Novi pristup računalnom programiranju u primarnom obrazovanju

Kukrić, Ines

SVEUČILIŠTE U ZAGREBU

UČITELJSKI FAKULTET

ODSJEK ZA UČITELJSKE STUDIJE

INES KUKRIĆ

DIPLOMSKI RAD

NOVI PRISTUPI RAČUNALNOM PROGRAMIRANJU U PRIMARNOM OBRAZOVANJU

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
UČITELJSKI FAKULTET
ODSJEK ZA UČITELJSKE STUDIJE
Zagreb

DIPLOMSKI RAD

Ime i prezime pristupnika: Ines Kukrić

TEMA DIPLOMSKOG RADA: Novi pristupi računalnom programiranju u primarnom obrazovanju

MENTOR: izv. prof. dr. sc. Mario Dumančić

SUMENTOR: dr. sc. Nataša Rogulja, predavač

Zagreb, 2020.

SAŽETAK

Tradicionalna se nastava mora i može značajno unaprijediti suvremenim tehnikama poučavanja i učenja. Dugo su već istraživačima predmet interesa načini na koje početnici usvajaju programske koncepte i izazovi s kojima se susreću u poučavanju početnika programera. Računarska znanost koja izučava računala i algoritamske procese ima za zadatak prepoznati što je učenicima zanimljivo. Učenici trebaju programirati odnosno kreirati i konkretno upotrebljavati algoritme, a ne programirati samo kako bi programirali. Uloga učitelja je identificirati kada učenici stvaraju miskoncepcije i modificirati način poučavanja primjeren stvaranju pravilnih mentalnih modela programskih koncepta. U ovom diplomskom radu naglasak je na razvoju računalnog razmišljanja i računalnog programiranja odnosno predloženi su metodički naputci za obradu programskog koncepta *while* petlje pod pretpostavkom da takvim pristupom učenici neće razviti pogrešne koncepte o petlji te da će koncept programske uvjetne petlje moći primijeniti i na druge programske jezike neovisno o sintaksi. Odabrana je *while* petlja, jer se pokazalo kako početnici imaju poteškoća u razumijevanju te programske strukture. Metodički naputci obrađeni su po teoriji konstruktivizma u Python programskom jeziku. Odabran je Python programski jezik jer se pokazalo kako je pogodan za početnike programere zbog jednostavne sintakse i više programskih paradigmi, što olakšava tranziciju na druge programske jezike.

Ključne riječi: programiranje, programski jezik Python, while petlja

SUMMARY

Traditional teaching must and can be significantly improved by modern teaching and learning techniques. Researchers have long been interested in the ways in which beginners adopt programming concepts and the challenges they face in teaching novice programmers. Computer science studies computers and algorithmic processes have the task of recognizing what is interesting to students. Students need to program or create and specifically use algorithms, not just program to program. The role of the teacher is to identify when students create misconceptions and modify the teaching method appropriate to creating proper mental models of programming concepts. In this thesis, the emphasis is on the development of computer thinking and computer programming, i.e. methodological guidelines for processing the programming concept of the while loop are proposed, assuming that such an approach will not develop erroneous concepts of the loop and that the concept of programming conditional loop will be able to be applied to other programming languages, regardless of syntax. The while loop was chosen because beginners were shown to have difficulty understanding that program structure. Methodical instructions are processed according to the theory of constructivism in the Python programming language. Python programming language was chosen because it proved to be suitable for novice developers due to its simple syntax and multiple programming paradigms, which facilitates the transition to other programming languages.

Keywords: programming, Python programming language, while loop

SADRŽAJ

UVOD

Računalne znanosti su u posljednjih nekoliko desetljeća snažno promijenile svijet oko nas. Začetnik umjetne inteligencije Alan Turing je 1950. godine u svojoj igri imitacije (engl. *imitation game*) predložio da ispitivač pokuša determinirati radi li se o čovjeku ili stroju i tako pokušao odgovoriti na postavljeno pitanje, „mogu li strojevi razmišljati?“ (Turing, 1950). Već 1997. godine, IBM-ovo računalo *Deep Blue* je pobijedilo svjetskog prvaka Kasparova u šahu. Odgojno-obrazovni proces (nastava) je interakcija odgoja i obrazovanja te potreba i interesa društvenih postignuća i vrijednosti. Jedna od temeljnih kompetencija 21. stoljeća je digitalna pismenost koja je neophodna pojedincu kako bi mogao upotrebljavati računalo i računalne sustave. Nastavni sadržaj treba prilagoditi potrebama suvremenog društva. Prema Pastuoviću (1999) ciljevi odgojno – obrazovnog procesa se ne određuju na temelju rezultata proučavanja stvarnih potreba učenikova razvoja i sustava vrijednosti prirodne i društvene sredine. Nastavni proces je premalo dinamičan, razvojan i kreativan. Učenik je pasivan primatelj gotovih znanja, a učitelj je temeljni subjekt. Ovakav pristup bi trebao u potpunosti biti zamijenjen suvremenom nastavom koja se temelji na konstruktivizmu. Utemeljitelj i začetnik konstruktivizma te učenja otkrivanjem je američki psiholog Jerome S. Brunner. Prema njemu, učenik se nalazi u centru paženje te postaje aktivni subjekt nastavnog procesa.

Tek 2016. godine računalno razmišljanje se prvi put spominje u okviru nastave predmeta Informatika u Nacionalnom kurikulumu nastavnog predmeta Informatika. Računalno razmišljanje i programiranje razvijaju sposobnosti logičkog razmišljanja i mentalnih procesa analize, usporedbe sinteze i apstrakcije, kreativnosti, zadavanja jasnih uputa te rješavanja problema odnosno logičkog povezivanja. Naglasak bi trebao biti na razvoju mentalnih sposobnosti koje su potrebne svakom pojedincu neovisno o budućoj odabranoj profesiji.

Učenjem programiranja učenik uči kako primijeniti pravilnu automatizaciju rješenja uporabom algoritamskog razmišljanja. Perma Kaczmarczyk (2010), učenicima je programska *while* petlja jedan od najproblematičnijih programskih koncepta za

usvojiti. U ovom diplomskom radu u kombinaciji načela konstruktivizma, PAR modela i postojećih metoda i strategija izrađeni su metodički naputci za jednostavnije usvajanje programskog koncepta petlje kako bi učenici mogli konkretno stvarati i primjenjivati algoritam *while* programske petlje na rješavanje zadanog problema te predviđati, analizirati i vrednovati rješenje pseudokoda testiranjem programskog koda u Python interpreterskom okruženju.

1. PROGRAMIRANJE U OSNOVNOJ ŠKOLI

1.1 Kurikulum

„Teorija kurikuluma nastala je i afirmirala se u Americi u prvoj polovici dvadesetog stoljeća“ (Cindrić, 2010, str. 78). Riječ *curriculum (lat.)*, tijek, slijed, kretanje u suvremenoj pedagogiji, označava strategiju reforme škole. Kurikulum je „obrazloženi sklop odluka o ciljevima, sadržajima, metodama i organizaciji učenja odnosno poučavanja“ (Meyer, 2002, str. 17). Nikada nije dovršen i povezan je s okruženjem pa ga to čini otvorenim sustavom.

1.2 Pregled zastupljenosti nastave programiranja

Dugo je pitanje obaveznosti nastave Informatike u osnovnim školama bilo tema rasprave. Način na koji se organizacija predmeta Informatike mijenjala s okruženjem u Republici Hrvatskoj, bit će prikazano u ovom poglavlju.

Prema reformskom projektu *Hrvatski nacionalni obrazovni standard* (HNOS), u kojem je naglasak na učeniku, a ne na sadržaju, izrađen je *Nastavni plan i program za osnovnu školu*[1] u Republici Hrvatskoj, koji se primjenjivao od 2006./2007. (do 2016). školske godine. Tom odlukom prestala je važiti Odluka o nastavnom planu i programu iz Prosvjetnog vjesnika iz 1999. godine za sve predmete osim za program tehničke kulture i izborni predmet informatike od 5. do 8. razreda. Nastavnim planom se određuje „koji će se nastavni predmeti i područja učiti u određenoj školi, kojim redoslijedom i u kojem vremenu“, a nastavni program je okvir kojim se „definira sadržaj odnosno popis nastavnih cjelina, tema i jedinica, a ne djelatnost“ (Cindrić, 2010, str. 78-79). Cilj programa je bio osposobljavanje učenika za upotrebljavanje računala te korištenje algoritamskog načina razmišljanja u svakodnevnim problemima. Ciljevi, zadaće i odgojno-obrazovna postignuća su detaljno i čvrsto određeni za svaki nastavni sat po razredima te nisu dopuštali fleksibilnost. U programu se za nastavnu cjelinu „Rješavanje

[1]URL:https://www.azoo.hr/images/AZOO/Ravnatelji/RM/Nastavni_plan_i_program_za_osnovnu_skolu_-_MZOS_2006_.pdf (1.srpnja 2020)

problema i programiranje“ savjetovalo korištenje jezika u kojima se programira uz pomoć grafičkog objekta u formi kornjače, kao što su Logo ili Python ili korištenje proceduralnih jezika poput C, BASIC, ili Fortran programskih jezika. Nije bilo preporučeno korištenje oba, a za svaki programski jezik su, u sklopu teme, bili razrađeni ključni pojmovi i obrazovna postignuća. Algoritam petlje te uporaba naredbi za petlju bez logičkog uvjeta se spominju prvi puta u 6. razredu. Petlja s logičkim uvjetom se obrađuje u 7. razredu, a tek u 8. razredu se spominje primjena programiranja u drugim predmetima kao što su matematika, fizika i kemija.

Nacionalni kurikulum temeljni je dokument na razini države, odnosno odgojno-obrazovnog sustava te se prema njemu izrađuju ostali dokumenti ili kurikulumi prije svega u školi. *Nacionalni okvirni kurikulum*[2] je dokument iz 2011. godine u kojem je naglasak na prijelaz sa sustava usmjerenog na sadržaj na kompetencijski sustav i ishode učenja (Cindrić, 2010, str. 88). U sklopu tehničkog i informatičkog područja u rješavanju problema pomoću računala spominju se „Osnove programiranja za učenike od 1. – 4. razreda osnovne škole (prvi ciklus), od 5. – 6. razreda (drugi ciklus) i od 7. – 8. razreda (treći ciklus)“. Tek u trećem ciklusu se u očekivanim učeničkim postignućima navodi upotrebljavanje programske petlje.

U travnju 2016. godine, na čelu s psihologom Borisom Jokićem, eksperimentalna radna skupina za provođenje *Cjelovite kurikularne reforme* – prve mjere kojom je započela realizacija *Strategije obrazovanja, znanosti i tehnologije* prihvaćene u Hrvatskome saboru 2014. godine, izdala je nacrt prijedloga *Okvir nacionalnoga kurikuluma*[3]. U sklopu strategije izdan je prijedlog – *Nacionalni kurikulum nastavnoga predmeta Informatika*[4]. Predmet Informatike su podijelili u četiri domene: E-društvo, Digitalna pismenost i komunikacija, Računalno razmišljanje i programiranje te Informacija i digitalna tehnologija. U posljednjih par godina u višim razredima se za programiranje najčešće koristi programski jezik Python. U ovom dokumentu svaka domena je podijeljena na ishod, razradu ishoda, razine usvojenosti te na preporuke za ostvarenje odgojno-obrazovnih ishoda. Učitelj/ica ima puno veću slobodu u izboru

[2] URL: https://www.azoo.hr/images/stories/dokumenti/Nacionalni_okvirni_kurikulum.pdf (1.srpnja 2020)
[3]URL:http://www.skole.hr/upload/portalzaskole/newsattach/12945/Nacrt_prijedloga_ONK_nakon_strucne_rasprave_%281%29.pdf (1. srpnja 2020.)
[4] URL: http://mzos.hr/datoteke/15-Predmetni_kurikulum-Informatika.pdf (2. srpnja 2020.)

metoda, pomagala i programskih jezika kojima će zadovoljiti ishode. Predmet Informatika se kroz cijelo osnovnoškolsko obrazovanje provodi kroz 70 sati godišnje.

