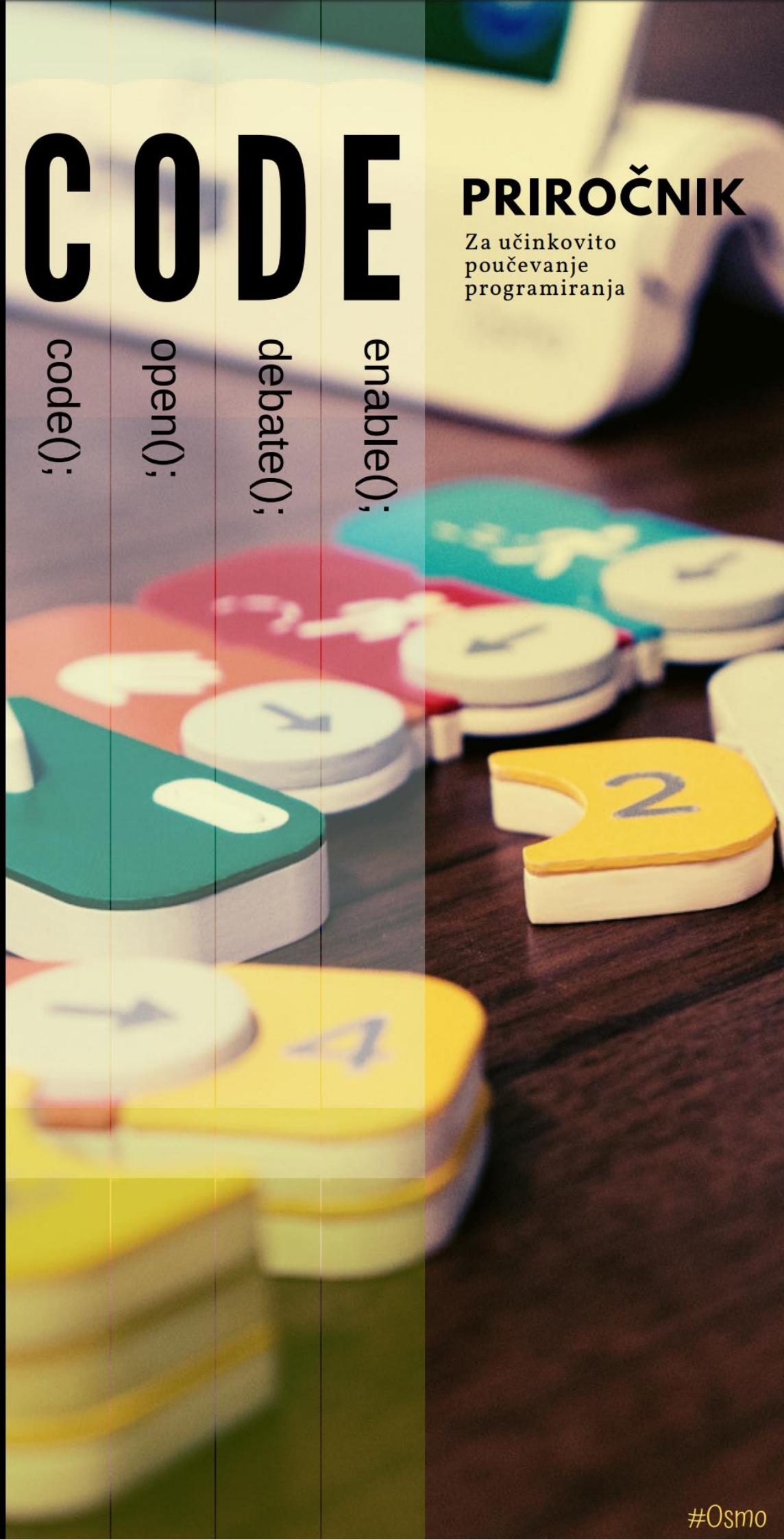
EDUCODE

PRIROČNIK

Za učinkovito poučevanje programiranja

enable();
debate();
open();
code();



Avrupa Komisyonu

TÜRKİYE CUMHURİYETİ AVRUPA BİRLİĞİ BAKANLIĞI

REPUBLIC OF TURKEY MINISTRY FOR EU AFFAIRS

TÜRKİYE ULUSAL AJANSI
TURKISH NATIONAL AGENCY

Erasmus+

#Osmo

required for publisher, authors and bibliographical fields, copyright, ...

CIP – v pridobivanju

KAZALO

Uvod

Uporaba informacijske in komunikacijske tehnologije (IKT), ki poteka na skoraj vseh področjih življenja, s sabo prinaša stalne in pomembne spremembe v razvoju in tehnologija vseskozi oblikuje naša življenja. Upoštevajoč to dejstvo, se zdi skoraj neizbežno, da se bomo tudi v prihodnosti kot družba še naprej radikalno spreminjali. Pojavili se bodo novi poklici, ki danes sploh še ne obstajajo. Zato je pomembno, da študente pripravimo na te spremembe tako, da jim pomagamo pri integraciji v IKT, saj bodo na ta način bolje pripravljeni na nove poklice prihodnosti. EU in Republika Turčija imata enako vizijo glede te tematike in postavljata učenje programiranja za prioriteto v izvedbi te skupne vizije. Na mednarodni ravni se učenje programiranja vedno bolj uveljavlja, veliko evropskih držav in Republika Turčija izjavljajo, da bo tovrstno izobraževanje obvezno na primarni in sekundarni ravni izobraževalnega sistema.

V luči tega je treba sedanje in bodoče učitelje na področju IKT/STEM (znanost, tehnologija, inženirstvo in matematika) izobraziti tako, da bodo pridobili znanje in izkušnje o tem, kako poučevati kodiranje. Prav tako je nujno, da povečamo in razširimo učne vire in vire poučevanja na tem področju. Upoštevajoč to dejstvo, se ta priročnik osredotoča na študente, ki bodo v prihodnje poučevali predmete, povezane z IKT/STEM, in jih opremlja z veščinami in znanjem, kako uporabiti kodiranje na začetku svoje poklicne kariere v šoli. Ta priročnik, ki smo ga razvili v skladu z namenom projekta »Povečanje kompetenc dodiplomskih študentov računalništva na področju učenja programiranja (EDUCODE)« s podporo strateškega partnerstva v višješolskem izobraževanju Erasmus+, je namenjen institucijam in profesionalcem, ki delajo na področju IKT.

Priročnik smo razvili postopoma. Najprej je projektna ekipa, sestavljena iz poklicnih akademikov iz štirih različnih držav, izvedla raziskave o potrebah in zahtevah učenja programiranja. Pomembno je namreč jasno artikuliranje tega, katere informacije in veščine so potrebne za dodiplomske študente, ki bodo poučevali IKT/STEM, da bodo lahko poučevali programiranje. Za ta namen smo izvedli raziskavo med 258 dodiplomskimi študenti poučevanja IKT z vseh štirih partnerskih univerz (Univerza Dokuz Eylül, Fakulteta za organizacijo in matematiko Univerze v Zagrebu, Univerza v Mariboru in Tehnološki inštitut Limerick), s katero smo pridobili ocene o njihovem obstoječem znanju in kompetencah na področju poučevanja kodiranja. Nato smo na podlagi rezultatov te analize razvili kurikulum. Kurikulum, ki smo ga razvili v tem projektu, odraža znanja, veščine in kompetence, ki bodo četi bodočih učiteljev IKT/STEM omogočili, da podajajo znanje o programiranju. Na koncu smo pripravili še priročnik za učitelje, ki podpira in dopolnjuje kurikulum. Priročnik so pregledali študentje, predavatelji in ljudje s sorodnih poklicnih področij in na podlagi njihovih povratnih informacij smo ga prilagodili in popravili do njegove končne vsebine.

Ta kurikulum in priročnik skušata opremiti študente poučevanja IKT/STEM z veščinami in s kompetencami, povezanimi z motiviranjem učencev za učenje programiranja, s tem, kako jim prikazati osnove računalništva, kako narediti predmet programiranja zanimiv in prijeten za učence. Poleg tega kurikulum in priročnik pokrivata teme, kot so, kako naučiti učence, da vzpostavijo smiselno sodelovanje z drugimi, kako razvijejo računalniško mišljenje in veščine reševanja problemov ter znajo poiskati sorodne vire. Knjiga vključuje tudi primere aktivnosti in delovnih listov. S tem priročnikom se bodo učitelji, akademiki in posamezniki-strokovnjaki s tega področja, naučili, kako lažje poučevati programiranjein ga uporabiti v razredu. Prav tako

bodo imeli priložnost, da s pomočjo pridobljenega znanja in izkušenj razvijejo nove načine in aktivnosti poučevanja. Priročnik in viri so prosto dostopni javni viri.

Posebna zahvala gre projektnim ekipam Univerze Dokuz Eylül, Univerze v Mariboru, Fakultete za organizacijo in matematiko Univerze v Zagrebu, Tehnološkemu inštitutu Limerick in APEC, ki so z vso skrbnostjo in veliko vnemo ustvarili ta priročnik.

# AVTORJI
develop();

PRIIMEK	IME	DRŽAVA IN UNIVERZA
Akpınar	Ercan	Turčija Dokuz Eylül University
Arslan	Kürşat	Turčija Dokuz Eylül University
Baran	Bahar	TurčijaDokuz Eylül University
Hajdin	Goran	Hrvaška University of Zagreb
Hoyne	Seamus	Irska Limerick Institute of Technology
Kermek	Dragutin	Hrvaška University of Zagreb
Krašna	Marjan	Slovenija University of Maribor
O'Brien	Elisabeth	Irska Limerick Institute of Technology
Oreški	Predrag	Hrvaška University of Zagreb
Özkaya	Yıldırım	Turčija APEC
Pesek	Igor	SlovenijaUniversity of Maribor
Radošević	Danijel	Hrvatska University of Zagreb
Vesel	Aleksander	SlovenijaUniversity of Maribor

Hvala vsem posebnim osebam, brez katerih ne bi bilo mogoče zaključiti tega projekta. Cenimo njihovo podporo in razumevanje. Oni so naši čudoviti prijatelji

3. projektni sestanek - Slovenija

O priročniku

V priročniku so zbrani postopki učnih ur ter podrobni opisi aktivnosti, ki jih lahko učitelji in predavatelji uporabijo za bolj učinkovito učenje programiranja.

Cilj priročnika je izboljšati kvaliteto poučevanja učiteljev in povečati možnosti učenja za tiste, ki se izobražujejo iz programiranja .

Priročnik je bil zasnovan tako, da je dostopen in uporaben tako za učitelje kot tudi za izvajalce usposabljanj za učitelje. Razdeljen je na pet enot, vsaka vsebuje posamezne naloge, ki pokrivajo področja učenja programiranja, pomembna in nujna za učinkovito učenje na tem področju.

Vsaka učna ura je zasnovana po podobnem formatu, tj.:

- naslov učne ure in predvidena časovna razporeditev
- učni izidi
- strategije poučevanja
- struktura učne ure
- metode ovrednotenja

Predvideno je, da se 5 enot izvede v približno 30 urah poučevanja (čeprav lahko institucije zastavijo tudi drugačno razporeditev). Načrtovana časovna razporeditev je opredeljena odstotkovno v odnosu do celotnega programa. Na primer, 1. enota zajema 20 % celotnega časovnega obsega (pri čemer 1. lekcija predstavlja 10 %), torej bi se naj 10 % celotnega programa namenilo tej lekciji. Pri 30-urnem programu to pomeni, da se ta lekcija izvede v 3 urah (predavanj/praktičnih vaj/delavnic). Cela enota (1. enota) se torej pokrije z 20 % časa, namenjenega temu program oz. v 6 urah od skupno 30. Ta odstotkovna razdelitev je navedena zgolj informativno, vsaka institucija oz. predavatelj si lahko časovno razporeditev priredi glede na svoje specifične potrebe.

Učni izidi izhajajo neposredno iz kurikuluma za učenje programiranja, ki je dostopen na spletni strani projekta in predavatelju omogočajo, da razume namene in cilje posamezne lekcije.

Strategije poučevanja in metode ovrednotenja kažejo možnosti izvedbe učne ure.

Struktura učne ure predstavi aktivnosti, ki sestavljajo učno uro.

Aktivnosti so sestavni del vsake učne ure. Vsaka aktivnost vsebuje naslov in kratek uvod. Sledi teorija, na kateri temelji izvedba aktivnosti. Nekatere aktivnosti imajo tudi seznam zahtevanih predznanj, ki jih predavatelj in študenti potrebujejo pred izvedbo aktivnosti. Nekatere aktivnosti, ki so naravnane na uporabno poučevanje programiranja, pa nimajo teoretičnega dela.

Vsaka aktivnost vsebuje tudi namige, kako jo uporabiti, in vire, ki pojasnjujejo in širijo znanje študentom ter pomagajo k učinkovitejši uporabi aktivnosti.

Nekatere učne ure navajajo tudi primere v obliki delovnih listov ali nalog, ki pomagajo pri njeni izvedbi.

Oznaka enote: UČNA ENOTA (U1-L1)

NASLOV: Pomen učenja programiranja in učitelji IKT /STEM (10 % od 20 %)

Učni izidi

- Razpravljati o osnovnih konceptih, na katerih temelji učenje kodiranja in programiranja.
- Pojasniti pomen računalniškega razmišljanja in reševanja problemov.
- Opisati odločilne vloge učiteljev IKT in STEM za uspešno učenje programiranja.

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
✕ Skupinsko delo	☐ Vprašanja in odgovori	☐ Projektno učenje
☐ Skupinsko učenje	☐ Kombinirano učenje	☐ Demonstracija
☐ Igranje vlog		

Struktura poglavja

- AKTIVNOST 1: Razlike med kodiranjem in programiranjem
- AKTIVNOST 2: Računalniško razmišljanje
- AKTIVNOST 3: Reševanje problemov
- AKTIVNOST 4: Ključna vloga učiteljev IKT

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	☐ Medsebojno ocenjevanje	✕ Ustno preverjanje znanja
☐ Domača naloga	☐ Vzorčni izdelek	☐ Projektna aktivnost
☐ Predstavitev		☐ Drugo

AKTIVNOST 1: Razlike med kodiranjem in programiranjem

Aktivnost je osredotočena na pojasnjevanje, ali kodiranje pomeni enako kot programiranje.

TEORIJA

Obstaja mnogo mnenj o tem, ali je kodiranje enako programiranju. Zato Tabela 1 primerja in pojasnjuje razlike med kodiranjem in programiranjem z različnih vidikov.

Tabela 1: Primerjava med kodiranjem in programiranjem[1]

PRIMERJAVA NA OSNOVI:	Kodiranje	Programiranje
definicije	Kodiranje je pravzaprav proces prepisovanja šifer iz enega jezika v drugega.	Programiranje je proces ustvarjanja in razvijanja izvršljivega strojnega programa, ki izvaja niz ukazov.
predlog	Osnovni cilj kodiranja je poenostavitev komunikacije med človekom in stroji.	Programiranje je proces uradnega zapisovanja kod tako, da človekov vložek in učinek stroja ostajata sinhronizirana.
veščin	Kodiranje je začetni korak uvajanja programiranja, tako da lahko imajo koderji manj strokovnega znanja in izkušenj kot programerji.	Programiranje je osnova komuniciranja med človekovimi mislimi in učinki na ravni strojev in to komuniciranje je običajno sestavljeno iz kompleksnih struktur. Programerji imajo veliko več strokovnega znanja in izkušenj kot koderji.
preprostosti	Kodiranje je začetni korak h kompleksnim programerskim poizvedovanjem in je bolj preprosto kot programiranje.	Programiranje se ukvarja z različnimi kompleksnimi situacijami in vprašanji z namenom, da bo stroj proizvedel ustrezen učinek. Gre pravzaprav za napredno različico kodiranja in drugih različnih pristopov in je torej veliko bolj kompleksno kot kodiranje.
pristopa	Ker gre za začetno stopnjo komuniciranja, se koderji običajno ukvarjajo zgolj z določenimi programskimi kodami in se ne poglabljajo v podrobnosti.	Programerji se običajno ukvarjajo s komunikacijskim pristopom na veliko bolj zahtevni ravni. Analizirajo in konceptualizirajo različne vidike komunikacije in jo postavijo tako, da stroj proizvede ustrezen učinek.
podpore	Koderji imajo veliko podporo skupnosti, ki jim pomaga pri uporabi različnih pristopov kodiranja v skladu z veljavnimi industrijskimi standardi.	Programiranje je pravzaprav širše področje kodiranja. Tudi programiranje ima veliko podporo in spodbudo skupnosti za konstanten napredek v skladu z veljavnimi standardi.
izpopolnjenih značilnosti	Kodiranje je večinoma del programskega pristopa, ki vključuje prevajalske zahteve, zapisovanje kodnih vrstic in njihovo uporabo za strojno berljive vnose.	Programiranje zajema veliko širši spekter, ki vključuje vse ključne parametre od iskanja in odpravljanja napak in pretvorbe do testiranja in uporabe. Ukvarja se s temeljno funkcionalno uporabnostjo človeškega vnosa za ustrezen učinek na ravni strojev.

Vedno se postavlja vprašanje, ali je lahko vsak uspešen programer tudi uspešen učitelj IKT/STEM. To vprašanje nima jasnega odgovora, ker je to odvisno tudi od osebnosti učitelja, njegove motiviranosti itd. V vsakem primeru pa je dobrodošlo, da ima programer določena pedagoška znanja o programiranju. To se bo jasno pokazalo v naslednjih lekcijah.

Obstoječe raziskave ne opredeljujejo starosti, pri kateri bi se bilo najbolje začeti učiti programiranje. Velja splošno sprejeto mnenje, da je smiselno začeti s poučevanjem programiranja nekje med 8. in 10. letom starosti, ko otrok začne oblikovati abstraktno mišljenje. Seveda obstajajo predpogoji za to, npr. otrok mora znati brati.

Postavlja se tudi vprašanje, ali lahko kdor koli postane programer. Odgovor je da, vendar je to odvisno od posameznika in njegovih učiteljev, motivacije, programskega jezika itd. Ne more pa vsak postati dober ali odličen programer, saj je za to potrebno veliko vaje in predanosti. Isto velja za kodiranje.

[1] https://www.educba.com/coding-vs-programming/

IZVEDBA

- Skupinsko delo
- Razprava za in proti
- Projektno učenje (kot primer)

ZA RAZMISLEK

- Ali je kodiranje in programiranje eno in isto?
- Če si programer, ali si potem tudi koder?
- Ali je lahko vsak uspešen programer tudi uspešen učitelj IKT/STEM?
- Pri katerem letu starosti naj bi se učenje programiranja začelo?
- Ali lahko programiranja učimo že otroke v osnovni šoli? Kaj pa starejše?
- Ali obstajajo predpogoji za učenje programiranja?
- Ali je lahko programer kdor koli?

VIRI

ars technica. (15. 9 2012). *Is it true that "not everyone can be a programmer"?* Pridobljeno iz ars technica: https://arstechnica.com/information-technology/2012/09/is-it-true-that-not-everyone-can-be-a-programmer/

Mechado, L. (9. 12 2017). *What is the best age to start learning programming?* Pridobljeno iz Quora: https://www.quora.com/What-is-the-best-age-to-start-learning-programming-1

Ohanesian, S. (31. 7 2018). *What is the difference between coding & programming?* Pridobljeno iz Julian Krinsky Camps & Programs: https://info.jkcp.com/blog/coding-vs-programming

Prottsman, K. (4. 12 2015). *Coding vs. Programming — Battle of the Terms!* Pridobljeno iz WebCite: http://www.webcitation.org/77YHuuIqu

Quora. (8. 4 2017). *Can anyone become a good programmer?* Pridobljeno iz Quora: https://www.quora.com/Can-anyone-become-a-good-programmer

Shaw, M. (1992). *We Can Teach Software Better.* Pridobljeno iz Carnegie Mellon University School of Computer Science: http://www.cs.cmu.edu/~Compose/ftp/TchSWBetter.pdf

AKTIVNOST 2: Računalniško razmišljanje

Aktivnost učne ure je osredotočena na računalniško razmišljanje.

TEORIJA

Wing (2017), eden izmed pionirjev računalniškega razmišljanja, se je strinjal z naslednjo definicijo: »Računalniško razmišljanje (RR) so miselni procesi, ki vključujejo formulacijo problema in izražanje njegovih rešitev tako, da jih lahko računalnik – človek ali stroj – uspešno izvede.«

Pri računalniškem razmišljanju gre za proces, v katerem se vzame določen problem, se ga razume in se razvije rešitev zanj. Nato se rešitev izrazi na tak način, da jo računalnik ali človek lahko razumeta.

Obstajajo štiri ključne komponente RR:

- razstavljanje
- prepoznavanje vzorca
- abstrakcija
- algoritmi

RR lahko razumemo kot temeljno veščino za vsakogar, ne le za računalničarje. Uporabno je tako v računalniškem kot v nemehaničnem procesu iskanja rešitev. RR ima potencial uporabe v širokem spektru disciplin kot kreativna priredba učenja.

ZA RAZMISLEK

- Zakaj je računalniško razmišljanje pomembno?
- Ali je za vsakega pomembno, da zna razmišljati na ta način?
- Ali se računalniško razmišljanje razlikuje od našega običajnega mišljenja?

IZVEDBA

- Skupinsko delo
- Zvrnjeno učenje
- Projektno učenje

VIRI

Google. (n. d.). *Exploring Computational Thinking.* Pridobljeno iz Google for Education: https://edu.google.com/resources/programs/exploring-computational-thinking/

ISTE. (3. 1 2012). *Computational thinking: A digital age skill for everyone.* Pridobljeno iz Youtube: https://www.youtube.com/watch?v=VFcUgSYyRPg

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142-158. Pridobljeno iz http://myweb.fsu.edu/vshute/pdf/CT.pdf

Togyer, J., & Wing, J. M. (n. d.). Research Notebook: Computational Thinking--What and Why? *The LINK - The magazine of Carnegie Mellon University's School of Computer Science*. Pridobljeno iz https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Wing, J. M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology, 25*(2), 7-14. Pridobljeno iz http://www.cs.cmu.edu/~wing/publications/Wing17.pdf

AKTIVNOST 3: Reševanje problemov

Aktivnost učne ure je osredotočena na reševanje problemov.

TEORIJA

Definicija: »Proces ali dejanje najti rešitev za problem,« slovar Merriam Webster Dictionary.

Druga zelo podobna definicija je: »Reševanje problemov je proces, v katerem identificiramo problem, razvijemo možne poti rešitve in ustrezno ukrepamo,« MIT – spletni tečaj reševanja problemov.

Ena komponenta evropskega okvirja digitalnih kompetenc za državljane je tudi reševanje problemov, kjer so definirane naslednje kompetence (Punie, 2017):

- reševanje tehničnih problemov
- prepoznavanje potreb in tehnološki odgovori
- kreativna uporaba digitalnih tehnologij
- prepoznavanje vrzeli v digitalnih kompetencah

Obstaja veliko virov o problemih na splošno in tudi vodnikov, kako se lotiti reševanja problemov. Bransford in Stein (1993) sta predlagala naslednji model (ang. IDEAL):

- Prepoznavanje problema
- Opredelitev konteksta problema
- Raziskovanja možnih rešitev
- Izpeljava najboljše rešitve
- Poglej nazaj in se uči.

Reševanje problemov in programiranje sta tesno povezana. Med programiranjem ljudje hkrati rešujejo probleme, ki se pojavljajo v procesu. Na splošno lahko definiramo naslednje, programsko naravnane, korake:

- razumeti problem
- analizirati problem in formulirati model
- razviti in zapisati algoritem s psevdokodo
- kodirati program
- testirati in poiskati napake programa
- oceniti rešitev in jo dokumentirati

IZVEDBA

- Skupinsko delo
- Razprava za in proti
- Projektno učenje

ZA RAZMISLEK

- Zakaj je reševanje problemov pomembno?
- Ali imajo veščine reševanja problemov ključni pomen za uspešno programiranje?
- Ali sta programiranje in reševanje problemov povezana? Razloži, kako?
- Ali obstajajo prav posebni koraki, po katerih naj bi potekalo reševanje problema?

VIRI

Gomes, A., & Mendes, A. (2019). *Problem solving in programming.* Pridobljeno iz https://pdfs.semanticscholar.org/1742/6220e02744ce9492bb4e56a0778333a79b18.pdf

Massachusetts Institute of Technology. (n. d.). *Introduction to problem solving skills.* Pridobljeno iz CC/MIT: https://ccmit.mit.edu/problem-solving/

Moursund, D. (2007). *Introduction to Problem Solving in the Information Age.* Oregon: College of Education, University of Oregon. Pridobljeno iz https://pages.uoregon.edu/moursund/Books/IAE-PS/PS-in-IA.pdf

Punie, Y. (2017). *DigComp 2.1: The Digital Competence Framework for Citizens.* Luxembourg: Publications Office of the European Union. Pridobljeno iz https://ec.europa.eu/jrc/en/publication/eur-scientific-and-technical-research-reports/digcomp-21-digital-competence-framework-citizens-eight-proficiency-levels-and-examples-use

AKTIVNOST 4: Ključna vloga učiteljev IKT

V tej aktivnosti slušatelji opišejo ključno vlogo učiteljev IKT za uspešno poučevanje in učenje programiranja.

TEORIJA

Slušatelji se lahko z uporabo interneta in odprtih virov sami naučijo programiranja in kodiranja. Na internetu je veliko virov in veliko dobrih programerjev je samoukov. Razlog, zakaj se večina ne uspe sama naučiti programiranja, leži v tem, da je krivulja učenja zelo strma in sčasoma izgubijo motivacijo.

Na drugi strani imajo učitelji IKT & STEM strokovno znanje, saj že znajo kodirati in lahko usmerjajo in svetujejo učečim, kako se ga lahko uspešno naučijo tudi sami.

IZVEDBA

- Razprava za in proti

ZA RAZMISLEK

- Ali se lahko programiranja naučimo zgolj iz spletnih virov oz. knjig? Ali je to lahko?
- Zakaj potrebujemo učitelje za poučevanje programiranja?

VIRI

Gutschank, J. (2019). *Coding in STEM Education.* Berlin, Germany: Science on Stage Deutschland e.V. Pridobljeno iz https://www.science-on-stage.eu/images/download/Coding_in_STEM_Education_web.pdf

Sentance, S., & Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies, 22*(2), 469–495.

