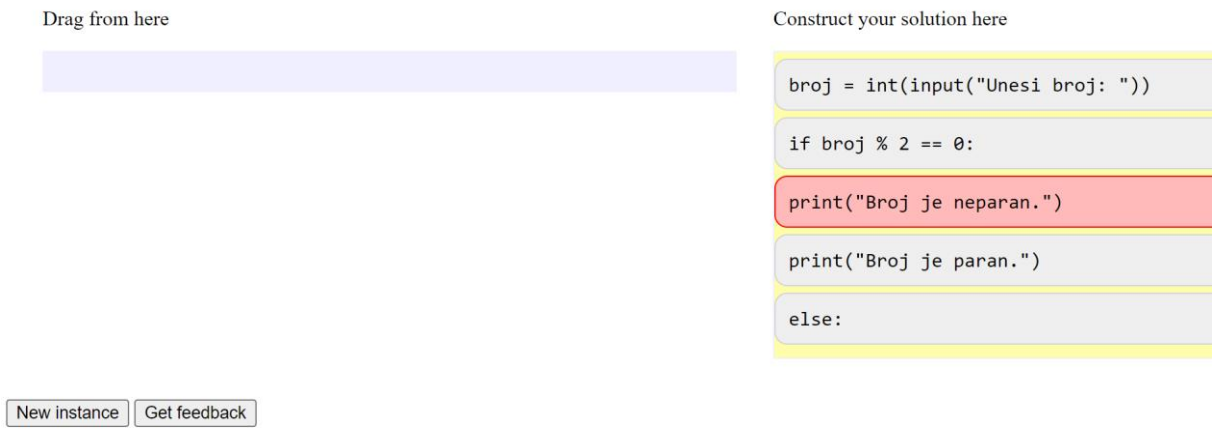
Svrha: Uključuje učenike povezujući zadatke programiranja sa stvarnim scenarijima, pojačavajući motivaciju i relevantnost.

Interaktivne i prilagodljive slagalice (engl. *Interactive and Adaptive Puzzles*)

Opis: Koristi interaktivno sučelje gdje se slagalice prilagođavaju na temelju uspjeha učenika. Sustav može pružiti savjete, ukloniti ometače ili ponuditi dodatnu podršku po potrebi (Ihantola & Karavirta, 2011).

Svrha: Pruža personaliziranu podršku i pomaže učenicima napredovati svojim tempom.

Primjer: Kod slaganja linija koda sustav učeniku pruža povratnu informaciju naglašavajući liniju koja nije u ispravnom redoslijedu (Slika 7). Ovaj proces pomaže učeniku da postupno razumije logiku uvjetnih naredbi, dobije personaliziranu podršku od sustava i poboljša svoje vještine programiranja kroz interaktivno učenje.



Slika 7 Povratna informacija

Slagalice usmjerene na procjenu (engl. *Assessment-Oriented Puzzles*)

Opis: Dizajnirane su posebno za ocjenjivanje razumijevanja i vještina učenika. Ove slagalice često imaju fiksni broj pokušaja ili dopuštenih poteza za postizanje rješenja (Ericson, Margulieux, & Rick, 2017).

Svrha: Koriste se za ocjenjivanje i procjenu kompetencija učenika u konceptima programiranja i sintakse.

2.4 Varijacije Pearsonovog problema

Od samog početka, razvijeno je mnogo varijacija Parsonovih problema koje su kasnije evaluirane. Ove analize su se uglavnom fokusirale na povećanje efikasnosti i smanjenje kognitivnog opterećenja, istovremeno maksimizirajući, ili barem održavajući, postignute obrazovne dobitke koje su Parsonovi problemi demonstrirali (Ericson, Margulieux, & Rick, 2017).

Postoje dva glavna načina uključivanja ometača (eng. *distractors*) u Parsonove probleme. Originalna metoda jednostavno je nasumično raspoređivala ometače zajedno s ispravnim blokovima koda i prepuštala učenicima da razlikuju koji bi trebali koristiti, proces koji će kasnije postati poznat kao "pomiješani ometači". Prethodni radovi su utvrdili da uključivanje pomiješanih ometača rezultira smanjenjem efikasnosti i završetka, dok povećava kognitivno opterećenje, te da povećanje broja ometača čini probleme težim (Ericson, Margulieux, & Rick, 2017). Nedavna alternativa pomiješanim ometačima uključuje vizualno grupiranje ometača s ispravnim blokom koda s kojim su povezani, što povratno vraća nešto efikasnosti, čineći potrebu za odabirom između više opcija eksplicitnijom.

Osim ometača, Parsonovi problemi mogu biti dizajnirani tako da su neosjetljivi na uvlačenje kao kod jezika poput C i Java, obično nazvani jednodimenzionalni, ili osjetljivi kao kod jezika poput Pythona, također nazvani dvodimenzionalni. Prethodni radovi pokazuju da su dvodimenzionalni problemi teži od svojih jednodimenzionalnih pandana (Ihantola & Karavirta, 2011).

Weinman i suradnici istražili su korištenje "Faded Parsons Problems" (Weinman, 2018). Oni se razlikuju od tradicionalnih Parsonovih problema po tome što ne sadrže statičke blokove koda. Umjesto toga, svaki blok sadrži neki skeletni kod s poljima za unos određenih vrijednosti i simbola koje treba ispuniti učenik.

Naposljetku, nedavni radovi istraživali su korisnost adaptivnih Parsonovih problema, gdje se, kako učenici šalju pogrešne odgovore, ometači eliminiraju, a ispravni blokovi se kombiniraju. Svrha ovih problema je postupno oblikovanje problema kako bi odgovarao trenutačnoj razini sposobnosti učenika. Cilj ovih problema je zadržati učenika koji ih rješava u zoni bliske razvoju, gdje su učenici izazvani, ali još uvijek sposobni napredovati, umjesto da stagniraju i postanu frustrirani.

2.5 Strategije rješavanja zadataka

Parsonove slagalice pružaju učenicima izazovne zadatke, potičući ih na razvijanje vlastitih strategija rješavanja programskih problema. Budući da učenici ne moraju pisati kod od nule, već složiti ponuđene linije koda u ispravan redoslijed, ključan aspekt ovakvih zadataka leži u prepoznavanju logičkog slijeda, sintaktičkih pravila i uvlaka, te ignoriranju distraktorskih linija koda. U nastavku su opisane osnovne strategije koje učenici mogu koristiti pri rješavanju zadataka (Parsons & Haden, 2006).

Razumijevanje zadatka

Prva i najvažnija strategija pri rješavanju Parsonovih slagalica jest temeljito razumijevanje zadatka. Učenici trebaju pročitati opis problema i razumjeti što se od njih traži. Na primjer, zadatak može zahtijevati iteraciju kroz niz, upotrebu uvjetnih naredbi ili manipulaciju varijablama. Prepoznavanje tih osnovnih zahtjeva ključno je za prepoznavanje ispravnih linija koda i njihovog redoslijeda. Ova strategija pomaže učenicima da identificiraju što je suvišno i koje linije koda čine osnovu za rješenje problema.

Eliminacija distraktora

Druga strategija uključuje eliminaciju distraktorskih linija koda. Parsonove slagalice često sadrže linije koda koje nisu relevantne za rješenje, ali su tu kako bi testirale učenikovo razumijevanje problema. Distraktori mogu biti sintaktički ispravni, ali logički netočni u kontekstu zadatka (Parsons & Haden, 2006). Učenici trebaju naučiti prepoznati takve linije i ignorirati ih pri slaganju koda. Ova metoda eliminacije može značajno postaviti fokus na one linije koje su ključne za rješenje.