U prosincu 2017. godine, u skladu s NOK-om, donesen je *Dokument tehničkog i informatičkog područja kurikuluma* kao prijedlog nakon javne rasprave na čelu s ministricom Blaženkom Divjak. U domeni „Rješavanje problema i programiranje“ u 3. ciklusu (6. – 8. razred) prvi puta je spomenuto da učenik primjenjuje uvjetne naredbe i petlje pomoću programskog jezika. Nakon javne rasprave i recenzija iste te godine Ministarstvo znanosti i obrazovanja izdalo je *Nacionalni kurikulum nastavnog predmeta Informatika* (prijedlog). U domeni „Računalno razmišljanje i programiranje“ već u 2. razredu jedan od ishoda je da učenik stvara niz uputa u kojem upotrebljava ponavljanje. U dokumentu iz 2018. godine, koji je nadopuna prethodno spomenutom dokumentu iz 2017. godine prvi put se napominje kako učenik u 4. razredu, pri rješavanju logičkog zadatka, može koristiti računalo. U istom se dokumentu ističe kako učenik u 5. razredu stvara algoritam.

U Zagrebu, 12. veljače. 2018. godine, Ministrica znanosti i obrazovanja – Blaženka Divjak kroz projekt *Škola za život* donijela je odluku o donošenju kurikuluma za nastavni predmet Informatike za osnovne škole i gimnazije u Republici Hrvatskoj. Domene su ostale iste, a odluka se primjenjuje za učenike od 5. – 6. razreda kada predmet Informatika mijenja status iz izbornoga u obavezni predmet školske godine 2018./2019. Nastava informatike za prva četiri razreda obavlja se tek u nekim školama kroz Izvannastavne aktivnosti ili preko Udruga koje roditelji plaćaju. Ovim kurikulumom od školske godine 2020./2021. nastava Informatike postaje izborna za prva četiri razreda. Orijentiranjem ka ishodima učenja u nastavi, pružena je puno veća sloboda pri organiziranju nastave Informatike, ali i učenja programiranja. Vidljivo je kako nije jednostavno odrediti u kojem dobnom razdoblju bi učenici trebali učiti pojedine programske koncepte.

2. OSNOVNI KONCEPTI PROGRAMIRANJA

2.1 Program i programiranje

Program je niz instrukcija koje određuju kako će se nešto izračunati. Izračun može biti matematički, primjerice rješavanje sustava jednadžbi, ali i simbolički, kao što je pronalaženje i zamjena teksta u dokumentu. Neke od osnovnih instrukcija su ulaz, izlaz, aritmetika, uvjetno izvršavanje i ponavljanje. Programiranje predstavlja proces pisanja naredbi, izraza i uputa koje računalo pokušava izvršiti. Programski kôd je skup naredbi i uputa koje formiraju program koji se nakon što je pokrenut, izvršava na računalu. Programski jezik služi za komunikaciju između čovjeka i računala (Downey, 2014). Osoba koja piše kôd, odnosno programira, zove se programer. Računalo razumije strojni jezik (engl. *machine code)* pisan u binarnom kodu koji se sastoji od nula i jedinica. Pisanje strojnim jezikom je nepraktično pa su razvijeni simbolički jezici (niži i viši), sintaksom bliži čovjeku, koji se prevode. Postoje tri vrste prevoditelja: asembler (*engl. assembler),* kompajler (*engl. compiler*) i interpreter (*engl.interpreter).* Razlikujemo niže programske jezike poput Asemblerskog jezika u kojem svaka instrukcija predstavlja instrukciju strojnog jezika, a program napisan u tom jeziku zove se izvorni program (*engl. source code)* i on se prevodi pomoću asemblera. Program napisan u asemblerskom jeziku nije moguće izvršavati na različitim računalima radi drukčije arhitekture računala. Kako bi se programera rasteretilo, osmišljeni su viši programski jezici s jednostavnijom sintaksom, koji se pomoću prevoditelja mogu izvršavati na svim računalima. U višim simbolički jezicima (C, C++, Java, Python itd.) napisani program se prevodi pomoću kompajlera i interpretera. Oba prevoditelja pretvaraju izvorni kôd iz višeg programskog jezika u strojni jezik odnosno binarni kôd. Razlika je u tome što interpreter prevodi program liniju po liniju, a kompajler prevodi cijeli kôd u binarni kôd i zbog toga je brži. Python koristi interpreter i to će biti detaljnije objašnjeno u četvrtom poglavlju ovog diplomskog rada.

Prema Budinu (2017) korištenje Python programskog jezika za poučavanje programiranja u osnovnoj školi ima mnogo prednosti. Početnici ga lako savladavaju, sintaksa je jednostavnija nego u programskim jezicima, poput Jave i C, a sve naučene

koncepte učenik može primijeniti pri učenju bilo kojih drugih programskih jezika kasnije. Budin ističe kako je Python izvrsna uvertira za učenje programskog jezika C u srednjim školama ili na fakultetima. Poznata izreka glasi *kako je C programski jezik latinski jezik među programskim jezicima*. U sljedećim potpoglavljima bit će prikazani koncepti Python C programksog jezika kako bi se ukazalo na sličnosti i razlike između dva programska jezika. Svaki učitelj/ica koji poučava programiranje trebao/la bi razlikovati niže navedene programske koncepte kako bi adekvatno oblikovao/la nastavne sate te tako pomogao/la učeniku da ostvari određene ishode učenja.

2.2 Tipovi podataka

Prema Downeyu (2014) vrijednost je jedna od osnovnih jedinica podataka s kojom program radi. Tip podataka je kategorija vrijednosti. Tipovi podataka označavaju potrebnu količinu memorije za pohranu podataka na računalu. Razlikujemo primitivne i izvedene tipove podataka. Primitivni tipovi podataka su ugrađeni u programski jezik, imaju istu vrijednost i u pravilu zauzimaju uvijek istu količinu memorije. Neznatno se broj bajtova određenog tipa podataka razlikuje od programskih jezika, ali i operacijskog sustava na kojem se program prevodi. Prema Kernighan i Ritchie (1978) proceduralni programski jezik C razlikuje *integer* (*byte, int, short, long*), *floating – points* (*float* i *double*), *character* (*char*), *void* i *boolean* (*true* i *false*). Primitivni tipovi podataka u Python programskom jeziku su *string* (znakovni tip podataka), *integer* – (cjelobrojni tip podataka), *float* – realni broj s pomičnim zarezom (numerički tip podataka), *boolean* (logički tip podataka).

Količinu memorije koju složeni ili „ne-primitivni“ tipovi podataka zauzimaju, za razliku od primitivnih, definira programer. Složeni tipovi podataka se koriste za pohranjivanje skupine vrijednosti. Neki od složenih tipova podataka su strukture, polja, liste, stabla, redovi, hrpe, gomile, datoteke itd. Python programski jezik ima puno više složenih tipova podataka nego C programski jezik.

Zaključujemo kako su primitivni tipovi podataka vrlo slični u većini programskih jezika. Treba naglasiti kako C ne poznaje *string* već poznaje niz *char* znakova. Ovo je važno uočiti jer pri korištenju znakova, niza znakova i datoteka u kodu C programskog jezika treba dobro poznavati koliko memorije zauzima koji tip podataka. Izbor ispravnog tipa podataka ima utjecaj na optimizaciju memorijskog prostora bilo to u strukturama podataka, ili alokaciji memorije u polju u C-u, kako ne bi došlo do gubljenja memorije ili usporavanja rada procesora.

Nadalje, radi ispravnog izvršavanja programskih naredbi važno je da računalo zna koji je tip podatka dodijeljen varijabli. Python jezik je jednostavniji od C programskog jezika, koji je vrlo osjetljiv na deklariranje tipa podataka, jer nema potrebe za deklariranjem tipa podataka varijable. **Varijabla** (engl. *variable)* je rezervirana memorijska lokacija za spremanje vrijednosti nekog tipa podatka, a koja se može mijenjati tijekom izrade i izvođenja programa. Ime varijable može biti proizvoljno dugačko i treba biti smisleno – ukazati za što se varijabla koristi. Ime varijable se najčešće sastoji od slova, brojki i donje crtice, ali mora počinjati sa slovom. Ime se dodjeljuje pomoću naredbe pridruživanja „=“. Doseg varijable može biti globalan – vidljiva u cijelom programskom kodu ili lokalan – vidljiva samo unutar funkcije. Varijable, odnosno tipovi podataka, se mogu eksplicitno i implicitno mijenjati iz jednog tipa podatka u drugi (engl. *type casting*).

2.3 Izrazi i operatori

Prema Downeyu (2014) izraz (engl. *expression)* je kombinacija varijabli i operatora koji prikazuju neku vrijednost. Iskaz (engl. *statement*) je jedinica koda koja se izvršava. Druga riječ koja se često koristi u hrvatskom jeziku za iskaz je *naredba.* Operatori su simboli koji predstavljaju računske operacije, a vrijednosti na koje se primjenjuju te operacije nazivaju se *operandi.* Vrsta operatora određuje vrstu izraza, npr. upotrebom logičkog operatora izraz postaje logički. Ako u jednom izrazu postoji više operatora treba poštivati prioritet izvršavanja operatora. Razlikujemo četiri osnovne

skupine operatora: aritmetičke operatore, operatore na bitovima, relacijske operatore i logičke operatore.

2.4 Uvjetne naredbe

Grananje je programska struktura koja omogućava različit tijek programa ovisno o rezultatu postavljenog uvjeta ili uvjetne naredbe. Uvjetna naredba (engl. *conditional statement)* određuje koja se grana izvršava, a sastoji se od uvjeta – logičkog izraza i tijela u kojem može biti još naredbi, petlji, naredbi za ispis (engl. *print*) itd. Kako bi razumjeli je li uvjet istinit ili lažan koristimo se tipom podataka *boolean*. Nad tim tipom podatka možemo koristiti logičke operatore. Npr. logički izraz „a < b“ provjerava je li „a manje od b“ pomoću logičkog operatora „<“. U C-jeziku postoje dvije vrste uvjetnih naredbi; naredbe za donošenje odluka i naredbe odabira. Naredbe za donošenje odluka se dijele na:

- *if*, gdje se svi uvjeti izvršavaju.
- *if-else,* gdje se izvršava *else* uvjet u slučaju da je *if* uvjet neistinit.
- *if-else-if,* gdje *if* izvršava prvi uvjet, *else if* – drugi uvjet, a *else* sve ostale uvjete.
- ulančana *if* naredba.

Naredba odabira je alternativa *if-else-if* izrazu i sastoji se od ključne riječi *switch* i mogućih slučajeva koji će se izvršiti nakon kojih slijedi *break* izraz.

Python programski jezik nema ovakvu alternativu, već ima *if* naredbu, *elif* – ako uvjet prije nije istinit izvršava se uvjet *elif* naredbe te *else* – izvršava se, ako uvjet koji prethodi nije istinit.

2.5 Petlje

Programska petlja je struktura koja omogućava izvršavanje dijelova koda više puta. Petlje se sastoje od četiri dijela: inicijaliziranja (dodjeljivanje vrijednosti) i/ili deklariranja (koji je podatkovni tip) varijable koja se inkrementira (povećava), izraza

kojeg testiramo, tijela petlje i inkrementa. C programski jezik ima *while, do-while i for petlju*. Python jezik ima *while* i *for petlju* kao osnovne oblike. Unutar uvjeta i tijela petlje često se koriste uvjetne naredbe i uvjetno grananje.

2.5.1 For petlja

Ova petlja se najčešće koristi kada je unaprijed poznat broj ponavljanja. U Python-u petlja može iterirati primjerice kroz *string*, polje ili koristeći funkciju *range()* koja zadano kreće od nule, gdje programer ima mogućnost sam odrediti početni broj i iterirati do određenog broja. Ugrađeno je da se petlja povećava za jedan, no dodavanjem trećeg parametra funkciji programer određuje za koliko će se petlja povećavati. U tijelu petlje se može koristiti ključna riječ *else* kojom se nalaže računalu da izvrši sve ono što se nije izvršilo *for* petljom, a pomoću *continue* može se zaustaviti trenutno izvođenje petlje i nastaviti sa sljedećim uvjetom. Petlja ne može biti prazna, ali ako iz nekog razloga jest, u Python-u se koristi naredba *pass* kako izlazna vrijednost programa ne bi bila pogreška. Petlje mogu imati različit izgled, ali su koncepti isti. Još jedan oblik je ugniježđena petlja, često korištena pri iteriranju kroz 2d polje u programskom jeziku C koje je zapravo u memoriji 1d polje, ali se ovakvim iteriranjem stvara privid dvodimenzionalnosti i lakše je manipulirati podatcima.