Oznaka enote: UČNA ENOTA (U1-L2)

NASLOV: Uporaba kodiranja (10 % od 20 %)

Učni izidi

- Razložiti pomen učenja programiranja v STEM.
- Opisati vključitev in uporabo programiranja v STEM, robotiki in internetu stvari.
- Predstaviti programsko in strojno opremo za učenje programiranja.

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
✕ Skupinsko delo	☐ Vprašanja in odgovori	☐ Projektno učenje
☐ Igranje vlog	☐ Kombinirano učenje	☐ Demonstracija

Struktura poglavja

Aktivnost 1: Problemi STEM so zabavni

Aktivnost 2: Koncept STEM v učilnici

Aktivnost 3: Orodja in viri za STEM

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	☐ Medsebojno ocenjevanje	✕ Ustno preverjanje znanja
✕ Domača naloga	☐ Vzorčni izdelek	☐ Projektna aktivnost
✕ Predstavitev		☐ Drugo

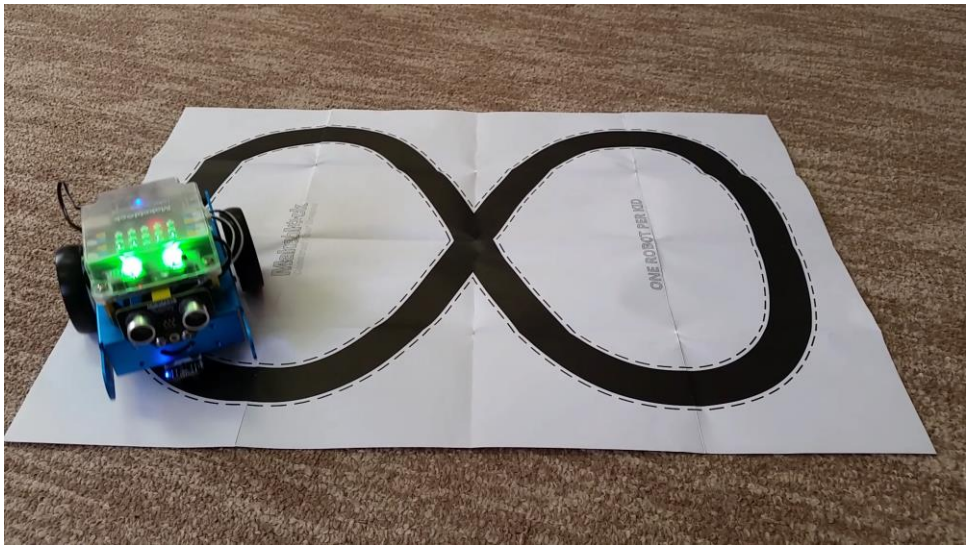
Aktivnost 1: Problemi STEM so zabavni

Učitelj povzame aktivnosti učne enote U1L1 in slušateljem zastavi nekaj vprašanj o njihovem poprejšnjem poznavanju STEM. Nato pokaže napravo STEM, kot je na primer preprost robot, in zastavi nekaj vprašanj glede delovanja robota in drugih naprav STEM. Cilj je spodbuditi slušatelje, da posamezno ali v skupini razpravljajo o možnih odgovorih. Učitelj jih pozove, da razmislijo o robotu ali kakšni drugi napravi STEM, ki jo uporabljajo, ter da razložijo, kako robot ali igrača STEM deluje.

Učitelj lahko za primer naprave STEM uporabi model avtomobila na daljinsko upravljanje. Gibanje modela avtomobila naprej in nazaj je lahko razloženo z opazovanjem, kako programska koda, ki skrbi za delovanje modela, nadzira gibanje modela ob vnosu različnih ukazov. Učitelj nato vzpodbudi slušatelje, da ob opazovanju in uporabi modela razpravljajo o inženirstvu, naravoslovju, tehnologiji, matematiki in umetnosti.

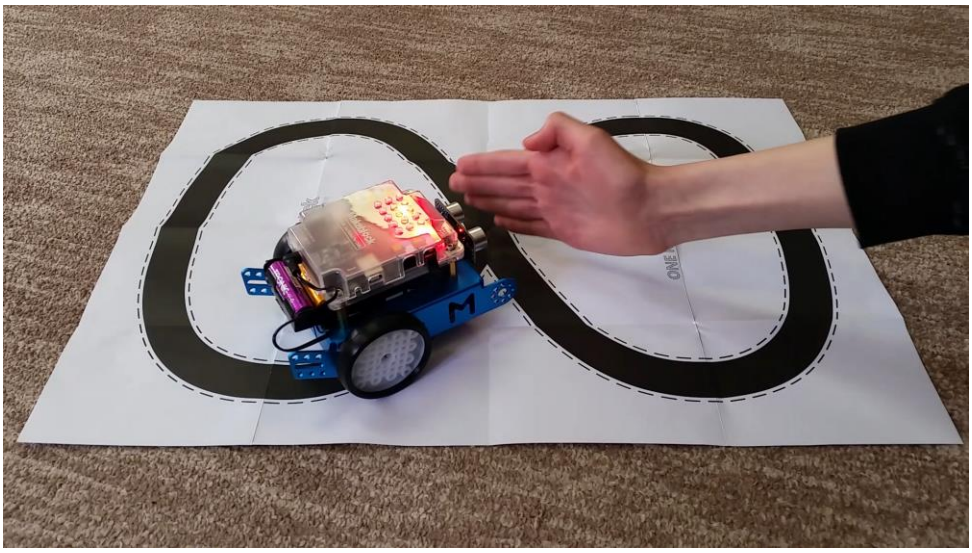
Učitelj lahko pokaže še številne druge naprave STEM, kot je na primer preprost robot, da vzpodbudi razmislek slušateljev glede uporabe kodiranja za robotske aplikacije ter jih seznani z osnovami te tematike.

Roboti so posebni računalniki, ki lahko komunicirajo s svojim okoljem, pri čemer uporabljajo senzorje, luči LED, zvočnike, motorje itd. Njihovo delovanje usmerja program, ki jim omogoča, da se odzivajo na različne razmere v okolju.



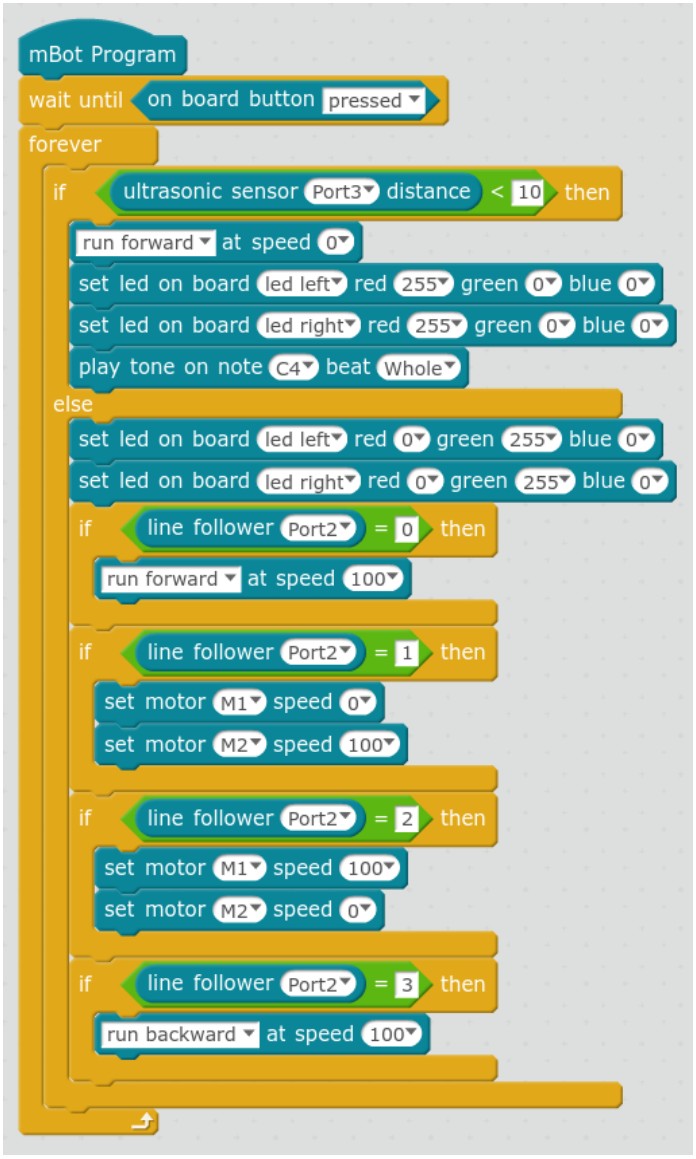
Slika 1: Robot mBot sledi črni črti

Kot primer lahko učitelj pokaže robota z imenom mBot (https://makeblock.com/steam-kits/mbot). Ta robot je zasnovan tako, da sledi črni črti, pri čemer ima prižgane zelene lučke LED (glej Sliko 1). Ko pride do ovire (lahko je škatla, knjiga ali roka, glej Sliko 2), se ustavi in pri tem piska ter spremeni lučke LED v rdeče. Ko je ovira odstranjena, nadaljuje svojo pot ob črti.



Slika 2: Robot mBot se ustavi pred oviro

Programska koda v programskem jeziku Scratch, ki usmerja robota, da sledi črni črti in se ustavi pred oviro, je predstavljena na Sliki 3.

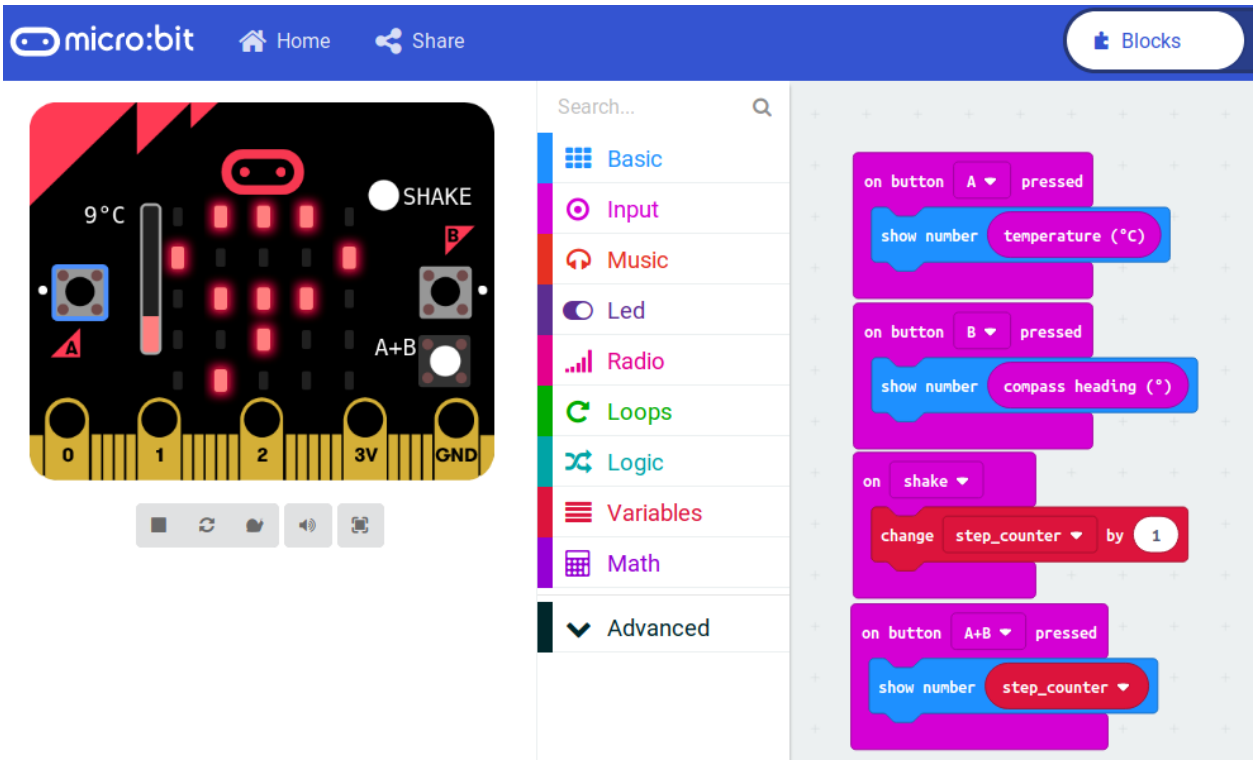


Slika 3: Programska koda v programskem jeziku Scratch, ki usmerja robota

Drugi primer naprave STEM je mikroračunalnik micro:bit, za katerega uporabljamo programski jezik Blocks (podoben programskemu jeziku Scratch).

Primer na Sliki 4 kaže programsko kodo, ki omogoča, da micro:bit kaže vrednost temperaturnega senzorja v stopinjah Celzija (ob pritisku na gumb A na napravi micro:bit) in vrednost senzorja magnetnega polja (kompasa) v kotnih stopinjah (0 - sever, 90 - vzhod, 180 - jug, 270 - zahod) (ob pritisku na gumb B na napravi micro:bit).

Naprava ima tudi senzor gibanja, ki se lahko uporabi na primer za štetje korakov ali za simulacijo meta igralne kocke. Če želimo videti število zaznanih korakov, moramo sočasno pritisniti gumba A in B.



Slika 4: Primer programske kode za micro:bit

IZVEDBA

- Razprava
- Učiteljevo opazovanje

ZA RAZMISLEK

- Kako dela robot?
- Kako vključiti in uporabiti kodiranje v STEM, robotiki in internetu stvari?

VIRI

makeblock. (n. d.). *mBot.* Pridobljeno iz makeblok: https://www.makeblock.com/steam-kits/mbot

micro:bit. (n. d.). *Power your imagination with code.* Pridobljeno iz micro:bit: https://microbit.org/code/

TechTarget network. (n. d.). *Robot.* Pridobljeno iz TechTarget network: https://searchenterpriseai.techtarget.com/definition/robot

AKTIVNOST 2: Koncept STEM v razredu

Učitelj razloži, kaj pomeni razumevanje koncepta STEM in njegova uporaba v razredu, vloge kodiranja v projektih STEM oz. aplikacijah in kako so ta področja med sabo povezana. Vsi slušatelji razpravljajo o pomenu, ki ga ima kodiranje za STEM. Učitelj se osredotoči na »T« v kratici STEM, saj kodiranje običajno uvrščamo na področje tehnologije.

TEORIJA

Koncept oz. pojem STEM je leta 2001 prvič omenil J. Ramaley, eden izmed direktorjev ameriških fundacij za nacionalno znanost NFS (National Science Foundations). STEM je angleška kratica za Science-znanost, Technology-tehnologija, Engineering-inženirstvo, in Mathematics-matematika. Vendar je po Whitu (2014) poučevanje STEM več kot zgolj kombinacija naslovov področij. Gre za filozofijo izobraževanja, ki predavatelju omogoča ustvarjanje učnega okolja, v katerem so prikazane določene veščine in teme, ki posnemajo resnični svet.

NFS definira STEM kot »*področja na splošno, ki ne vključujejo zgolj splošnih kategorij matematike, naravoslovja, inženirstva, računalništva in informatike, temveč tudi takšne družbene/vedenjske vede, kot so psihologija, ekonomija, sociologija in politologija.*« (Breiner, Sheats Harkness, Johnson, & Koeler, 2012, str. 4)

Znanost, tehnologija, inženirstvo in matematika, ki predstavljajo štiri temeljne kamne koncepta STEM, zasedajo pomembno mesto v akademskih karierah študentov, še zlasti sta zanje pomembna znanost in matematika.

Za ta področja lahko podamo naslednje definicije:

- Znanost (Science): sistematično preučevanje narave in obnašanja materialnega in fizičnega vesolja, temeljujoč na opazovanju, preizkušanju in meritvah, ter formuliranje zakonov, ki opisujejo ta dejstva, s splošnimi pojmi (https://www.dictionary.com, 2019).
- Tehnologija (Technology): veja znanja, ki se ukvarja z ustvarjanjem in uporabo tehničnih sredstev ter njihove medsebojne povezave z življenjem, družbo in okoljem, ter vključuje predmete, kot so industrijsko oblikovanje, inženirstvo, uporabne znanosti in čista znanost (https://www.dictionary.com, 2019).
- Inženirstvo (Engineering): obrt ali veda o praktični uporabi čistih znanosti, kot sta fizika ali kemija, pri npr. izdelavi motorjev, mostov, stavb, rudnikov, ladij in kemičnih tovarn (https://www.dictionary.com, 2019).
- Matematika (Mathematics): skupina povezanih ved, vključno z algebro, geometrijo in računanjem, ki se ukvarjajo s preučevanjem številk, količin, oblik in prostora ter njihovimi medsebojnimi povezavami, pri čemer uporabljajo specializirane zapise (https://www.dictionary.com, 2019).

POUČEVANJE KONCEPTA STEM

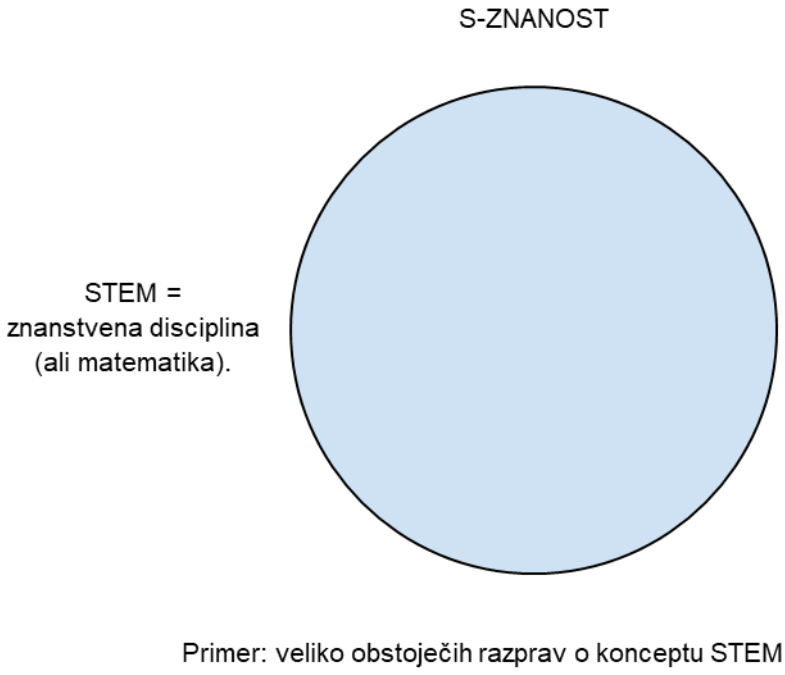
Izobraževanje STEM zadeva poučevanje in učenje na področjih STEM, je pristop, ki učečim omogoča neposredno učenje – da vidijo in razumejo, kako stvari delujejo, in lahko prenesejo svoje znanje na nove in različne probleme. Tisti, ki imajo ustrezne informacije o konceptu

STEM, lahko z njim rešujejo težave, na katere naletijo v svojem vsakdanjem življenju in izboljšajo svojo uporabo tehnologij.

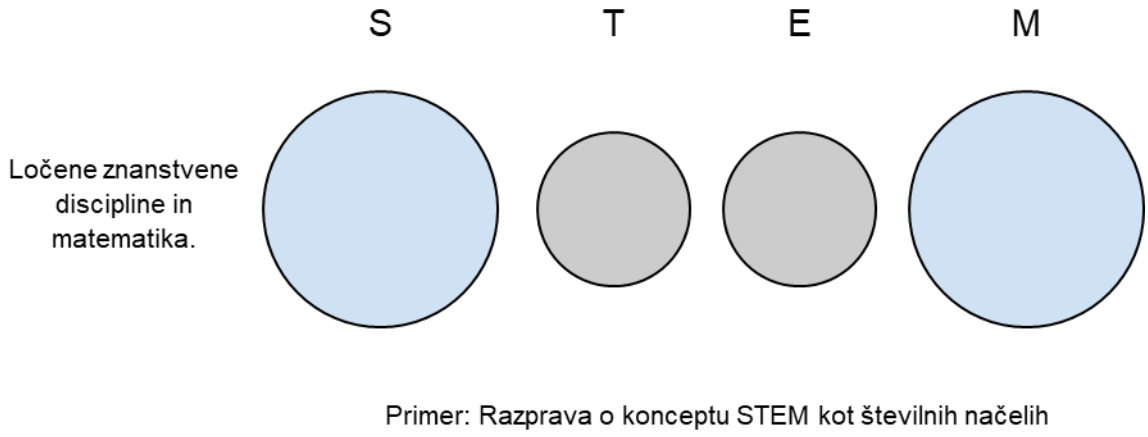
Bybee (2013) navaja, da je splošen cilj izobraževanja STEM izboljšati pismenost koncepta STEM v družbi. Njegova definicija pismenosti STEM se nanaša na posameznika, ki:

- ima znanje, odnos in veščine, da prepozna vprašanja in probleme v življenjskih situacijah, razloži naravni in umetni svet ter na podlagi dokazov pride do zaključkov o vsebinah, povezanih s konceptom STEM;
- razume značilne lastnosti disciplin STEM kot vrste človekovega znanja, raziskovanja in oblikovanja;
- se zaveda, kako discipline STEM oblikujejo naša materialna, intelektualna in kulturna okolja;
- se je pripravljen ukvarjati z zadevami, povezanimi s konceptom STEM ter z idejami znanosti, tehnologije, inženirstva in matematike kot ustvarjalen, zavzet in razmišljujoč državljan (str. 101).

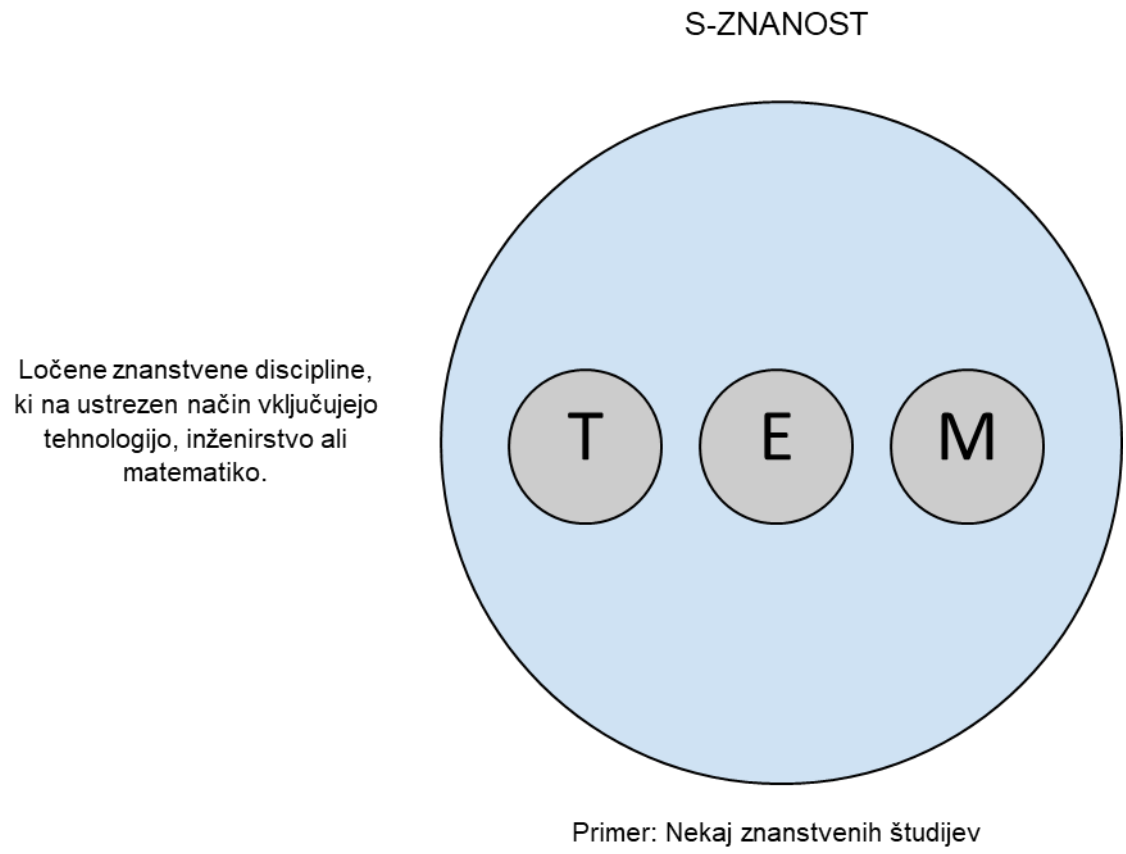
Obstajajo različni pogledi na izobraževanje STEM, ki so ponazorjeni na Slika 5, Slika 6, Slika 7. Cilj je, da se izobraževanje STEM pokaže na jasen in razumljiv način.



Slika 5: Pojmovanje koncepta STEM kot ene discipline (Bybee, 2013)



Slika 6: Pojmovanje koncepta STEM kot ločene znanstvene discipline in matematike (Bybee, 2013)



Slika 7: Ločene znanstvene discipline, ki vključujejo druge discipline (Bybee, 2013)

ZA RAZMISLEK

- Zakaj je koncept STEM pomemben?
- Katere so prednosti in slabosti izobraževanja STEM?

VIRI

Breiner, J., Sheats Harkness, S., Johnson, C., & Koeler, C. M. (2012). What is STEM? A discussion about Conceptions of STEM in education and partnerships. *School Science and Mathematics, 112*(1), 3-11.

Bybee, R. W. (2013). *The Case for STEM Education: Challenges and Opportunities.* NSTA Press (National Science Teachers Association).

Mills, A. (4. 10. 2017). *What is STEM education?* Pridobljeno iz Phys.org: https://phys.org/news/2017-10-stem.html

STEAM Powered Family. (19. 3. 2018). *What is STEM and STEAM? A guide for parents and educators.* Pridobljeno iz STEAM Powered Family: https://www.steampoweredfamily.com/education/what-is-stem/

White, D. (2014). What is STEM education and why is it important? *Florida Association of Teacher Educators Journal, 14*, 1-8.