Korištenje petlji i uvjeta

Jedna od osnovnih vještina koje učenici trebaju razviti pri rješavanju Parsonovih slagalica je pravilno korištenje petlji i uvjetnih naredbi. Kao što je opisano u zadacima, učenici moraju prepoznati kako i gdje koristiti for petlje ili if uvjete. Pravilno razumijevanje logičkog toka

petlje i uvjetnih blokova omogućuje učenicima da točno postave linije koda koje će ponavljati ili testirati određene uvjete unutar programa.

Strukturna uvlaka (indentacija)

Važan aspekt Parsonovih slagalica je pravilno korištenje uvlaka, koje definiraju blokove koda unutar petlji ili uvjetnih naredbi. Jedan od ključnih izazova za učenike početnike je prepoznavanje gdje je potrebno dodati uvlake kako bi struktura koda bila ispravna (Shah, 2020). Učenici često moraju koristiti metodu pokušaja i pogrešaka kako bi prilagodili uvlake na pravim mjestima. Aplikacija nudi vizualnu pomoć pri uvlačenju linija koda, omogućujući učenicima lakše razumijevanje blokova koda unutar petlji i uvjetnih izraza.

Provjera rješenja i povratne informacije

Jedan od najvažnijih koraka u procesu rješavanja Parsonovih slagalica je provjera rješenja i korištenje povratnih informacija koje aplikacija pruža. Nakon što učenici slože linije koda, aplikacija provjerava rješenje i vraća povratnu informaciju o točnosti rješenja. Ako rješenje nije točno, učenici dobiju obavijest te mogu pokušati ponovo. Posebna značajka aplikacije omogućuje označavanje točnih linija koda nakon tri neuspjela pokušaja, čime učenici dobiju smjernice za daljnje korake. Ovo je važna strategija za učenike jer im omogućuje da isprave pogreške i bolje razumiju logiku koda.

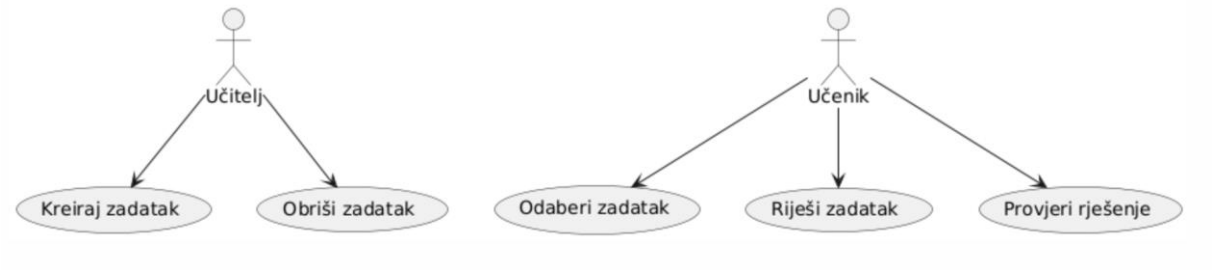
3. Aplikacija za kreiranje i rješavanje Parsonovih slagalica

U kontekstu obrazovnih alata za učenje programiranja, Parsonove slagalice pokazale su se kao učinkovit način podučavanja programskih vještina. Za potrebe rada razvijena je aplikacija koja omogućava učiteljima kreiranje i upravljanje zadacima, a učenicima rješavanje tih zadataka pomoću slaganja ispravno strukturiranog Python koda. Aplikacija za kreiranje i rješavanje Parsonovih slagalica predstavlja koristan alat za učitelje i učenike u učenju programiranja. Integracijom različitih tehnologija, aplikacija omogućava jednostavno kreiranje zadataka i interaktivno učenje kroz slaganje koda. Ova aplikacija demonstrira kako tehnologija može poboljšati proces učenja i učiniti ga učinkovitijim i interaktivnijim.

3.1 Modeliranje zahtjeva

Prije samog početka izrade aplikacije, potrebno je definirati funkcionalnosti aplikacije. Aplikacija pruža osnovne funkcionalnosti za učitelje i učenike. Učitelji imaju mogućnost kreiranja zadataka, pri čemu unose naziv i pripadajući kod, također, učitelji mogu brisati zadatke koje su prethodno kreirali. S druge strane, učenici mogu odabrati zadatak iz dostupne liste, povlačiti i slagati blokove koda kako bi riješili zadatak, te provjeriti ispravnost svog rješenja putem aplikacije. Ove funkcionalnosti omogućuju jednostavno upravljanje zadacima za učitelje, dok učenici mogu aktivno vježbati programiranje rješavanjem zadataka.

Funkcionalnosti aplikacije prikazat će se dijagramom slučajeva korištenja (Slika 8) koji se kreira prije samog početka izrade aplikacije. Iz dijagrama se jasno vide navedene funkcionalnosti aplikacije te se prikazuje način na koji korisnici integriraju sa sustavom.

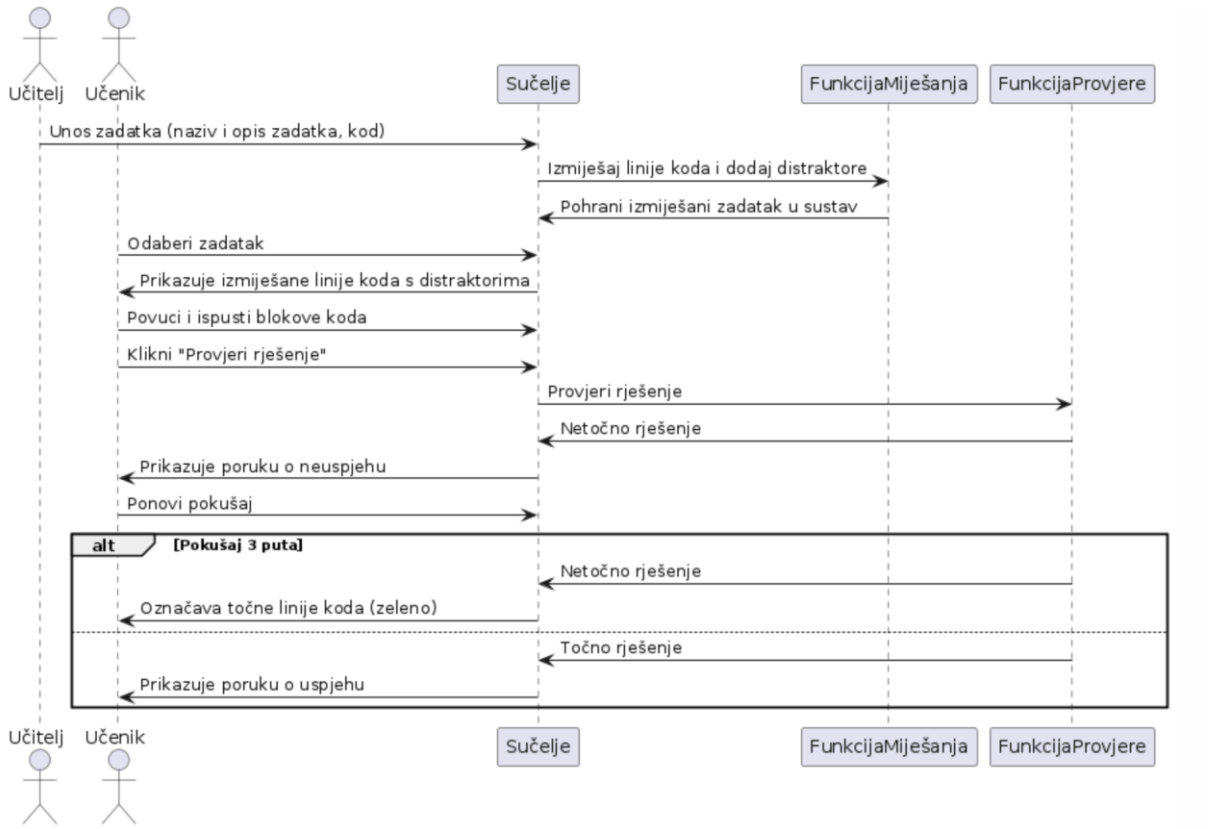


Slika 8 Dijagram slučajeva korištenja

3.2 Modeliranje tijeka

Dijagram slijeda (Slika 9) ilustrira ključne interakcije između učitelja, učenika i aplikacijskih funkcija unutar sustava. Proces započinje kada učitelj unese zadatak, uključujući naziv i