2.5.2 While petlja

U mnogim programskim jezicima *while* petlja omogućava programskom kodu da se izvršava sve dok je logički uvjet istinit. Za razliku od C-a, u kojem postoji *do-while* petlja koja se izvršava minimalno jednom, Python razlikuje *while* petlju s dvije različite naredbe.[5] *Continue* – zaustavlja trenutnu iteraciju i nastavlja sa sljedećom i *break* – naredba pomoću koje se petlja zaustavlja i ako je uvjet istinit. Može se dogoditi da se petlja uopće ne izvrši jer je uvjet lažan (engl. *False*). Specifično za Python, dozvoljena je *else* naredba na kraju *while* petlje pomoću koje izvršavamo izraz nakon što izađemo iz

[5] URL: https://wiki.python.org/moin/WhileLoop / (5. lipnja 2020)

while petlje. Postoje i ugniježđene petlje. Moguće su i beskonačne petlje u kojima se uvjet nikada ne evaluira u laž. Ovakve se petlje vrlo često slučajno izvršavaju kod početnika programera. Naglasak u ovom diplomskom radu je na ***while*** **petlji** jer se pokazala kao zahtjevan programerski koncept za usvajanje i potpuno razumijevanje.

2.6 Funkcije

Funkcije su globalni objekti odnosno naredbe koje izvršavaju određene zahtjeve samo kada su „pozvane“ preko imena. Prednost funkcija je raščlanjivanje problema na manje probleme, programski kôd je pregledniji i rješavanje problema je jednostavnije. Razlikujemo ugrađene funkcije kao *printf* u C programskom jeziku – šalje formatirani zapis na izlazni tok i *scanf* – čita unos s ulaznog toka, te korisničke funkcije koje su samostalni dijelovi koda. Funkcija se definira tako da se naznači tip funkcije koji može biti *int, char, float, double* i *void*, dodijeli ime i definira lista parametara. Vrijednosti iz funkcija se vraćaju ključnom riječju *return* osim u slučaju *void* funkcije koja nema povratnu vrijednost. Funkcije se izvršavaju pozivima kojima se prosljeđuju parametri. Mogu se pozivati u petljama, izrazima ili argumentima drugih funkcija. Jednom napisana funkcija se može pozivati s različitim argumentima (broj argumenata mora odgovarati broju parametara), više puta tijekom koda. U Pythonu, funkcija se definira ključnom riječju *def*. Funkcija ne mora imati argumente ili, ako nije poznato koliko se argumenata prosljeđuje, dodaje se * pa funkcija prima *tuple* argumenata. *Tuple* je sekvenca slična listi, ali za razliku od liste, koristi zagrade i nepromjenljiva je. Rekurzivna funkcija je funkcija koja sama sebe poziva prilikom izvršenja programa, a rekurzivna funkcija u Python programskom jeziku ima limit od 1000 ponavljanja. Nakon toga program ne dopušta funkciji da se pozove. Svakim rekurzivnim pozivom funkcije udvostručava se količina potrebna za alokaciju memorije na stogu. Ovakvim ograničenjem sprječava se *stack overflow.*

3. METODIKA RAČUNALNOG RAZMIŠLJANJA I PROGRAMIRANJA

3.1 Metodika obrazovanja

„Metodika znanstvenog područja bavi se svim znanostima iz procesa obrazovanja. Obrazovanje shvaćamo kao zadovoljavanje spoznajnih, doživljajnih i psihomotornih interesa pojedinca aktivnim usvajanjem i daljim razvijanjem određenih kulturnih i civilizacijskih dostignuća“ (Bognar i Matijević, 1993, str. 15). Metodika je konkretizacija didaktike, koja je dio pedagogije – znanosti o odgoju i obrazovanju. Metodika obrazovanja se odnosi na kognitivni aspekt obrazovanja kada je ono proces stjecanja znanja i vještina, ali i doživljavanja.[6]

„Uporaba računala u obrazovanju počela je šezdesetih godina 20. stoljeća, odmah nakon što su računala postala dostupna obrazovnim institucijama“ (Perić i sur., 2017, str. 3). Međutim u Hrvatskoj i dalje ne postoji „univerzalna“ metodika Informatike kao što postoji metodika Hrvatskoga jezika ili metodika Matematike te zbog toga učitelji Informatike poučavaju programske koncepte na različite i pomalo zastarjele načine, ne samo u osnovnim školama već i u višem obrazovanju. Prema novom kurikulumu projekta *Škola za život*, težište obrazovnog procesa u predmetu Informatika treba biti na rješavanju problema i programiranju kako bi se poticalo razvijanje računalnog načina razmišljanja koje omogućuje razumijevanje, analizu i rješavanje problema i algoritama te bi se takvi načini razmišljanja trebali prenositi u druga područja nastave i u svakodnevni život.[7]

3.2 Računalno razmišljanje

Pojam računalno razmišljanje (engl. *computational thinking)* prvi je osmislio Seymour Papert (1980) – jedan do glavnih autora Logo programskog jezika. Prema Wingu (2008, 2017) računalno razmišljanje obuhvaća misaone procese uključene u

[6] URL: https://www.enciklopedija.hr/natuknica.aspx?id=40439 (5. svibnja 2020)
[7] URL: https://skolazazivot.hr/wp-content/uploads/2020/06/INF_kurikulum.pdf (5. svibnja 2020)

formuliranje i izražavanje rješenja na takav način da ih čovjek ili stroj može učinkovito provesti, i ne podrazumijeva samo rješavanje već i osmišljavanje problema. Prema istraživanju (Selby & Woollard, 2014) računalno razmišljanje se sastoji od:

- Misaonih procesa – naglasak je na računalnom razmišljanju kao misaonom procesu rješavanja problema preko pronalaženja rješenja;
- Apstrakcija – prema Wingu (2008) apstrakcija je najviši i drukčiji način razmišljanja;
- Dekompozicije – „raščlanjivanje“ problema na manje probleme;
- Algoritama – „postupak koji opisuje kako se neki problem rješava“ (Kusalić, 2014. str 38);
- Evaluacije – pronalazak najefektivnijeg rješenja problema;
- Generalizacije – pronađeno rješenje jednog problema moći primijeniti na slične probleme ili prepoznati uzroke;
- Automatizacije – rješenje odnosno algoritam koji se izvodi na računalu ili pomoću robota.

Prema Denningu i Tedreu (2019) računalno razmišljanje je mentalna vještina dizajniranja programa, objašnjavanja i interpretiranja odnosno to je univerzalna vještina koja potiče preciznost i sustavnost, a može se primijeniti u različitim situacijama svakodnevnog života. Računalno razmišljanje se može učiti i bez računala pa je kao takvo vrlo primjereno za prva četiri razreda, čak i za vrtić. Posljednjih godina istražuju se najbolji pristupi poučavanju programiranja i računalnog razmišljanja u osnovnoj školi. Godine 2018. održana je Internacionalna konferencija o računalnom razmišljanju (CTE2018) gdje su prikazana istraživanja na tom području iz raznih dijelova svijeta. U jednom istraživanju (Faber, i sur., 2017) autori su prikazali kako se programski koncepti poput algoritama, petlji, varijabli i uvjeta mogu poučavati preko *Unplugged programming,* pa su tako za nastavni sadržaj usvajanja *ponavljanja* predložili upotrebu igraćih karata, za *sortiranje*, sortiranje igračaka po određenom uvjetu i sl. Ovakav pristup je u suštini poučavanje programskih koncepata poštivajući načelo zornosti, koristeći didaktički materijal i igru kao nastavnu metodu. Nastavni plan i program za osnovnu školu iz 2006. godine navodi kako predmet Informatika treba omogućiti

učenicima upoznavanje s *informacijskom i komunikacijskom tehnologijom*. U eksperimentalnom projektu Škola za život, predmet Informatika je podijeljen na četiri domene: *e-Društvo, Digitalna pismenost i komunikacija, Informacije i digitalna tehnologija te domenu Računalno razmišljanje i programiranje*. Tek 2016. godine se u prijedlogu Nacionalni kurikulum nastavnog predmeta Informatika prvi put računalno razmišljanje spominje u okviru nastave predmeta Informatika. Računalno razmišljanje nije sinonim za programiranje, ali poučavanje računalnog razmišljanja prije i tijekom učenja programiranja uvelike olakšava formiranje ispravnih mentalnih modela programskih koncepta.

3.3 Računalno programiranje

Glavni cilj poučavanja uvodnog programiranja je shvaćanje osnovnih programskih koncepta koji se kasnije jednako tako dobro mogu primjenjivati i na druge programske jezike (Bubica i Boljat, 2014). Prema Mayeru (1989) istraživanja su pokazala kako je najproduktivniji način učenja programskog jezika prvo shvatiti temeljne koncepte na prirodnom jeziku. Posljednjih nekoliko godina, istraživači se zalažu za *unplugged programming* poučavanje iz kojeg se kasnije prelazi u konkretno programiranje na računalu. Istraživači Kong i sur. (2018) su prikazali kako učenje programiranja utječe na razvoj i drugih kognitivnih sposobnosti kao što su kreativno razmišljanje, matematičke vještine i rasuđivanje. Istraživanja iz područja kognitivne psihologije (Pea i Kurland, 1984) upućuju na to kako novo učenje treba biti povezano s prethodnim znanjem. Kako bi došlo do kognitivnih promjena, učenici moraju biti aktivno uključeni u učenje novih koncepta. Temeljne ideje na osnovu kojih se izvodi nastava su didaktička načela, poput načela primjerenosti, zornosti, interesa, svjesnosti i aktivnosti, sistematičnosti i postupnosti, trajnosti znanja, vještina i navika. Izbor metoda i oblika rada u nastavi ovisi o učitelju. Do sada se računalno programiranje u školama poučavalo pomoću LOGO, Scratch i Python programskog jezika. Premda su se Mayer i drugi istraživači dotakli načina poučavanja programiranja i prije više od trideset godina, tek posljednjih deset godina mogu se uočiti male promjene u tom smjeru. Novi pristup poučavanju posljednjih godina ogleda se kroz nastavne metode koje proizlaze iz teorije

konstruktivizma, gdje se učenik nalazi u centru pažnje te postaje aktivni subjekt nastavnog procesa. Naglasak je na konstruiranju znanja odnosno na aktivnom učenju, a ne na pasivnom apsorbiranju. De Zan (1999) smatra kako bi se tradicionalna nastava u potpunosti trebala zamijeniti suvremenom u kojoj bi se učiteljevo poučavanje trebalo zamijeniti samostalnim radom i otkrivanjem učenika, a izlaganje problema zamijeniti samostalnim rješavanjem problema i planiranjem. Poučavanje se provodi ako dođe do nerazumijevanja, u suprotnom učitelj treba oblikovati istraživački proces i uspostaviti uvjete za samoučenje. Suvremene metode aktivnog poučavanja su istraživačka, simuliranje, igra i učenje putem rješavanja problema koje je najviši oblik učenja. Neke od metoda aktivnog učenja su: metoda kreativnog pisanja, meditativni oblici, igra po ulogama, projektno učenje, imaginacija, kreativni rad. [8] Jedna od ključnih strategija poučavanja programiranja, u okviru konstruktivizma, koja potiče prepoznavanje pogrešaka je strategija kognitivnog konflikta (engl. *Cognitive Conflict*). Smatra se kako prethodno stečeno znanje i postojeći koncepti utječu na to kako će učenik interpretirati i organizirati nove informacije koje primi. Suvremena nastava se klasificira kao problemska nastava, istraživački usmjerena nastava, učenja kroz igru i učenja otkrivanjem. Strategija koja naglašava angažman učenika je kognitivno naukovanje (engl. *Cognitive Apprenticeship)* koja djeluje u tri faze: modeliranje, skaliranje i nestajanje. U fazi modeliranja učitelj pruža učenicima konceptualni model procesa. Nakon toga učenici u fazi skaliranja sami rješavaju zadatak pod nadzorom učitelja. U ovoj je fazi iznimno važno da učitelj pomaže učenicima.