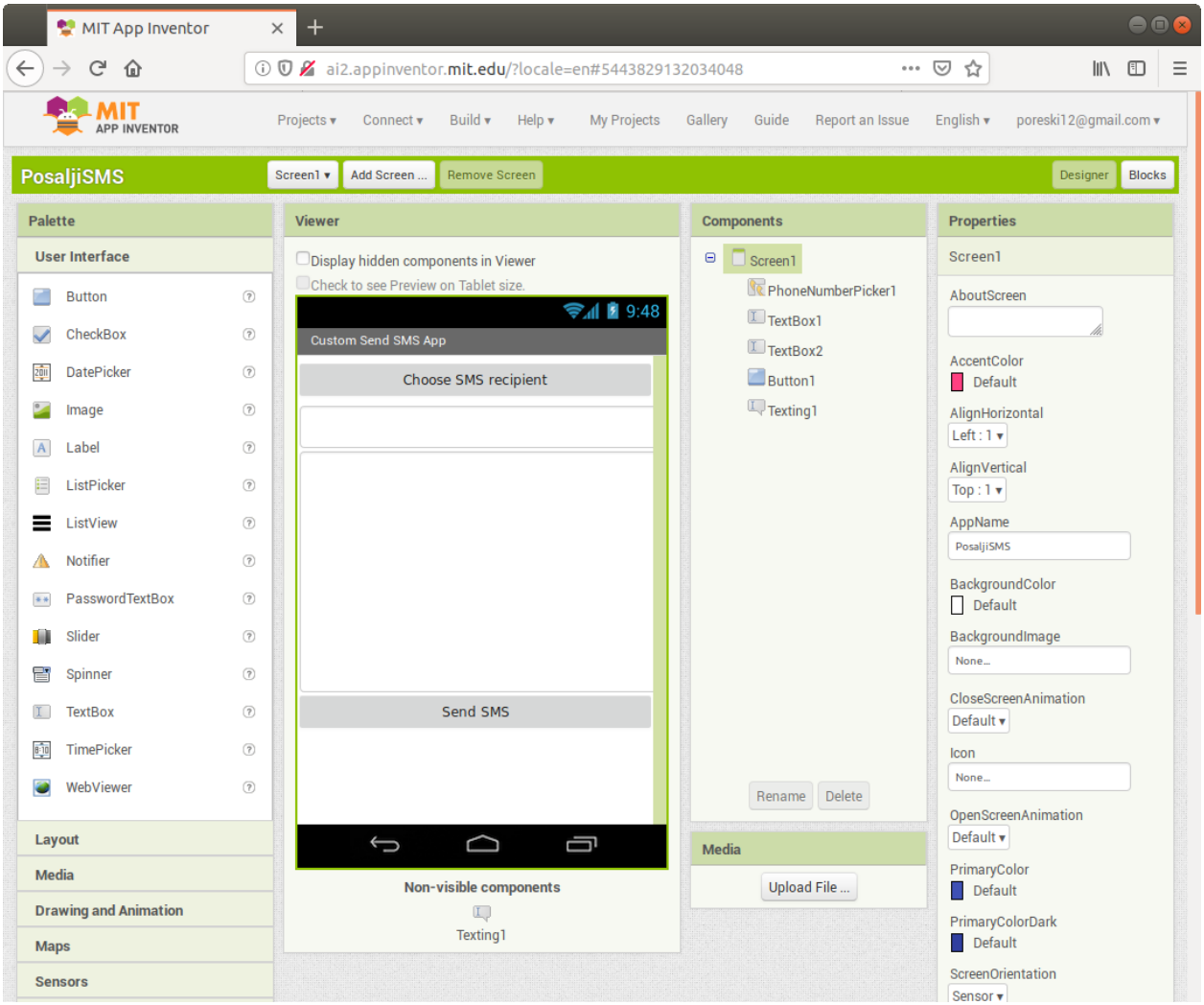
AKTIVNOST 3: Orodja in viri STEM

IZVEDBA

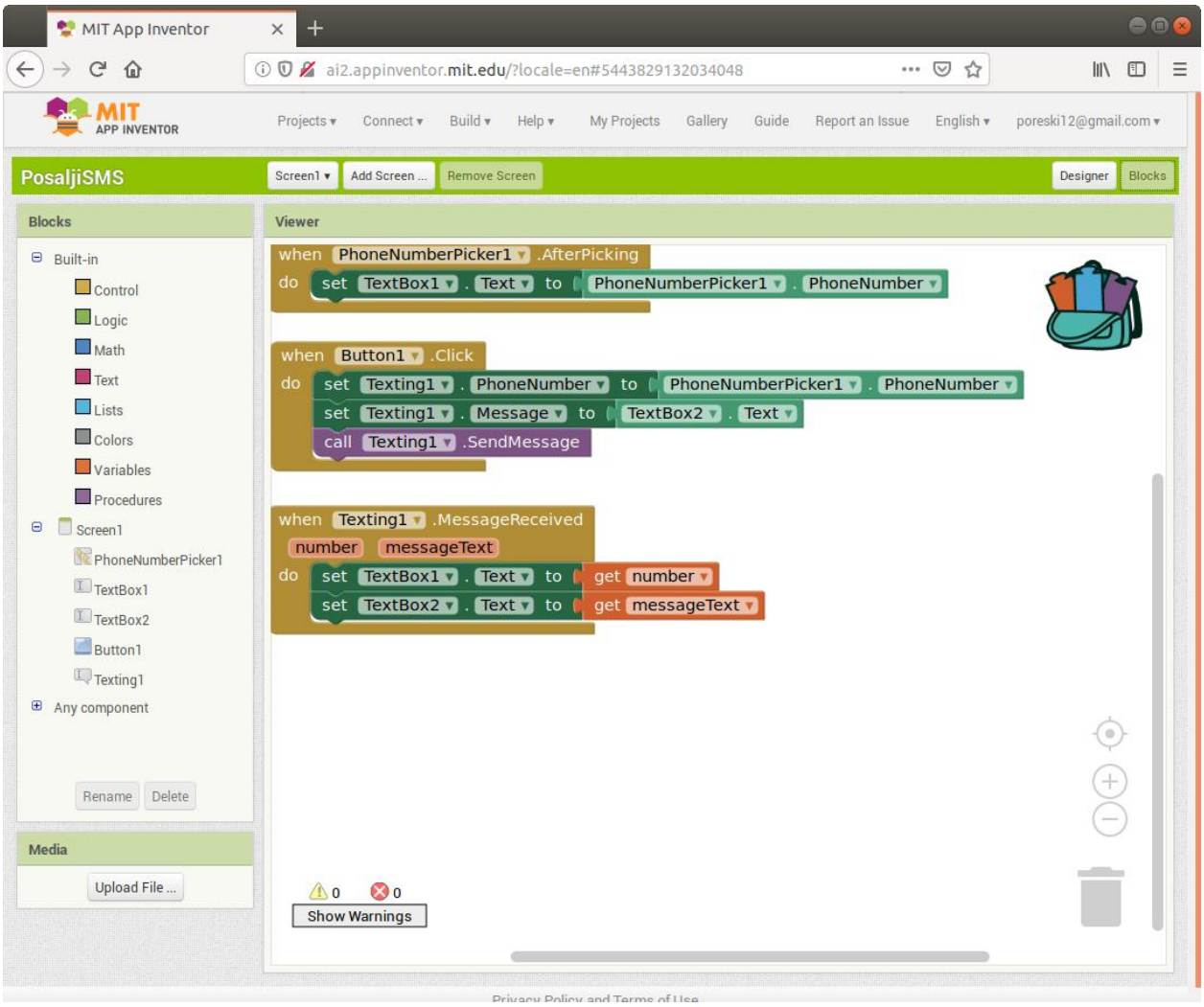
Učitelj pozove slušatelje, da najdejo spletni vir, npr. znanstveni članek/raziskavo, ki izpostavlja nekaj orodij in uporabo STEM, povezanih s programiranjem. Slušatelji delajo v skupini in so usmerjeni na to, da najdejo orodja STEM in jih ovrednotijo na podlagi kriterijev, ki jih pripravi učitelj.

Primeri orodij in uporab STEM, povezanih s kodiranjem, so za slušatelje na voljo na https://www.commonsense.org/education/top-picks/stem-apps-for-higher-order-thinking.

Dodatni primer, ki opisuje preprost način kodiranja za pametne telefone (mikro računalniki s senzorji), je na voljo na spletni strani MIT App Inventor http://appinventor.mit.edu/explore/. Slika 8 in Slika 9 prikazujeta aplikacijo, izdelano po meri, ki omogoča pošiljanje in sprejemanje kratkih sporočil (SMS).



Slika 8: MIT App Inventor – programerski pogled



Slika 9: MIT App Inventor – pogled v programskem jeziku Blocks

Spletni viri se lahko ovrednotijo po naslednjih kriterijih:

- katera področja STEM so vključena
- interdisciplinarnost
- upoštevana orodja in aplikacije
- ali so usmerjeni na strojno ali na programsko opremo
- ali so komercialni ali nekomercialni
- prijazen vmesnik
- spletna pomoč

VIRI

Common sense education. (n. d.). *STEM Apps for Higher-Order Thinking*. Pridobljeno iz Common sense education: https://www.commonsense.org/education/top-picks/stem-apps-for-higher-order-thinking

DCSTEM network. (n. d.). *Resources & Tools*. Pridobljeno iz DCSTEM network: https://www.dcstemnetwork.org/resources-and-tools/

DS Examiner. (14. 7. 2014). *The Ultimate STEM Guide for Kids: 239 Cool Sites About Science, Technology, Engineering and Math*. Pridobljeno iz Master's in data science:

https://www.mastersindatascience.org/blog/the-ultimate-stem-guide-for-kids-239-cool-sites-about-science-technology-engineering-and-math/

MIT App inventor. (n. d.). *MIT App inventor*. Pridobljeno iz MIT App inventor: http://appinventor.mit.edu/explore/

Techno Kids. (n. d.). *TechnoCode*. Pridobljeno iz Techno Kids: K-12 Technology projects: https://www.technokids.com/store/middle-school/technocode/scratch-for-kids.aspx

Oznaka enote: UČNA ENOTA (U2-L1)

NASLOV: Učno gradivo in orodja za učenje programiranja (10 % od 20 %)

Učni izidi

- Ovrednotiti primerna spletna mesta za učenje programiranja.
- Izbrati primerna motivacijska orodja (npr. učni material za računalništvo brez računalnika, vizualizacija algoritmov), da se navduši učence za učenje programiranja.

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
✕ Skupinsko delo	☐ Vprašanja in odgovori	☐ Projektno učenje
☐ Igranje vlog	☐ Kombinirano učenje	☐ Demonstracija

Struktura poglavja

Aktivnost 1: Motivacijska aktivnost – primer iz realnega življenja

Aktivnost 2: Koncept motivacije

Aktivnost 3: Učenje na podlagi primerov, računalništvo brez računalnika in vizualizacija algoritmov

Aktivnost 4: Učenje programiranja na podlagi primera

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	☐ Medsebojno ocenjevanje	✕ Ustno preverjanje znanja
✕ Domača naloga	☐ Vzorčni izdelek	☐ Projektna aktivnost
✕ Predstavitev		☐ Drugo

AKTIVNOST 1: Motivacijska aktivnost – primer iz realnega življenja

Ta aktivnost je osredotočena na kriterije ovrednotenja spletnih strani za učenje programiranja.

IZVEDBA

Cilj te aktivnosti je usmeriti pozornost slušateljev na temo učne ure in jim pokazati, kako ustvariti predlogo za sprejemanje odločitev pri ovrednotenju spletnih strani. Učitelj lahko začne učno uro z nekaj vprašanji.

Spodnja tabela se lahko pokaže slušateljem kot primer, ki jih usmerja po tem, ko so že podali svoje odgovore. Argumenti za in proti se lahko razlikujejo na podlagi komentarjev.

Tabela 2: Ovrednotenje različnih spletnih strani

	Greentoe	amazon.com	Walmart
Redna dostava	✓	✓	✗
Dostava po meri naročnika	✗	✓	✗
Brezplačno pošiljanje	✓	✓	✓
Politika vračanja	30 dni	30 dni	90 dni
Za	hitro		
Proti			

Ovrednotenje ustreznih spletnih strani za učenje programiranja

Učitelj slušatelje razdeli v skupine in jih pozove, da poiščejo ustrezne spletne strani za učenje programiranja na podlagi naslednjih kriterijev:

- dostopnost tečajev
- brezplačni tečaji programiranja
- prijazen vmesnik
- uporaba namigov
- spletna pomoč
- potrdila
- prednosti in slabosti

ZA RAZMISLEK

- Kakšne strategije uporabljate za nakupovanje preko spleta?
- Katera spletna mesta so vam bolj všeč? Zakaj?

VIRI

Anderson, L. W., & Krathwohl, D. R. (2001). A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. New York: Longman.

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Wittrock, M. C. (n. d.). REVISED Bloom's Taxonomy Action Verbs. Pridobljeno iz AZUSA

Pacific University: https://www.apu.edu/live_data/files/333/blooms_taxonomy_action_verbs.pdf

Bigby, G. (29. 1. 2019). Top 25 Amazing Websites to Learn How to Code. Pridobljeno iz Dyno Mapper: https://dynomapper.com/blog/410-top-25-websites-to-learn-to-code

Video. Hour of Code Intro. (n. d.). Frozen - Hour of Code Introduction. Pridobljeno iz Code.org: https://studio.code.org/s/frozen/stage/1/puzzle/1

w3school.com. (n. d.). Python Tutorial. Pridobljeno iz w3school.com: https://www.w3schools.com/python/default.asp

Wilson, L. O. (2016). Anderson and Krathwohl – Bloom's Taxonomy Revised. Pridobljeno iz The Second Priciple: https://thesecondprinciple.com/teaching-essentials/beyond-bloom-cognitive-taxonomy-revised/

AKTIVNOST 2: Koncept motivacije

Ta aktivnost se osredotoča na razlago motivacijskih konceptov in na eno izmed motivacijskih teorij (npr. Kellerjev motivacijski model ARCS).

TEORIJA

Motivirani učenci so eden izmed temeljev razredne pedagogike.

Kaj je motivacija?

»Motivacija je definirana kot dejanje oz. proces motiviranja; stanje, ko smo motivirani; motivacijska sila, stimulant ali vpliv; spodbuda; zagon; nekaj (kot potreba ali želja), ki navede osebo ali študenta k dejanju; in porabi truda za doseganje rezultata.« (Vero & Puka, 2017)

John Kellerjev model ARCS vključuje štiri korake za ohranjanje ali povečevanje motivacije pri učnem procesu: pozornost (Attention), relevantnost (Relevance), zaupanje (Confidence), zadovoljstvo (Satisfaction) (Keller, 1987).

- **Pozornost**: Se nanaša na učenčevo zanimanje. Ključno je, da spodbudimo in ohranimo učenčevo zanimanje in pozornost.
- **Relevantnost**: Učni proces mora pokazati uporabnost vsebine, tako da si učenci lahko zgradijo most med vsebino in realnim svetom.
- **Zaupanje**: Ta dejavnik se osredotoča na razvijanje pričakovanja na uspeh med učenci. Pričakovanje uspeha omogoča učencem, da nadzirajo svoje učne procese. Obstaja korelacija med ravnijo zaupanja in pričakovanjem uspeha. Zato je pomembno, da učencem povemo, kolikšne so možnosti, da jim uspe.
- **Zadovoljstvo**: Obstaja neposredna povezava med motivacijo in zadovoljstvom. Učenci morajo biti zadovoljni s tem, kar so dosegli med učnim procesom (ARCS model of motivation, n. d.)

Ta model vsebuje nekaj metod in strategij, ki učitelju pomagajo, da ohranja vsakega izmed motivacijskih dejavnikov. Učitelj lahko navede nekaj primerov za vsak motivacijski dejavnik, npr. za spodbuditev čutil lahko na humoren način navede nekaj primerov iz realnega sveta ali pa slušatelje povpraša o njihovih preteklih izkušnjah.

IZVEDBA

Učitelj definira motivacijo na podlagi virov, navedenih spodaj. Izpostavi lahko pomen motivacije pri učenju (spodnja vprašanja so postavljena v to smer).

Na področju izobraževanja obstaja veliko motivacijskih teorij, toda učitelj lahko izbere nekaj najbolj znanih ter predstavi njihove glavne značilnosti, npr. Maslowa hierarhija potreb (1954), McClellandova teorija potreb (1961), »Herzbergova teorija dveh dejavnikov« (Rily, 2005, str. 3) in Kellerjeva teorija ARCS (1987).

ZA RAZMISLEK

- Zakaj je motivacija pomembna pri izobraževanju?
- Katere so najbolj znane teorije o motivaciji?
- Katerih pet elementov ima ključni vpliv na učenčevo motivacijo?

VIRI

ARCSmodel.com. (n. d.). *Attention, relevance, confidence, satisfaction*. Pridobljeno iz ARCSmodel.com: https://www.arcsmodel.com/

Chand, S. (n. d.). *Motivation Theories: Top 8 Theories of Motivation – Explained!* Pridobljeno iz YourArticleLibrary: The next generation library: http://www.yourarticlelibrary.com/motivation/motivation-theories-top-8-theories-of-motivation-explained/35377

Keller, J. M. (1987). Development and use of the ARCS model of instructional design. *Journal of instructional development, 10*(3), 2-10.

Learning theories. (2005-2019). *ARCS model of motivational design theories (Keller).* Pridobljeno iz Learning theories: https://www.learning-theories.com/kellers-arcs-model-of-motivational-design.html

Seifert, K., & Sutton, R. (2009). *Educational Psychology (Second Edition).* Zurich, Switzerland: The Saylor Foundation. Pridobljeno iz https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2012/06/Educational-Psychology.pdf

Sookdeo, C. (26. 4 2012). *Motivational Theories In Education.* Pridobljeno iz SlideShare: https://www.slideshare.net/ChristinaSookdeo/motivational-theories-12696671

Texas Tech University. (n. d.). *ARCS model of motivation.* Pridobljeno iz The Texas A&M University System: http://www.tamus.edu/academic/wp-content/uploads/sites/24/2017/07/ARCS-Handout-v1.0.pdf

Vero, E., & Puka, E. (2017). The Importance of Motivation in an Educational Environment. *Formazione & Insegnamento XV, 15*(1), 57-66.

Williams, K. C., & Williams, C. (2011). Five Key Ingredients for Improving Student Motivation. *Research in Higher Education Journal, 12*(1), 104-122.

AKTIVNOST 3: Učenje na primerih, računalništvo brez računalnika in vizualizacija algoritmov

Ta aktivnost vključuje definicijo učenja na podlagi primerov, računalništva brez računalnika in vizualizacije algoritmov.

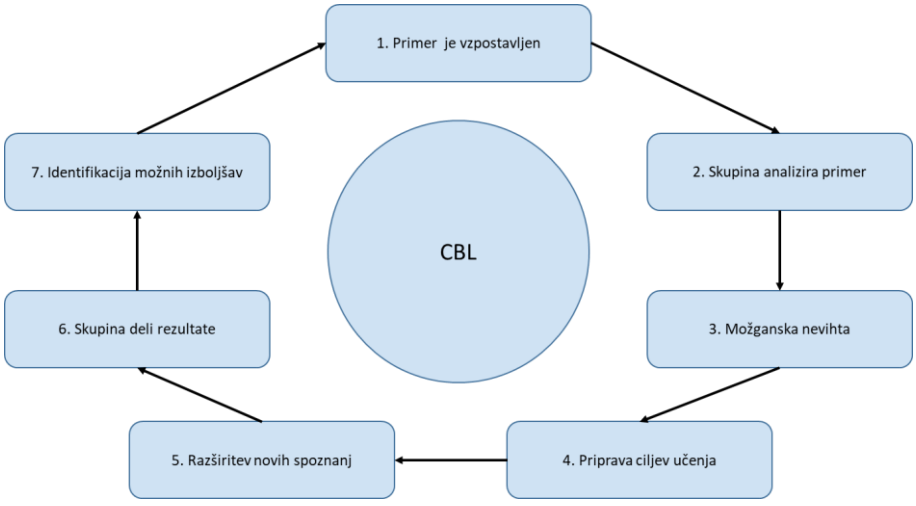
TEORIJA

Učenje na primerih (Case-based learning CBL) je model učenja, ki ima podobna izhodišča kot projektno naravnano učenje. V ožjem smislu je učenje na primerih precej podobno učenju na podlagi problemov, toda temelj tega pristopa je pripeljati učence do razprave, ki temelji na primerih iz realnega življenja ali na specifičnih okoliščinah.

Ta metoda je osredotočena na učenca. V tem pristopu udeleženci skupaj preučujejo primer, medtem ko si ustvarjajo svoje lastne informacije. Vloga učitelja je, da z metodo sodelovanja vodi učence, medtem ko analizirajo vprašanja in iščejo več primernih odgovorov nanje.

Za razlago učenja na primerih se lahko uporabi naslednji model/seznam.

1. Postavi se primer.
2. Primer analizira več skupin.
3. Ideje se zberejo in soočijo.
4. Oblikuje se učne cilje.
5. Širijo se nove ugotovitve.
6. Rezultati se delijo z drugimi skupinami.
7. Prepoznajo se področja za napredek.



Slika 10: Pristop učenja na primerih (CBL)[2]

Pomembno je tudi, da pri učenju na primerih definiramo vloge učitelja in učencev ter značilnosti tovrstnega učenja: (source:http://edutechwiki.unige.ch/en/Case-based_learning#What_is_case-based_learning.3F)

[2] Source: https://emj.bmj.com/content/emermed/22/8/577.full.pdf

Značilnosti

- Osredotočenost na učenca.
- Sodelovanje in medsebojna pomoč med udeleženci.
- Razprava o specifičnih situacijah, tipični primeri iz realnega sveta.
- Vprašanja, ki nimajo zgolj enega pravilnega odgovora.

Učenci

- Vključeni so v karakterje in okoliščine zgodbe.
- Identificirajo probleme, kot jih zaznajo.
- Povežejo pomen zgodbe s svojim življenjem.
- Prispevajo svoje predhodnje znanje in načela.
- Postavljajo trditve in vprašanja ter zagovarjajo svoja stališča.
- Oblikujejo strategije za analiziranje podatkov ter predstavijo možne rešitve.
- Mogoče se ne bodo strinjali, zato pride včasih do kompromisov.

Učitelj

- Posrednik.
- Spodbuja raziskovanje primera in preučuje dejanja akterjev v luči svojih odločitev.

Primeri

- Temeljijo na dejstvih.
- Kompleksni problemi, napisani za stimulacijo razprave v razredu in skupinsko analizo.
- Vključujejo interaktivna, na učence osredotočena raziskovanja realističnih in specifičnih situacij.

RAČUNALNIŠTVO BREZ RAČUNALNIKA

»Računalništvo brez računalnika je zbirka prosto dostopnega izobraževalnega materiala, namenjenega poučevanju računalništva s pomočjo iger in ugank, ki uporabljajo karte, vrvice, barvice in veliko tekanja naokoli.« (https://csunplugged.org/en/)

Na pristop računalništva brez računalnika lahko gledamo kot na srce računalništva. Poleg tega ta pristop omogoča prednosti, kot so ukvarjanje z računalniki oz. celo dostopanje do njih, brez da bi se morali naučiti programskega jezika. Osnovno načelo tega pristopa je medsebojna pomoč, ne pa učenje o računalnikih. S tem ta pristop tudi cilja na odstranjevanje nepravilnih informacij o računalniškem področju.

Aktivnosti računalništva brez računalnika imajo nekaj skupnih značilnosti: (Nishida, in drugi, 2009):

- Ni računalnika:
  - Kot že naslov pove, v aktivnostih tega pristopa ni neposredne uporabe računalnikov.
- Igre:
  - Aktivnosti temeljijo na igri oz. izzivu, ki vsebuje zanimanje, radovednost in motivacijo. Ti omogočajo, da učenci aktivnost doživljajo kot igro.

- Kinestezija:
  - V aktivnostih se uporabljajo fizični predmeti, kot so karte ali uteži. S tem se vključi kinestetična zaznava, zaradi nenavadnih predmetov pa tudi humor (npr. uporaba sadja pri uri računalništva).
- Usmerjeno na učence:
  - Igre običajno vključujejo interakcijo z drugimi učenci. Posledično se učenci trudijo priti do odgovora s pomočjo poskušanja in delanja napak.
- Preprosta uporaba:
  - Aktivnosti pogosto vključujejo lahko dostopno opremo, ki jo ponavadi najdemo v vsaki šoli. Priprava in uporaba opreme je preprosta.
- Smisel zgodbe:
  - Zgodba običajno vsebuje fantastične elemente (npr. pirate, skrivnostna sporočila), ki vključijo učence v izvedbo učne ure.

Cilj tehnologije vizualizacije algoritma **(AV)** je, da pomaga učencem razumeti, kako delujejo algoritmi. Poleg tega lahko to tehnologijo definiramo kot »grafično predstavitev« dinamičnega vedenja računalniških algoritmov.

»Tehnologija vizualizacije algoritmov se je iz programske opreme, naravnane na paketni sistem, ki omogoča predavateljem ustvarjanje animiranih filmov, razvila do visoko interaktivnih sistemov, ki omogočajo učencem, da sami dinamično raziskujejo konfigurirane animacije algoritmov; in interaktivnih programerskih okolij, ki omogočajo učencem, da si hitro ustvarijo svoje lastne vizualizacije.« (Donmez & Inceoglu, 2008, str. 531)

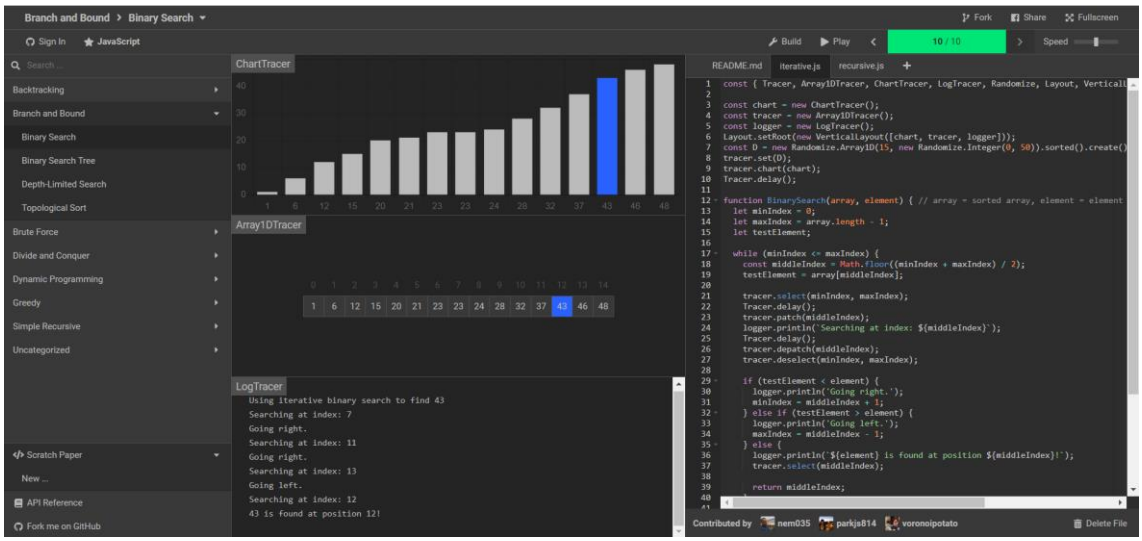
»Programska oprema vizualizacije algoritmov se uporablja (Hundhausen, Douglas, & Stasko, 2002, str. 260):

- za pomoč predavateljem pri vizualizaciji algoritmičnih operacij v učni uri;
- za pomoč učencem pri učenju o temeljnih algoritmih v računalništvu;
- za pomoč predavateljem, da med uradnimi urami npr. izsledijo napake študentov v programih povezanih seznamov;
- za pomoč učencem pri učenju o temeljnih operacijah abstraktnih podatkovnih tipov v računalniškem laboratoriju.«

Učitelj lahko slušateljem pokaže nekaj orodij za vizualizacijo, ki jih lahko pomagajo naučiti, kako ustvariti programsko kodo v tem orodju. Ta orodja se lahko najdejo na spletu:

- https://algorithm-visualizer.org/
- https://visualgo.net/en
- http://www.algomation.com/ ali programska oprema;
- http://www.flowgorithm.org/

Denimo, predavatelj lahko ustvari programsko kodo na spletni strani https://algorithm-visualizer.org/ in učenci lahko takoj vidijo rezultat ter analizirajo proces izvajanja kode.