pripadajući kod. Sučelje aplikacije tada poziva funkciju koja izmiješa linije koda i dodaje jedan ili dva distraktora. Zadatak se zatim pohranjuje u sustav i postaje dostupan učenicima. Učenik odabire zadatak i sučelje prikazuje izmiješane linije koda sa distraktorima. Učenik povlači i slaže blokove koda te kada klikne na gumb "Provjeri rješenje", aktivira funkciju koja provjerava je li učenik pravilno složio kod. Ako rješenje nije točno, aplikacija vraća poruku o neuspjehu. U slučaju da učenik tri puta pokuša riješiti zadatak, a rješenje nije ispravno, označavaju se ispravne linije koda zelenom bojom, što služi učeniku kao smjernica za točno rješenje. Nakon uspješnog rješavanja zadatka, učeniku se prikazuje poruka o uspjehu. Dijagram slijeda jasno prikazuje kako aplikacija koristi različite funkcije za miješanje koda i spremanje zadatka, provjeru rješenja i prikaz povratnih informacija korisnicima.

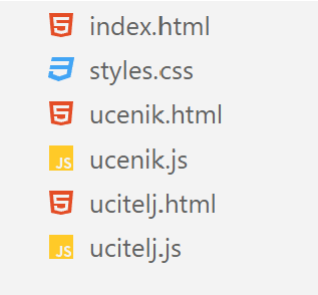


Slika 9 Dijagram slijeda

3.3 Struktura i komponente aplikacije

Aplikacija je razvijena korištenjem Visual Studio Code okruženja i sastoji se od nekoliko ključnih datoteka (Slika 10):

- HTML Datoteke: index.html, ucenik.html, ucitelj.html
- JavaScript Datoteke: ucenik.js, ucitelj.js
- CSS Datoteka: styles.css



Slika 10 Prikaz datoteka

Aplikacija koristi kombinaciju HTML, CSS i JavaScript za frontend, dok je Python korišten za definiranje zadataka i provjeru rješenja. Svaka komponenta ima specifičnu funkciju.

HTML i CSS se koriste za strukturiranje i stiliziranje sučelja. HTML datoteke definiraju raspored elemenata na stranici, dok CSS datoteka osigurava vizualni izgled aplikacije, uključujući boje, fontove i raspored elemenata. HTML datoteke su temelj svakog sučelja, omogućavajući korisnicima intuitivnu navigaciju i interakciju s aplikacijom. CSS datoteka dodatno omogućava prilagodbu vizualnog izgleda, osiguravajući da aplikacija bude estetski ugodna i funkcionalna.

JavaScript osigurava dinamičku manipulaciju DOM-a i interaktivnost sučelja. „ucenik.js“ i „ucitelj.js“ sadrže funkcije koje omogućuju kreiranje zadataka, upravljanje zadacima i interakciju učenika s blokovima koda. Ove datoteke upravljaju logikom aplikacije, omogućavajući korisnicima brzu i učinkovitu interakciju s aplikacijom.

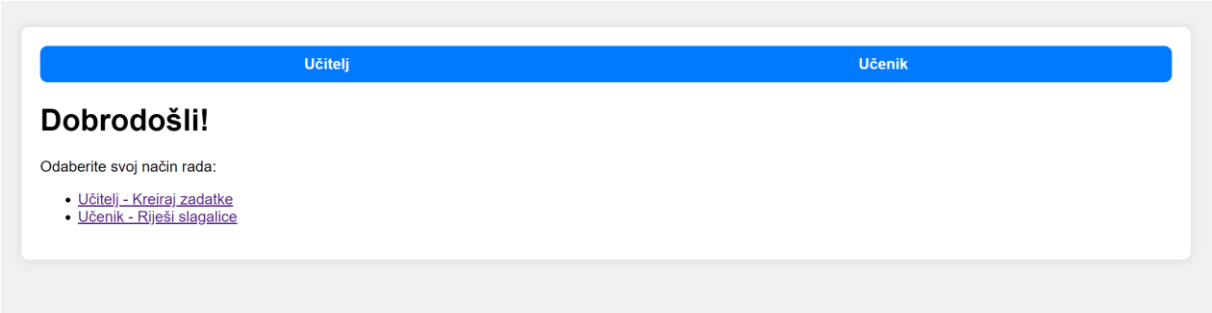
Za kreiranje zadataka i logiku provjere rješenja koristimo Python. Python kod unesen od strane učitelja se koristi kao referentni kod prilikom provjere ispravnosti složenih blokova koda od strane učenika.

3.4 Izgled i funkcionalnosti aplikacije

Aplikacija mora biti dizajnirana s naglaskom na jednostavnost, pristupačnost i funkcionalnost kako bi zadovoljila potrebe korisnika. Korisničko iskustvo treba biti intuitivno, omogućujući lako snalaženje bez potrebe za složenim uputama. Jasna struktura i dosljedan dizajn su ključni za smanjenje potencijalnih grešaka i frustracija korisnika.

3.4.1 Početno sučelje

Početno sučelje aplikacije služi kao ulazna točka za korisnike, omogućujući im odabir između dvije glavne opcije rada: sučelje za učitelje ili sučelje za učenike (Slika 11). Ovo sučelje je dizajnirano tako da bude jednostavno i intuitivno, omogućujući korisnicima brz i jednostavan pristup željenoj funkcionalnosti. Intuitivni dizajn i jednostavna navigacija osiguraju da korisnici mogu lako započeti rad s aplikacijom, bilo da su u ulozi učitelja ili učenika.