Bubica (2014) naglašava kako je za odabir strategija poučavanja iznimno važno poznavati na koji način početnici programeri usvajaju teške koncepte. Oni mogu slijediti sintaktička pravila, ali često ne znaju riješiti problem. Iz ovog razloga naglasak treba biti na usvajanju programskih koncepta, a ne samo na sintaksi. Zato je Python programski jezik iznimno pogodan. Kao i u drugim predmetima treba se poučavati od jednostavnog ka složenom. Međutim, kognitivne sposobnosti učenika ne treba podcjenjivati. Prema Bubici (2008) važno je pomoći učenicima stvoriti održive mentalne modele koji se mogu ostvariti percepcijom, imaginacijom te konstruktivnim raspravama. Mnoga istraživanja pokazuju kako korištenje „tema“, poput igara i robota ima dobar učinak na

[8] URL: https://www.os-kamenica.com/nastava/suvremene-metode-i-oblici-poucavanja (1. rujna 2020)

učenje programskih koncepta. „Tradicionalno“ programiranje se često poučava izvan konteksta stvarnog svijeta. Na tragu da se ovo ispravi izrađena su mnoga okruženja u kojima se koristi vizualizacija objekta. Jedno takvo okruženje je *Alice,* u kojem se vizualni objekti, poput životinja i relacije između njih te metode koje se koriste recimo za njihovo kretanje, automatski prevode u Java kôd, *Logo Blocks, Play* i drugi. Prema Linn i Dalbey (1989) ciljevi početnog programiranja su usvajanje znanja o svojstvima programskog jezika, razvijanje vještine oblikovanja programa i stjecanje sposobnosti prijenosa stečenog znanja pri rješavanju problema. Oblikovanje programa je tehnika u kojoj se problem rješava kombiniranjem svojstva jezika u jednu cjelinu. Smatra se kako učenik razumije algoritam ako je sposoban objasniti njegove korake prirodnim jezikom. Učenici često nailaze na problem pri shvaćanju algoritama.

U skladu s navedenim izazovima u procesu poučavanja i učenja uvodnog programiranja, u sljedećem poglavlju prikazana je osobna analiza autorice ovog diplomskog rada u poučavanju *while* petlje korištenjem suvremenih nastavnih metoda.

3.4 Zašto novi pristupi?

Svrha nastavnog procesa je pripremiti učenike da „budu razumni i značajni ljudi“ (Basariček, 1882, str. 54). Još davno prije Basaričeka, Ciceron je rekao da je *razum gospodar i kralj svega*. Bez obzira na filozofski stav učitelja, učenici bi nakon završenog školovanja trebali moći analizirati, generalizirati, klasificirati i kritički prosuđivati svijet oko sebe. Prema Bloomovoj taksonomiji iz 1956. godine, učenici koji žele shvatiti programske koncepte trebali bi razviti vještine iz kognitivne domene poput analiziranja, sinteze, primjene i vrednovanja nastavnog materijala odnosno programskog koda i algoritama. Prema Piagetu (1977) dijete tek u formalno-operacijskoj fazi kognitivnog razvoja može apstraktno razmišljati. Apstrakcija je temeljni koncept računalnog razmišljanja te potiče uporabu metakognitivnih vještina i omogućuje rad na složenim problemima razdvajajući ih u više jednostavnih problema. Iako kognitivne sposobnosti pojedinca ne napreduju jednakom brzinom kao i tehnologija, to ne znači da poučavanje programskih koncepta treba ostati isto kao i prije 20 godina. Kada učitelji razumiju

način na koji učenici uče i konstruiraju mentalne modele vezane uz određeni problem, tada mogu organizirati pravilno izvođenje nastave.

Veliki nedostatak trenutne organizacije školskog sustava je nedovoljna međupredmetna korelacija i nepovezanost nastavnog sadržaja sa svakodnevnim znanjima i iskustvima. Prema Poljaku (1988) nastavne metode su: metoda demonstracije, metoda praktičnih radova, metoda crtanja, metoda pismenih radova, metoda čitanja i rada na tekstu, metoda razgovora i metoda usmenog izlaganja. PAR (engl. *Present, Apply, Review*) model učenja je metoda aktivnog učenja novih nastavnih sadržaja organiziranih u tri dijela. Autor PAR metode je Geoff Petty, koji je revidirao Bloomovu taksonomiju i aktivno poučavanje te je prilagodio suvremenom dobu. U prvom, *Prezentiraj* dijelu ovog modela učenja učenik spoznaje nove pojmove koji su predstavljeni usmenim izlaganjem, problemskim pitanjem i razgovorom, čitanjem i radom na tekstu ili gledanjem video zapisa. U drugom, *Apliciraj* dijelu učenici su izloženi aktivnostima koje od njih zahtijevaju primjenu izrečenog i učenik tada stvara koncepte kojima daje smisao. U ovom dijelu učenik pred sobom ima produkt svojeg rada u kojem uočava što je ispravno usvojio, a što nije. Zadnji dio, *Revidiraj,* uključuje naglašavanje i povezivanje ključnih koncepta i pojmova s novim znanjem. Prema istraživanju Sweeney i Posavec (2017), primjena ovog modela u nastavi prirode pokazuje bolje rezultate kod onih učenika koji su učili po ovom modelu, nego kod onih koji su radili po tradicionalnom modelu. Na posljetku, nastavu bi trebalo organizirati postavljajući pitanja i organizirajući probleme koji su visoko na Bloomovoj taksonomiji, gdje bi svaka nastavna jedinica trebala sadržavati sva tri dijela: dio analize (*Zašto?*), sinteze (*Kako?*) i evaluacije.

Kada se upotrebljava pojam „računalno razmišljanje“ često se stavlja naglasak na upotrebu programiranja i IT vještina, umjesto na razvijanje intelektualnih sposobnosti ili kognitivnog aspekta pojedinca kroz proces programiranja. Računalno razmišljanje bi trebalo činiti temelj za učenje programskih koncepta. Koncept je utemeljena znanstvena spoznaja, a učenici često o nekom pojmu imaju predznanje odnosno *predkoncepcije*. Gick i Holyoak (1980, 1983) smatraju kako bi se pri poučavanju programiranja trebali koristiti konkretni modeli i kako izravna poduka o konceptualnim modelima može omogućiti analogno učenje i spoznaju te primjenu stečenog znanja u novim situacijama.

Nakon što učenik prikupi i sintetizira potrebne informacije, stvara konceptualni model te ga pretvara u relacijski model, odnosno logički modelira informacije. Dakle, konceptualno modeliranje je način na koji učenik kognitivno procesuira veze između objekata i entiteta. Prema Mayeru (1981) dva su polazišna načina poučavanja, prema kognitivnoj psihologiji, koja bi početnicima olakšala učenje programiranja. Prvo je korištenje konkretnih modela, a drugo poticanje objašnjavanja tehničkih informacija vlastitim riječima.

Za nastavu informatike u osnovnoj školi predviđeno je 70 sati godišnje, gdje je predmet Informatika podijeljen u četiri domene. Tako domenu *Računalno razmišljanje i programiranje* učitelj može obrađivati i do osamnaest nastavnih sati, što predstavlja dovoljno vremena za detaljnu razradu programskih koncepta i utvrđivanje jesu li učenici formirali pravilne mentalne modele. Prema Linn i Dalbey (1989) sastavnice vještine oblikovanja programa su: utvrđivanje ciljeva programa analizom zadataka; planiranje rješenja zadatka; dekompozicija složenog problema na jednostavnije; implementacija i sinteza rješenja u jedinstveni program; vrednovanje rješenja testiranjem; lociranje logičkih pogrešaka i promjene rješenja radi postizanja konkretnog programa. Programski problemi mogu biti vrlo složeni pa je potrebno razložiti program na više dijelova kako bi se svaki mogao nezavisno promatrati. U osnovnoj školi ovakav pristup je teško ostvariti pri učenju određenih programskih koncepta. Posebno je uočeno kako je učenicima shvaćanje *while* petlje iznimno zahtjevno. Do sada je naglasak bio na poučavanju ili preko *grafičkog prozora* za rad s kornjačinom grafikom ili preko matematičkih funkcija koje ne prate u potpunosti princip zornosti i apstraktnosti. Odnosno, *while* petlja se često poučava izvan konteksta stvarnog svijeta ili predmetnog okruženja, što učenicima mlađe dobi predstavlja problem pri kreiranju pravilnih mentalnih modela. Učenici imaju problem s konkretizacijom odnosno vizualizacijom „što i kako petlja radi“, jer se ta petlja najčešće koristi kao dio većih programa. U svom istraživanju Kong i sur. (2018.) naglašavaju kako programiranje pozitivno utječe na pravilno rješavanje problema (engl. *problem solving*). Pri učenju algoritama učenik kreira rješenje danog problema, program se izvršava nekoliko puta te odmah pruža povratnu informaciju. Svaki put kada se kôd izvrši, u program se unose unaprijed pripremljeni podaci, a rezultat se uspoređuje s vrijednošću pripremljenih podataka o odgovorima. Za analizu učenja algoritma koristi se

programski jezik tekstualnog tipa za usporedbu nizova i linija izvornog koda koji pruža povratne informacije o pogreškama ili rezultatima. Ovakav stil učenja algoritama nije razvijen u blok programiranju (engl. *block programming*), odnosno u programskom jeziku Scratch koji se trenutno koristi osnovnim školama, kao jedan od programskih jezika za učenje programiranja. Iz ovog razloga su kao didaktički materijal korištene datoteke u sklopu obrađenih nastavnih jedinica (vidi poglavlje 5).

„Bitno je da nastavnik dobro zna ono što predaje. To je *conditio sine qua non*“[9] (Pavleković, 1997, str 270). Istraživanje iz 2011. godine ukazalo je kako učitelji imaju siromašno znanje o onome što učenici zaista nauče na uvodnom predmetu programiranja (Linxiao, 2007). Programiranje je specifičan način razmišljanja pomoću kojega se od postavljenog problema dolazi do rješenja izvedivog na računalu. Kod učenika može doći do miskoncepcija u okvirima shvaćanja instrukcija koje se zadaju računalu. Iako se u poučavanju osnovnih programskih koncepta treba ići od konkretnog ka apstraktnom ili analognog ka digitalnom, učitelj treba posebnu pažnju posvetiti procesu poučavanja algoritama. Tijekom osnovnoškolskog obrazovanja, učenik u pravilu ne stekne odgovarajuće razumijevanje o načinu rada računala i izvršavanju programskog kôda. Uloga učitelja je da temeljito objasni razlike između zadavanja instrukcije čovjeku i zadavanja instrukcije računalu. Dakle, učenik je aktivni subjekt koji računalu govori što treba učiniti, a ne obrnuto.

Vizualizacija softvera (engl. *software visualization*) je pristup i tehnika grafičke prezentacije algoritama i programskog koda. Cilj ovog pristupa je olakšati razumijevanje „nevidljivih“ dijelova koda. Jedno od ovakvih okruženja (engl. *environment)* je već prethodno spomenut *Alice*. Međutim, Naps i sur. (2003) su u istraživanju utvrdili kako se ovakav pristup nije pokazao učinkovitim prema raznim očekivanjima. Iako pri početnom učenju programiranja, sintaktičke pogreške mogu preusmjeriti pažnju s programskih koncepata i formiranja ispravnih mentalnih modela, izostanak sintakse može predstavljati veliki problem pri neizbježnoj tranziciji u programski jezik sa sintaksom. Tehnike vizualizacije se koriste preko dvadeset godina i još se nisu pokazale dovoljno učinkovitima.

[9] Uvjet bez kojeg se ne može

U ovom diplomskom radu prikazan je drukčiji pristup u poučavanju računalnog programiranja i usvajanja programskog koncepta *while* petlje. U kombinaciji načela konstruktivizma, spomenutog PAR modela i postojećih metoda i strategija izrađeni su metodički naputci kako bi učenici mogli konkretno stvarati i primjenjivati algoritam na rješavanje zadanog problema, predviđati te analizirati i vrednovati rješenje pseudokoda testiranjem programskog koda u interpreterskom okruženju. Za poučavanje *while* petlje koristila se međupredmetna korelacija s nastavnim satom Hrvatskog jezika u okviru komunikacijskog i problemskog sustava. U priloženim metodičkim naputcima za obradu *while* petlje u višim razredima osnovne škole ili po učiteljevoj procjeni, nastavni se sadržaj *while* petlje produbljuje kroz tri nastavne jedinice u vidu dvostrukog nastavnog sata. Pri tome su, kao temeljni didaktički materijal za izvođenje nastavne jedinice, korištene datoteke. Prikazani metodički naputci čine samostalni rad autorice ovog diplomskog rada (vidi poglavlje 5), koja smatra kako konkretni modeli čine neizostavni dio nastavnog procesa tijekom učenja i poučavanja programskih koncepata i to osobito u razvijanju „računalnog razmišljanja“ kod učenika nižih razreda. Nakon konkretiziranja računalnog problema u predmetnom okruženju učenik lakše prelazi u digitalno okruženje programskog jezika Python, pri čemu rješava konkretne probleme upotrebom algoritama. U posljednjoj fazi, kada učenik stekne djelomično ili potpuno razumijevanje rada algoritma, može prijeći na programiranje uz pomoć grafičkog objekta u formi kornjače.