Slika 11: Program za vizualizacijo algoritma

ZA RAZMISLEK

- Zakaj je pomembno učenje na primerih?
- Kakšna je razlika med učenjem na podlagi primerov in učenjem na podlagi problemov?
- Zakaj je vizualizacija algoritmov koristna?
- Kaj spodbuja računalništvo brez računalnika?
- Ali nam lahko pristop računalništva brez računalnika pomaga razumeti algoritme?

VIRI

CS Unplugged. (n. d.). *CS Unplugged Topics.* Pridobljeno iz CS Unplugged: https://csunplugged.org/en/topics/

Donmez, O., & Inceoglu, M. M. (2008). A Web Based Tool for Novice Programmers: Interaction in Use. *ICCSA '08 - International conference on Computational Science and Its Applications* (str. 530 - 540). Perugia, Italy: Springer-Verlag.

Grissom, S., McNally, M. F., & Naps, T. (2003). Algorithm visualization in CS education: comparing levels of student engagement. *SoftVis '03 Proceedings of the 2003 ACM symposium on Software visualization* (str. 87-94). San Diego, CA: ACM.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A Meta-Studyof AlgorithmVisualization Eļectiveness. *Journal of Visual Languages and Computing, 13*, 259-290. doi:doi:10.1006/S1045-926X(02)00028-9

immersed. (n. d.). *Gamification versus game-based learning.* Pridobljeno iz Immersed games: http://www.immersedgames.com/gamification-vs-game-based-learning/

Kapp, K. M. (2012). *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education.* San Francisco, CA: John Wiley & Sons, Inc.

Kapp, K. (2014). GAMIFICATION: Separating Fact From Fiction. *Chief Learning Officer*, 42-52. Pridobljeno iz http://ww.w.cedma-europe.org/newsletter%20articles/Clomedia/Gamification%20-%20Separating%20Fact%20from%20Fiction%20(Mar%2014).pdf

Michael, D., & Chen, S. (2006). *Serious Games: Games That Educate, Train, and Inform.* Boston, MA: Thomson Course Technology PTR. Pridobljeno iz https://anagroudeva.files.wordpress.com/2013/06/serious_games__games_that_educate__train__and_inform.pdf

Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T. C., & Kuno, Y. (2009). A CS unplugged design pattern. *40th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2009* (str. 231-235). Chattanooga, TN: ACM.

AKTIVNOST 4: Učenje programiranja na podlagi primera

V tej aktivnosti se pokaže različne primere učenja na podlagi primerov.

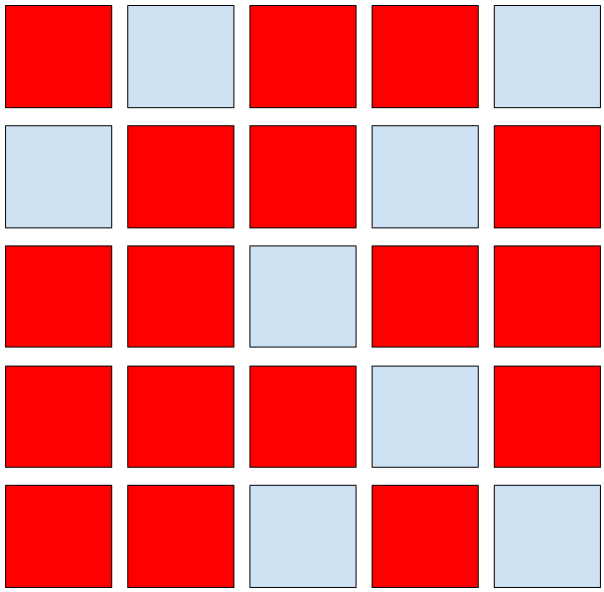
IZVEDBA

Učitelj lahko na začetku ure uporabi naslednje primere v skladu z učnim procesom učenja na podlagi primerov. Primer se predstavi slušateljem in ko ga vsi razumejo, začnejo razpravo o njem in se odločijo za pristop, ki je po njihovem mnenju najboljši za dani primer.

Primer 1: Prepoznavanje in popravljanje napak s pomočjo barvnih kartic

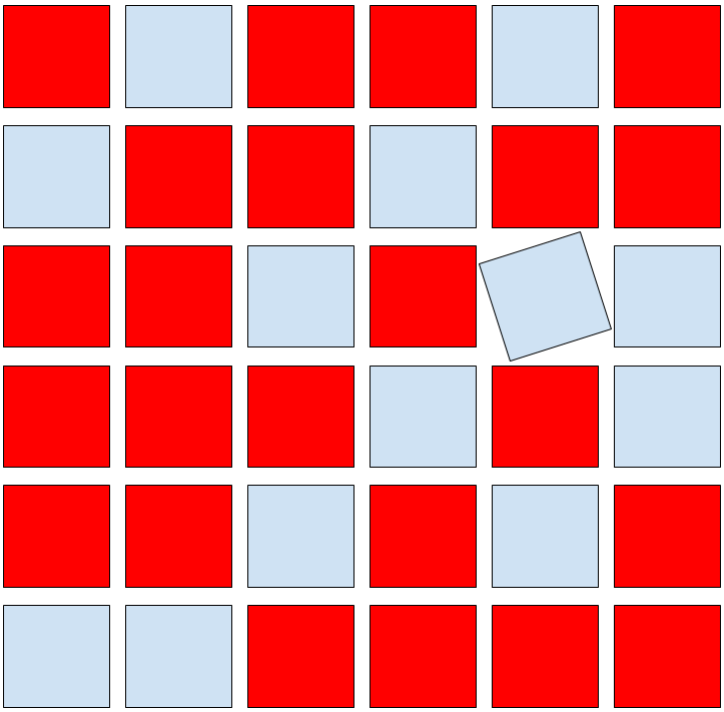
Ideja za ta primer prepoznavanja in popravljanja napak najdete v (Card Flip Magic—Error Detection & Correction, 2002)

Učitelj izbere slušatelja, ki naredi kvadrat 5x5 iz magnetnih kartic, ki so na eni strani pobarvane z eno barvo npr. rdečo, na drugi strani pa z drugo barvo npr. modro. Slušatelj naj izbere poljubno barvo ter postavi kartice v kvadrat (Slika 13.)



Slika 12: Magnetne kartice v kvadratu 5x5

Učitelj naj naključno doda še po eno vrstico vodoravno in navpično in pove, da bi rad naredil stvari malo težje. Ti dodatni vrstici sta ključni za zagotovitev parnega števila vrstic in stolpcev. Sedaj imamo magnetne kartice v kvadratu 6x6 in učitelj obrne kartice tako, da bo v vsaki vrstici in v vsakem stolpcu enako število rdečih kartic.



Slika 13: Magnetne kartice v kvadratu 6x6 z eno obrnjeno kartico

Zdaj učitelj pozove slušatelja, da obrne eno kartico, medtem ko si on oz. ona pokrije oči. Ko odpre oči, učitelj brez težav najde obrnjeno kartico, saj se je razmerje v tisti vrstici in stolpcu porušilo – število rdečih kartic je sedaj neparno, kar olajša iskanje obrnjene kartice (Slika 14).

V naslednjem koraku učitelj organizira slušatelje, da se to igro igrajo v parih.

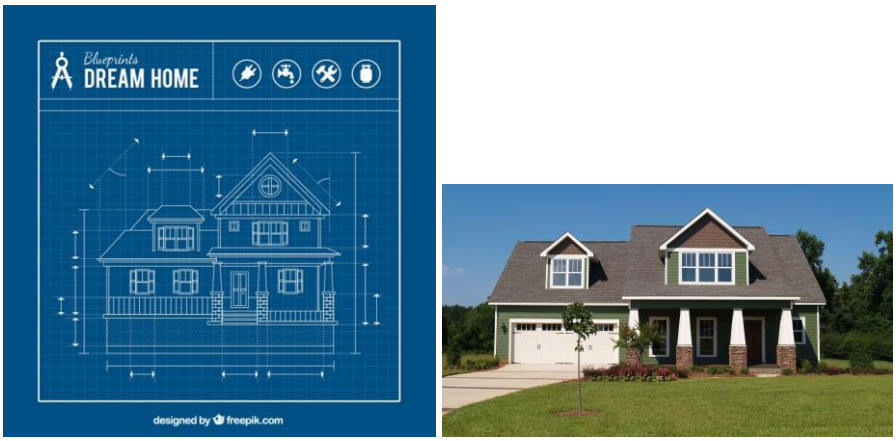
Primer 2: Razredi in objekti

Za razlago razlike med razredi in objekti v programskih jezikih učitelj pokaže primer načrta za hišo (razred) in več stvarnih hiš (dejanskih objektov, narejenih na podlagi načrtov za hišo) (Slika 15).

Slika 14: Načrt postane stvarni predmet

Objektno programiranja pomeni uporabo v naprej definiranih programskih modularnih enot (objektov, razredov, podrazredov itd.) za to, da je programiranje hitrejše in se lažje vzdržuje. Objektno usmerjeni jeziki pomagajo upravljati kompleksnost v velikih programih. Objekti zapakirajo podatke in operacije na njih, tako da so javno dostopne le operacije, notranje podrobnosti o podatkovni strukturi pa so skrite. To zakrivanje podatkov je olajšalo programiranje velikih formatov, saj se lahko programer osredotoči na vsak del programa posebej. Poleg tega lahko objekti izhajajo iz bolj splošnih, od katerih »dedujejo« njihove zmožnosti. Takšna hierarhija je omogočila definiranje specializiranih objektov brez da bi se moralo ponavljati tisto, kar vsebujejo bolj splošni objekti (Hemmendinger, 2000).

Več objektov lahko definiramo na podlagi razreda. V danem primeru pa je lahko vsaka hiša (objekt) drugačne barve, ima lahko drugačna vhodna vrata, eno ali dve kopalnici itd.

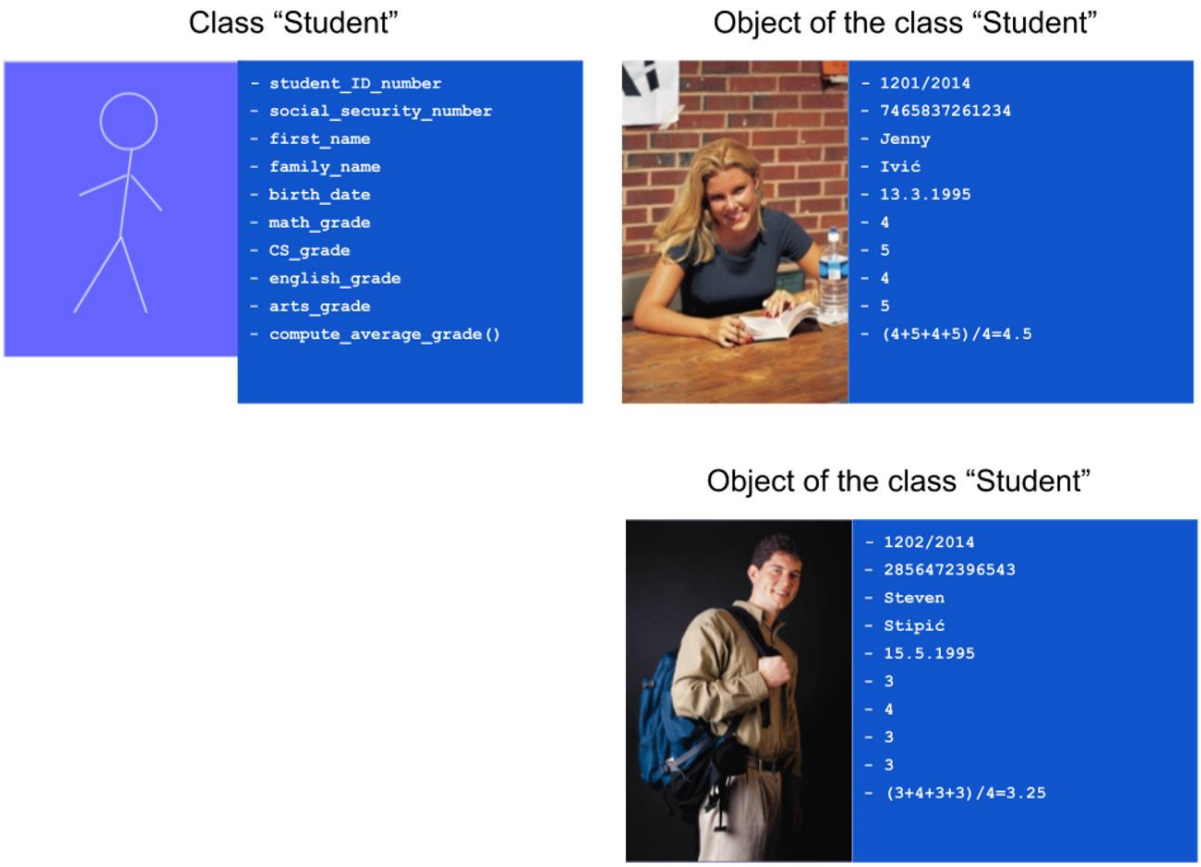


Slika 15. Objekt 2 – izvedba razreda Objekt 3 – izvedba razreda

1. Učitelj slušateljem pokaže načrt hiše.
2. Pozove jih, naj naredijo hiško iz lego kock.
3. Skupaj opazujejo, da so si vse hiše podobne, da pa se hkrati razlikujejo, npr v barvi strehe, sten ali vrat.

Primer 3: Lastnosti razredov in objektov

Naveden je seznam lastnosti in funkcij študentov (**razredne** značilnosti in metode) in veliko **objektov** – dejanski študenti in študentke.



Slika 16: Značilnosti razredov in objektov

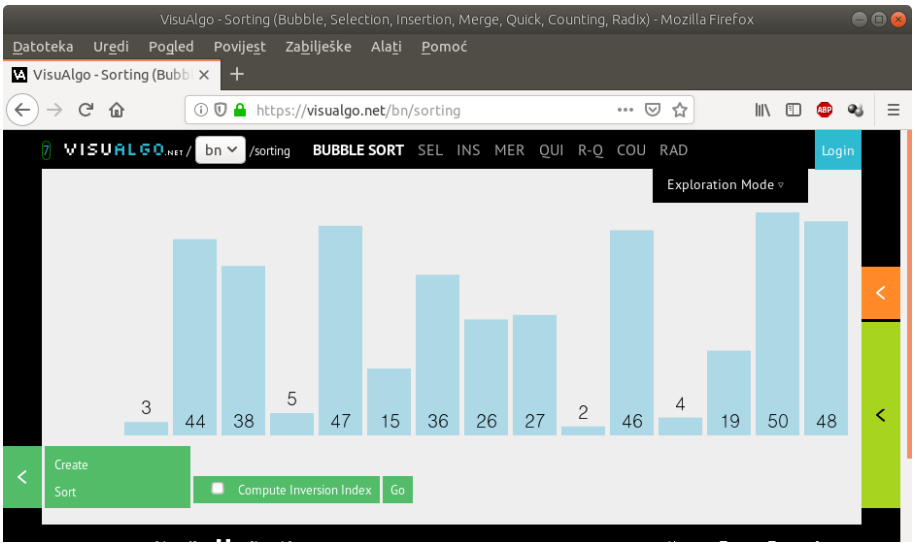
1. Učitelj pozove slušatelje, naj opišejo svoje skupne značilnosti ali lastnosti.
2. Slušatelji za opis samih sebe uporabijo iste značilnosti.

Primer: Vizualizacija algoritma

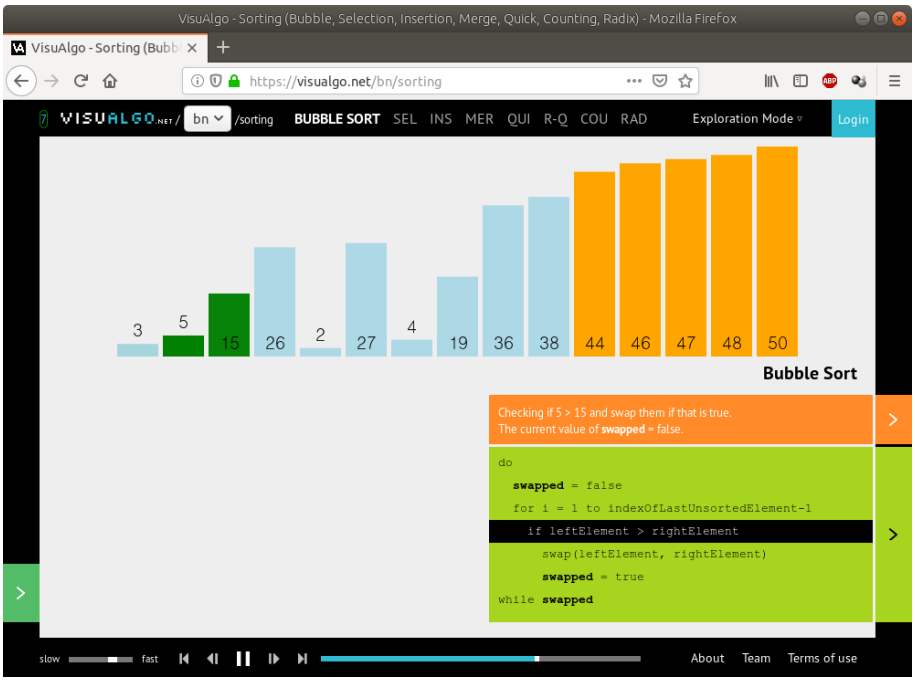
Animacija algoritma urejanja z mehurčki – https://visualgo.net/bn/sorting

V tem primeru se vizualizacija algoritma uporabi za razlago natančnega delovanja algoritma urejanja z mehurčki (Sliki 19. in 20.)

Vrednosti (stolpci) pred začetkom vizualizacije algoritma:



Slika 17: Pred začetkom vizualizacije algoritma



Slika 18: Vizualizacija algoritma med procesom urejanja

Oznaka enote: UČNA ENOTA (U2-L2)

NASLOV: Učno gradivo in orodja za poučevanje programiranja (10 % od 20 %)

Učni izidi

- Prepoznati skupne platforme, ki lahko olajšajo poučevanje programiranja.
- Najti prosto dostopne digitalne vire npr. e-textbooks/video/movie/cartoon itd., ki se lahko uporabijo za poučevanje programiranja.

Metodologija

Strategije poučevanja

☐ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
☐ Skupinsko delo	☐ Vprašanja in odgovori	☐ Projektno učenje
☐ Igranje vlog	☐ Kombinirano učenje	☐ Demonstracija

Struktura poglavja

Aktivnost 1: Sodelovalna orodja za učenje programiranja

Aktivnost 2: Prosto dostopno učno gradivo

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	☐ Medsebojno ocenjevanje	✕ Ustno preverjanje znanja
☐ Domača naloga	☐ Vzorčni izdelek	☐ Projektna aktivnost
☐ Predstavitev		☐ Drugo

AKTIVNOST 1: Sodelovalna orodja za učenje programiranja

Cilj aktivnosti je, da pomaga slušateljem iskati in najti nekaj skupnih orodij, ki podpirajo sinhrono in asinhrono komunikacijo v učenju programiranja.

IZVEDBA

Učitelj najprej opozori na pomembnost deljenja kod z drugimi pri kodiranju, denimo s prijatelji, sošolci ali z učitelji. Aktivnost se lahko začne s postavitvijo nekaj vprašanj o deljenju in sodelovanju.

Nato lahko učitelj povzame odgovore slušateljev ter jih razvrsti v tri kategorije. Prva kategorija je lahko »kodiranje s prijatelji«. Učitelj lahko to kategorijo razloži kot »najti primerno orodje za pisanje kod in njihovo deljenje s prijatelji in učitelji; čemur sledi skupni pregled programa in skupno iskanje rešitev.« Druga kategorija je »v kodiranju nihče ne mara (šolske) table«. To kategorijo lahko učitelj komentira z naslednjimi besedami »v kodiranju je uporaba tehnologije zelo pomembna – namesto da uporabljamo tradicionalen način poučevanja, ki je pokaži-in-naredi, lahko za podporo poučevanju in učenju uporabimo urejevalnik v realnem času.« Zadnja kategorija je lahko »poučujemo z orodji«, ki se jo lahko razloži kot: »Delite vaše kode s svojimi sošolci in prijatelji. Poskušajte jih učiti tako, da bodo delali napake pri kodiranju, tako da bodo takoj v realnem času videli rezultate kodiranja.«

Učitelj nato pozove slušatelje, da poiščejo najmanj pet orodij, ki omogočajo skupno sodelovanje med učenci in učitelji. Lahko se oblikujejo tudi kriteriji za izbiro orodij, tako da slušatelji lahko vidijo razliko med njimi. Kriteriji so lahko naslednji:

- prost dostop
- odprtokodni
- sinhrono/asinhrono
- na spletu/ni na spletu
- preprosta uporaba
- podporni video posnetki
- brskanje po dokumentaciji
- osebno testiranje
- podpora protokola za prenos datotek FTP/SFTP

Aktivnost se lahko zaključi z razpravo o orodjih, ki so jih slušatelji našli, in njihovimi mnenji o njih.

ZA RAZMISLEK

- Ali res moramo deliti svoje kode z drugimi?
- Ali meniš, da to pomaga razvijati boljše programiranje?
- Ali moramo delati skupaj?

VIRI

Capterra. (n. d.). *Collaboration Software*. Pridobljeno iz Capterra: https://www.capterra.com/collaboration-software/

Ciobanu, D. (24. 10 2018). *Best Tools for Code Collaboration.* Pridobljeno iz designmodo: https://designmodo.com/code-collaboration/

Eberman, A. R. (9. 4. 2018). *Collaborative Learning: Why Coding Is The Best Field For Kids To Learn From Each Other.* Pridobljeno iz Tekkie Uni: https://tekkieuni.com/blog/collaborative-code-learning/

Gaebel, D. (27. 2. 2018). *9 Real-Time Code Collaboration Tools for Developers.* Pridobljeno iz envatotuts+: https://webdesign.tutsplus.com/articles/real-time-code-collaboration-tools-for-developers--cms-30494

Haberer, T. (8. 11. 2017). *Why Learning to Code Should Be Collaborative.* Pridobljeno iz Atomicdust: https://www.atomicdust.com/learning-to-code/

Robogarden. (13. 06. 2018). *How coding improve collaboration and cooperation?* Pridobljeno iz Robogarden: https://robogarden.ca/blog/how-coding-improve-collaboration-and-cooperation

AKTIVNOST 2: Prosto dostopno učno gradivo

Cilj te aktivnosti je, da slušatelji najdejo prosto gradivo (npr. knjige, video, filme, risanke), ki se lahko uporabijo pri poučevanju kodiranja, ter da razpravljajo o rezultatih.

IZVEDBA

Učitelj najprej razloži, kako bi bilo precej preprosto najti zanimive brezplačne učbenike o kodiranju. Ti učbeniki so lahko tako za začetnike kot tudi za zelo izkušene programerje. Učitelj navede nekaj primerov iz teh učbenikov (gl. Tabelo spodaj). Nato se izbere enega izmed njih ter se ga skuša opisati na podlagi značilnosti knjige.

Učitelj lahko da slušateljem na voljo nekaj časa (dva ali tri dni), da poiščejo gradivo, ki bi bilo uporabno za učenje programiranja, in napišejo kratek sestavek o njem.

Tabela 3: Primeri brezplačnih učbenikov programiranja

Naslov knjige	Avtor/ji	Raven	Uporabnost
A Byte of Python	Swaroop C H	Raven težavnosti za bralce: 4 od 10	Dobro orodje za začetniško programiranje
Node.js Succinctly	Emanuella Delbono	Zahtevnejša	
97 Things Every Programmer Should Know		Začetniška	
Game Programming Patterns		Zahtevnejša	
Mastering iOS game Development		Zahtevnejša	

Primer: Učbenik

Poiščite učbenik “A Byte of Python” in razpravljajte o naslednjih vprašanjih:

- Ali je to brezplačen učbenik za programski jezik, ki uporablja jezik Python?
- Ali se ga lahko uporabi za vaje in priročnik za jezik Python?
- Ali je ta knjiga namenjena začetnikom?
- Ali ne potrebujemo predhodnjih informacij, preden začnemo uporabljati to knjigo?
- Ali je ta knjiga primerna za univerzitetno raven?

VIRI

Open Culture. (n. d.). *Free Textbooks: Computer Science.* Pridobljeno iz Open Culture: The best free cultural & educational media on the web: http://www.openculture.com/free-computer-science-textbooks

Packt. (n. d.). *Free programming ebooks and videos.* Pridobljeno iz Packt publishing: https://www.packtpub.com/packt/offers/free-learning

tutorialzine. (2018). *10 Free Programming Books You Should Read in 2018.* Pridobljeno iz tutorialzine: https://tutorialzine.com/2018/01/10-free-programming-books-you-should-read-in-2018

WebFX. (n. d.). *15 Free Books for People Who Code.* Pridobljeno iz WebFX Digital Marteking That DRivers Results: https://www.webfx.com/blog/web-design/free-books-code/

Oznaka enote: UČNA ENOTA (U3-L1)

NASLOV: Metode poučevanja programiranja (10 % od 30 %)

Učni izidi

- Razložiti pomembnost uporabe različnih strategij, metod in tehnik poučevanja
- Opisati različne tehnike poučevanja programiranja, osredotočene na učitelja.