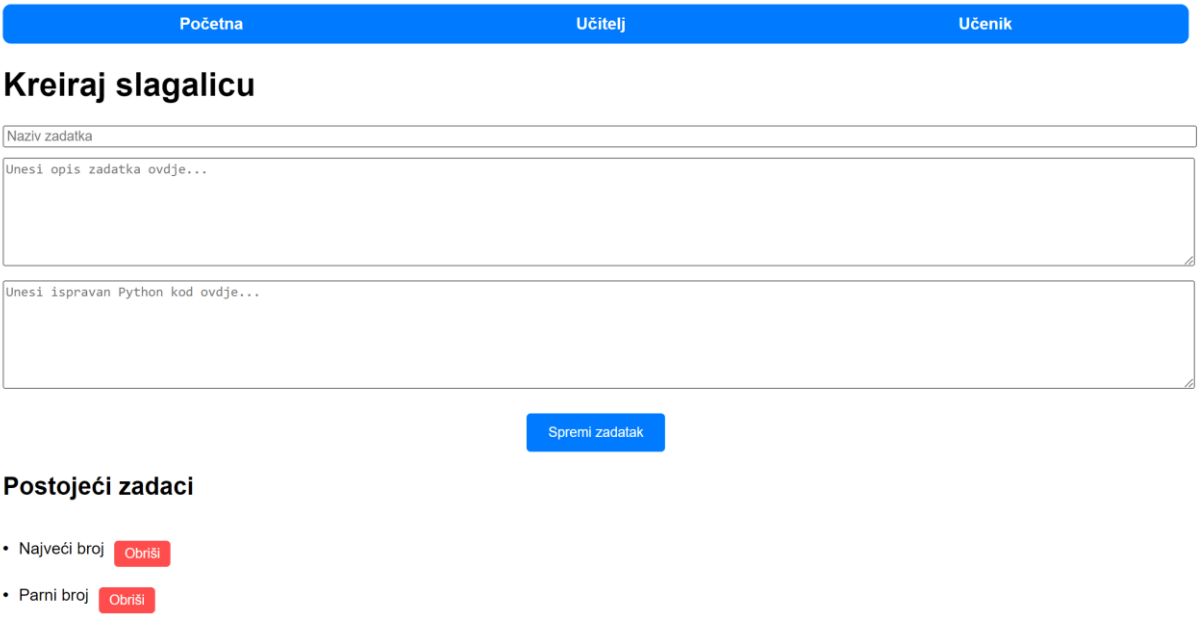


Slika 11 Početno sučelje

3.4.2 Funkcionalnosti sučelja za učitelje

Sučelje za učitelje (Slika 12) omogućava kreiranje, upravljanje i brisanje zadataka. Učitelji mogu dodati naslov i opis zadatka te unijeti strukturirani Python kod. Ovaj kod će se kasnije koristiti kao referenca za generiranje blokova koda koje učenici trebaju složiti.

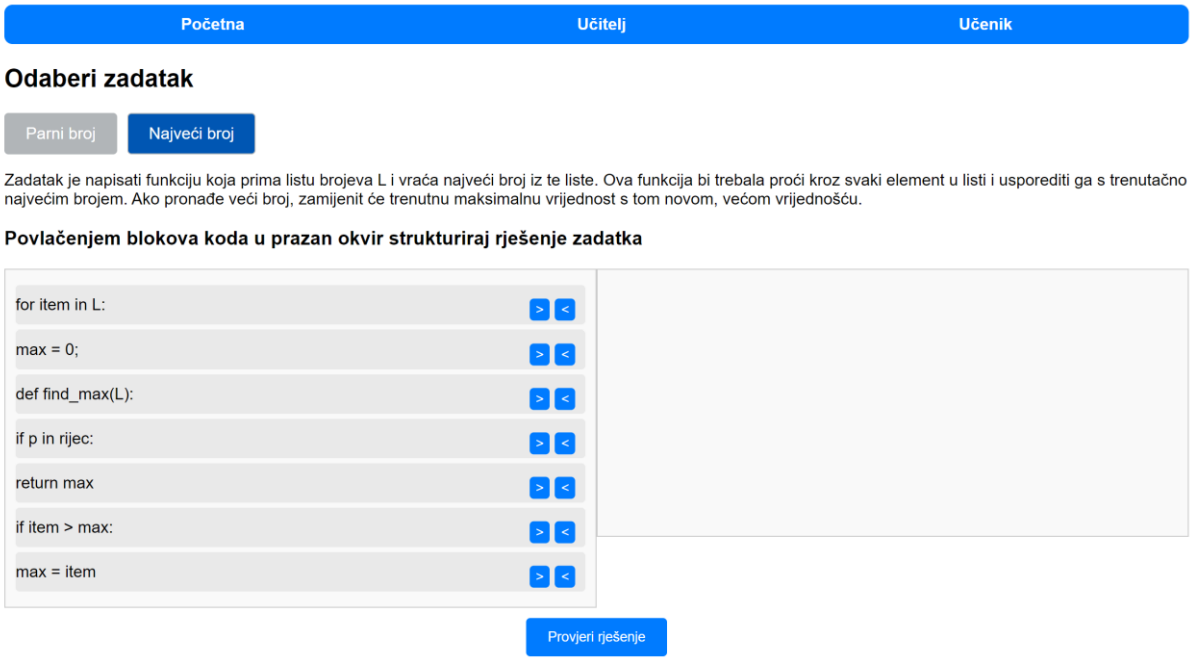
Sučelje omogućava prikaz svih kreiranih zadataka u formi liste. Svaki zadatak u listi prikazan je s nazivom zadatka i opcijom za brisanje. Ova funkcionalnost omogućava učiteljima jednostavno upravljanje zadacima, pregled trenutnog stanja i eventualno uklanjanje nepotrebnih zadataka.



Slika 12 Sučelje za učitelja

3.4.3 Funkcionalnosti sučelja za učenike

Sučelje za učenike (Slika 13) omogućava rješavanje zadataka pomoću slaganja blokova koda. Učenici imaju pristup listi dostupnih zadataka koje su učitelji kreirali. Klikom na naziv zadatka, prikazuje se opis zadatka, a sučelje generira blokove koda koji su nasumično poredani. Svaki blok predstavlja dio koda koji učenik treba pravilno posložiti kako bi riješio zadatak.



Slika 13 Sučelje za učenika

Osim ispravnih linija koda, među blokovima koda nalaze se i **distraktori** – linije koda koje ne pripadaju ispravnom rješenju. Ove distraktorske linije su nasumično odabrane, obično jedna ili dvije, iz niza od 10 linija koda (Slika 14), te su ubačene kako bi povećale izazovnost zadatka i potaknule učenike na kritičko razmišljanje prilikom prepoznavanja ispravnih linija.

```
const distractors = [
    { text: 'for i in L:', indent: 0 },
    { text: 'x = 5 + 10', indent: 1 },
    { text: 'if x > 10:', indent: 1 },
    { text: 'print(i)', indent: 0 },
    { text: 'for i in range(N)', indent: 1 },
    { text: 'for j in range(1, N)', indent: 0 },
    { text: 'while i < 10:', indent: 0 },
    { text: 'if p in rijec:', indent: 1 },
    { text: 'return i', indent: 0 },
    { text: 'rijeci = []', indent: 2 },
];
```

Slika 14 Distraktori

Učenici koriste metodu povuci i ispusti (engl. *drag and drop*) kako bi premjestili blokove koda iz jednog okvira, gdje su generirani, u prazan okvir ili obrnuto. Nakon premještanja blokova, učenici mogu strukturirati linije koda pomoću tipki koje označavaju uvlake, omogućujući

dodavanje ili uklanjanje uvlaka prema potrebi. Pomoću ove funkcionalnost učenici na intuitivan način vježbaju logičko razmišljanje i pravilno strukturiranje koda.

Nakon što učenik posloži sve blokove koda, može provjeriti rješenje klikom na gumb "Provjeri rješenje". Aplikacija će tada usporediti učenikov složeni kod s referentnim kodom koji je učitelj unio prilikom kreiranja zadatka. Ako je rješenje ispravno, učenik dobiva povratnu informaciju o uspjehu, inače dobiva povratnu informaciju kako kod nije dobro posložen.

Dodatno, ako učenik pokuša riješiti zadatak tri puta bez uspjeha, aplikacija će mu pružiti dodatnu pomoć. Nakon trećeg pokušaja, linije koda koje su točno postavljene obojat će se zelenom bojom (Slika 15), čime učenik dobiva vizualnu povratnu informaciju o ispravno postavljenim dijelovima koda. Ova funkcionalnost omogućuje učenicima uvid u napredak i pruža smjernice koje im mogu pomoći da lakše isprave preostale pogreške i dovrše zadatak uspješno.

Slika 15 Vizualna povratna informacija

3.5 Primjer zadatka

Da bi se jasnije prikazala strategija rješavanja Parsonovih slagalica, prikazan je detaljan primjer zadatka uz korake rješavanja koje učenik može slijediti. Ovaj primjer pokazuje kako učenik može primijeniti opisane strategije pri rješavanju zadatka pomoću aplikacije.

3.5.1 Zbrajanje brojeva od 1 do 10

Opis zadatka: Potrebno je složiti ispravan Python kod koji izračunava zbroj brojeva od 1 do 10. Linije koda su izmiješane, a neke od njih su distraktori koji nisu relevantni za rješenje.