4. PYTHON

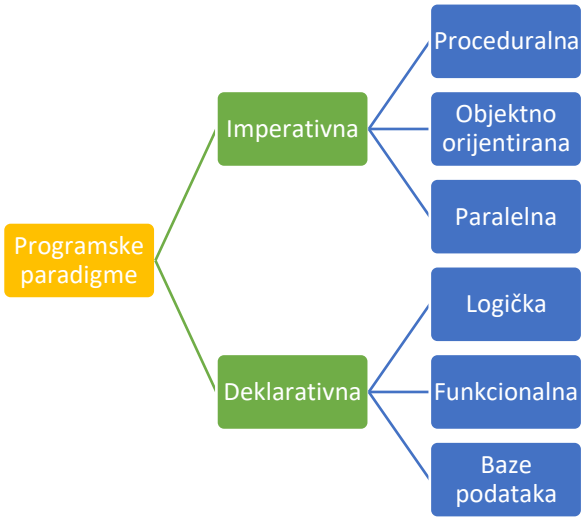
U drugom poglavlju ovog diplomskog rada dan je pregled nekih osnovnih programskih koncepta u Python i C programskom jeziku. Iz pregleda je moguće zaključiti kako je to samo uvod u apstraktne tipove podataka, strukture podataka, naprednije algoritme, klase i objekte i druge programske koncepte. U ovom poglavlju detaljnije je objašnjen programski jezik Python, koji je korišten kao temelj za razradu novih pristupa poučavanja programiranja u osnovnoj školi.

Godine 1989. programer Guido Van Rossum je kao hobi napravio Python programski jezik visoke razine, a ime mu je dao prema „*Monthy Python's Flying Circus*"[10]. U uvodnom dijelu knjige *Programming Python* iz 2013. godine Von Rossum je objasnio kako je jezik nastao. Njegova inicijalna ideja je bila stvoriti elegantan i snažan jezik. Kako on naglašava, jedan od elegantnih svojstava jezika je grupiranje blokova bez korištenja zagrada (*eng. indentation for statement grouping).* Takvim stilom programski kôd je vizualno pregledniji, a formatiranje koda postaje univerzalno svim programerima, što omogućava lakše čitanje koda. Zbog ovakve jednostavnosti, pažnja programera početnika je usmjerena više na algoritme nego na sintaktička i semantička pravila, kao što je čest slučaj u C programskom jeziku. Python programski jezik je jedan od najpopularnijih jezika u industriji, koriste ga Google, NASA, CERN i drugi. On je derivacija ABC programskog jezika dizajniranog za poučavanje. Iz ovih razloga je iznimno popularan za učenje prvih programskih koncepata (Tollervey, 2015, str 5). Istraživanje uspjeha početnika programera indicira kako izbor prvog programskog jezika uvelike utječe na uspjeh u savladavanju ostalih programskih jezika (Pillay, 2005).

[10] British television sketch comedy series (Encyclopedia Britannica)

4.1 Programske paradigme Python jezika

Programska paradigma je stil ili način programiranja. Ovaj termin uveo je Robert W. Floyd 1979.godine. Prema njemu, pri poučavanju računalnog programiranja naglasak je na sintaksi i pravilima, a zanemaren je proces dizajna programa. Jedan programski jezik odnosno program može imati više programskih paradigmi. Postoji ih mnogo, a najčešća je podjela na imperativne i deklarativne paradigme.



Slika 1. Podjela programskih paradigmi

Programski jezik C pripada imperativnoj tj. proceduralnoj paradigmi. Imperativna paradigma je najstarija paradigma i naglasak je na slijednom izvođenju niza naredbi. Proceduralna paradigma je tip imperativne paradigme koja se gradi korištenjem više procedura, koje se nazivaju funkcije ili potprogrami. Python pripada proceduralnoj, objektno-orijentiranoj i funkcionalnoj paradigmi, zbog čega je naglasak na rješavanju zadatka, a ne na jeziku. Programske paradigme, njihov razvoj i primjenu, proučava metodologija programiranja. Od učitelja se očekuje da prepozna važnost izbora programskog jezika Pyhton za učenje programskih koncepata, što podrazumijeva poznavanje programskim paradigmi obuhvaćenih tim jezikom.

4.1.1 Proceduralna programska paradigma

Proceduralno programiranje se sastoji od procedura i funkcija – dijelova koda koji se mogu ponavljati. Prema Whiteu (2005) procedure se još nazivaju i potprogrami. Ti potprogrami su zaduženi za izvršavanje određenog skupa instrukcija, a svaka se procedura može pozvati u bilo kojem trenutku izvršavanja koda pa čak i unutar samih sebe (rekurzija). Potprogrami se izvršavaju u obliku metoda i funkcija pravilnim redoslijedom. U svojoj knjizi, Balagurusamy (2013) objašnjava kako je kod proceduralne paradigme naglasak na izvršavanju algoritama, gdje se podatci mogu kretati iz jedne metode u drugu, a programski kôd se programira odozgo prema dolje. Kako se funkcije koriste u svim programima veoma je važno da učenici usvoje principe programskog rada u sklopu proceduralne paradigme. Ujedno, ova paradigma dio je programskog jezika Python i olakšava tranziciju na složenije programske jezike, poput C-a.

4.1.2 Objektno-orijentirana programska paradigma

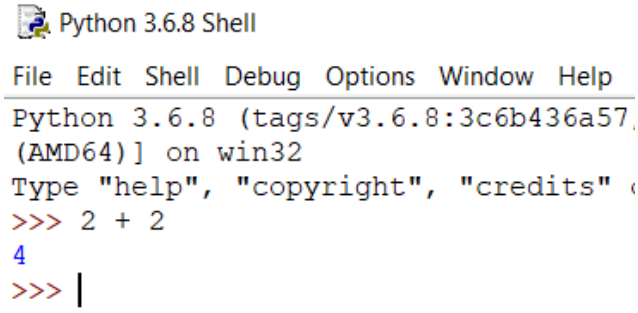
Prema Stefik i Bobrow (1985) svi jezici kojima je svojstvena ova paradigma sadrže *objekt* – entitet koji kombinira svojstvo podataka i postupaka, te izvodi izračunavanja (engl. *computations).* Entitet je stvarni ili apstraktni objekt koji posjeduje svojstva odnosno atribute (Kramberger, 2018, str. 2). Nadalje, Stefik i Bobrow (1985) objašnjavaju kako se sve radnje odvijaju pomoću slanja „poruka“ između objekata, a te poruke su indirektni proceduralni pozivi odnosno metode. U ovoj paradigmi nailazi se na podjelu između klasa i instanci. Klasa je opis jednog ili više objekata, a instance su konkretni primjeri objekata. Ova paradigma je posebna po svojim konceptima, a to su: enkapsulacija, kopiranje i prijenos, apstrakcija, nasljeđivanje i polimorfizam. Ujedno predstavlja dobre temelje za kasnije razumijevanje objektno - orijentiranih programskih jezika, poput Jave i C++. Razlike između drugih programskih jezika i Pythona su u tome što konstruktor (koji služi za izradu objekta) u drugim jezicima ima isto ime kao i klasa, a kod Pythona to ime glasi *innit*. Nadalje, Python ne sadrži preopterećene metode (engl. *method overloading)*.

4.1.3 Funkcionalna programska paradigma

Ova programska paradigma se sastoji od funkcija. Glavni program (engl. *main)* je sam po sebi funkcija koji prima argumente te izvodi program. Primjer takve funkcije (funkcionalne programske paradigme) u Pythonu je *lambda* funkcija, anonimna funkcija koja može imati samo jedan izraz. Jedna od razlika između funkcionalne i proceduralne paradigme je redoslijed izvođenja programskog koda, gdje je kod proceduralne paradigme taj redoslijed od iznimne važnosti. Nadalje, jednom deklarirane varijable više se ne mogu mijenjati. Funkcionalna paradigma se temelji na matematici i modularnosti (Hughes,1989).

4.2 Razvojno okruženje

Kao što je već rečeno u prvom poglavlju, interpreter je prevoditelj programskog koda. Prema Downeyu (2012) programi napisani u Python programskom jeziku se izvode s interpreterom. Zbog ovakvog načina prevođenja koda, Python je sporiji nego primjerice Java programski jezik, kod kojeg se kôd kompajlira. Postoje dva načina korištenja interpretera: interaktivni i skriptni način. U interaktivnom načinu program ispisuje rezultat odmah, a niz znakova ">>>" označava da je interpreter spreman izvršiti naredbu:

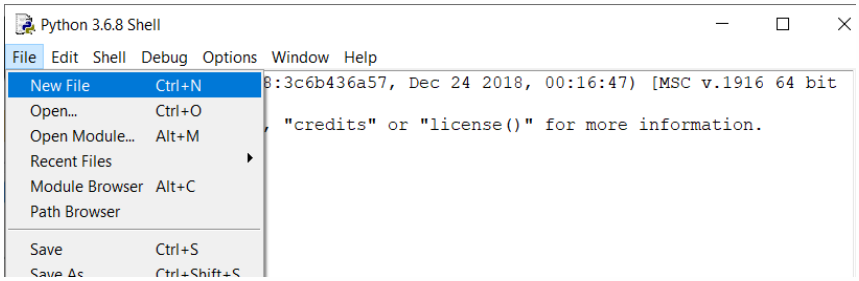


Slika 2. Prikaz izvršavanja naredbe u interaktivnom načinu rada

Drugi način je skriptni način rada. Napisani kôd se pohranjuje u datoteku s ekstenzijom .py koja se naziva skripta. Takve datoteke mogu se produbiti s modulima koje je kasnije moguće pokrenuti u razvojnoj okolini pomoću naredbe IMPORT. Iako je

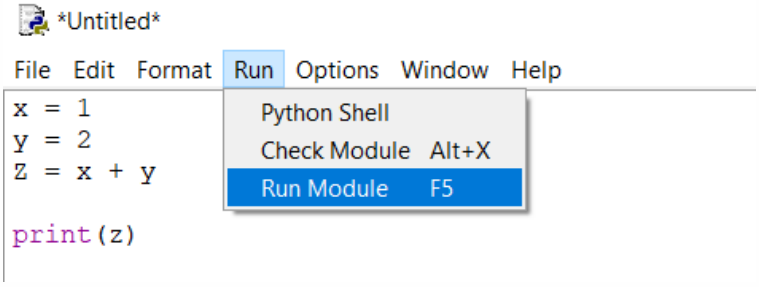
interaktivni način rada bolji za kraće kodove, pri izlazu iz programa sve naredbe se brišu.

Programski kodovi korišteni u ovom diplomskom radu napisani su u uređivaču teksta, u IDLE[11] okruženju. IDLE okruženje ima dvije vrste prozora, ljusku (engl. *shell)* i uređivač teksta (engl. *editor)*. Python okruženje odnosno ljuska može se pokrenuti unosom riječi IDLE u *Start menu.* Zatim se odabirom naredbe *New File* otvara uređivač teksta u kojem su napisani programski kodovi ovog diplomskog rada.



Slika 3. Stvaranje nove datoteke (skripte)

Kako bi se pokrenuo isti programski kôd odabiru se naredbe *Run > Run Module.*



Slika 4. Pokretanje modula iz skriptnog načina rada

Rezultati programskog koda ispisuju se unutar ljuske (engl. *shell).*

4.3 Datoteke

Osnovni programski koncepti Python jezika, potrebni za razumijevanje metodičkih naputaka u razradi nastavnih cjelina u sklopu ovog rada, objašnjeni su u prvom poglavlju. Složeniji koncepti jezika, osim datoteka (vidi poglavlje 5) koje su

[11] Integrated Development and Learning environment URL: https://docs.python.org/3/library/idle.html (25. kolovoza 2020)

potrebne za izvršavanje programskih kodova, izlaze izvan okvira ovog rada te nisu detaljno objašnjeni.