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
✕ Skupinsko delo	✕ Vprašanja in odgovori	☐ Projektno učenje
☐ Skupinsko učenje	☐ Kombinirano učenje	☐ Demonstracija
☐ Igranje vlog		

Struktura poglavja

AKTIVNOST 1:Poučevanje, osredotočeno na učitelja

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	☐ Medvrstniško ocenjevanje	✕ Ustno preverjanje znanja
☐ Domača naloga	☐ Vzorčni izdelek	☐ Projektna aktivnost
☐ Predstavitev		☐ Drugo

AKTIVNOST Poučevanje, osredotočeno na učitelja

ZAHTEVANA ZNANJA

Osnovno znanje o:

- različnih tehnikah, osredotočenih na učitelja (metoda predavanja, vprašanja in odgovori, navajanje dejstev in vaja, demonstracija in vaja), in
- metodah, strategijah in tehnikah poučevanja.

TEORIJA

Pri poučevanju, ki v ospredje postavlja učitelja, so učenci osredotočeni predvsem na njega. Učitelj razlaga in učenci ga poslušajo ter odgovarjajo na njegova vprašanja.

Metodo »neposrednega poučevanja«, ki je ena izmed najbolj pogosto uporabljenih pri poučevanju, osredotočenem na učitelja, lahko definiramo kot »poučevanje, osredotočeno na učitelja, ki je popolnoma v rokah učitelja; vse izhaja iz učitelja, ki ima hkrati popolni nadzor nad potekom poučevanja ter visoka pričakovanja glede napredka pri učencih.«

Demonstracija in vaja: običajno za učno uro kodiranja učitelji najraje uporabljajo to metodo. Govorjenje učitelja IKT/STEM je lahko pri uporabi te tehnike dolgočasno ali pa zanimivo, odvisno od njegove priprave in načrtovanja. Zatorej bi po dolgem predavanju učitelj moral uporabiti nekaj motivacijskih strategij, ki bi pritegnile pozornost slušateljev. Uskladitev hitrosti učiteljevega predavanja in hitrosti, s katero mu lahko slušatelji sledijo, se mora doseči simultano, tako da slušatelji lahko uporabljajo primere, ki jih demostrira učitelj, tudi na svojih računalnikih. Poleg tega se mora učitel slišati v vsakem kotičku laboratorija/učilnice.

IZVEDBA

- Predstavitev
- Razprava
- Uporaba tabel za primerjave, na primer:

	Negativne strani	Pozitivne strani
Metoda učne ure		
Demonstracija in vaja		

OPOMBE

Večina študentov (bodočih učiteljev) si mogoče predstavlja, da je poučevanje, ki v ospredje postavlja učitelja, najlažji način izvajanja učnih ur. Vendar je treba opozoriti na to, da lahko med dolgimi predavanji o kodiranju nekateri učenci izgubijo zanimanje in motivacijo. Zato je dobro, da jim predstavite, se z njimi pogovorite in jih naučite strategij in tem, ki dvigujejo motivacijo in zainteresiranost učencev med dolgimi predavanji.

ZA RAZMISLEK

- Naštejte pomembne oblike pouka, osredotočenega na učitelja.
- Katere težave se pojavljajo pri poučevanju programiranja?
- Zakaj je nujno, da uporabljamo različne metode poučevanja?

- Ali je ena strategija dobra za vse situacije, ki nastanejo v razredu, ko poučujemo programiranje?
- Kaj lahko storimo, da premagamo težave, ki se pojavljajo pri poučevanju programiranja?
- Kako pogosto naj učitelj pri poučevanju uporablja metode, osredotočene na učitelja?
- Kateri so razlogi za in proti učenju, ki v ospredje postavlja učitelja?

VIRI

Brown, N. C., & Wilson, G. (2018). Ten quick tips for teaching programming. *PLoS Computational Biology, 14*(4).

Nuryan Issayan, L. (2011). *Teacher-Centered Vs Learner-Centered Approach: Teacher Behavior.* LAP Lambert Academic Publishing.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137–172.

Santrock, J. W. (2009). *Educational Psychology (4th edition).* New York: McGraw-Hill Education.

Oznaka enote: UČNA ENOTA (U3-L2)

NASLOV: Metode poučevanja programiranja (10 % od 30 %)

Učni izidi

- Opisati različne tehnike poučevanja programiranja, ki v ospredje postavljajo učence.
- Vključiti učenje na podlagi problema/projekta/igre v poučevanje programiranja.

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
☐ Skupinsko delo	✕ Vprašanja in odgovori	☐ Projektno učenje
☐ Skupinsko učenje	☐ Kombinirano učenje	☐ Demonstracija
☐ Igranje vlog		

Struktura poglavja

Aktivnost 1: Učenje, osredotočeno na učence

Aktivnost 2: Problemsko učenje

Aktivnost 3: Projektno učenje

Aktivnost 4: Učenje na podlagi igre

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	☐ Peer assessment	✕ Ustno preverjanje znanja
☐ Domača naloga	✕ Vzorčni izdelek	☐ Projektna aktivnost
☐ Predstavitev	☐ Homework	☐ Drugo
☐ Short Paper		

AKTIVNOST 1: Učenje, osredotočeno na učence

ZAHTEVANA ZNANJA

Osnovno znanje o tehnikah poučevanja, osredotočenega na učence (učenje na podlagi problema, projektno učenje, učenje na podlagi primerov).

TEORIJA

Fokus poučevanja se prestavi z učitelja na učence. Pri pouku programiranja to pomeni, da ima vsak učenec na voljo napravo, na kateri dela. Učenci se vključijo v projekt in morajo sodelovati, da ga dokončajo; skupine niso strogo razdeljene v prostoru. Učenci se učijo v skladu s svojimi zmožnostmi.

IZVEDBA

- Razprava
  - Spomnite se svojih šolskih dni. Poskušajte se spomniti na tiste učitelje, ki ste jih imeli za dobre učitelje. Kako je ime prvemu, ki ste se ga spomnili? Kaj ga je delalo dobrega učitelja?
- Predstavitev

ZA RAZMISLEK

- Zakaj je pomembno, da uporabljamo poučevanje, osredotočeno na učence?
- Kako dolgo naj bi trajal pouk, osredotočen na učence?
- Kateri so negativni in kateri pozitivni vidiki poučevanja, ki v osredje postavlja učence?
- Kaj menite o učinkovitosti poučevanja, osredotočenega na učence?
- Kako lahko izvajamo ovrednoteje pri poučevanju, osredotočenem na učence?

VIRI

Biggs, J. (2006). What the Student Does: teaching for enhanced learning. *Higher Education Research & Development*, 57-75.

Doyle, T. (2011). *Learner-Centered Teaching: Putting the Research on Learning into Practice.* Sterling, Virginia: Stylus Publishing.

Guzdial, M. (2016). *Learner-Centered Design of Computing Education: Research on Computing for Everyone.* Morgan & Claypool.

Weimer, M. (2002). *Learner-Centered Teaching.* New York: Jossey-Bass John Wiley & Sons.

AKTIVNOST 2: Problemsko učenje

TEORIJA

Vprašajte: »Kaj je to – problem?« Po določitvi tega, kaj je problem, naj vam slušatelji navedejo nekaj problemov iz vsakdanjega življenja. Navedite nekaj primerov, kot so: odpiranje vrat za izhod iz sobe, popravilo pipe, ki pušča, kako priti od letališča do hotela, menjava izdelka, kupljenega preko spleta ipd.

Slušateljem razložite nekaj kriterijev, ki se lahko upoštevajo pri izbiri in oblikovanju problemov za srednješolce. Dva pomembna dejavnika, ki vplivata na določitev problema, sta lahko: 1) preučiti predznanje dijakov, 2) določitev konceptov, ki jih želite naučiti (zanke, tabele, funkcija, spremenljivka itd.).

Na tej točki aktivnosti je zelo pomembno, da poudarimo pomen predzanja učencev o programiranju. Njihovo predznanje lahko namreč vpliva na njihovo pozornost in zadovoljstvo z učno uro. Če smo učencem predstavili preveč kompleksen problem, bodo mogoče izgubili zanimanje zanj. Če pa je problem preveč preprost, se bodo kratkočasili s kakšnim drugim početjem. Podajte primer »zapiranja zamaška« kot primer preprostega problema. Procedura je 1) primite za zgornji del zamaška, 2) obrnite zamašek v smeri urinega kazalca. Ta problem je linearen in preprost. Podajte primer kompleksnega problema: kako najhitreje in hkrati najceneje priti od šole do centra mesta. To je kompleksen primer, ki vključuje nekaj pogojev.

Za usmeritev pozornosti na dejstvo, da so programski koncepti za učence abstraktni, lahko predstavite nekaj vrst konceptov in metod poučevanja konceptov. Učenci morajo vedeti, da morajo učitelji IKT/STEM uporabljati analogije, da si bodo lahko učenci koncepte realno predstavljali. Pogovorite se o vrstah konceptov (nadrejeni/podrejeni/enakovredni koncept) in o konceptih metod poučevanja (induktivni in deduktivni pristop poučevanja).

IZVEDBA

- Predstavitev
- Skupinsko delo
- Primer delovnega lista

ZA RAZMISLEK

- Ali problemsko učenje izboljša sposobnost reševanja problemov?
- Kako lahko merimo in vrednotimo v problemskem učenju?
- Kaj naj bi vključevalo dobro okolje za problemsko učenje?

VIRI

Shimic, G., & Jevremovic, A. (2010). Problem-based learning in formal and informal learning environments. *Interactive Learning Environments, 20*(4), 351-367.

AKTIVNOST 3: Projektno učenje

TEORIJA

Projektno učenje je metoda počevanja, s katero učenci pridobijo znanje in veščine tako, da daljše obdobje delajo na kompleksnem vprašanju, problemu ali izzivu, ki ga preučujejo in se nanj odzivajo.

IZVEDBA

- Predstavitev
- Razprava
    - Začnite z vprašanjem: »Kakšne so razlike med problemskim učenjem in projektnim učenjem?« Opišite projektno učenje in slušateljem razložite proces razvoja projekta.
    - Slušatelji lahko povedo/ustvarijo svoje lastne predloge projektov za učence. Nato navedite nekaj primerov projektov, ki se lahko uporabijo za tovrstno delo.
    - Na primer: učitelj IKT/STEM želi od svojih učencev projekt v programskem jeziku Scratch. Cilj tega projekta je razvitje igre s težavnostnimi ravnmi in točkami. Učenci morajo uporabiti vse svoje znanje, ki so ga pridobili tekom šolskega leta.

ZA RAZMISLEK

- Kateri so pozitivni in kateri negativni vidiki projektnega učenja?
- Kakšna je razlika med problemskim učenjem in projektnim učenjem?
- Kako ovrednotiti projekte po zaključku projektnega dela?

VIRI

Bell, S. (2010). Project-Based Learning for the 21st Century: Skills for the Future. *The Clearing House, 83*(2), 39-43.

Blumenfeld, P., Soloway, E. M., Krajcik, J. S., Guzdial, M., & Palincsar, A. (2011). Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. *Educational Psychologist, 26*(3), 369-398. Pridobljeno iz https://www.researchgate.net/profile/Yael_Seker/post/How_do_you_approach_the_design_of_a_group_task_in_higher_education/attachment/59d623d979197b8077982283/AS%3A309004255858690%401450683763438/download/Blumenfeld+et+al_Motivating_project_based_learning.pd

Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools. *Computers & Education, 97*, 129-141.

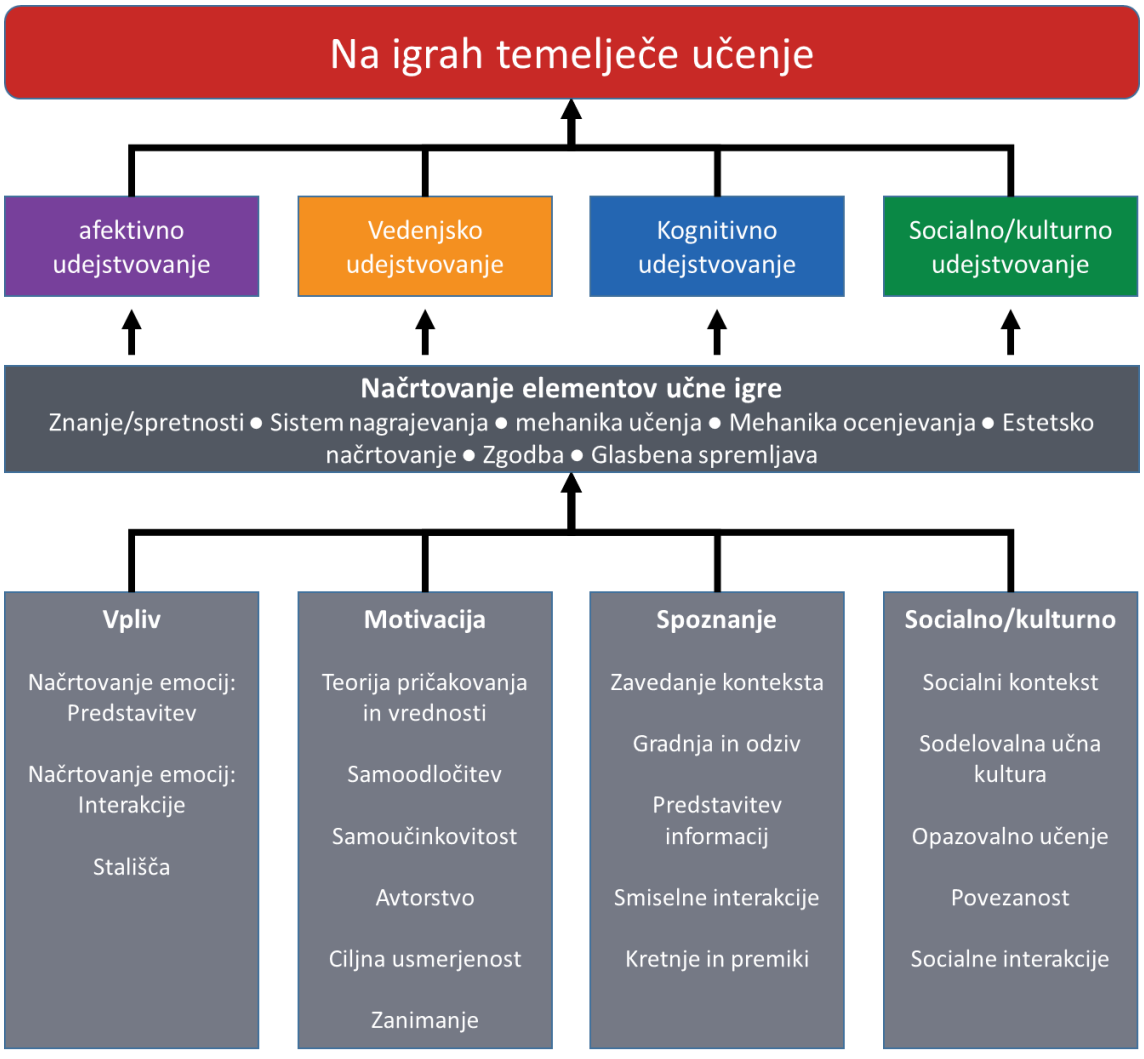
AKTIVNOST 4: Učenje na podlagi igre

TEORIJA

Učenje na podlagi igre je pristop poučevanja, kjer učenci raziskujejo bistvene vidike igre v kontekstu učenja, ki ga ustvarijo učitelji. Učitelji in učenci sodelujejo z namenom, da povečajo globino in razširijo poglede na izkušnjo igranja igre.

Dobre aplikacije učenja na podlagi igre nas lahko popeljejo v virtualna okolja, ki se zdijo znana in ki jih občutimo kot domača in relevantna.

Prva prioriteta učenja na polagi igre ni narava igre same. Prioriteta je, da si olajšamo učenje. Kot vemo, lahko z virtualnimi okolji zelo dobro zapolnimo prostor med učno izkušnjo in službami iz realnega sveta, zato je učenje na podlagi igre pogosto uporabljeno kot motivacijsko orodje. Temelje učenja na podlagi igre lahko razložimo na naslednji način, s katerim opišemo različne vrste vključenosti.



Slika 19: Ogrodje integriranega dizajna igrivega učenja na podlagi igre[3]

[3] source: https://files.eric.ed.gov/fulltext/EJ1090277.pdf

V igrifikaciji se uporablja zgolj nekaj elementov igre. Učenci ne igrajo igre od začetka do konca; sodelujejo v prizorih, ki vključujejo elemente video ali mobilnih igric, kot so osvajanje točk, premagovanje ovir ali pridobivanje značk, da uspešno zaključijo nalogo.

Učenci običajno ne ločijo med igrifikacijo in učenjem na podlagi igre, zato lahko učitelj pojasni razliko med igrifikacijo in učenjem na podlagi igre. Lahko poudari njune glavne značilnosti, tako da bo razlika bolj jasna. Poleg tega lahko navede nekaj specifičnih primerov za igrifikacijo in učenje na podlagi igre.

IZVEDBA

- Predstava
- Razprava

ZA RAZMISLEK

- Kakšne so značilnosti igralnega okolja v vsakdanjem življenju?
- Ali lahko prenesemo elemente igre iz vsakdanjega življenja v učno okolje?
- Kaj menite, kateri elementi bi bili primerni za presojo učenja na podlagi igre?
- Ali bi lahko vsako učno uro izvedli na podlagi igre?

VIRI

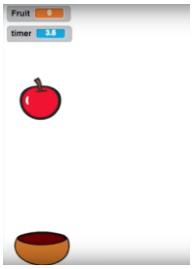
Wang, L.-C., & Chen, M.-P. (2010). The effects of game strategy and preference-matching on flow experience and programming performance in game-based learning. *Innovations in Education and Teaching International, 47*(1), 39-52.

Wilson, A., Hainey, T., & Connolly, T. (2013). Using Scratch with Primary School Children: An Evaluation of Games Constructed to Gauge Understanding of Programming Concepts. *International Journal of Game-Based Learning, 3*, 93-109.

Primer 1: DELOVNI LIST – Problemsko učenje

1. Preučite naslednji primer problemskega učenja.
2. Razvijte svoj lasten primer problemskega učenja, ki temelji na tem delovnem listu
   - Seznanite se s predznanjem slušateljev.
   - Napišite nove koncepte, ki se jih bodo naučili.
   - Določite problem.
   - Predlagajte rešitev.

IGRA UJEMI JABOLKO
V PROG. JEZIKU SCRATCH



Predhodno znanje
Učenci morajo obvladati gibanje, pokazati, skriti, bloke spremenljivih dolžin v jeziku Scratch.

Koncepti
Operatorski bloki v programskem jeziku Scratch
“>”, “<” in bloki “izberi naključno 1 do 10”.

Problem
Stopnjevanje težavnosti v igri »Ujemi jabolka v košaro«.
Jabolka padajo na naključno mesto na ekranu. Naloga igralca je, da skuša ujeti jabolka v košaro, preden padejo na tla.

Rešitev
Učenec ki bo kodiral, mora dodati bloke »spremenljivke gibanje, pokazati, skriti« v določenem logičnem zaporedju, ki bo ponazarjalo gibanja padanja jabolk. Za prikaz položaja jabolk na zaslonu mora učenec uporabiti operatorja »>« in »<«. Po tem, ko se jabolko dotakne košare, se mora jabolko skriti, število točk pa se povečati za 10. Ko se jabolko dotakne košare, mora rezultat >70 spremeniti raven.

Primer 2: DELOVNI LIST – primer učenja na podlagi igre

1. Preučite naslednji primer učenja na podlagi igre.
2. Razvijte svoj lasten primer učenja na podlagi igre, ki temelji na tem delovnem listu.

IZBERI KARTE/VZAMI KARTE

LOGIČNE OPERACIJE

Predhodno znanje
Pred to igro je treba poznati pomen logičnih operatorjev.

Cilj
Poučevanje »logični operaterji; in, ali, ne, večji, manjši«

Orodja
Orodja: karte, krog, štoparica, svinčniki 3 različnih dolžin

Karte
Na vsako od kart napišite »vzami najdaljšo in najkrajšo«, »vzami najkrajšo in ne najdaljšo«, vzami najdaljšo ali najkrajšo« itd.

Naloga
Slušatelji izberejo karto, ki kaže hrbtno stran. Takoj, ko preberejo, kaj piše na njej, morajo to nalogo tudi izvesti.

Oznaka enote: UČNA ENOTA (U3-L3)

NASLOV: Metode poučevanja programiranja (10 % od 30 %)

Učni izidi

- Primerjati prednosti in pomanjkljivosti poučevanja programiranja, osredotočenega na učitelja, s tistim, ki je osredotočeno na učence.
- Izbrati primerne metode, kako pritegniti učence k programiranju.

Metodologija

Strategije poučevanja

☐ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
✕ Skupinsko delo	✕ Vprašanja in odgovori	☐ Projektno učenje
☐ Skupinsko učenje	☐ Kombinirano učenje	☐ Demonstracija
☐ Igranje vlog		

Struktura poglavja

AKTIVNOST 1: Prednosti in pomanjkljivosti poučevanja programiranja, osredotočenega na učitelja, in poučevanja, osredotočenega na učence

AKTIVNOST 2: Izbira metode poučevanja

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	✕ Peer assessment	✕ Ustno preverjanje znanja
☐ Domača naloga	☐ Vzorčni izdelek	☐ Projektna aktivnost
✕ Predstavitev		☐ Drugo
☐ Kratko poročilo		

AKTIVNOST 1: Prednosti in pomanjkljivosti poučevanja programiranja, osredotočenega na učitelja, in poučevanja, osredotočenega na učence

Ta aktivnost se osredotoča na primerjavo med poučevanjem, osredotočenim na učitelja, in poučevanjem, osredotočenim na učence, ki sta bili opisani v predhodnih učnih urah (U3L1 in U3L2).

IZVEDBA

- Skupinsko delo
- Razprava
- Uporaba analize prednosti in pomanjkljivosti ter priložnosti in nevarnosti (SWOT) ali tabele za primerjavo obeh pristopov (gl. delovni list 1)
- Primerjava med poučevanjem, osredotočenim na učence, in tistim, osredotočenim na učitelja (gl. delovni list 2)

ZA RAZMISLEK

- Ali so bolj učinkovite metode poučevanja, osredotočenega na učitelja ali na učence?
- Kateri metodi bi dali prednost vi, če bi bili učitelji?

VIRI

Concordia University Portland Oregon. (2012 (2018)). *Which is Best: Teacher-Centered or Student-Centered Education?* Retrieved from A blog by Concordia University - Portland: https://education.cu-portland.edu/blog/classroom-resources/which-is-best-teacher-centered-or-student-centered-education/

Khaled, A. (2013). Teacher-Centered Versus Learner-Centered Teaching Style. *The Journal of Global Business Management, 9*(1), 22-34. Retrieved from http://www.jgbm.org/page/3%20Ahmed%20Khaled%20Ahmed.pdf

Primer 1: DELOVNI LIST 1 Analiza SWOT

Primerjava med poučevanjem, osredotočenim na učence, in poučevanjem, osredotočenim na učitelja.

1. Izpolnite delovni list analize prednosti in pomanjkljivosti ter priložnosti in nevarnosti (SWOT) poučevanja programiranja, osredotočenega na učence.
2. Izpolnite drugi delovni list analize poučevanja programiranja, osredotočenega na učitelja.
3. Primerjajte in razpravljajte o rezultatih.

Notranji dejavniki za učitelja oz. znotraj šole	Prednosti (Katere so prednosti/vrednote?)	Pomanjkljivosti (Katere so možne kritike?)
Zunanji dejavniki, ki izhajajo iz skupnosti	Priložnosti (Kakšen potencial širitve imajo metode poučevanja kodiranja, osredotočene na učence, glede na spremembe v poučevanju/skupnosti/tehnologiji?)	Nevarnosti (Kateri so zunanji problemi oz. ovire, ki so posledica sprememb v poučevanju/ skupnosti/tehnologiji?)

Primer 2: DELOVNI LIST 2

Primerjava med poučevanjem, osredotočenim na učence, in tistim, osredotočenim na učitelja:

1. Navedite pozitivne in negativne vidike pouka programiranja, osredotočenega na učence.
2. Navedite pozitivne in negativne vidike pouka programiranja, osredotočenega na učitelja.
3. Primerjajte prednosti in pomanjkljivosti pouka, osredotočenega na učence, in pouka, osredotočenega na učitelje.

Tabela 4: Primerjalna tabela med poučevanjem, osredotočenim na učence, in poučevanjem, osredotočenim na učitelja

Pouk, osredotočen na učence:		Pouk, osredotočen na učitelja	
Pozitivni vidiki	**Negativni vidiki**	**Pozitivni vidiki**	**Negativni vidiki**
Prednosti in pomanjkljivosti pouka, osredotočenega na učence		**Prednosti in pomanjkljivosti pouka, osredotočenega na učitelja**	

AKTIVNOST 2: Izbira metode poučevanja

TEORIJA

Mishra in Koehler (2006) v svoji knjigi o poznavanju vsebine, tehnologije in pedagogike *Technological pedagogical content knowledge (TPACK).* Poudarjata, da sta vsebina (to, kar učite) in pedagogika (kako učite) temelj za katero koli tehnologijo, ki jo želite uporabiti pri poučevanju. http://www.rt3nc.org/edtech/the-tpack-model/

IZVEDBA

- Razpravljajte o modelu TPACK in izpostavite pomembnost pedagoškega elementa v teoretskem okviru TPACK.
- Skupinsko delo
- Razprava
- Študija primera metode poučevanja (YouTube), ki mu sledi analiza učne ure.

ZA RAZMISLEK

- Kaj vpliva na učiteljevo kvaliteto poučevanja v razredu? Znanje, povezano z vsebino; pedagoško znanje ali tehnološko znanje?
- Učitelj bi moral v vseh razredih poučevati z isto učno metodo – razprava?
- Katere so prednosti tega, da učence poučujemo na različne načine?
- Katerim metodam boste vi dali prednost, ko boste postali učitelji?

VIRI

Geddis, A. N. (2006). Transforming subject-matter knowledge: the role of pedagogical content knowledge in learning to reflect on teaching. *International Journal of Science Education, 15*(6), 673-683.

Mishra, P., & Koehler, M. J. (2006). Technological Pedagogical Content Knowledge: A Framework for Teacher Knowledge. *Teachers College Record, 108*(6), 1017-1054.