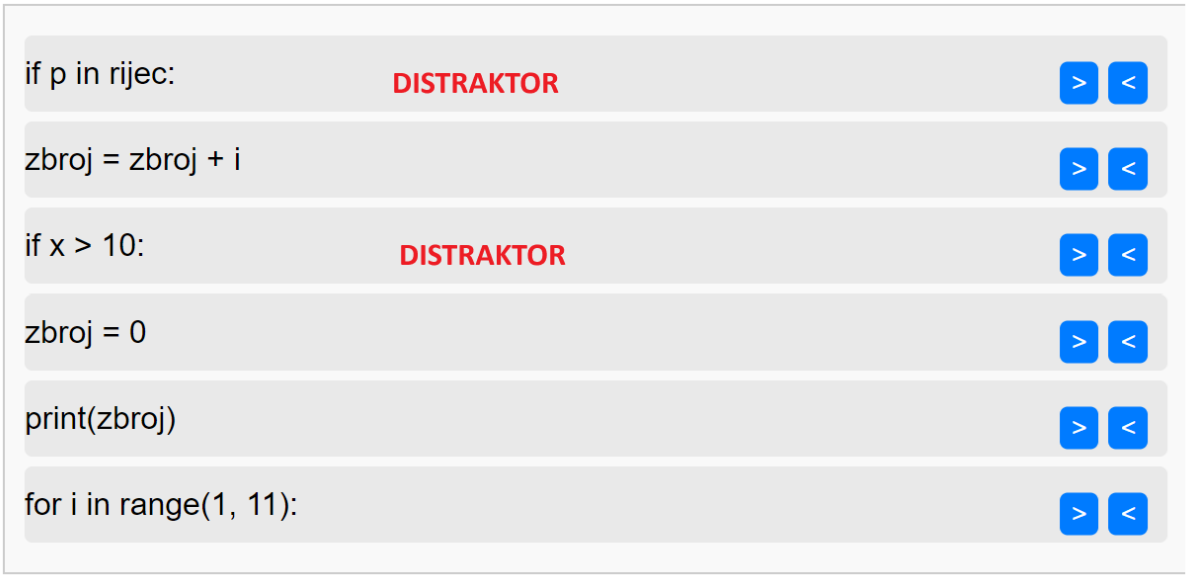
Izgled zadatka (Slika 16):

Slika 16 Zadatak: Zbrajanje brojeva od 1 do 10

Da bi učenici uspješno riješili Parsonsove slagalice, prvo moraju temeljito razumjeti što se traži u zadatku. U ovom primjeru, učenik treba složiti ispravan Python kod koji izračunava zbroj brojeva od 1 do 10. Prvi korak u procesu rješavanja je prepoznavanje osnovnih elemenata zadatka (Slika 16). Učenik mora shvatiti da je potrebno koristiti **for petlju** za iteraciju kroz brojeve, te varijablu koja će čuvati zbroj tih brojeva. Ovo razumijevanje zadatka ključno je za prepoznavanje potrebnih linija koda.

Sljedeći korak uključuje prepoznavanje ključnih linija koda koje su potrebne za ispravno rješenje. Učenik prepoznaje da su linije poput inicijalizacije varijable `zbroj = 0`, zatim petlje `for i in range(1, 11):` te linije za zbrajanje `zbroj = zbroj + i` ključne za rješenje zadatka. Ispis konačnog rezultata, tj. `print(zbroj)`, također je važna komponenta. Na ovaj način, učenik može izdvojiti ključne dijelove koda, dok se ostale linije mogu eliminirati kao distraktori.

U ovom primjeru, linije poput `if p in rijec:` ili `if x > 10:` mogu djelovati zbunjujuće, ali nisu relevantne jer ne odgovaraju zahtjevima zadatka (Slika 17). Eliminacija distraktora pomaže učeniku da se usmjeri na ključne dijelove koda, omogućujući mu da lakše složi ispravan redoslijed.



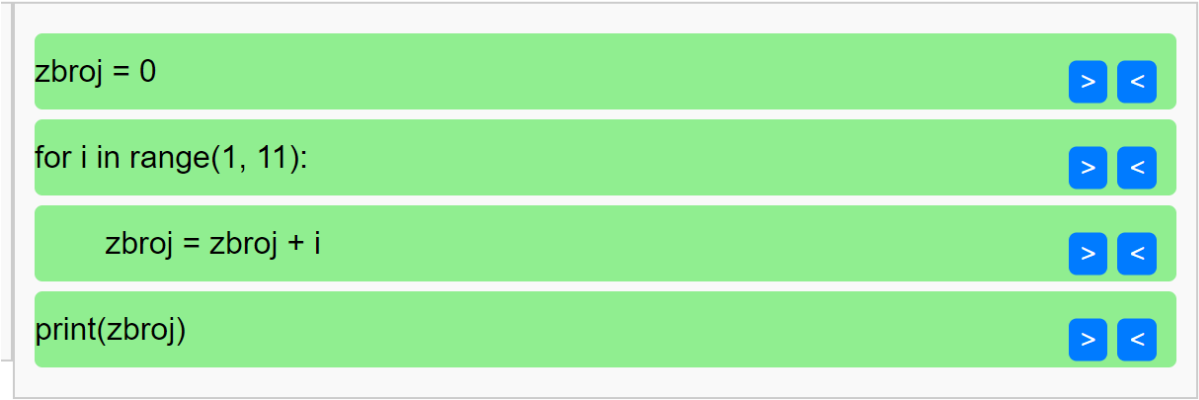
Slika 17 Označeni distraktori

Nakon eliminacije distraktora, učenik mora pravilno koristiti petlju. Prepoznavanje ispravne petlje `for i in range(1, 11):` omogućuje učeniku da pravilno iterira kroz traženi raspon brojeva. U ovom trenutku, učenik mora paziti na sintaksu petlje i pravilno postavljanje unutarnjih linija koda. Uvjeti i petlje čine temelj programskih zadataka, te ih je ključno pravilno razumjeti i primijeniti.

Jedan od izazova u rješavanju Parsonsovih slagalica je pravilno korištenje uvlaka (indentacija). Učenik mora obratiti pažnju na to kako su linije koda strukturirane unutar petlje. Na primjer, zbrajanje brojeva `zbroj = zbroj + i` mora biti unutar petlje, dok ispis rezultata `print(zbroj)` mora biti izvan nje kako bi se rezultat ispisao tek nakon što se izračuna zbroj. Ispravna uvlaka pomaže učeniku da organizira kod na način koji osigurava ispravno izvođenje programa.

Nakon što učenik složi kod, može provjeriti rješenje klikom na gumb "Provjeri rješenje". Ako rješenje nije točno, aplikacija će prikazati povratnu informaciju koja učeniku daje smjernice za ispravljanje pogrešaka. Ako učenik tri puta pokuša složiti rješenje, a ne uspije, aplikacija će označiti ispravne linije koda zelenom bojom, čime pruža dodatnu pomoć. Kroz ovaj proces provjere i povratnih informacija, učenik može ispravljati greške i učiti iz vlastitih pokušaja.

Konačno, nakon što učenik ispravno složi sve linije koda, dobije povratnu informaciju o uspjehu (Slika 18). Ovaj postupak ne samo da omogućuje učeniku rješavanje zadatka, nego ga i uči važnim programskim konceptima poput rada s petljama, varijablama, uvlakama i prepoznavanja ispravnog logičkog slijeda u kodu.



Slika 18 Ispravno rješenje zadatka

3.5.2 Funkcija koja provjerava parne brojeve u listi

Opis zadatka: Zadatak je napisati funkciju koja prima listu brojeva, funkcija bi trebala proći kroz svaki element u listi i ispisati samo parne brojeve.