Datoteka je složeni tip podatka, ali jednostavan za korištenje. Prvo se inicijalizira putanja do tekstualne datoteke koja se želi otvoriti. Važno je da učenici znaju mjesto na koje su pohranili datoteku. Tekstualna datoteka ima ekstenziju *.txt.*

```
datoteka = "C:\\Users\\ines9\\pjesma.txt"
```

Slika 5. Definiranje putanje datoteke

Zatim se pomoću funkcije *open()* otvara datoteka. Za potrebe ovog diplomskog rada obradit će se samo metoda (engl. *mode*) "r", koja označava čitanje datoteke (engl. *read*), kako se ne bi izgubio fokus na učenju *while* petlje. Postoje još i *"a" – append, "w" – write* i *"x" create* metode[12].

```
with open (datoteka, 'r') as
mojapjesma:

redak = mojapjesma.readline()
```

Slika 6. Otvaranje datoteke

Funkcija *readline()* se poziva nad pjesmom i vraća vrijednosti jednog retka. Koristi se ova funkcija jer se pomoću nje i petlje čita red po red, a u konačnici i cijela pjesma, što je i cilj koji se želi postići. Ovakav način programiranja je sličniji radu s datotekama u C-u nego u Pythonu, ali je pogodan za objašnjavanje koncepta *while* petlje.

Python programski jezik ima regularne izraze (engl. *regular expressions*) koji se koriste za rad s datotekama i omogućavaju lakše korištenje istih. Može ih se koristiti tako da se u IDLE upiše naredba *import re*. Međutim, u metodičkim naputcima se nisu koristili jer se smatra kako bi se upotrebom tih izraza izgubio fokus s glavnog predmeta poučavanja odnosno poučavanja *while* petlje.

[12] URL: https://www.w3schools.com/python/python_file_handling.asp (5. srpnja 2020)

5. METODIČKI NAPUTCI

U ovom poglavlju prikazani su metodički naputci za obradu nastavnog sadržaja programske strukture *while* petlje, za više i/ili niže razrede osnovne škole (prema procjeni učitelja). Nastavni sadržaj razrađen je kroz tri nastavne jedinice u vidu dvostrukog nastavnog sata, pri čemu se koristila međupredmetna korelacija s nastavnim satom Hrvatskog jezika. Razrada metodičkih naputaka temelji se na novom, drukčijem pristupu poučavanja i učenja *while* petlje, jer se u nizu istraživanja pokazalo kako je ta struktura učenicima izuzetno zahtjevan i problematičan programski koncept (Kaczmarczyk i sur., 2010; Sorva, 2012). U skladu s tim, isključivo su prikazane nastavne jedinice i njima odgovarajuće etape nastavnoga sata, bez pozivanja na postojeće nastavne cjeline i teme iz kurikuluma za osnovnoškolsko obrazovanje. Kao temeljni didaktički materijal za izvođenje nastavne jedinice korištene su datoteke. Prikazani metodički naputci čine samostalni rad autorice ovog diplomskog rada.

1. PRVA NASTAVNA JEDINICA

Prva nastavna jedinica sastoji se od tri etape. Na slikama 7 i 8 prikazani su programski kôd ***while*** petlje i izlaz (izlazna vrijednost) programskog koda prve nastavne jedinice. U sljedećim poglavljima detaljno su razrađene etape.

```
#inicijaliziraj putanju datoteke
datoteka = "C:\\Users\\ines9\\pjesma.txt"

#otvori datoteku i pročitaj ju te ju zapamti pod imenom "mojaPjesma"
with open (datoteka, 'r') as mojaPjesma:
    #varijabla za brojanje redaka
    brojacRedaka = 0
    #pročitaj prvi redak
    redak = mojaPjesma.readline()
  #izvršavaj naredbe unutar while petlje dok postoji redak
    while redak:
        #ispiši red koji si pročitao
        print(redak.strip())
        #pročitaj novi redak
        redak = mojaPjesma.readline()
        #povećaj brojacRedaka za 1
        brojacRedaka = brojacRedaka + 1
    #kada while petlja završi s izvršavanjem, ispiši broj redaka
    print("Broj redaka u mojoj pjesmi je: ", brojacRedaka)
```

Slika 7. Programski kôd while petlje prve nastavne jedinice

```
Ne, ja se nisam oprostio s njom
kad nestade i na svoju stranu.
Sam slusah svojih nada lom
U jednom zabacenom restoranu.

Kako je bilo? Nije tesko reci!
U zamoru oglasila se trublja,
I vlak je krenuo obicno i lijeno,
Sa svime sto jos ljubljah.
Broj redaka u mojoj pjesmi je:  9
>>>
```

Slika 8. Izlaz programskog koda prve nastavne jedinice

1.1 Prva etapa

Prva etapa nastavne jedinice, za poučavanje *while* petlje u sklopu predmeta Informatika, čini emocionalno-intelektualna motivacija i temelji se na uvodnom dijelu nastavne jedinice iz predmeta Hrvatski jezik, uz poštivanje načela aktivnosti i samostalnosti te motivacije.

U ovom dijelu sata učenici stvaraju vlastitu pjesmu na temelju određenih smjernica poštivajući pravopisne norme. Učitelj može koristiti bilo koji drugi tekstualni oblik koji smatra odgovarajućim kako bi se postiglo razumijevanje *while* petlje odnosno korisnik za međupredmetnu korelaciju. Učenici mogu raditi individualno ili u paru. Primjereno tijeku nastavnog sata, učitelj najavljuje novi nastavni sadržaj *while* petlje na samom početku, kako bi učenici mogli razumjeti namjeru pisanja pjesme te slijediti daljnje upute. Motivacija ovisi o namjerama učitelja (što želi postići), ali poželjno je da se učenika uvede u srž nastavnog sata na samom početku.

Primjeri motivacijskih pitanja: *„Može li računalo „čitati“ poput čovjeka? Može, na primjer kamerom. Međutim, može li računalo pročitati pjesmu koju vi napišete na njemu?“*

Motivacijskim pitanjima analizira se i utvrđuje svrha programa i algoritma te učenika vodi u formiranju pravilnih mentalnih modela programskog koncepta *while* petlje. Pri tome je potrebno upozoriti učenike na dijakritičke znakove; veliko i malo slovo.

Kao što je rečeno u trećem poglavlju, programske instrukcije koje se zadaju računalu su detaljne i računalo ne može „samostalno“ zaključiti *„što je učenik želio reći“.* Pjesmu koju će učenici unositi u tekstualni uređivač treba pisati bez dijakritičkih znakova. Preporučuje se upotreba *Notepad* tekstualnog uređivača. Pri tome je poželjno naglasiti kako računalo može „pročitati“ i dijakritičke znakove i da to učenici, ako žele, mogu proučiti sami.

U ovom metodičkom naputku, kao ogledni primjer, koriste se prve dvije strofe pjesme ***Bez oproštaja***, autora **Dobriše Cesarića**. Zadaća ove etape ili dijela nastavnog sata je pobuditi pažnju učenika, i interes za suradnju i učenje. Učitelj može mijenjati

imena varijabli prikazanih u metodičkim naputcima. Komentari su označeni crvenom bojom i započinju znakom **#.**

Aktivnost 1:

1. *Učitelj učenicima zadaje pisanje pjesme na određenu temu po uputama koje sam izabere. Duljina pjesme se izražava u stihovima odnosno redovima. Dio vlastitog stvaralačkog rada učenici mogu raditi samostalno ili u paru.*
2. *Učenici zapisuju pjesmu u tekstualni uređivač. Naslov pjesme je naslov datoteke. Datoteka treba imati ekstenziju .txt*

1.2 Druga etapa

U drugoj etapi prve nastavne jedinice učitelj dijeli nastavne listiće sa pseudokodom. Učenik treba pretpostaviti cilj programa analizom zadatka odnosno analizom pseudokoda i zapisati vlastito mišljenje o tome „što će program ispisati nakon što se izvrši“. U ovom dijelu sata učenik stvara koncept programskog algoritma. Učitelj na kraju nastavne jedinice provjerava je li koncept ispravno (mentalni model) ili neispravno formiran (miskoncepcija). Učitelj treba nastojati utvrditi način na koji učenici razmišljaju kako bi prilagodio nastavni sadržaj njihovim kognitivnim sposobnostima.

Aktivnost 2:

1. *Nastavni listić: Dok postoji redak u tvojoj pjesmi čitaj pjesmu i zabilježi brojkom koliko redaka ima. Zapiši broj iza operatora pridruživanja:*
   brojacRedaka = *...*
2. *Usmeno komentiranje rezultata – smatraju li učenici da će rezultat biti drukčiji i što misle o čemu ovisi rezultat?*

1.3 Treća etapa

U trećoj etapi prve nastavne jedinice učitelj detaljno i zorno nastoji objasniti programski kôd. Učitelj ističe kako se *while* petlja izvršava sve dok je uvjet *(while redak-dok postoji redak*) istinit, odnosno dok se naredbom ili uvjetom ne uzrokuje prekid izvršavanja koda.

Na nastavnom listiću izostavljeni su samo najjednostavniji dijelovi programskog koda. Kôd treba biti ispisan na papiru i ako je moguće u boji. Pri objašnjavanju učitelj ne ulazi u detalje rada datoteka, već jezikom primjerenim učenicima pojašnjava rad računala prilikom korištenja datoteka. Na primjer: „*Računalo otvara datoteku kako bi ju moglo pročitati*“; „*Računalo mora prepoznati kada će prijeći u novi red“*.

Funkcija ***strip()*** se poziva nad retkom i vraća kopiju vrijednosti retka bez razmaka. Bez ove funkcije, iza svakog ispisanog reda, slijedio bi prazan red. U oglednom primjeru korištenom za izradu metodičkih naputaka pjesma (***Bez oproštaja***, autora **Dobriše Cesarića**) se sastoji od dvije strofe. Između strofa postoji jedan prazan redak, što će program isto zabilježiti kao (novi) redak i zbog toga će broj redaka biti devet, a ne osam kao što bi učenici mogli očekivati.

Aktivnost 3:

1. *Učitelj usmeno objašnjava uvjet i rad while petlje.*
2. *Učitelj objašnjava nove naredbe za rad s datotekama – otvaranje datoteke, čitanje redaka s funkcijom* ***readline(),*** *te ispis redaka pomoću* ***strip()*** *funkcije.*
3. *Podjela i rješavanje nastavnih listića (slika 9).*
4. *Učitelj analizira jesu li svi učenici ispravno riješili zadatak na papiru.*
5. *Učenici prepisuju kôd na računalo poštivajući pravila sintakse te pokreću program pomoću naredbe* ***Run****.*
6. *Učenici usmeno komentiraju izlazne vrijednosti programa, odnosno vrijednosti varijable* ***brojacRedaka****.*

Moguća pitanja za analizu: *Zašto je devet redaka, a osam je stihova (redaka) u pjesmi? Zna li računalo što je stih? Na koji način bi se mogla ispisati pjesma bez while petlje? Podudara li se vrijednost varijable s očekivanom vrijednošću iz prve aktivnosti?*

```
#1.zad: zapiši gdje se nalazi tvoja datoteka
datoteka = "_____"

with open (datoteka, 'r') as mojaPjesma:
    brojacRedaka = 0
    redak = mojaPjesma.readline()

    while redak:
        print(redak.strip())
        redak = mojaPjesma.readline()

        #2.zad: povećaj brojacRedaka za 1
        brojacRedaka = brojacRedaka _____

    print("Broj redaka u mojoj pjesmi je: ", brojacRedaka)
```

Slika 9**. **Nastavni listić prve nastavne jedinice s programskim kodom

Obrazovni ishodi prve nastavne jedinice: (1) Psihomotoričko područje: Učenik će stvoriti tekstualnu datoteku; (2) Kognitivno područje: Učenik će objasniti vrijednost varijable ***brojacRedaka*** i ***'r'* način** uporabe datoteke. Učenik će predvidjeti ponašanje algoritma i izlazne vrijednosti te provjeriti ispravnost algoritma izvođenjem programa; (3) Afektivno područje: Učenik će razvijati suradnički odnos prema radu, aktivno sudjelovanje u radu i slijediti upute pomoću kojih se lakše postiže cilj i rješava problem.