Taylor, E., Breed, M., Hauman, I., & Homann, A. (2013). Choosing learning methods suitable for teaching and learning in computer science. *IADIS International Conference e-Learning* (str. 74-82). Prague, Czech Republi: IADIS - International association for development of the information society.

Vitkutė Adžgauskienė, D., & Vidžiūnas, A. (2012). Problems in Choosing Tools and Methods for Teaching Programming. *Informatics in Education, 11*(2), 271–282.

Oznaka enote: UČNA ENOTA (U4-L1)

NASLOV: Ocenjevanje pri pouku programiranja (7,5 % od 15 %)

Učni izidi

1. Opisati metode ocenjevanja pri pouku programiranja .
2. Uporabiti različne programe, ki lahko podpirajo ocenjevanje (npr. Kahoot, Socrative in drugi pripomočki).
3. Preverjanje plagiatorstva v okviru pouka programiranja .

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Diskusija	☐ Vaje in utrjevanje
✕ Skupinsko delo	✕ Vprašanja in odgovori	☐ Projektno učenje
☐ Sodelovalno učenje	☐ Kombinirano učenje	☐ Demonstracija
☐ Igranje vlog		

Struktura poglavja

Aktivnost 1: Metode ocenjevanja

Aktivnost 2: Avtorske pravice in licence

Aktivnost 2: Plagiatorstvo pri pouku programiranja

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravil	☐ Pisno preverjanje
✕ Kontrolni seznam/ocenjevalna lestvica	✕ Medsebojno ocenjevanje	✕ Ustni zagovor
☐ Domače delo	☐ Vzorčni izdelek	☐ Projektna aktivnost
☐ Predstavitev		☐ Drugo

AKTIVNOST 1: Metode ocenjevanja

Namen te aktivnosti je seznaniti udeležence, da razmislijo o ocenitviprogramiranja .

ZAHTEVANA ZNANJA

Ljudje ne razumejo, kaj je programska oprema. Na splošno jo obravnavajo kot programe. Vendar programska oprema poleg programov potrebuje tudi dokumentacijo za nadgradnjo in vzdrževanje programa med življenjskim ciklom. Inženirska disciplina, ki zajema vse vidike načrtovanja in razvoja programske opreme, se imenuje programski inženiring. (Sommerville, 2011).

Programski inženirji razvijajo programsko opremo z uporabo modelov za razvoj programske opreme. Eden najstarejših razvojnih modelov je model »waterfall«, ki je sestavljen iz naslednjih faz: specifikacija zahtev, načrtovanje, izvajanje, testiranje in vzdrževanje programske opreme. Te faze bi lahko priredili na razvoj programa v izobraževanju. Specifikacija zahtev je opis naloge (problem), ki ga poda učitelj. Oblikovanje programske opreme opredeljuje temeljno idejo rešitve, npr. algoritem in je običajno zapisana v obliki diagrama. Izvajanje pomeni pisanje programske kode na podlagi danega algoritma. Testiranje zajema simulacijo izvajanja programa ali njegovo dejansko izvedbo – preverjanje, ali program izpolnjuje zahteve, npr. daje pravilen izhod na dani vhod. Če program ne uspe, npr. ima napake, se je potrebno vrniti v fazo izvajanja in najdene napake odpraviti.

Vzdrževanje programske opreme ni samo »popravljanje napak«. Gre za spremembo računalniške kode po dostavi programske opreme in jo lahko izvedemo na več različnih načinov: korektivno, prilagodljivo, popolno ali preventivno (McCormack, 2005). Vzdrževanje pogosto ni vključeno v izobraževanje programiranja na nižjih nivojih.

Preizkušanje programov bi bilo mogoče izvesti z ročnim sledenjem programski kodi (na papirju) ali z razhroščevanjem določene rešitve v interaktivnih orodjih za odpravljanje napak. Ročno sledenje dani rešitvi pomeni, da je treba začeti od začetka programa in slediti spremembam spremenljivk. Ko pride do spremembe spremenljivke, se sprememba zabeleži na papirju. Po ukazih v programu se oblikuje seznam vrednosti spremenljivk v programu in izhodu na koncu. Če je izhod enak želenemu izhodu, program nima napak. Po drugi strani pa je napaka ugotovljena, zato je treba analizirati, zakaj se napaka pojavlja v procesu in poiskati napačen ukaz v programu.

Odpravljanje napak (večinoma v integriranem razvojnem okolju – IDE) ponuja številne možnosti za poglobljeno analizo izvajanja programa. Deluje tako kot pri ročnem razhroščevanju, vendar ga opravi stroj. Možno je, da zaženete program ukaz za ukazom, si ogledate vrednosti za izbrano spremenljivko, nastavite prekinitvene točke na določenih ukazih v programu, kjer se izvajanje programa zaustavi, ko izvajanje programa doseže ta ukaz.

Ocena kodiranja je proces, pri katerem avtorje programske kode ocenjujejo glede na njihove kompetence pri izdelavi programske kode. Na splošno ta proces uporabljamo za najemanje pravih posameznikov za namene kodiranja. Proces zagotavlja tudi vpogled v to, kaj je testiran posameznik sposoben narediti in kaj se mora še naučiti. Pri izobraževanju se ocena kodiranja ujema z oceno, ki jo študent dobi za svojo nalogo. (Bienkowski, Snow, Rutstein, & Grover, 2015)

Ocena kode je sestavljena iz štirih vidikov:

- **Natančnost** (**pravilnost**) (sape, n. d.) Pomeni, da računalniška koda deluje pravilno v vsakem ponavljanju in za kateri koli pričakovani vnos.
- **Učinkovitost** (technopedia, n. d.) je povezana z izvajanjem računalniške kode. Težave je mogoče rešiti na različne načine in nekateri pristopi so učinkovitejši (hitrejši in/ali porabijo manj pomnilniškega prostora) od drugih.
- **Robustnost** (Bishop, 1998-2002) je povezana z obravnavo posebnih primerov (mejni primeri in nepričakovani vnosi). Računalniški program se ne sme prekiniti zaradi nepričakovanega stanja. Program se preneha izvajati samo zaradi ukaza za prenehanje delovanja.
- **Jasnost** je povezana s tem, kako dobro je organizirana in razumljiva izvorna koda.

Programsko inženirstvo uporablja dva izraza – funkcionalne in nefunkcionalne zahteve. Funkcionalne zahteve so zelo blizu natančnosti. Nefunkcionalne zahteve imajo širši pogled na programsko opremo. Imajo nabor metrik (hitrost, velikost, enostavnost uporabe, zanesljivost, robustnost, prenosljivost) (Sommerville, 2011). Te zahteve se skladajo s standardi kakovosti ISO / IEC FCD 25010, kjer je navedenih 8 glavnih kategorij (funkcionalna primernost, učinkovitost delovanja, združljivost, uporabnost, zanesljivost, varnost, vzdržljivost, prenosljivost).

TEORIJA

Ocenjevanje je ena od nalog, ki jih učitelji opravljajo v učilnicah. Eden od ciljev ocenjevanja je podati pravilno oceno, a to ni edini namen. Ocenjevanje se uporablja za:

- pomoč učiteljem, da izboljšajo svoje razumevanje trenutnega znanja svojih učencev
- podajanje povratne informacije učencem o njihovem razumevanju ocenjevane teme.

Ocenjevanje lahko služi kot refleksija za učitelja in učence/študente glede poučevanja in učenja v programiranju.

Učitelj lahko oceni študentsko delo z različnimi tipi nalog (projektna dokumentacija, laboratorijske vaje, teoretična vprašanja itd.). V tej dejavnosti je poudarek na ocenjevanju programiranja.

IZVEDBA

Namen te aktivnosti je, da udeleženci začnejo razmišljati o ocenjevanju kodiranja.

Razpravljajte o različnih vrstah vprašanj s področja izobraževanja (*razvoj rešitve, sledenje dani rešitvi/razhroščevanje dane rešitve, najti cilj dane rešitve, dokončanje dane rešitve, itd.*)

- **Razložite izvedbo programske kode**
  Pripravite primer programske kode in želeni rezultat. Koda naj ne bo dokončana, temveč blizu zaključka. Uporabite skupno orodje (kot so Kahoot, Socrative …), da pridobite povratne informacije od učencev, kako nadaljevati s kodiranjem.
- **Ocenite programsko kodo**
  Priskrbite program; zaženite ga; postavljajte vprašanja (kaj, če …). lahko odgovarjajo,

kje v kodi kaj spremeniti. Učenci bi morali videti razliko med dobro in slabo zastavljeno kodo => neposredni vpliv za olajšanje popravkov.

- **Razpravljajte o ocenjevanju programa (nesprejemljiv, delno sprejemljiv ali sprejemljiv)**
  Priprava seznama značilnosti za ocenjevanje programske kode, glasujte o značilnostih in povežite z danim primerom.
- **Razpravljajte o mejnih primerih**
  Razpravljajte o številu krožnih napak; razpravljajte o problemu mejnih primerov; pretvarjanje tipov; težave z dinamičnimi strukturami podatkov in sprostitvijo pomnilnika (Zakaj pride do izgubljanja pomnilnika?), čiščenje pomnilnika in njegov vpliv na sisteme realnega časa.
- **Dopolnite dani program**
  Predstavite problem in naredite program nedokončan. Učenec mora program dokončati na tak način, da reši problem.
- **Pravilnost danega programa**
  Učenci morajo določiti, ali dani program na pravilen način rešuje dani problem.
- **Najti cilj programa**
  Učencem se predstavi programska koda, oni pa morajo ugotoviti, kateri problem rešuje.
- **Sledenje programski kodi**
  Predstavi se koda, učenci pa jo morajo analizirati (izslediti njene spremenljivke itd.)

Primer 1: Metoda ocenjevanja – urejanje z mehurčki (izgradnja programa)

Za namen razlage se lahko uporabi (neučinkovit primer) algoritma »mehurčnega urejanja« za ureditev celih števil. Postopek se začne z branjem številk, ki se jih uredi, in nato se urejene številke pokaže.

Algoritem: Za ročno izvedbo lahko uporabimo polstrukturirano programsko kodo, ki se jo lahko prenese v kateri koli računalniški jezik.

```
i = 1; //števec števil za sortiranje
// bere števila po vrsti in jih vstavlja v listo
While new_number do
 Read new_number into number_list[i];
 Increment i;
EndWhile

Numbers = i-1; // zakaj?
// sortiramo števila
For i = 1 to numbers do
 For j = 1 to numbers do
 If number_list[i]>number_list[j] then swap_numbers;
 EndFor
EndFor

// izpišemo sortirana števila
Print number_list;
```

Postopno izboljšujte kodo tako, da boste prišli do kodiranja v enem izmed programskih jezikov.

```
#include <iostream>
using namespace std;

int number_list[20], max_size; //za začetek postavimo, da lahko sortiramo le 20 števil

// preberemo števila
int read_list()
{
  int i = 0;
  int number = 1;

  while(number > 0) {
    cout << "Input number (0 for end)";
    cin >> number;
    number_list[i] = number;
    i++;
  } //end while
  return i-1;
}

// sortiramo števila
void bubble_sort(int size)
{
 int i, j, temp;
 for(i = size; i> 0; i--){
   for(j = 0; j<i; j++){
     if (number_list[j]>number_list[j+1]){
       temp = number_list[j];
       number_list[j] = number_list[j+1];
       number_list[j+1] = temp;
     }
   } //end for j
 } //end for i
} //buble_sort

// izpišemo števila
void print_list(int size)
{
  int i;
  for(i=0; i<=size; i++) {
    cout<<number_list[i]<<" ";
  }
  cout<<"\n";
}

// glavni program
int main()
{
  max_size = read_list();
  print_list(max_size);
  bubble_sort(max_size);
  print_list(max_size);
  return 0;
}
```

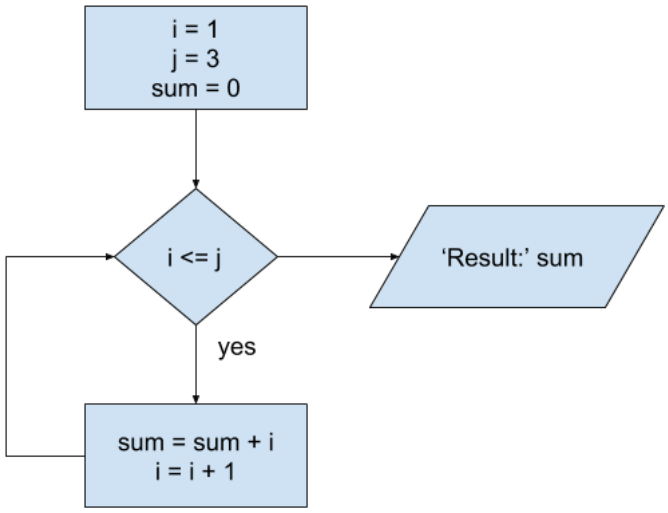
Vnesite kodo v računalnik, jo prevedite in izvedite z razhroščevanjem.

ZA RAZMISLEK

- Kakšen bi bil izhod za vhod 3 2 1?
- Kaj bi se zgodilo, če bi vnesli “a”?
- Kaj bi se zgodilo, če bi vnesli število 1234567890?
- Razpravljajte o tem, kako lahko preizkušamo kodo za kateri koli dani vhod.
- Katere vhode moramo preizkusiti? (nedovoljeni vhod (ukaz v C-ju, niz, realen), dovoljeni vhod (števila; razprava -max_int in max_int), mejni primeri).

Primer 2: Metoda ocenjevanja – diagram poteka

Razložite namen programa:



Slike 20: Diagram poteka preprostega algoritma

Program Python:

```
i = 1
j = 3
sum = 0
while(i <= j):
   sum = sum + i
   i = i + 1
print("Result: ", sum)
```

ZA RAZMISLEK

- Kakšen je namen programa?
- Prikaži vrednost spremenljivke sum.
- Izračunajte vsoto celih števil v obsegu od 1 do 3 in jo na koncu pokažite.
- Primerjajte vrednosti spremenjivk i in j.

VIRI

Bienkowski, M., Snow, E., Rutstein, D., & Grover, S. (december 2015). *Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science: A First Look.* Pridobljeno iz Principled Assessment of Computational Thinking: https://pact.sri.com/downloads/Assessment-Design-Patterns-for-Computational%20Thinking-Practices-Secondary-Computer-Science.pdf

Bishop, M. (1998-2002). *Robust Programming.* Pridobljeno iz Matt Bishop - Department of Computer Science - University of California, Davis: http://nob.cs.ucdavis.edu/bishop/secprog/robust.html

McCormack, J. (2005). *Topic 25: Software Maintenance and Re-engineering.* Pridobljeno iz CSE2305 Object-Oriented Software Engineering: http://users.monash.edu/~jonmc/CSE2305/Topics/13.25.SWEng4/html/text.html

sape. (n. d.). *Accuracy.* Pridobljeno iz sape - Software and Programmer Efficiency Research Group: http://sape.inf.usi.ch/accuracy

Sommerville, I. (2011). *Software Engineering.* Boston: Addison-Wesley.

technopedia. (n. d.). *Code Efficiency.* Pridobljeno iz technopedia: https://www.techopedia.com/definition/27151/code-efficiency

AKTIVNOST 2: Avtorske pravice in licence

Ta aktivnost se nanaša na avtorske pravice in različne licence programske opreme.

TEORIJA

Zakaj licenca?

Ko nekdo ustvari programsko opremo, nastane intelektualna lastnina (programska oprema in dokumentacija), ki je zaščitena z avtorskim pravom prav tako kot literarno ali umetniško delo.

Avtorska pravica za ustvarjeno delo, bodisi knjigo ali programsko opremo, pomeni, da se lastniki – običajno avtorji originala ali njihovi delodajalci – odločijo, kdo lahko to delo kopira, prilagaja ali deli in pod kakšnimi pogoji. Samo avtor je tisti, ki odloča o tem. Vsak, ki kopira, spreminja ali deli delo nekoga drugega brez avtorjevega dovoljenja, je v prekršku.

Tako torej zgolj licenca zagotavlja dovoljenje za uporabo, kopiranje, spreminjanje ali deljenje programske opreme.

Običajno avtor/lastnik programske opreme vključi opozorilo glede pravnega statusa za potencialne uporabnike programa. Uporaba programa lahko poteka na ravni izvorne ali objektne kode. Pravni status je običajno znan pod pojmom licenca za programsko opremo. Obstajata dve vrsti licenc: prosto dostopna programska licenca in odprtnokodna licenca.

Licenca za programsko opremo je lahko ena izmed naslednjih: GNU General Public License (GPL), BSD license, MIT license, Apache license, Eclipse Public License, European Union Public Licence (EUPL), Academic Free License (AFL) itd. Če je originalna izvorna koda napisana pod eno licenco, potem običajno ni mogoče označiti razširitve te izvorne kode z drugo licenco. Tabela kaže primerjavo med nekaterimi najbolj znanimi prosto dostopnimi in odprtokodnimi licencami programske opreme. Veliko podrobnejšo primerjavo najdemo na spletnih straneh, namenjenih prosto dostopnim programskim licencam in odprtokodnim licencam[4].

Tabela 5: Primerjava med nekaterimi prosto dostopnimi in odprtokodnimi licencami programske opreme

	GNU GPLv3	BSD License	FreeBSD License	MIT License	Apache License	Eclipse Public License	European Union Public Licence	Academic Free License
GPL kompatibilen	Da	No	Da	Da	Da (>= 2.0)	Opcijsko	Da	Ne
DFSG kompatibilen	Da	Da	Da	Da	Da	Da		
FSF odobren	Da	Da	Da	Da	Da		Da	Da
OSI odobren	Da	Ne	Da	Da	Da	Da	Da	Da
Copyleft	Da	Ne	Ne	Ne	Ne	Omejen	Da	Ne
Povezan s kodo z drugo licenco	Ne (razen GPLv3)	Da	Da	Da	Da	Da		Da

[4] E.g. http://www.webcitation.org/77uoCLli9

IZVEDBA

- Individualno delo
- Skupinsko delo
- Razprava za in proti
- Demonstracija učitelja
- Skupinsko učenje
- Projektno učenje (glej primere)

ZA RAZMISLEK

- Kaj pomeni licenca?
- Kaj varuje intelektualno lastnino (programsko opremo in dokumentacijo)?
- Kakšna je razlika med prosto dostopno programsko licenco in odprtokodno licenco?
- Ali moraš plačati, če uporabljaš programsko opremo, ki ima prosto dostopno programsko licenco?
- Če vi in vaša programska oprema uporabljate katero izmed izvornih kod, ki ima licence GNU GPL, ali jo lahko delite in računate zanjo? Ne glede na to, če jo delite gratis ali zastonj, ali morate prejemnikom predati enako svobodo za razmnoževanje, deljenje oz. prilagajanje?

Primer 1: Licenca za programsko opremo – licenčna pogodba za končnega uporabnika (End User Licence Agreement – EULA)

Preberite si licenčne pogodbe za končne uporabnike nekaterih programov:

- WinRAR (https://www.win-rar.com/winrarlicense.html?&L=0)
- LibreOffice (https://www.libreoffice.org/about-us/licenses/).

ZA RAZMISLEK

- Ali morajo posamezniki kupiti WinRAR (LibreOffice)?
- Kaj je mogoče zaračunati pri prodaji programskega paketa LibreOffice?

VIRI

EU. (n. d.). *What is the EUPL?* Pridobljeno iz EUPL [European Union Public Licence]: https://eupl.eu/

GNU Operating System. (n. d.). *Various Licenses and Comments about Them.* Pridobljeno iz GNU Operating System: https://www.gnu.org/licenses/license-list.html

Peterson, S. K. (7. 11. 2017). *What's the difference between open source software and free software?* Pridobljeno iz opensource.com: https://opensource.com/article/17/11/open-source-or-free-software

AKTIVNOST 3: Plagiatorstvo v izobraževanju kodiranja

Tema te aktivnosti je plagiatorstvo na splošno ter posebnosti plagiatorstvo v programiranju.

TEORIJA

Plagiatorstvo je širok pojem, ki označuje različne stvari, toda vsi so povezani s pravicami intelektualne lastnine. Torej lahko rečemo, da pod plagiatorstvo spadajo: spreminjanje dela nekoga drugega v svoje lastno; kopiranje besed ali idej nekoga drugega brez navajanja avtorja; neuporabljanje narekovajev, ko nekoga citiramo; dajanje nepravilnih informacij o viru navedenih besed; spreminjanje besed, a hkrati kopiranje strukture originalnega stavka brez navajanja avtorja; kopiranje tako velikega števila besed ali idej iz originala, da predstavljajo večino tvojega dela, pa če navedeš vir ali ne (gl. poglavje o pravilih »poštene uporabe«) (p.org, 2017).

Plagiatorstvo ni kriminalno dejanje, je pa neetično početje, ki ima resne posledice v akademskih in industrijskih krogih. Za učitelje je pomembno, da učence naučijo, da kopiranje od drugih (brez pravilnega označevanja) v izobraževalnih ustanovah ni primerno. Pomembno je, da razlikujemo med kopiranjem in sodelovanjem. Učenci drug drugemu tudi pomagajo pri pisanju ali razhroščevanju programov. Količina dane pomoči in raven medsebojne pomoči študentov, ki je sprejemljiva za predavatelja, je lahko zelo različna,« (Wagner, 2000)

Predavatelji lahko med procesom ocenjevanja ročno pregledujejo učenčeve naloge glede plagiatorstva pri programiranju. Vendar ta tehnika ni uspešna in zanesljiva, če je treba oceniti veliko študentskih nalog.

Za pomoč predavatelju so lahko nekatera programska orodja za odkrivanje plagiatorstva, s katerimi lahko odkrije podobnosti med študentskimi nalogami, in na podlagi katerih se lahko odloči, ali gre za plagiatorstvo ali ne. Nekatera od teh orodij so brezplačna, obstajajo pa tudi komercialna orodja za odkrivanje programskega plagiatorstva. Primeri: JPlag, SIM, Sherlock, Plaggie, MOSS. Nekatera izmed teh orodij so namenjena enemu specifičnemu programskemu jeziku (npr. Java), zato da se prepozna sintaksa tega jezika in se poveča učinkovitost zaznavanja plagiatorstva. Nekatera druga orodja so lahko uporabna v več programskih jezikih ali navadnih besedilih, če se uporabljajo v določenih parametrih, nastavljenih za delovanje v besedilnem načinu. Naloga orodja je, da najde podobnosti med dvema študentskima nalogama, toda še vedno je učitelj tisti, ki odloči, ali gre za plagiatorstvo ali ne.

Učenci lahko poskušajo v svojih nalogah skriti izvorno kodo svojega sošolca (izvirnik ali narejene spremembe). To imenujemo zakrivanje in obstaja veliko tehnik za to, npr. oblikovanje izvorne kode, odstranitev komentarjev, preimenovanje razredov, metod in spremenljivk, vrstni red vrstic itd.

Dobra orodja za odkrivanje plagiatorstva v programiranju običajno nimajo težav z odkrivanjem večine tehnik zakrivanja[5].

[5] https://www.sciencedirect.com/science/article/pii/S1877050915032780

IZVEDBA

Navodila:

Razpravljajte o plagiatorstvu na splošno in plagiatorstvu pri računalniškem kodiranju.

Razložite, kaj je plagiatorstvo in navedite nekaj primerov, učenci/študenti povedo svoje mnenje in nato razpravljajte o tem

- Individualno delo
- Skupinsko delo
- Razprava za in proti
- Demonstracija učitelja
- Skupinsko učenje
- Projektno učenje (glej primere)

Primer: Zakrivanja

Primerjajmo izvorni program 1 in program 2, zakrit s preimenovanjem spremenljivk.

Program Python 1:

```
i = 1
j = 3
sum = 0
while(i <= j):
    sum = sum + i
   i = i + 1
print("Result: ", sum)
```

Program Python 2:

```
k = 1
l = 3
res = 0
while(k <= l):
    res = sum + k
   k = k + 1
print("Result: ", res)
```

ZA RAZMISLEK

- Ali lahko v primeru, ko študent napiše večino izvorne kode za svojega sošolca/sošolko, oba obtožimo plagiatorstva?
- Ali je odkrivanje plagiatorstva bolj natančno, če ga učitelj izvaja ročno?
- Kaj pomeni zakrivanje?
- Kaj naj učitelj naredi, ko mu programsko orodje za odkrivanje plagiatorstva poroča o podobnostih med nalogama študentov?

Primer: Različne ravni zakrivanja

Preučite primer programske kode in njenih zakritij.

Zakrivanje 1:

```
//sort the array
void bubble sort(int size)
{
int i, j, temp;
 for(i = size; i > 0; i--) {
   for(j = 0; j<i; j++){
     if (number list[j]>number list[j+1]) {
        temp = number list[j];
        number_list[j] = number_list[j+1];
        number_list[j+1] = temp;
     }
   } //end for j
 } //end for i
} //buble sort
```

```
//new algorithm sort the array
void new sort(int size)
{
int a, b, tmp;
 for(a = size; a > 0; a--) {
    for(b = 0; b<a; b++){
       if (number list[b]>number list[b+1]) {
          tmp = number list[b];
          number_list[b] = number_list[b+1];
          number_list[b+1] = tmp;
       }
    } //end for b
  } //end for a
} //new new sort
```

Zakrivanje 2:

```
//sort the array
void bubble_sort(int size)
{
int i, j, temp;
 for(i = size; i > 0; i--) {
    for(j = 0; j<i; j++) {
       if (number_list[j]>number_list[j+1]){
          temp = number_list[j];
          number_list[j] = number_list[j+1];
          number list[j+1] = temp;
       }
    } //end for j
  } //end for i
} //buble_sort
```

```
//new algorithm sort the array
void new_sort(int size)
{
int a, b, c, tmp;
 for(a = size; a > 0; a--) {
    for(b = 0; b<a; b++) {
      c=b+1;
      if (number_list[b]>number_list[c] {
         tmp = number_list[b];
         number list[b] = number list[c];
         number_list[c] = tmp;
      }
    } //end for b
  } //end for a
} //new sort
```

Zakrivanje 3:

```
//sort the array
void bubble_sort(int size)
{
int i, j, temp;
 for(i = size; i > 0; i--) {
   for(j = 0; j<i; j++) {
      if (number_list[j]>number_list[j+1]) {
         temp = number_list[j];
         number list[j] = number list[j+1];
         number list[j+1] = temp;
       }
    } //end for j
  } //end for i
} //buble sort
```

```
//new algorithm sort the array
void new_sort(int size)
{
int a, b, c, tmp;
 for(a = size; a > 0; a--) {
    for(b = 0; b<a; b++) {
       c=b+1;
       if (number_list[c]<number_list[b]) {
          tmp = number list[c];
          number list[c] = number list[b];
          number_list[b] = tmp;
       }
     } //end for b
   } //end for a
} //new sort
```

ZA RAZMISLEK

- Zakaj spadajo 1., 2. in 3. primer med zakrivanje?