Izgled zadatka (Slika 19):

Slika 19 Zadatak: Provjera parnih brojeva u listi

Zadatak zahtijeva da učenik iterira kroz listu brojeva i ispituje je li broj paran. U ovom slučaju, učenik mora znati koristiti petlju za iteraciju kroz listu te uvjetnu naredbu if za provjeru parnosti brojeva (Slika 19). Prepoznavanje ovih osnovnih elemenata, poput petlje i uvjeta za provjeru brojeva, ključno je za rješavanje zadatka.

U sljedećem koraku, učenik mora prepoznati ključne linije koda. Prvi korak u tom procesu je razumijevanje funkcije koju treba izraditi. Budući da zadatak traži provjeru parnih brojeva u listi, učenik zaključuje da je najvažnije prvo pronaći liniju koja definira funkciju, a to je `def`

`provjeri_parne_brojeve(lista):`. Ova linija je ključna jer označava početak funkcije koja obrađuje listu brojeva.

Nakon što je prepoznao naziv funkcije, učenik dalje zaključuje da je potrebna petlja koja prolazi kroz svaki element liste. Stoga, linija `for broj in lista:` postaje sljedeća važna linija, jer omogućuje iteraciju kroz sve brojeve u listi. Učenik razumije da se svaki broj mora provjeriti, pa je prirodno da se uvjet `if broj % 2 == 0:` koristi za provjeru parnosti brojeva. Ovaj uvjet pomaže učeniku da shvati koji brojevi trebaju biti ispisani, a koji ne.

Konačno, linija `print(broj)` koristi se za ispisivanje parnih brojeva. Kroz ove ključne linije, učenik postepeno slaže kod, eliminirajući distraktore koji nisu relevantni za zadata, u ovom slučaju linija koda `for j in range(1, N):` je distraktor (Slika 20).

Slika 20 Označeni distraktori

Nakon eliminacije distraktora, učenik mora pravilno koristiti petlju i uvjet. Korištenje ispravne petlje `for broj in lista:` omogućuje iteriranje kroz sve elemente liste, dok uvjet `if broj % 2 == 0:` osigurava da se ispisuju samo parni brojevi. Uvjeti i petlje često su ključni aspekti zadataka i moraju biti pažljivo primijenjeni kako bi rješenje bilo točno.

Kao i u prethodnom zadatku, jedan od izazova je pravilno korištenje uvlaka. Učenik mora obratiti pažnju na to kako postaviti uvjet unutar petlje. Linija `print(broj)` mora biti uvučena unutar bloka if, kako bi se ispisao broj samo ako je uvjet zadovoljen. Ispravna uvlaka organizira kod na način koji osigurava njegovo ispravno izvođenje.

Nakon što učenik složi sve linije koda, može provjeriti rješenje. Ako učenik ne uspije riješiti zadatak nakon tri pokušaja, aplikacija će označiti točne linije zelenom bojom, pomažući mu da

ispravi rješenje. Nakon što učenik ispravno rješi zadatak, dobije povratnu informaciju o uspjehu (Slika 21).

Slika 21 Ispravno rješenje zadatka

3.5.3 Funkcija koja provjerava najveći broj u listi

Opis zadatka: Zadatak je napisati funkciju koja prima listu brojeva L i vraća najveći broj iz te liste. Ova funkcija bi trebala proći kroz svaki element u listi i usporediti ga s trenutačno najvećim brojem. Ako pronađe veći broj, zamijenit će trenutnu maksimalnu vrijednost s tom novom, većom vrijednošću.

Izgled zadatka (Slika 22):

Slika 22 Zadatak: Pronađi najveći broj u listi

Zadatak zahtijeva da funkcija prođe kroz listu brojeva i usporedi svaki broj s trenutnim maksimumom. Ako se pronađe veći broj, varijabla koja čuva maksimalnu vrijednost treba biti

ažurirana. Ovaj proces ključan je za ispravno rješenje zadatka (Slika 22). Učenik mora prepoznati osnovne elemente, poput petlje i uvjetne naredbe, te varijable koja čuva najveći broj.

Nakon razumijevanja zadatka, učenik mora prepoznati ključne linije koda. Potrebno je prvo pronaći liniju koja definira funkciju, a to je `def najveci_broj(L):`.

Zatim je bitno pregledati sve varijable. Varijabla `max = 0` inicijalizira se na 0, što će poslužiti kao početna vrijednost za usporedbu. Ova varijabla privremeno pohranjuje najveću vrijednost koju pronađe funkcija. Nakon što je varijabla postavljena, ključni sljedeći korak je korištenje for petlje, koja služi kao alat za iteriranje kroz svaki element u listi.

For petlja `for broj in L:` omogućuje prolazak kroz sve elemente liste. U svakoj iteraciji, trenutni broj iz liste uspoređuje se s trenutnom maksimalnom vrijednošću pohranjenom u varijabli max. Ako trenutni broj zadovoljava uvjet `if broj > max:`, ažurira se maksimalna vrijednost s novom većom vrijednošću. Na taj način, petlja osigurava da se svaki element u listi pregleda i usporedi, što je ključno za rješavanje ovog zadatka.

Nakon što se svi brojevi u listi usporede, linija `return max` vraća konačnu vrijednost varijable max, koja predstavlja najveći broj u listi.

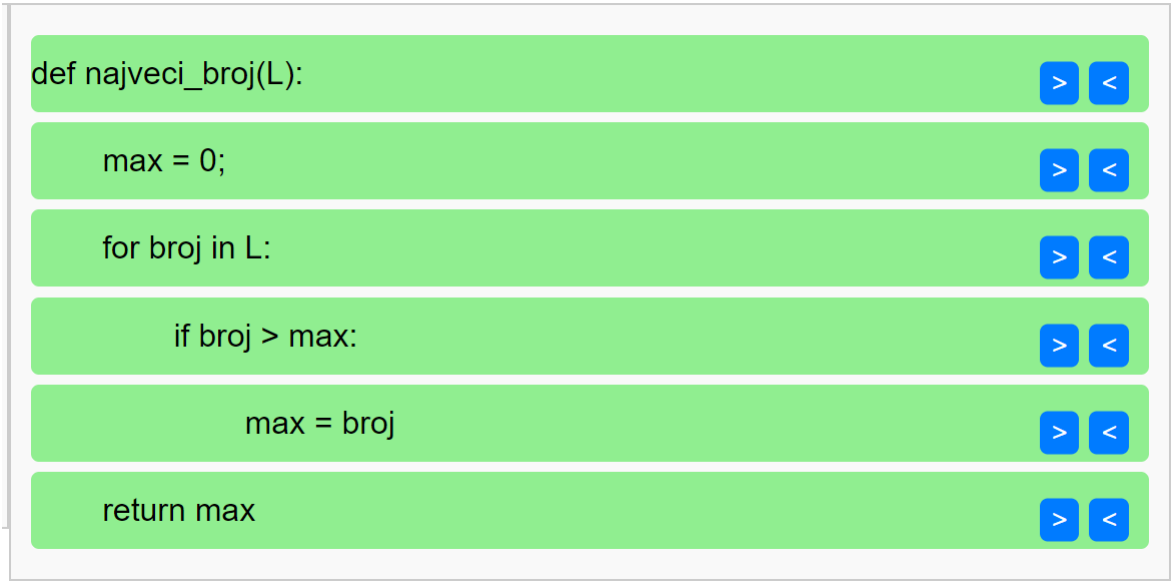
Distraktori, poput linije `for i in range(N):` ili `while i < 10:` unutar petlje, mogu zbuniti učenika jer nisu relevantni za rješenje (Slika 23). Eliminacija distraktora ključna je za usmjeravanje učenika prema pravilnom razumijevanju logike zadatka, koja traži najveći broj u listi.