2. DRUGA NASTAVNA JEDINICA

U drugoj nastavnoj jedinici uvodi se naredba ***break*** unutar ***if*** naredbe. Naredbi *break* uvijek prethodi uvjet *if* po kojem se uvjetuje zbog čega će se prekinuti izvođenje petlje. U uvjetnoj *if* naredbi relacijski operator koji se koristi za provjeru je operator „je jednako" (= =). Ovaj se operator razlikuje od operatora pridruživanja „ =", koji u nastavi matematike znači jednakost. Uvođenjem *break* naredbe usvaja se jedna od karakteristika *while* petlje – petlja će se izvršavati sve dok je uvjet istinit odnosno dok se ne zada uvjet s kojim bi se prekinuo rad petlje. Pri tome se koristi modificirani programski kôd iz prethodne nastavne jedinice kako bi ostali dosljedni principu postupnosti, odnosno postepeno uvodili nove pojmove.

```
#inicijaliziraj putanju datoteke
datoteka = "C:\\Users\\ines9\\pjesma.txt"
#otvori datoteku i pročitaj ju te ju zapamti pod imenom "mojaPjesma"
with open (datoteka, 'r') as mojaPjesma:
   #varijabla za brojanje redaka
   brojacRedaka = 0
   #pročitaj prvi redak
   redak = mojaPjesma.readline()
 #izvršavaj naredbe unutar while petlje dok postoji redak
   while redak:
      #ispiši red koji si pročitao
      print(redak.strip())
      #pročitaj novi redak
      redak = mojaPjesma.readline()
      #povećaj brojacRedaka za 1
      brojacRedaka = brojacRedaka + 1
      #prekini petlju nakon što pročitaš treći red
      if brojacRedaka == 3:
         break
```

Slika 10. Programski kôd while petlje s if naredbom druge nastavne jedinice

```
Ne, ja se nisam oprostio s njom
kad nestade i na svoju stranu.
Sam slusah svojih nada lom
>>>
```

***Slika 11**. Izlaz programskog koda druge nastavne jedinice*

2.1 Prva etapa

U uvodnom dijelu sata učitelj ponavlja nastavni sadržaj koji se obrađivao na prethodnom satu/satima. Učitelj dijeli nastavne listiće sa pseudokodom i daje svoj komentar. Učenici rješavaju pseudokod koristeći vlastitu pjesmu kao vanjsko pomoćno kognitivno sredstvo.

Aktivnost 1:

1. *Nastavni listić: Dok postoji redak u tvojoj pjesmi čitaj pjesmu i bilježi brojkom koliko redaka ima. Ako (kad) pročitaš treći red, stani sa prepisivanjem pjesme.*
2. *Prokomentirati s učenicima koliko su redaka prepisali. Ima li njihova pjesma smisla? Sviđa li im se? Zašto treći redak , je li to mogao bilo koji drugi redak odnosno broj?*

2.2 Druga etapa

U drugoj nastavnoj etapi učitelj dijeli nastavne listiće. Nakon što ih učenici riješe, komentiraju se rješenja. Vodi se heuristički razgovor s temeljnim pitanjem: „*Imaju li učenici pretpostavku o tome što naredba* ***break*** *radi?*“

Učitelj objašnjava naredbu ***break***, posebno identificirajući njenu sintaksu odnosno mjesto ***break*** naredbe unutar programskog koda. Provjeru ispravnosti koda učitelj radi demonstracijom putem projektora ili usmenim izlaganjem/komentiranjem. Dodatnu provjeru ispravnosti koda učitelj može napraviti i u trećoj etapi nastavne jedinice evaluacijom koda pomoću računala.

Aktivnost 2:

1. *Podjela i rješavanje nastavnih listića (slika 12).*
2. *Učitelj provjerava jesu li svi učenici ispravno riješili nastavni listić.*

```
#1.zad: zapiši gdje se nalazi tvoja datoteka
datoteka = "_____"

with open (datoteka, 'r') as mojaPjesma:
    brojacRedaka = 0
    redak = mojaPjesma.readline()

    while redak:
        print(redak.strip())
        redak = mojaPjesma.readline()
        brojacRedaka = brojacRedaka + 1

#2. zadatak: dopuni if naredbu, ako je brojacRedaka jednak 3
    if _____
        break
```

Slika 12**. **Nastavni listić druge nastavne jedinice s programskim kodom

2.3 Treća etapa

U trećoj etapi učenik treba biti sposoban integrirati prethodno znanje o ***if*** naredbi i uskladiti ga s informacijama o *while* petlji koje je stekao u prethodnoj nastavnoj jedinici.

Postavlja se pitanje: *„Koje se naredbe odnosno funkcije ponavljaju pomoću while petlje?“* Odgovor: Funkcija ***print***, ***readline()*** i naredba ***inkrementiranja***.

Učitelj može promijeniti *if* naredbu promjenom vrijednosti s kojom se uspoređuje varijabla **brojacRedaka**, a može promijeniti i relacijski operator. Relacijski operatori su (>, <, >=, <=, = =, !=).

Aktivnost 3:

1. *Učenici uspoređuju izlazne vrijednosti programa s vrijednostima iz Aktivnosti 1.*
2. *Učenici rade izmjene unutar if naredbe – uvećavanjem ili smanjivanjem broja s kojim se uspoređuje brojčana vrijednost varijable brojacRedaka.*
   Pitanja: Zašto se pjesma nije ispisala do kraja? Koje naredbe while petlja ponavlja? Koliko je puta izvršena (ponovljena) while petlja?

Obrazovni ishodi druge nastavne jedinice: (1) Psihomotoričko područje i kognitivno područje: Učenik će zapisati lokaciju svoje datoteke. Učenik će napisati naredbu *if* prethodno zadanu pseudokodom. Učenik će predvidjeti ponašanje algoritma te provjeriti ispravnost algoritma izvođenjem odnosno rješavanjem programa. Učenik će dovršiti programski kôd kako bi program ispravno radio. Učenik će procijeniti, povezati i zaključiti u kojem će trenutku *break* petlja zaustaviti izvršavanje programa. Učenik će analizirati koje se naredbe ponavljaju unutar *while* petlje; (3) Afektivno područje: Učenik će razvijati suradnički odnos prema radu, aktivno sudjelovanje u radu i slijediti upute pomoću kojih se lakše postiže cilj i rješava problem.

3. TREĆA NASTAVNA JEDINICA

U trećoj nastavnoj jedinici sadržaj *while* petlje modificira se pretraživanjem pjesme. Pretraživati se može bilo koja riječ. Učitelj može u prvom nastavnom satu zadati učenicima da u svojoj pjesmi pokušaju što više puta upotrijebiti određenu riječ koju će kasnije pretraživati u ovoj nastavnoj jedinici. Učenici mogu pretraživati glagole, imenice i sl. Međutim, učenici moraju točno naznačiti koju riječ traže, pazeći na pravopis. Odabran je ovakav pristup, kako bi učenici znali da se programski kôd piše radi rasterećivanja i automatizacije rada (aktivnosti) čovjeka. Kao što je prije rečeno, instrukcije računalu moraju biti precizne. U ovoj nastavnoj jedinici uveden je i skraćeni način pisanja operatora zbrajanja. Učitelj ne mora pratiti ovakav način zapisa, ako smatra da će to kognitivno preopteretiti učenike.

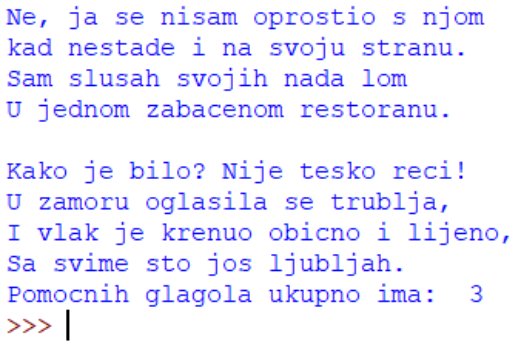
```
#inicijaliziraj putanju datoteke
datoteka = "C:\\Users\\ines9\\pjesma.txt"

#inicijaliziraj riječ/i koju želiš pretražiti u tekstu
#obavezan je razmak na kraju riječi kako se ne bi pretraživao string "je" unutar riječi poput "nije"
pomGlagolJe = "je "
pomGlagolSam = "Sam "

#otvori datoteku i pročitaj ju te ju zapamti pod imenom "mojaPjesma"
with open (datoteka, 'r') as mojaPjesma:
    #varijabla za brojanje redaka
    brojacRedaka = 0
    #varijabla za brojanje pomoćnih glagola
    brojPomGlagola = 0
    #pročitaj prvi redak
    redak = mojaPjesma.readline()
  #izvršavaj naredbe unutar while petlje dok postoji redak
    while redak:
        #ispiši red koji si pročitao
        print(redak.strip())
        #pročitaj novi redak
        redak = mojaPjesma.readline()
        #povećaj brojacRedaka za 1
        brojacRedaka = brojacRedaka + 1
        #ako je pomoćni glagol "je" u retku, zabilježi to
        if pomGlagolJe in redak:
         brojPomGlagola += 1
        if pomGlagolSam in redak:
         brojPomGlagola += 1

    print("Pomocnih glagola ukupno ima: ", brojPomGlagola)
```

Slika 13. Programski kôd while petlje s pretraživanjem datoteke treće nastavne jedinice



***Slika 14**. Izlaz programskog koda treće nastavne jedinice*

3.1 Prva etapa

U prvoj etapi učitelj dijeli nastavne listiće sa pseudokodom i daje svoje komentar. Učenici rješavaju pseudokod koristeći svoju pjesmu kao vanjsko pomoćno kognitivno sredstvo.

Aktivnost1:

1. *Nastavni listić: Dok postoji redak u tvojoj pjesmi čitaj pjesmu i pronađi pomoćni glagol „je“ i pomoćni glagol „sam“. Zabilježi koliko si pomoćnih glagola pronašao/la. Pridruži varijabli brojPomGlagola vrijednost.*
   brojPomGlagola = …
2. *Učitelj s učenicima komentira što misle kako će računalo pronaći njihove pomoćne glagole. Je li to uopće moguće? Na što moraju paziti?*

3.2 Druga etapa

Kao i u prethodnim nastavnim jedinicama učenik dopunjava nastavni listić. Važno je pripaziti da su učenici stavili razmak iza riječi koje traže kako se ne bi pretraživao *string* unutar *stringa*. *String* nije istoznačnica riječi. Nadalje, važno je pripaziti i na velika i mala slova, jer Python programski jezik razlikuje riječi „sam“ i „Sam“.

Aktivnost 2:

1. *Podjela i rješavanje nastavnih listića (slika 15).*
2. *Učitelj objašnjava uvjet „**if pomGlagolJe in redak:**“*
3. *Učenici pokreću program. Ispravljaju moguće pogreške i komentiraju dobivene rezultate te ih uspoređuju s rezultatima iz Aktivnosti 1.*

```
#1.zad: zapiši gdje se nalazi tvoja datoteka
datoteka = "_____"

#2. dodijeli vrijednost varijabli koju ćeš pretraživati
pomGlagolJe = "____"
pomGlagolSam = "____"
with open (datoteka, 'r') as mojaPjesma:
    brojacRedaka = 0
    brojPomGlagola = 0
    redak = mojaPjesma.readline()
#3. napiši uvjet while petlje
    while _____:
        print(redak.strip())
        redak = mojaPjesma.readline()
        brojacRedaka = brojacRedaka + 1
        if pomGlagolJe in redak:
            brojPomGlagola += 1
        #4.zad: povećaj brojacRedaka za 1
        if pomGlagolSam in redak:
            _________________
    print("Pomocnih glagola ukupno ima: ", brojPomGlagola)
```

Slika 15**. **Nastavni listić treće nastavne jedinice s programskim kodom

3.3 Treća etapa

Treća etapa je završna etapa učenja programskog koncepta *while* petlje. U završnoj etapi važno je sistematizirati i evaluirati dosad naučeno. Iskažu li učenici nedoumice, učitelj ih treba objasniti. Ako učitelj smatra da su učenici usvojili programski koncept *while* petlje, na način da ga mogu primjenjivati i u drugim programskim jezicima te da nisu stvorili *miskoncepcije,* po osobnom izboru može produbiti učenje *while* petlje. Pritom naglasak ostaje na konkretnoj upotrebi *while* petlje. U nastavku je dan prijedlog dodatnih aktivnosti u kojima učenici samostalno mijenjaju pokazne primjere:

- *Učenik po osobnom izboru pretražuje riječ.*
- *Učenik koristi* ***break*** *naredbu kada naiđe na riječ koju pretražuje.*
- *Učenik radi na pjesmi drugog učenika.*
- *Učenik mijenja uvjet petlje.*
- *Učenici u paru zadaju jedan drugome* ***while*** *petlju te evaluiraju rješenje.*
- *Učitelj objašnjava naredbu* ***continue****.*

Dodatne aktivnosti se mogu izvoditi tek kada je učenik u potpunosti usvojio koncept programske strukture *while* petlje, što obuhvaća: (1) Razlikovanje logičkog uvjeta (je li istina ili nije) ili relacijskih uvjeta (veće od, manje od, veće ili jednako od, manje ili jednako od, je jednako, nije jednako); (2) Identificiranje mjesta uvjeta u programskom kodu; (3) Opisivanje problema i prepoznavanje koraka u rješavanju problema; (4) Pisanje uvjeta *while* petlje; (5) Sposobnost (usmenog) opisivanja (govoreni jezik) dijelova koda koji se ponavlja unutar *while* petlje i dijelova koda koji se ne ponavlja; (6) Sposobnost (usmenog) opisivanja (govoreni jezik) načina rada *break* naredbe i prepoznavanje trenutka zaustavljanja rada programa prilikom korištenja te naredbe; (7) Sposobnost kreiranja i opisivanja primjera *while* petlje iz svakodnevnog života ili predmetnog okruženja.