VIRI

Divya, L., Divya, P., Sony, L., Sreeprabha, S., & Elizabeth, B. (2014). Software Plagiarism Detection Techniques: A Comparative Study. *International Journal of Computer Science and Information Technologies, 5*(4), 5020-5024.

p.org. (2017). *What is Plagiarism?* Pridobljeno iz p.org: https://www.plagiarism.org/article/what-is-plagiarism

Turnitin. (2017). *Plagiarism and Programming: How to Code Without Plagiarizing.* Pridobljeno iz Turnitin: https://www.turnitin.com/blog/plagiarism-and-programming-how-to-code-without-plagiarizing-2

Wagner, N. R. (2000). *Plagiarism by Student Programmers.* Pridobljeno iz Department of Computer Science, The University of Texas at San Antonio: http://www.cs.utsa.edu/~wagner/pubs/plagiarism0.html

DODATEK Aktivnosti 1

Primer – metoda ocenjevanja – natančnost in mejni primeri

Razpravljajte o natančnosti in o mejnih primerih zakrivanja.

Možen primer:

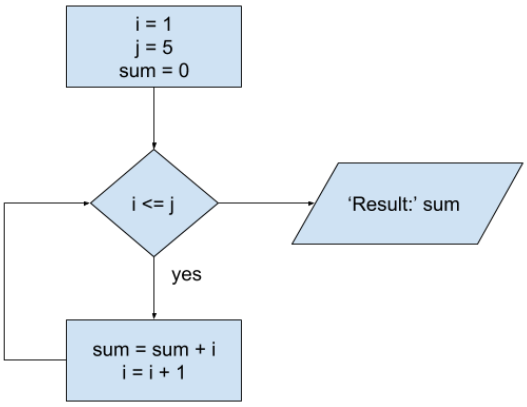
```
Read num;
Root = sqrt(num); {sqrt mean square root }

If root*root = num than
 Print "Calculation is accurate for ", num;
```

ZA RAZMISLEK:

- Zakaj zgornja koda ni robustna?
- Kaj moramo storiti v realni situaciji, da bi dobili zahtevano natančnost?
- Kakšni so razlogi za nenatančnost v nekaterih primerih?

Primer – metoda ocenjevanja – ročna izvedba

Poiščite vrednost vsote spremenljivk na koncu programa (ročno sledenje).



Program Python:

```
i = 1
j = 5
sum = 0
while(i <= j):
      sum = sum + i
      i = i + 1
print("Result: ", sum)
```

Slika 21. Diagram poteka za nalogo za ročno izvedbo kode

Dve možnosti ročne izvedbe

Ročna izvedba 1
Spremenljivke:
i=1, 2, 3, 4, 5, 6
j=5
vsota=0, 1, 3, 6, 10, 15

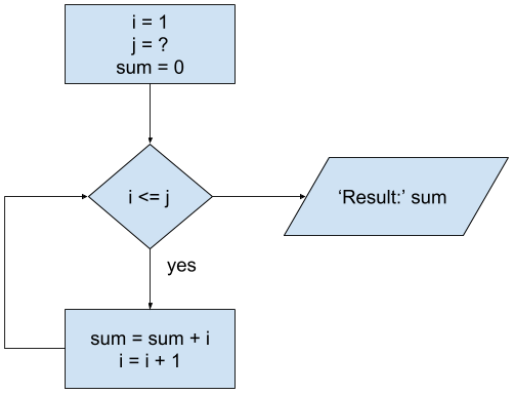
Izhod:
Result 15

Ročna izvedba 2

step	eval	i	j	sum
init		1	5	0
while (1 <=5)	TRUE	1	5	1
while (2 <=5)	TRUE	2	5	2
while (3 <=5)	TRUE	3	5	6
while (4 <=5)	TRUE	4	5	10
while (5 <=5)	TRUE	5	5	15
while (6 <=5)	FALSE	6	5	15
print		6	5	15

Primer – metoda ocenjevanja – analiza kode

Poiščite vrednost spremenljivke j tako, da bo na koncu programa vsota vsaj 7 (ročno sledenje):



Spremenljivke:
i=1, 2, 3, 4, 5
j=1, 2, 3, 4
vsota=0, 1, 3, 6, 10

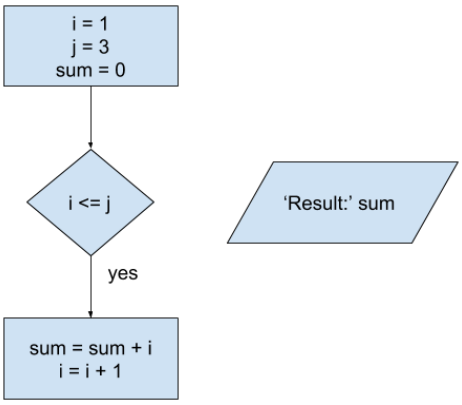
Izhod:
Result: 10

Program Python:

```
i = 1
j = ?
sum = 0
while(i <= j):
    sum = sum + i
    i = i + 1
print("Result: ", sum)
```

Primer – metoda ocenjevanja – popravki kod

Dokončajte diagram tako, da se ujema s programom Python. Ugotovite, kakšen program bo izpisan na zaslonu (ročno sledenje):



Spremenljivke:
i=1, 2, 3
j=3
vsota=0, 1, 3

Izhod:
Result: 0
Result: 1
Result: 3

Program Python:

```
i = 1
j = 3
sum = 0
while(i < j):
    sum = sum + i
    i = i + 1
print("Result: ", sum)
```

Primer – metode ocenjevanja – vredotenje kode

Razložite:

Po vrednotenju lahko program razvrstimo v eno izmed naslednjih kategorij:

1. Sintaktično pravilen, izvedba brez napak, optimalna izvedba
2. Sintaktično pravilen, izvedba brez napak
3. Sintaktično pravilen, ima napako v izvedbi
4. Sintaktično nepravilen in se ne more izvesti

Opis programa: izračunaj vsoto sodih števil v obsegu od 1 do 3 in jo na koncu pokaži.

Ocenite 4 izvedbe programa glede na zgoraj navedene kategorije. Opišite, zakaj je izbrana določena kategorija.

Program Python 1:

```
i = 1
j = 3
sum = 0
while(i <= j):
   sum = sum + i
   i = i + 1
print("Result: ", sum)
```

Odgovor: Sintaktično pravilen, izvedba brez napak, optimalna izvedba. Vse je, kot mora biti.

Program Python 2:

```
i = 1
j = 3
sum = 0
while(i <= j):
   sum = sum + i
   i = i + 1
   print("Result: ", sum)
```

Odgovor: Sintaktično pravilen, izvedba brez napak. Program izpiše vsoto za vsako vrednost. Bolje bi bilo, če bi v 7. vrstici umaknili ukaz print iz zanke »while« (izbrisali zamik pred ukazom print(..)).

Program Python 3:

```
i = 1
j = 3
sum = 0
while(i < j):
   sum = sum + i
   i = i + 1
print("Result: ", sum)
```

Odgovor: Sintaktično pravilen, napaka v izvedbi. Program ima logično napako v 4. vrstici, zato ker uporabi obseg od 1 do 2 namesto 1 do 3. Pogoj v ukazu while mora biti i <= j.

Program Python 4:

```
i = 1
j = 3
sum = 0
qhile(i < j):
   sum = sum + i
   i = i + 1
print("Result: ", sum)
```

Odgovor: sintaktično nepravilen, se ne more izvesti. Program ima sintaktično napako v 4. vrstici, "**qhile**" ni ključna beseda, moralo bi biti "**while**".

ZA RAZMISLEK

- Ali se vsi programi izvajajo brez napak?
- Kako se imenuje aktivnost/proces, ki odkriva napake v programu?
- Kaj potrebujemo za ročno odhroščevanje?
- Opišite razmerje med programom in programsko opremo.
- Ali razhroščevanje v IDE omogoča način, da vidimo vrednosti za določene spremenljivke?

Oznaka enote: UČNA ENOTA (U4-L2)

NASLOV: Ocenjevanje v poučevanju programiranja (7,5 % od 15 %)

Učni izidi

1. Oceniti kode in delovne primere projektov, ki so jih ustvarili učenci.
2. Uporabiti ocenjevanje sovrstnikov v kontekstu kodiranja.
3. Izvesti formativno preverjanje kot nadaljevalni korak pri dajanju povratnih informacij učencem.

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Razprava	☐ Vaja in utrjevanje
✕ Skupinsko delo	✕ Vprašanja in odgovori	☐ Projektno učenje
☐ Skupinsko učenje	☐ Kombinirano učenje	☐ Demonstracija
☐ Igranje vlog		

Struktura poglavja

Aktivnost 1: Projekti programiranja za učence

Aktivnost 2: Medsebojno ocenjevanje

Aktivnost 3: Formativno preverjanje pri računalništvu brez računalnika

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
✕ Kontrolni seznam/ocenjevalna lestvica	✕ Medsebojno ocenjevanje	✕ Ustno preverjanje znanja
✕ Domača naloga	☐ Vzorčni izdelek	✕ Projektna aktivnost
✕ Predstavitev		☐ Drugo

AKTIVNOST 1: Projekti programiranja za učence

V tej aktivnosti učenec dobi projektno nalogo, v okviru katere mora razviti programsko kodo. Nato se koda oceni po dani predlogi.

TEORIJA

V programskih kodah se lahko pojavi veliko različnih napak in okvar (Calvanese, 2005). Najbolj preprosta in trivialna je napaka v sintaksi.

Sintaktična napaka je napaka, ki jo lahko odkrije računalnik. Pomeni, da programski ukaz ni napisan v skladu z vnaprej določeno strukturo.

Lahko se pojavi kot:

- **Napačno črkovanje** rezerviranih besed (npr. **rihgt** namesto **right** v programskem jeziku Logo; **begn** namesto **begin** v programskem jeziku Pascal, **imput** namesto **input** v programskem jeziku Python).
- **Izpustitev** posebnih znakov, ki so nujni za dokončanje sintakse (npr. **writeln(Hello World)**; namesto **writeln("Hello World");** v programskem jeziku Pascal; **to poly a** namesto **to poly :a** v programskem jeziku Logo.
- **Nepopolnost** ukaza (npr. **x := sin(x;** manjka zaklepaj pred podpičjem).
- **Napačni argumenti** v procesu ali klicu funkcije (npr. **Canvas.Line(x,y);** manjkata drugi točki (x2, y2) v klicu funkcije).
- Uporaba **Nedeklarirane (nenajavljene spremenljivke** v striktnih programskih jezikih (npr. **int a, b; number = a+b;** število ni deklarirano/najavljeno).

Pomenske napake so bolj zapletene, ker gre za problem v logiki programa. Lahko se kažejo na različne načine. Navajamo nekatere izmed njih, vendar ta seznam ni popoln:

- Napačen operator (npr. **sum := a – b;** uporabiti je treba **+**)
- Napačen klic procedure (npr. **y := sin(x);** kjer bi moral biti **y:=cos(x);**)
- Zamenjava argumenta (npr**. flower(:size :leaf)** toda moralo bi biti **flower(:leaf :size)**)
- Napačna logika (npr. **if x=0 then y = a/x else y=maxint;** toda moralo bi biti **if x<>0 then y:=a/x** …)

Pomenske napake se zaznanajo, kadar kodiranje proizvede napačni izhod, lahko bi tudi rekli, da ima program okvaro. V nekaterih primerih tovrstne napake hitro odkrijemo, v nekaterih primerih pa jih je zelo težko najti. Zatorej prirporočamo, da preverite logiko, preden se lotite kodiranja programa. Preverjanje programa je aktivnost, ki bi morala odkriti obstoj pomenskih napak. Razhroščevanje programa pa bi moralo najti mesta, kjer se nahajajo pomenske napake in poskrbeti za njihove popravke.

IZVEDBA

Učenci dobijo individualne naloge ter se lotijo dela. V dodatku je navedenih nekaj primerov teh nalog. Učitelje spodbujamo, da poiščejo še dodatne primere.

Predlagamo naslednjo predlogo za ocenjevanje kodiranja. Učitelje spodbujamo, da pripravijo svoje lastne predloge, ki bodo primerne za ocenjevanje nalog učencev.

Opisno:

- *Kodna vrstica*
- *komentarji*
- *število postopkov / funkcij*
- *število logičnih poti*

Izkušnja uporabnika::

- *ali program deluje: da, vedno/ večinoma/ za pričakovane vnose/ za nekatere vnose/ ne*
- *ali je program razumljiv: da/ delno/ ne*
- *ali je program natančen: da/ delno/ ne*
- *ali je program predvidljiv: da/ delno/ ne*
- *ali je program dovolj hiter: da/ne*

Preverjanje programa:

Vhodi:

- *pričakovani vhodi:*
- *vhodi mejnih vrednosti:*
- *nepričakovani vhodi:*

Izhodi:

- *pričakovani izhodi glede na vhode:*
- *nepričakovani izhodi glede na vhode:*

Optimizacija:

- *Ali je koda preprosta za optimiziranje: da / ne*

Vzdrževanje:

- *Ali je koda razumljiva: da/ ne*

- Individualno delo
- Skupinsko delo
- Razprava za in proti
- Demonstracija učitelja
- Skupinsko učenje
- PBL – Projektno poučenje (glej primere)

Primer: računalnik za računanje Fibonaccijevega števila

Učenci naj najdejo algoritem, ki izpiše vrednost za dano pozitivno število/indeks elementa glede na naslednja začetna števila in njihove vrednosti:

Algoritem gre takole:

$$F_1 = 1$$
$$F_2 = 1$$
$$F_i = F_{i-1} + F_{i-2}$$

To je znano Fibonaccijevo število/zaporedje.

i	1	2	3	4	5	6	7	8	9	10
vrednost	1	1	2	3	5	8	13	21	34	55

Rešitve

Program Python 1:

```
num = int(input("Enter a positive number (> 0): ")
vi_1 = 1
vi_2 = 1
i = 0
if num <= 0:
   print("Please enter a positive integer")
elif num == 1:
   print("Value: ", vi_1)
elif num == 2:
   print("Value: ", vi_2)
else:
   while i <= num:
      vi = vi_1 + vi_2
      vi_1 = vi_2
      vi_2 = vi
      i += 1
   print("Value: ", vi)
```

V tem programu funkcija izračuna vrednost za dani argument na osnovi ponovitve in jo vrne. Izpis vrednosti, vrnjen iz klica funkcije, ima enako število kot argument.

Program Python 2:

```
def Fibonacci(n):
   if n <= 0:
      return "Incorrect number"
   elif n == 1:
      return 1
   elif n == 2:
      return 1
   else:
      return Fibonacci(n - 1) + Fibonacci(n - 2)

num = int(input("Enter a positive number (> 0): ")
print(Fibonacci(num))
```

ZA RAZMISLEK

- Rešitev izvorne kode za dan problem (nalogo) se lahko zapiše na papir ali v katerega izmed programov (še zlasti IDE) na računalnik.

- Ocenjevanje rešitve se razlikuje glede na medij, kjer je ta rešitev shranjena.
- Sintaktična napaka je večja, če je izvorna koda zapisana na računalnik, ker jo tam lahko preveri IDE, prevajalnik ali tolmač.

Če rešitev nima nobene sintaktične napake, jo lahko ocenimo na enega izmed dveh načinov:

- Na temelju pravilnosti izvedbe. V tem primeru sta možnosti zgolj: opravil ali padel.
- Na temelju izčrpnosti. Predvidena rešitev mora biti pripravljena in razdeljena na več delov (koraki in algoritem), kjer vsak del predstavlja določen % od celotne rešitve. Glede na dele, ki jih je mogoče najti v učenčevi rešitvi, mu učitelj/sošolec sešteva % (točke) od celotne vsote.

Pravilnost izvedbe se lahko oceni na dva načina oz. enega izmed njiju:

- Avtomatsko z določenimi orodji – običajno uporabimo, če imamo veliko učencev in kratke algoritemske probleme z znanim vhodom (ena ali več testnih datotek) in izhodom (datoteka). V tem primeru sta zgolj dve možnosti: opravil ali padel. Če je rešitev padla, jo lahko učitelj oceni ročno.
- Ročno – učitelj ali sošolci.

Primer: Ocenjevanje kode

Uporabite program Python 1 in izvedite ocenjevanje na podlagi sledeče tabele:

Del	Vrstice	%
Vnos številke	1 - 1	5
Uvodna nastavitev spremenljivk	2 - 4	15
Preverjanje neveljavnega števila	5 - 6	10
Preverjanje prvih dveh števil	7 - 10	20
Izračun za druga števila	11 - 16	45
Izpis rezultata	17 - 17	5
Skupaj		100

Program Python 1:

```
1   num = int(input("Enter a positive number (> 0): ")
2   vi_1 = 1
3   vi_2 = 1
4   i = 0
5   if num <= 0:
6    print("Please enter a positive integer")
7   elif num == 1:
8    print("Value: ", vi_1)
9   elif num == 2:
10   print("Value: ", vi_2)
11  else:
12   while i <= num:
13   vi = vi_1 + vi_2
14   vi_1 = vi_2
15   vi_2 = vi
16   i += 1
17   print("Value: ", vi)
```

AKTIVNOST 2: Medsebojno ocenjevanje

Ta aktivnost se osredotoča na medsebojno ocenjevanje v poučevanju programiranja.

TEORIJA

Medsebojno ocenjevanje izvajajo učenci tako, da ocenijo delo svojih sošolcev. Podajo povratne informacije (vključijo lahko tudi ocene) o kvaliteti opravljenega dela. Tovrstno ocenjevanje je zelo koristno za projekte kodiranja, saj od učencev zahteva višjo stopnjo znanja o Bloomovi taksonomiji, in sicer glede analiziranja in ovrednotenja. Toda še pred ocenjevanjem mora skupina razviti in se strinjati s kriteriji ocenjevanja, ki se lahko podajo v obliki navodil ipd. Vključevanje učencev v razvijanje kriterijev jim odpre globlji pogled v učenje in jim omogoča temeljito poznavanje teme. Kriteriji se postavijo pred začetkom dejanskega dela, da lahko učenci pripravijo svoje programe v skladu s temi kriteriji in uporabijo samoocenjevanje svojega dela.

IZVEDBA

Ovrednotenje medsebojnega ocenjevanja in poročanje:

- Ocenjevanje izbranih projektov učencev v razredu (aktivnost 1) morajo izvesti sošolci. Pari se lahko določijo s pomočjo spletne učilnice ali pa jih določijo učitelji.
- Ena skupina učencev mora pripraviti smernice ocenjevanja določenega projekta. Nato morata vsaj še dve skupini oceniti ta isti projekt na podlagi pripravljenih smernic in nato pojasniti, kako sta to izvedli in ali se njihova mnenja v čem razlikujejo.

Pripravite skupne rezultate nalog in ovrednotite poročila učencev.

Poročila učencev se lahko naložijo v aktivnost naloga v spletni učilnici. Strukturirani podatki se lahko pridobijo z uporabo standardnega vprašalnika, ki temelji na različnih metodah ocenjevanja. Z učenci preučite pridobljene podatke in pojasnite najbolj pogoste napake …

- Skupinsko delo
- Razprava za in proti
- Demonstracija učitelja
- Skupinsko učenje

ZA RAZMISLEK

- Zakaj je medsebojno ocenjevanje dobro za to, da učenci snov bolje razumejo?
- Ali je pomembno, da se skupina strinja glede kriterijev? Kdo pripravi kriterije?
- Ali lahko kriterije pripravimo šele na koncu naloge?

AKTIVNOST 3: Formativno preverjanje v poučevanju računalništva brez računalnika

Ta aktivnost se osredotoča na različne možnosti formativnega preverjanja v poučevanju računalništva brez računalnika.

ZAHTEVANA ZNANJA

- Formativno preverjanje (Frey & Fisher, 2011) (Moos & Brookhart, 2009).

TEORIJA

Formativno preverjanje je zelo uporabno v poučevanju kodiranja. Ko učenci kodirajo, jim učitelj da povratno informacijo o kvaliteti celega programa in kvaliteti programske kode, kar pomaga tudi, kadar imajo učenci težave s programom. Tovrstno ocenjevanje daje učencem takojšno povratno informacijo o njihovih veščinah kodiranja, učitelju pa vpogled v učenčevo znanje in sposobnosti.

IZVEDBA

- Individualno delo
- Skupinsko delo
- Razprava za in proti
- Demonstracija učitelja
- Skupinsko učenje

ZA RAZMISLEK

- Katera vrsta ocenjevanja se uporablja?
- Kakšne so razlike med formativnim in končnim preverjanjem?
- Kaj je namen formativnega preverjanja?
- Zakaj naj učitelj uporablja vprašanja in pozive?
- Kdaj naj učitelj uporabi namige?

VIRI

Anderson, L. W., & Krathwohl, D. R. (2001). A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives. New York: Longman.

Calvanese, D. (2005). Unit 10: Program errors and exception handling. Pridobljeno iz Introduction to Programming: https://www.inf.unibz.it/~calvanese/teaching/05-06-ip/lecture-notes/uni10/uni10-main.html

Frey, N., & Fisher, D. (2011). The Formative Assessment Action Plan: Practical Steps to More Successful Teaching and Learning. Alexandria, VA: ASCD.

Moos, C. M., & Brookhart, S. M. (2009). Advancing Formative Assessment in Every Classroom: A Guide for Instructional Leaders. Alexandria, VA: ASCD.

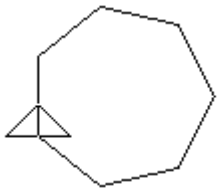
DODATEK Aktivnosti 1

Primeri nalog

https://www.myassignmenthelp.net/algorithm-assignment-help

Grafična naloga:

- Pripravite program, ki nariše kateri koli konveksni mnogokotnik z enakimi stranicami (Slika 22).
- Preparavite program, ki nariše rožo (Slika 23).
- Pripravite rekurzivni program, ki nariše črto. Vnos je velikost prve stranice (Slika 24).
- Pripravite program, ki nariše drevo. Vnos je velikost drevesnega debla (Slika 25).



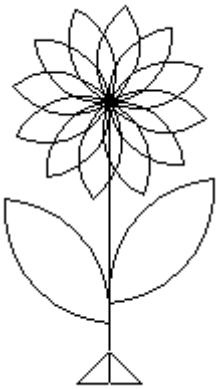
mnogokotnik 40 7

'40 slikovnih točk stran in 7 strani, za razumevanje kotov in števila strani.

koda je za program MSWLogo ali FMSLogo, ki je prost za uprabo (http://fmslogo.sourceforge.net/)

```
to poly :a :n
 repeat :n [
 fd :a
 rt 360/:n
 ]
end
```

Slika 22: Enakostranični mnogokotnik



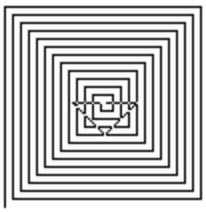
roža 10 12

10 slikovnih točk za velikost in 12 listov v cvetu. To je dobro za algoritem deli-in-vladaj.

koda je za program MSWLogo ali FMSLogo, ki je prost za uprabo (http://fmslogo.sourceforge.net/)

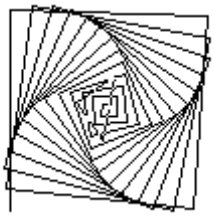
```
to flower :a :n
 repeat :n [
 right leaf :a*0.6
 rt 360/:n
 ]
end

to left leaf :a
 repeat 2 [
 repeat 9 [
 fd :a
 lt 10 ]
 lt 90
 ]
end

to right_leaf :a
 repeat 2 [
 repeat 9 [
 fd :a
 rt 10 ]
 rt 90
 ]
end

to rose :a :n
 fd :a*3
 left_leaf :a
 fd :a
 right_leaf :a
 fd :a*10
 flower :a :n
 bk :a*14
end
```

Slika 23: Roža



special 100

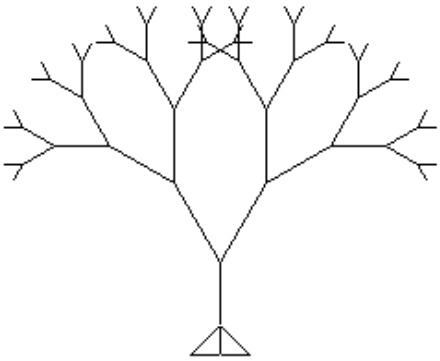
(100 slikovnih točk začetna velikost druge strani), še zlasti dobro za učenje rekurzivnega algoritma. Obstaja veliko variacij te oblike.



koda je za program MSWLogo ali FMSLogo, ki je prost za uprabo (http://fmslogo.sourceforge.net/)

```
to special :a
if :a > 0 [
 fd :a
 rt 90
 special :a-2
]
end
```

Slika 24: Rekurzivno risanje



drevo 70

(70 slikovnih točk deblo drevesa), dobro za razumevanje rekurzivnih in ponavljajočih se vzorcev. Lahko ga nadgradimo še z uporabo poljubnega kota.

koda je za program MSWLogo ali FMSLogo, ki je prost za uprabo (http://fmslogo.sourceforge.net/)

```
to tree :a
 fd :a ;tree trunk
 crochet :a
 bk :a
end

to crochet :a
 if :a > 4 [
 lt 30 fd :a crochet :a-10 bk :a
 rt 60 fd :a crochet :a-10 bk :a
 lt 30
 ]
end
```

Slika 25: Rekurzivno drevo

Naloga – obdelava števil:

- Pripravite program, ki bo sprejemal števila in prikazal (min, maks, vsoto, in povprečje).
- Pripravite program, ki bo vzel dva seznama števil in pripravil enoten seznam urejenih števil.
- Pripravite program, ki bo sprejemal cela števila in prikazal samo tiste, ki so deljivi z/s (nekim poljubnim številom).
- Pripravite program, ki bo sprejel realno število in ...

Naloga – obdelava besedila:

- Pripravite program, ki bo prejel besedilo in prikazal število besed (število določenih znakov, npr. »a« v besedilu).
- Pripravite program, ki bo sprejel stavek in razvrstil besede iz stavka.
- Pripravite program, ki bo sprejel stavek in prikazal besede v obrnjenem vrstnem redu (razširite na prikaz besed z obrnjenim vrstnim redom znakov – dobra vaja za rekurzivnost).
- Pripravite program, ki bo prebral besedilni dokument in prikazal njegovo statistiko (koliko stavkov, besed, znakov z ali brez presledkov).