Slika 23 Označeni distraktori

Kada učenik eliminira distraktore, sljedeći korak je pravilna primjena petlje i uvjeta. Petlja `for broj in L:` omogućuje iteraciju kroz sve elemente liste, dok uvjet `if broj > max:` provjerava treba li ažurirati maksimalnu vrijednost. Učenik mora obratiti pozornost na sintaksu i logički slijed kako bi rješenje bilo ispravno.

Kao i u prethodnim zadacima, učenik mora pravilno koristiti uvlake. Uvjet if mora biti unutar petlje, dok linija `return max` mora biti izvan petlje, kako bi se funkcija vratila nakon što svi elementi liste budu pregledani.

Nakon što učenik ispravno riješi zadatak, dobije povratnu informaciju o uspjehu (Slika 24).

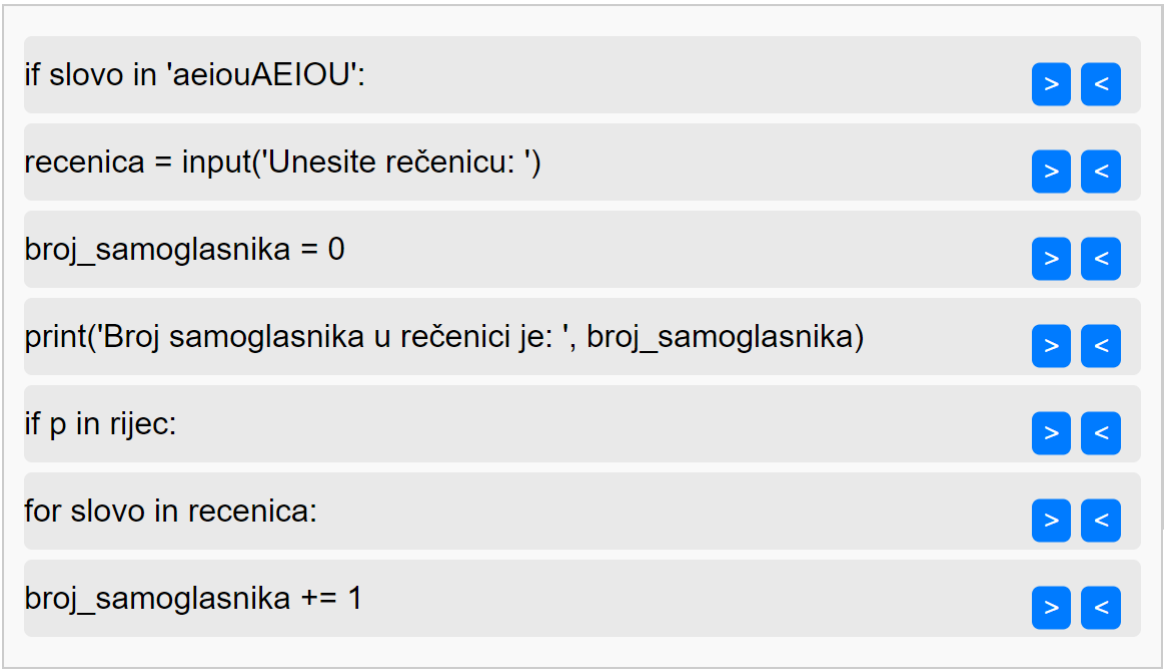


Slika 24 Ispravno rješenje zadatka

3.5.4 Brojanje samoglasnika u rečenici

Opis zadatka: Zadatak je napisati Python program koji od korisnika prima rečenicu putem input naredbe i zatim koristi petlju kako bi izbrojao koliko se samoglasnika nalazi u toj rečenici. Na kraju, program ispisuje ukupan broj samoglasnika koristeći naredbu print.

Izgled zadatka (Slika 25):

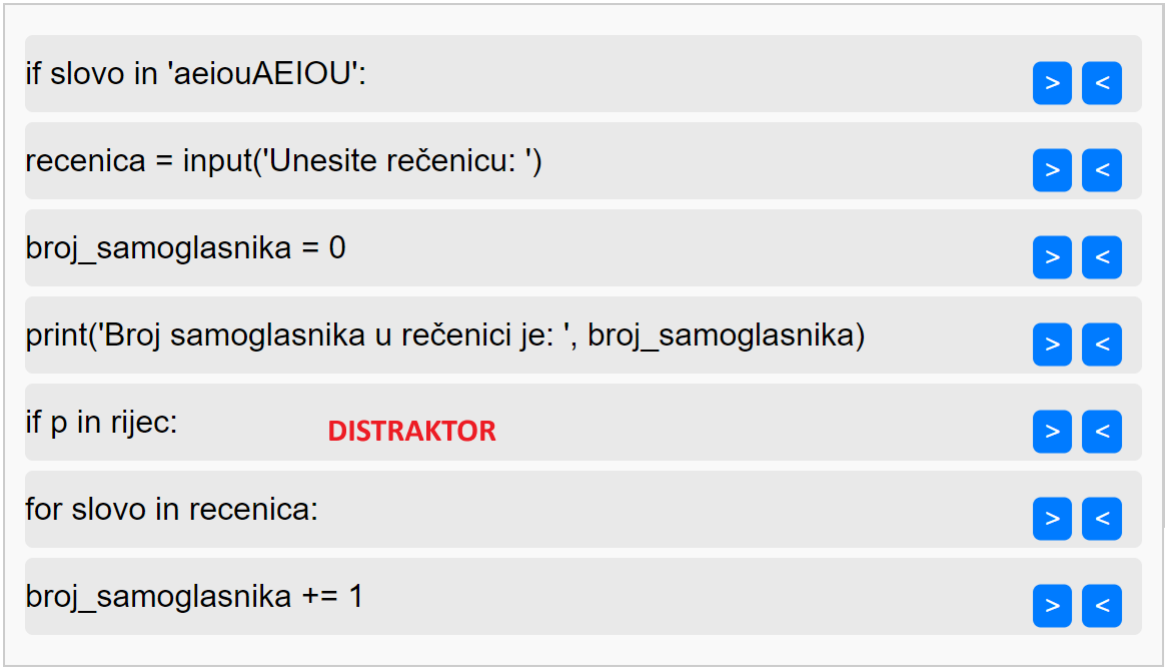


Slika 25 Zadatak: Broj samoglasnika u rečenici

Učenik prvo mora razumjeti što se od njega traži. Zadatak zahtijeva unos rečenice putem input naredbe, zatim iteraciju kroz znakove u rečenici i provjeru je li znak samoglasnik. Konačno,

potrebno je ispisati ukupan broj samoglasnika. Ključno je da učenik prepozna potrebu za petljom i uvjetom kako bi se provjerili svi znakovi u stringu.

Zadatak uključuje distraktore poput pogrešnih uvjeta za prepoznavanje samoglasnika. Na primjer, linije poput `if p in rijec:` mogu zbuniti učenika (Slika 26). Eliminacija ovih distraktora pomaže učeniku da se usmjeri na ključne linije koda koje vode do ispravnog rješenja.



Slika 26 Distraktori

Učenik mora prvo prepoznati da program započinje uzimanjem korisničkog unosa pomoću naredbe input, koja od korisnika traži da unese rečenicu. Ovo je ključno jer se sav daljnji proces temelji na ovoj rečenici. Stoga, prva linija koda mora biti `recenica = input('Unesite rečenicu: ')`.