6. ZAKLJUČAK

U odgojno-obrazovnom procesu u Hrvatskoj, za poučavanje i učenje računalnog programiranja, koriste se LOGO, Scratch i Python programski jezici. Dugogodišnja istraživanja su pokazala kako svladavanje nastavnog sadržaja iz područja računalnog programiranja kod početnika programera u većoj mjeri ovisi o metodama učenja i poučavanja (Bergin i Reilly, 2006, prema Caspersen, 2007). To ukazuje kako postojeća tradicionalna nastava, osobito na području osnovnoškolskog računalnog programiranja, zahtjeva temeljitu promjenu u pristupu poučavanja odnosno zahtjeva uvođenje suvremenih nastavnih metoda, poput aktivnog učenja, učenja putem rješavanja problema, primjena igara, kako bi se učenike motiviralo za učenje i aktivno sudjelovanje u nastavnom procesu. U procesu učenja programiranja, računalno razmišljanje je jedna od ključnih mentalnih vještina dizajniranja programa, jer potiče preciznost i sustavnost u oblikovanju programskog koda, te pozitivno utječe na pravilno rješavanje i osmišljavanje problema.

Python programski jezik pokazao se najpogodnijim zbog svoje jednostavne sintakse te funkcionalne, proceduralne i objektno-orijentirane programske paradigme. Pa tako učenje Python programskog jezika omogućava lakšu tranziciju na druge programske jezike, poput objektno-orijentiranog Java ili proceduralnog C programskog jezika.

U ovom diplomskom radu prikazan je novi, drukčiji pristup poučavanja i učenja programske strukture *while* petlje, upotrebom Python programskog jezika, kroz stvaralački pismeni rad učenika. Odabrana je *while* petlja, jer se u nizu istraživanja pokazalo kako je ta programska struktura učenicima izuzetno zahtjevna i problematična (Kaczmarczyk i sur., 2010; Sorva, 2012). Prikazani metodički naputci prilagođeni su za više i/ili niže razrede osnovne škole, pri čemu se koristila međupredmetna korelacija s nastavnim satom Hrvatskog jezika. Razrada nastavnoga sadržaja obuhvaća tri nastavne jedinice, sa sveukupno devet nastavnih etapa i odgovarajućih programskih kodova. Naglasak prikazanog pristupa poučavanja i učenja nije samo na usvajanju programskog koncepta, već i na razlikovanju i definiranju svrhe upotrebe programskog jezika,

strukture *while* petlje i njene primjene u predmetnom okruženju. Prikazani metodički naputci čine samostalni rad autorice ovog diplomskog rada.

LITERATURA

1. Astrachan, O., Hambrusch, S., Peckham, J., & Settle, A. (2009). The present and future of computational thinking. ACM SIGCSE Bulletin, 41(1), 549–550.
2. Basariček, S. (1882). Pedagogija, II. dio: Obće obukoslovje. Zagreb: Hrvatsko pedagoški - književni zbor.
3. Beazley, D., K. Jones, B. (2013). Python Cookbook, Third Edition, O'Rilley Media, Inc.
4. Budin L., Brođanac P., Markučić Z., Perić S., Škvorc D., Babić M., (2017). Računalno razmišljanje i programiranje u Pythonu, Zagreb: ELEMENT.
5. Caspersen, M. E. (2007). Educating Novices in The Skills of Programming. Doctoral Thesis. Department of Computer Science, University of Aarhus, Denmark. Dostupno na: http://www.cs.au.dk/~mec/dissertation/Dissertation.pdf [2.9.2020.]
6. Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational Thinking - a guide for teachers.
7. De Zan, I. (1999). Metodika nastave Prirode i društva, Zagreb: Školska knjiga.
8. Denning, P., Tedre, M. (2019). Computational Thinking, Cambridge: MIT Press.
9. Downey A. (2014.) Naučite Python, Zagreb: Dobar plan.
10. Downey, A. (2012). Think Python, How to Think Like a Computer Scientist, Green Tea Press.
11. E. Balagurusamy, (2013). Object Oriented Programming with C++, McGraw-Hill Education - Europe.
12. E. Mayer, R., (1981). The Psychology of how Novices Learn Computer Programming, ACM.
13. Faber, H., Wierdsma, M., Doornbos, R. P., van der Ven, J. S., & de Vette, K. (2017). Teaching computational thinking to primary school students via unplugged programming lessons. Journal of the European Teacher Education Network, 12, 13-24.
14. Floyd W. R. (2005). The Paradigms of Programming, Standford University

15. Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. Cognitive Psychology, 12, 306-355.
16. Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. Cognitive Psychology, 15, 1-38.
17. Hughes,J. (1989). Why Functional Programming Matters, The Computer Journal, Volume 32, Issue 2, Pages 98–107, Dostupno na: https://doi.org/10.1093/comjnl/32.2.98[25.8.2020]
18. Kaczmarczyk, L. C., Petrick, E. R., East, J. P., Herman, G. L. (2010). Identifying Student Misconceptions of Programming. Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE '10, 107–111.
19. Kernighan, B., Ritchie, D. (1978). The C Programming Language. Second Edition, Prentice Hall.
20. Kong, S.C., Andone, D., Biswas, G., Crick, T., Hoppe, H.U., Hsu, T.C., Huang, R.H., Li, K.Y., Looi, C.K., Milrad, M., Sheldon, J., Shih, J.L., Sin, K.F., Tissenbaum, M., & Vahrenhold, J. (Eds.). (2018). Proceedings of the International Conference on Computational Thinking Education 2018. Hong Kong: The Education University of Hong Kong.
21. Kramberger, T., Duk, S., Kovačević R., (2018). Baze podataka, Tehničko veleučilište u Zagrebu.
22. Latour, B. (1986). 'The Powers of Association'. Power, Action and Belief. A new sociology of knowledge? Sociological Review monograph 32. Law, J. (Ed). Routledge & KeganPaul, London: 264-280.
23. Linn, M., C., Dalbey, J. (1989). Cognitive Consequences of Programming Instruction, in Studying the Novice Programmer, Soloway, E., & Spohrer, J. C. (ur.), Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers, 57-81.
24. Loyola Marymount University, Los Angeles, »Programming Paradigms,« Loyola Marymount University, Dostupno na: http://cs.lmu.edu/~ray/notes/paradigms/ [20.8.2020]
25. Lutz, M. (1996). Programming Python, O'Rilley Media, Inc.

26. Ma, Linxiao, (2007). Investigating and Improving Novice Programmers' Mental Models of Programming Concepts, University of Strathclyde, Department of Computer & Information Science.
27. Mayer, R., Dyck. J., (1989). Teaching for Transfer of Computer Program Comprehension Skill, Journal of Educational Pyschology, Vol. 81, No. I, 16 – 24.
28. Meyer, H. (2002). Didaktika razredne kvake. Zagreb: Educa.
29. Mischel, W. (1993). Behavioral conceptions. In W. Mischel, Introduction to personality, New York: Harcourt Brace, 295 – 316.
30. Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., i dr. (2003, December). Evaluating the Educational Impact of Visualization. ITiCSE-WGR '03: Working group reports from ITiCSE on Innovation and technology in computer science education ACM, 124 – 136.
31. Pastuović, N. (1999). Edukologija: integrativna znanost o sustavu cjeloživotnog obrazovanja i odgoja. Zagreb: Znamen.
32. Pavleković M. (1997). Metodika nastave matematike s informatikom, Zagreb: ELEEMNT.
33. Piaget, J. (1977). The Essential Piaget. Gruber, HE; Voneche, JJ. eds. New York: Basic Books.
34. Pillay, N., Jugoo, Vikash R., (2005). An Investigation into Student Characteristics Affecting Novice Programming Performance, inroads – The SIGCSE Bulletin, Volume 37, Number 4, pp. 107-110.
35. Roy D. Pea, D. Midian Kurland. (1984). On the cognitive effects of learning computer programming. New Ideas in Psychology, Elsevier, 2(2), 137-168.
36. S. M. Maurizio Gabbrielli, (2010). Programming Languages: Principles and Paradigms, London: Springer.
37. Selby, C. C. (2014). How Can the Teaching of Programming Be Used to Enhance Computational Thinking Skills, University of Southampton, Southampton Educational School, Doctoral Thesis.
38. Selby, C. C. (2015). Relationships: computational thinking, pedagogy of programming, and bloom's taxonomy. In Proceedings of the Workshop in

Primary and Secondary Computing Education on ZZZ (pp. 80–87). New York: ACM.

39. Selby, C. C., & Woollard, J. (2014). Refining an Understanding of Computational Thinking.
40. Sorva, J. (2012). Visual Program Simulation in Introductory Programming Education. Doctoral Disertation 61. Aalto University, School of Science, Department of Computer Science and Engineering.
41. Stefik, M., & Bobrow, D. G. (1985). Object-Oriented Programming: Themes and Variations. AI Magazine, 6(4), 40. Dostupno na: https://doi.org/10.1609/aimag.v6i4.508[20.8.2020.]
42. Sweeney, A. i Posavec, M. (2017). Utjecaj primjene PAR modela poučavanja u konstruktivističkom pristupu na usvajanje pojmova i koncepata. *Napredak, 158.* (4.), 503-525. Dostupno na: https://hrcak.srce.hr/188297[1.9.2020.]
43. Tollervey, N., (2015). Python in Education, O'Rilley Media, Inc.
44. Turing, A. (1950). *Mind*, Volume LIX, Issue 236, October 1950, 433–460. Dostupno na: https://phil415.pbworks.com/f/TuringComputing.pdf [25.8.2020]
45. White, G., Sivitanides, (2005). M. Cognitive Differences Between Procedural Programming and ObjectOriented Programming. Inf Technol Manage 6, 333–350. Dostupno na: https://doi.org/10.1007/s10799-005-3899-2 [20.8.2020]
46. Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical transactions. Series A, Mathematical, Physical, and Engineering Sciences, 366(1881), 3717–3725. Dostupno na: https://doi.org/10.1098/rsta.2008.0118 [2.7.2020]
47. Wing, J.M. (2017). Computational thinking's influence on research and education for all. Italian Journal of Educational Technology, 25(2), 7-14. doi: 10.17471/2499-4324/922.

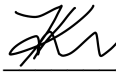
POPIS SLIKA

Izjava o samostalnoj izradi rada

Diplomski rad pod naslovom *Novi pristupi računalnom programiranju u primarnom obrazovanju* izrađen je samostalno na temelju korištenja navedene literature uz mentorstvo izv. prof. dr. sc. Maria Dumančića i sumentorstva dr. sc. Nataše Rogulje koji se temelji na kritičkom pregledu literature i dosadašnjih istraživanja. Zahvaljujem se roditeljima, mentoru i sumentoru na razumijevanju i usmjeravanju pri studiranju i izradi diplomskog rada.

Vlastoručnim potpisom potvrđujem izjavu o samostalnoj izradi rada.

Studentica:



(potpis)