Učenik mora pravilno koristiti varijable. Prvo je potrebno definirati varijablu koja će pohraniti broj samoglasnika, na primjer, `broj_samoglasnika = 0`. Ova varijabla će se koristiti za povećavanje brojača svaki put kada program pronađe samoglasnik. Potrebno je prepoznati da zadatak zahtijeva prolazak kroz svaki znak u rečenici. To se postiže korištenjem for petlje: `for slovo in recenica:`, koja iterira kroz sve znakove u rečenici.

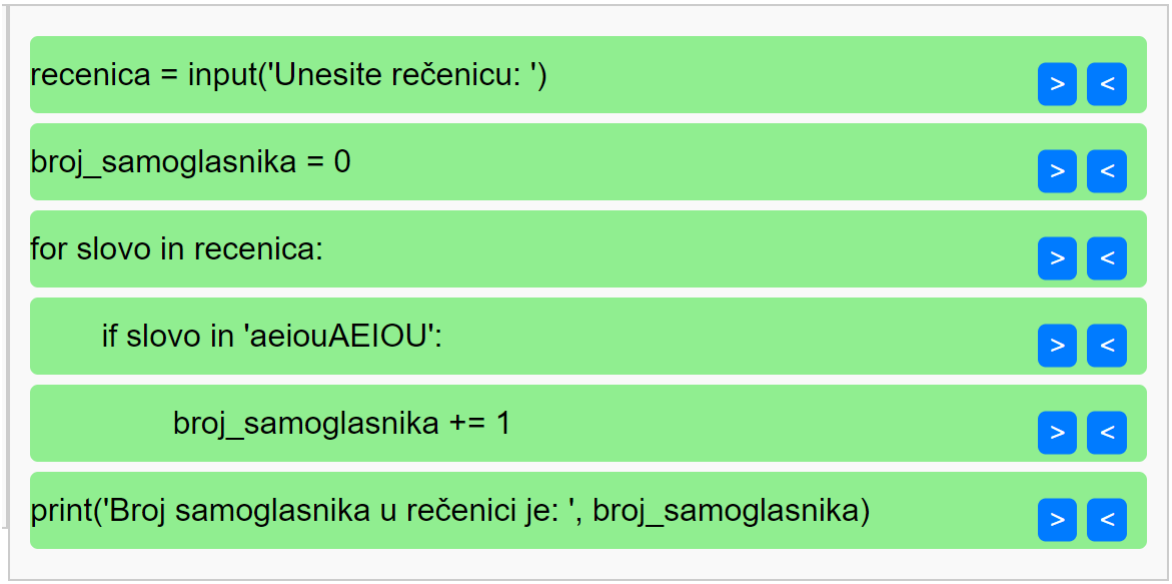
Zatim se koristi if uvjet za provjeru je li trenutni znak samoglasnik. Ova provjera može se obaviti linijom `if slovo in 'aeiouAEIOU':`, koja provjerava je li znak jedno od pet mogućih samoglasnika, uključujući i velika slova. Ako uvjet bude zadovoljen, varijabla

broj_samoglasnika povećava se za 1. Ova petlja i uvjet ključni su za rješavanje zadatka jer omogućuju učeniku da prepozna i izbroji sve samoglasnike u rečenici.

Učenik mora paziti na ispravnu uvlaku unutar petlje. Uvjetna provjera mora biti uvučena unutar petlje, tako da se svaki znak pravilno provjeri. Učenik također mora paziti da sve linije koje trebaju biti unutar petlje (kao što je povećanje brojača) budu pravilno uvučene.

Korištenje print naredbe na kraju programa označava kraj logike programa i pruža korisniku povratnu informaciju o rezultatu operacije.

Nakon što učenik unese sve linije koda, može pokrenuti program kako bi provjerio radi li ispravno. Ako program ne radi kako treba, aplikacija može pružiti povratne informacije o ispravnim i pogrešnim linijama. Nakon što učenik ispravno rješi zadatak, dobije povratnu informaciju o uspjehu (Slika 27).



Slika 27 Ispravno rješenje zadatka

2 Zaključak

U ovom radu istraženi su različiti aspekti Parsonovih slagalica, njihov povijesni razvoj, karakteristike, vrste, te njihova uloga u učenju programiranja. Parsonove slagalice, prvi put uvedene 2006. godine od strane Dalea Parsonsa i Patricije Haden, postale su važan alat za učenje programiranja zahvaljujući svojoj jednostavnosti i učinkovitosti (Parsons & Haden, 2006). Njihov glavni cilj je pomoći učenicima u razumijevanju sintakse i strukture programskog jezika na angažirajući i interaktivan način.

Karakteristike Parsonovih slagalica uključuju rad s fragmentima koda koje učenici trebaju pravilno poredati kako bi stvorili funkcionalan program. Ove slagalice omogućuju trenutne povratne informacije, što učenicima pomaže da brzo prepoznaju i isprave svoje pogreške. Različite vrste Parsonovih slagalica, kao što su osnovno strukturiranje linija, rad s ugniježđenim strukturama, dodavanje distraktora i djelomične linije, omogućuju prilagodbu različitim razinama znanja i potrebama učenika.

Proučavanje i praksa pokazali su da Parsonove slagalice značajno smanjuju kognitivno opterećenje studenata, povećavajući pritom njihovu učinkovitost i razumijevanje programskih koncepata. Uključivanje distraktora, kao i razvoj adaptivnih slagalica, dodatno poboljšava proces učenja prilagođavajući se trenutnom nivou znanja učenika i održavajući ih u zoni proksimalnog razvoja, gdje su izazvani, ali još uvijek sposobni napredovati.

Pored toga, Parsonove slagalice pokazale su se korisnima ne samo za vježbanje i učenje već i kao alat za ocjenjivanje znanja učenika. Njihova sposobnost da automatski ocjenjuju rješenja i pružaju detaljne povratne informacije čini ih vrijednim alatom za nastavnike.

Kombinacijom tradicionalnih metoda poučavanja i inovativnih pristupa kao što su Parsonove slagalice, obrazovni programi mogu pružiti sveobuhvatan i učinkovit pristup učenju programiranja. Tradicionalne metode osiguravaju potrebnu teorijsku podlogu, dok inovativne metode pružaju priliku za praktično iskustvo i aktivno učenje.

3 LITERATURA

Ericson, B. J., Margulieux, L. E., & Rick, J. (2017). Solving Parsons problems versus fixing and writing code. *Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17)* (str. 20-29). New York, NY, USA: Association for Computing Machinery. Dohvaćeno iz https://dl.acm.org/doi/pdf/10.1145/3141880.3141895

Fincher, S. (1999). What are we doing when we teach programming? *In Frontiers in Education '99* (str. 12a41-5). IEEE. Dohvaćeno iz https://ieeexplore.ieee.org/document/839268

Ihantola, P., & Karavirta, V. (2011). Two-dimensional Parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education: Innovations in Practice, 10*, str. 119-132. Dohvaćeno iz https://jite.org/documents/Vol10/JITEv10IIPp119-132Ihantola944.pdf

Lister, R. (2004). The trials and tribulations of teaching software engineering to a novice programmer. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education.*

Luxton-Reilly, A. (2016). Learning programming in pairs: a pedagogical tool. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education.*

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. *Proceedings of the 8th Australasian Conference on Computing Education (ACE '06), Volume 52* (str. 157-163). AUS: Australian Computer Society, Inc. Dohvaćeno iz https://dl.acm.org/doi/pdf/10.5555/1151869.1151890

Shah, M. (2020). *Exploring the use of Parsons problems for learning a new programming language.* Electrical Engineering and Computer Sciences University of California at Berkeley. Dohvaćeno iz https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-88.pdf

Weinman, N. (2018). Using Parsons problems to teach CS1 students Python. *Proceedings of the 49th ACM technical symposium on Computer Science Education*.

POPIS SLIKA