Primer: Razpravljajte o različicah rešitev za programsko opremo

Teme razprave: dober slog programiranja (omejitev števila vrstic v funkciji), uporaba komentarjev, dobra/slaba struktura programa, uporaba splošnih načel za imena spremenljivk in funkcij; uporaba opisnih konstant proti konstantam samim.

```
to poly :a :n
 repeat :n [
 forward :a
 right 360/:n
 ]
end
```

```
to poly :a :n
 repeat :n [ fd :a rt 360/:n ]
end
```

ZA RAZMISLEK:

- Katerega izmed dveh programov je lažje brati?
- Kje je lažje najti napake?

Primer: Razpravljajte o različicah rešitev za programsko opremo

```
to desni_list :a
; :a je spremenljivka, ki določa velikost lista

; narisati moramo dva loka in se vrniti na izhodiščno točko
; Torej moramo z risanjem narediti 360 stopinjski kot
;
; izračunajmo kako narišemo list
; najprej imamo repeat 2 => 2
; potem imamo repeat 9 => 9
; znotraj zanke repeat se pomaknemo za kot 10 stopinj
; v desno in v zanki naredimo (9*10) => 90 degrees
; potem se obrnemo za 90 degrees
; če izračunamo kote, ki jih želvica prehodi ob risanju dobimo
; 2*(9*10+90) = 2*180 = 360 stopinj

repeat 2 [
; list rože je zunanji kot četrtine 36 kotnika.
; 9*10 stopinj nam naredi 90 stopinj in to je četrtina 36 kotnika
   repeat 9 [
      forward :a
      right 10
   ] ; konec notranje zanke repeat 9

; narisali smo en del lista, sedaj pa se obrnemo za
; 90 stopinj in narišemo še drug del lista.

   right 90
] ; konec zunanje zanke repeat 2
end
```

```
to desni_list :a
 repeat 2 [ repeat 9 [
 fd :a rt 10 ] rt 90
 ]
end
```

Za razmislek:

- Katerega izmed dveh programov je lažje brati?
- Katerega izmed dveh programov je lažje razumeti?
- Zakaj je dobro uporabljati komentarje?

Oznaka enote: UČNA ENOTA (U5-L1)

NASLOV: Postavitev načrta učne ure za relevantne teme kodiranja (15 %)

Učni izidi

- Določiti učne izide učne ure kodiranja.
- Ustvariti učni načrt za učno uro kodiranja.
- Razviti časovnico za načrt učne ure.

Metodologija

Strategije poučevanja

✕ Predavanja	✕ Skupinsko delo	☐ Igranje vlog
✕ Razprava	☐ Vprašanja in odgovori	☐ Kombinirano učenje
☐ Vaja in utrjevanje	✕ Projektno učenje	☐ Demonstracija

Struktura poglavja

Aktivnost 1: Določitev učnih izidov pri učni uri kodiranja

Aktivnost 2: Ustvarjanje učnega načrta za učno uro kodiranja

Aktivnost 3: Razvijanje časovnice za načrt učne ure

Možne metode ovrednotenja

✕ Učiteljevo opazovanje	☐ Analiza opravljenih del	☐ Pisno preverjanje znanja
☐ Kontrolni seznam/ocenjevalna lestvica	✕ Medsebojno ocenjevanje	✕ Ustno preverjanje znanja
✕ Domača naloga	☐ Vzorčni izdelek	✕ Projektna aktivnost
☐ Predstavitev		☐ Drugo

AKTIVNOST 1: Določitev učnih izidov pri učni uri kodiranja

TEORIJA

Zakaj je pomembno, da definiramo cilje učne ure? Veliko novih učiteljev si postavlja to vprašanje. Kadar razmišljamo o ciljih poučevanja, moramo poznati tradicionalni pristop, ki je osredotočen na cilje, ter sodobni prostop, ki se naslanja na učne izide. Ko razmišljamo o ciljih in učnih izidih, moramo imeti v mislih možne načine, kako jih lahko ovrednotimo, in veliko lažje je ovrednotiti učne izide kot pa cilje.

	Cilj	**Učni izid**
Pomen	Namen, kamor je usmerjeno prizadevanje.	Nekaj, kar želimo doseči s svojim trudom ali dejanji; namen; tarča.
Obseg	Širok: v smislu, da so cilji splošni in niso dovolj specifični, da bi jih lahko merili.	Natančen: učni izidi so določeni za vsako nalogo posebej.
Specifičnost	Splošen: splošne namere proti dosegu nečesa.	Specfičen: natančno določena dejanja za izpolnitev specifične naloge.
Merljivost	Cilji niso vedno natančno izmerljivi ali otipljivi.	Mora biti izmerljiv in otipljiv.
Osredotočenost	Osredotočen na učitelja.	Osredotočen na učence.
Časovni okvir	Dolgoročni.	Srednje do kratkoročni.

Tabela 6: Razlike med cilji in učnimi izidi[6]

Učni izidi so sestavljeni iz treh delov, pri čemer sta prva dva dela obvezna:

- I) **glagol,** ki definira tip in raven aktivnosti,
- II) **kratek opis** aktivnosti,
- III) kako bo aktivnost (izid) **ovrednotena**.

Osnovna pravila za postavljanje učnih izidov so:

- vsak učni izid mora imeti vsaj en glagol,
- če ima več glagolov, se morajo ti dopolnjevati in biti logično povezani,
- učni izid mora biti merljiv (kako bo ovrednoten),
- izogibati se moramo dolgim in kompleksnim učnim izidom,
- ko oblikujemo učne izide, se moramo izogibati glagolom, kot so: vedeti, učiti, razumeti …
- izogibamo se primerjanj, kot so »boljši«, »več« …

Dodatne ideje najdete med viri na koncu tega poglavja, ki navajajo podrobnejše informacije o učnih izidih. Pomembno je, da se vadimo v pisanju učnih izidov. Prav je, da pri tem uporabljamo potreben material (aktivne glagole) in zagovarjamo temo učne ure. Obstaja več načinov, kako postaviti učne cilje, in osnovnih pravil, ki jim morajo učni cilji slediti, pa tudi njihove izjeme. Določenih glagolov in besednih zvez se moramo izogibati. Prav tako moramo imeti v mislih ravnotežje med skupinami učnih izidov (kognitivno, psihomotorično in afektivno področje).

[6] Adapted from: Weber State University. Goals vs. Objectives. Retrieved from: https://weber.instructure.com/courses/307280/pages/goals-vs-objectives

IZVEDBA

- Razprava
- Delo v parih
- Individualno delo

ZA RAZMISLEK

- Zakaj je pomembno, da razumemo cilj?
- Kakšna je razlika med cilji in učnimi zidi v procesu poučevanja?
- Katera pravila moramo upoštevati, ko postavljamo učne izide?
- Naštejte nekaj glagolov in besednih zvez, ki se jih moramo pri postavljanju učnih izidov izogibati, in razložite, zakaj.
- Zakaj naj učitelji pri načrtovanju učne ure izhajajo iz učnih izidov?

VIRI

Anderson, L. (n. d.). *Bloom's Revised Taxonomy: Cognitive, Affective, and Psychomotor.* Pridobljeno iz Arkansas State University: https://www.astate.edu/dotAsset/7a3b152c-b73a-45d6-b8a3-7ecf7f786f6a.pdf

CEDEFOP - European Centre for the Development of Vocational Training. (2012). *Curriculum reform in Europe: The impact of learning outcomes.* Pridobljeno iz CEDEFOP - European Centre for the Development of Vocational Training: http://www.cedefop.europa.eu/en/publications-and-resources/publications/5529

CEDEFOP: European Centre for the Development of Vocational Training. (2017). *Defining, writing and applying learning outcomes: A European handbook.* Pridobljeno iz CEDEFOP: European Centre for the Development of Vocational Training: http://www.cedefop.europa.eu/en/publications-and-resources/publications/4156

Center for Excellence in Learning and Teaching. (n. d.). *Tips on Writing Course Goals/Learning Outcomes and Measurable Learning Objectives.* Pridobljeno iz Iowa State Universtiy: Center for Excellence in Learning and Teaching: http://www.celt.iastate.edu/teaching/preparing-to-teach/tips-on-writing-course-goalslearning-outcomes-and-measureable-learning-objectives/

AKTIVNOST 2: Ustvarjanje učnega načrta za učno uro kodiranja

ZAHTEVANA ZNANJA

- Motivacijske tehnike iz pedagogike in psihologije
- Znanje, povezano z vedenjem in obnašanjem s področja pedagogike
- Didaktični principi
- Organizacija pouka
- Metode in tehnike poučevanja
- Metode in tehnike učenja
- Znanje programiranja/kodiranja

TEORIJA

Ko načrtujemo učno uro, moramo začeti s primeri, ki so učencem zelo dobro znani – njihove lastne izkušnje s šolo. Razmislek o tem, kako je pouk običajno organiziran, nam lahko pomaga pri pripravi našega lastnega načrta učne ure. Upoštevati moramo obvezne elemente za vsak del pouka: uvod, osrednji del in zaključek. Vsak del ima lahko določene variacije svojih pod-elementov in pristopov. Dobro je, če osnovni bazen naših idej temelji na osebni izkušnji s šolo.

Ključni elementi uvodnega dela so:

- strinjanje razreda
- motivacija
- predznanje

Ko načrtujemo osrednji del ure, moramo imeti v mislih možne pristope in kombinacije: učenje novih znanj in veščin, vaja, vrednotenje ali katera koli kombinacija teh treh. Motivacija in vedenje (tako profesionalno kot splošno) bi prav tako morala biti obvezna v tem osrednjem delu. Vsaka aktivnost mora biti povezana z učnimi izidi, določenimi za to učno uro.

V zaključnem delu mora vsak učitelj načrtovati aktivnosti, osredotočene na zaključna navodila, povezana z upravljanjem pouka, vajo in ovrednotenjem učne ure. Če je na voljo dovolj časa, se lahko ura zaključi z vedenjskimi elementi, povezanimi z osrednjim delom. To bo pripomoglo k holističnemu pristopu in predstavilo celo učno uro kot smiselno celoto.

Preučite različne predloge za načrtovanje učne ure in poskusite analizirati vsak del posebej. Ena izmed predlog je navedena med viri na koncu tega poglavja. Brez skrbi jo lahko prilagajate ali poiščete druge podobne vire, ki so na voljo na spletu ali med vašimi običajnimi relevantnimi viri za učitelje.

Učitelji morajo ohranjati jasno povezavo med učnimi izidi, načrtom učne ure in njegovo predlogo.

Ko učitelj načrtuje učno uro, se mora odločiti za vrstni red aktivnosti in izbrati, katere aktivnosti se lahko izvedejo znotraj učne ure. Prav tako mora imeti učitelj vseskozi v mislih učne izide, ki jih je postavil za to uro, in mora poskrbeti, da so ti elementi jasno povezani med sabo.

IZVEDBA

- Razprava
- Delo v parih
- Medsebojno ocenjevanje
- Individualno delo

ZA RAZMISLEK

- Zakaj je pomembno načrtovanje v procesu poučevanja?
- Kaj mora biti načrtovano v naprej?
- Ali so načrti učne ure enaki njihovim izvedbam?
- Kako lahko načrt prilagodimo nepričakovanim situacijam? Ali lahko katero izmed prilagoditev predvidimo že v fazi načrtovanja učne ure?
- Katere dobre prakse naj upoštevamo, ko delamo načrt za učno uro?

VIRI

Alexandria City Public School. (2017). *Tips for Teachers: Key Elements of Effective Lesson Delivery.* Pridobljeno iz Alexandria City Public School: https://www.acps.k12.va.us/cms/lib/VA01918616/Centricity/Domain/801/Teaching%20Tips%20for%20Teachers%20Vol.%202.pdf

All you need is code. (n. d.). *Lesson Plans.* Pridobljeno iz All you need is code: http://www.allyouneediscode.eu/lesson-plans

Hour of Code. (n. d.). *Educator Hour of Code: Lesson Plan Outline.* Pridobljeno iz Hour of Code: https://hourofcode.com/files/EducatorHourofCodeLessonPlanOutline.docx

Singapore Management University. (n.d.). *Lesson planning.* Pridobljeno iz Teaching@SMU: https://cte.smu.edu.sg/approach-teaching/integrated-design/lesson-planning

AKTIVNOST 3: Razvijanje časovnice za načrt učne ure

TEORIJA

Načrtovanje je zelo pomemben del procesa poučevanja. Učitelji morajo slediti (in ustvarjati) nujne logične povezave med načrtovanimi aktivnostmi v letnem učnem načrtu. Temeljna pravila za nove učitelje za ocenitev trajanja aktivnosti so:

- Aktivnost, ki jo bodo izvajali, bo trajala približno toliko časa, kot ga sami potrebujejo, da jo izvedejo;
- Aktivnost, ki jo bodo vodili oni, učenci pa jim bodo sledili, bo trajala približno 1,5 x toliko, kot če bi jo izvajali sami;
- Če bodo aktivnost izvajali (zgolj) učenci, jim bo vzela približno 2 x toliko časa, kot ga vzame vam (učitelju).

Učitelj mora upoštevati morebitne velike razlike (heterogene skupine) v znanju in veščinah učencev, ki zelo vplivajo na čas izvedbe načrtovanih aktivnosti. Učenci, ki so vešči v določenih temah, bodo mogoče aktivnost izvedli tako hitro kot učitelj; medtem ko mogoče nekateri učenci aktivnosti sploh ne bodo mogli izvesti samostojno.

Ko razvijamo časovnico, moramo vedeti, da imata uvodni in zaključni del ure svoj namen in da ju ne smemo ignorirati ali ju imeti za nepomembna elementa učne ure, katerima namenimo le 1-3 minute časa. Ne obstaja eno in edino pravilo za uspešno postavitev časovnega načrta. Ta temelji na praksi, didaktičnih principih in logičnih povezavah med ključnimi elementi, ki definirajo učno uro. Zato se moramo osredotočiti na praktično delo in vaje kot tudi na delavnice in tutorstvo.

IZVEDBA

- Razprava
- Delo v parih
- Medsebojno ocenjevanje

ZA RAZMISLEK

- Zakaj je pomembna časovnica pri načrtovanju učne ure?
- Kako lahko ocenimo, kako dolgo bo aktivnost trajala?
- Ali obstaja razlika med trajanjem aktivnosti, če jo izvaja učitelj ali učenci, ki jih vodi učitelj, ali pa učenci sami?
- Katere informacije potrebujemo za uspešno časovno načrtovanje učne ure?
- Kateri so primeri dobrih praks, ki bi jih naj uporabili pri časovnem načrtovanju učne ure?

VIRI

Singapore Management University. (n.d.). *Lesson planning.* Pridobljeno iz Teaching@SMU: https://cte.smu.edu.sg/approach-teaching/integrated-design/lesson-planning

Usmani, F. (28. 1 2019). *Three Tools to Estimate Activity Duration.* Pridobljeno iz PM Study Circle: https://pmstudycircle.com/2012/06/three-tools-estimate-activity-duration/

Seznam literature

Alexandria City Public School. (2017). *Tips for Teachers: Key Elements of Effective Lesson Delivery.* Pridobljeno iz Alexandria City Public School: https://www.acps.k12.va.us/cms/lib/VA01918616/Centricity/Domain/801/Teaching%20Tips%20for%20Teachers%20Vol.%202.pdf

All you need is code. (n. d.). *Lesson Plans.* Pridobljeno iz All you need is code: http://www.allyouneediscode.eu/lesson-plans

Anderson, L. (n. d.). *Bloom's Revised Taxonomy: Cognitive, Affective, and Psychomotor.* Pridobljeno iz Arkansas State University: https://www.astate.edu/dotAsset/7a3b152c-b73a-45d6-b8a3-7ecf7f786f6a.pdf

Anderson, L. W., & Krathwohl, D. R. (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives.* New York: Longman.

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., . . . Wittrock, M. C. (n. d.). *REVISED Bloom's Taxonomy Action Verbs.* Pridobljeno iz AZUSA Pacific University: https://www.apu.edu/live_data/files/333/blooms_taxonomy_action_verbs.pdf

ARCSmodel.com. (n. d.). *Attention, relevance, confidence, satisfaction*. Pridobljeno iz ARCSmodel.com: https://www.arcsmodel.com/

ars technica. (15. 9 2012). *Is it true that "not everyone can be a programmer"?* Pridobljeno iz ars technica: https://arstechnica.com/information-technology/2012/09/is-it-true-that-not-everyone-can-be-a-programmer/

BBC. (n. d.). *Introduction to computational thinking.* Pridobljeno iz BBC Bitsize: https://www.bbc.com/bitesize/guides/zp92mp3/revision/1

Bell, S. (2010). Project-Based Learning for the 21st Century: Skills for the Future. *The Clearing House, 83*(2), 39-43.

Bienkowski, M., Snow, E., Rutstein, D., & Grover, S. (12 2015). *Assessment Design Patterns for Computational Thinking Practices in Secondary Computer Science: A First Look.* Pridobljeno iz Principled Assessment of Computational Thinking: https://pact.sri.com/downloads/Assessment-Design-Patterns-for-Computational%20Thinking-Practices-Secondary-Computer-Science.pdf

Bigby, G. (29. 1 2019). *Top 25 Amazing Websites to Learn How to Code.* Pridobljeno iz Dyno Mapper: https://dynomapper.com/blog/410-top-25-websites-to-learn-to-code

Biggs, J. (2006). What the Student Does: teaching for enhanced learning. *Higher Education Research & Development*, 57-75.

Binstock, A. (6. 3 2003). *Obcuscation: Cloaking your Code from Prying Eyes.* Pridobljeno iz Destination.NET: https://web.archive.org/web/20080420165109/http://www.devx.com/microsoftISV/Article/11351

Bishop, M. (1998-2002). *Robust Programming.* Pridobljeno iz Matt Bishop - Department of Computer Science - University of California, Davis: http://nob.cs.ucdavis.edu/bishop/secprog/robust.html

Blumenfeld, P., Soloway, E. M., Krajcik, J. S., Guzdial, M., & Palincsar, A. (2011). Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. *Educational Psychologist, 26*(3), 369-398. Pridobljeno iz https://www.researchgate.net/profile/Yael_Seker/post/How_do_you_approach_the_design_of_a_group_task_in_higher_education/attachment/59d623d979197b8077982283/AS%3A309004255858690%401450683763438/download/Blumenfeld+et+al_Motivating_project_based_learning.pd

Breiner, J., Sheats Harkness, S., Johnson, C., & Koeler, C. M. (2012). What is STEM? A discussion about Conceptions of STEM in education and partnerships. *School Science and Mathematics, 112*(1), 3-11.

Brown, N. C., & Wilson, G. (2018). Ten quick tips for teaching programming. *PLoS Computational Biology, 14*(4).

Bybee, R. W. (2013). *The Case for STEM Education: Challenges and Opportunities.* NSTA Press (National Science Teachers Association).

Calvanese, D. (2005). *Unit 10: Program errors and exception handling*. Pridobljeno iz Introduction to Programming: https://www.inf.unibz.it/~calvanese/teaching/05-06-ip/lecture-notes/uni10/uni10-main.html

Capterra. (n. d.). *Collaboration Software*. Pridobljeno iz Capterra: https://www.capterra.com/collaboration-software/

CEDEFOP - European Centre for the Development of Vocational Training. (2012). *Curriculum reform in Europe: The impact of learning outcomes.* Pridobljeno iz CEDEFOP - European Centre for the Development of Vocational Training: http://www.cedefop.europa.eu/en/publications-and-resources/publications/5529

CEDEFOP: European Centre for the Development of Vocational Training. (2017). *Defining, writing and applying learning outcomes: A European handbook.* Pridobljeno iz CEDEFOP: European Centre for the Development of Vocational Training: http://www.cedefop.europa.eu/en/publications-and-resources/publications/4156

Center for Excellence in Learning and Teaching. (n. d.). *Tips on Writing Course Goals/Learning Outcomes and Measurable Learning Objectives.* Pridobljeno iz Iowa State Universtiy: Center for Excellence in Learning and Teaching: http://www.celt.iastate.edu/teaching/preparing-to-teach/tips-on-writing-course-goalslearning-outcomes-and-measureable-learning-objectives/

Chand, S. (n. d.). *Motivation Theories: Top 8 Theories of Motivation – Explained!* Pridobljeno iz YourArticleLibrary: The next generation library: http://www.yourarticlelibrary.com/motivation/motivation-theories-top-8-theories-of-motivation-explained/35377

Ciobanu, D. (24. 10 2018). *Best Tools for Code Collaboration.* Pridobljeno iz designmodo: https://designmodo.com/code-collaboration/

Common sense education. (n. d.). *STEM Apps for Higher-Order Thinking*. Pridobljeno iz Common sense education: https://www.commonsense.org/education/top-picks/stem-apps-for-higher-order-thinking

Concordia University Portland Oregon. (2012 (2018)). *Which is Best: Teacher-Centered or Student-Centered Education?* Pridobljeno iz A blog by Concordia University - Portland: https://education.cu-portland.edu/blog/classroom-resources/which-is-best-teacher-centered-or-student-centered-education/

CS Unplugged. (2002). *Card Flip Magic—Error Detection & Correction.* Pridobljeno iz CS Unplugged - Computer Science without a computer: https://classic.csunplugged.org/wp-content/uploads/2014/12/unplugged-04-error_detection.pdf

CS Unplugged. (n. d.). *CS Unplugged Topics.* Pridobljeno iz CS Unplugged: https://csunplugged.org/en/topics/

DCSTEM network. (n. d.). *Resources & Tools*. Pridobljeno iz DCSTEM network: https://www.dcstemnetwork.org/resources-and-tools/

Divya, L., Divya, P., Sony, L., Sreeprabha, S., & Elizabeth, B. (2014). Software Plagiarism Detection Techniques: A Comparative Study. *International Journal of Computer Science and Information Technologies, 5*(4), 5020-5024.

Donmez, O., & Inceoglu, M. M. (2008). A Web Based Tool for Novice Programmers: Interaction in Use. *ICCSA '08 - International conference on Computational Science and Its Applications* (str. 530 - 540). Perugia, Italy: Springer-Verlag.

Doyle, T. (2011). *Learner-Centered Teaching: Putting the Research on Learning into Practice.* Sterling, Virginia: Stylus Publishing.

DS Examiner. (14. 7 2014). *The Ultimate STEM Guide for Kids: 239 Cool Sites About Science, Technology, Engineering and Math*. Pridobljeno iz Master's in data science: https://www.mastersindatascience.org/blog/the-ultimate-stem-guide-for-kids-239-cool-sites-about-science-technology-engineering-and-math/

Eberman, A. R. (9. 4 2018). *Collaborative Learning: Why Coding Is The Best Field For Kids To Learn From Each Other.* Pridobljeno iz Tekkie Uni: https://tekkieuni.com/blog/collaborative-code-learning/

EU. (n. d.). *What is the EUPL?* Pridobljeno iz EUPL]European Union Public Licence]: https://eupl.eu/

Frey, N., & Fisher, D. (2011). *The Formative Assessment Action Plan: Practical Steps to More Successful Teaching and Learning.* Alexandria, VA: ASCD.

Gaebel, D. (27. 2 2018). *9 Real-Time Code Collaboration Tools for Developers.* Pridobljeno iz envatotuts+: https://webdesign.tutsplus.com/articles/real-time-code-collaboration-tools-for-developers--cms-30494

Geddis, A. N. (2006). Transforming subject-matter knowledge: the role of pedagogical content knowledge in learning to reflect on teaching. *International Journal of Science Education, 15*(6), 673-683.

GNU Operating System. (n. d.). *Various Licenses and Comments about Them.* Pridobljeno iz GNU Operating System: https://www.gnu.org/licenses/license-list.html

Gomes, A., & Mendes, A. (2019). *Problem solving in programming.* Pridobljeno iz https://pdfs.semanticscholar.org/1742/6220e02744ce9492bb4e56a0778333a79b18.pdf

Google. (n. d.). *Exploring Computational Thinking.* Pridobljeno iz Google for Education: https://edu.google.com/resources/programs/exploring-computational-thinking/

Grissom, S., McNally, M. F., & Naps, T. (2003). Algorithm visualization in CS education: comparing levels of student engagement. *SoftVis '03 Proceedings of the 2003 ACM symposium on Software visualization* (str. 87-94). San Diego, CA: ACM.

Gutschank, J. (2019). *Coding in STEM Education.* Berlin, Germany: Science on Stage Deutschland e.V. Pridobljeno iz https://www.science-on-stage.eu/images/download/Coding_in_STEM_Education_web.pdf

Guzdial, M. (2016). *Learner-Centered Design of Computing Education: Research on Computing for Everyone.* Morgan & Claypool.

Haberer, T. (8. 11 2017). *Why Learning to Code Should Be Collaborative.* Pridobljeno iz Atomicdust: https://www.atomicdust.com/learning-to-code/

Hemmendinger, D. (2000). *Object-oriented programming.* Pridobljeno iz Encyclopaedia Britannica: https://www.britannica.com/technology/object-oriented-programming

Hour of Code. (n. d.). *Educator Hour of Code: Lesson Plan Outline.* Pridobljeno iz Hour of Code: https://hourofcode.com/files/EducatorHourofCodeLessonPlanOutline.docx

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A Meta-Studyof AlgorithmVisualization E¡ectiveness. *Journal of Visual Languages and Computing, 13*, 259-290. doi:doi:10.1006/S1045-926X(02)00028-9

immersed. (n. d.). *Gamification versus game-based learning.* Pridobljeno iz Immersed games: http://www.immersedgames.com/gamification-vs-game-based-learning/

ISTE. (3. 1 2012). *Computational thinking: A digital age skill for everyone.* Pridobljeno iz Youtube: https://www.youtube.com/watch?v=VFcUgSYyRPg

Kapp, K. (2014). GAMIFICATION: Separating Fact From Fiction. *Chief Learning Officer*, 42-52. Pridobljeno iz http://ww.w.cedma-europe.org/newsletter%20articles/Clomedia/Gamification%20-%20Separating%20Fact%20from%20Fiction%20(Mar%2014).pdf

Kapp, K. M. (2012). *The Gamification of Learning and Instruction: Game-based Methods and Strategies for Training and Education.* San Francisco, CA: John Wiley & Sons, Inc.

Keller, J. M. (1987). Development and use of the ARCS model of instructional design. *Journal of instructional development, 10*(3), 2-10.

Khaled, A. (2013). Teacher-Centered Versus Learner-Centered Teaching Style. *The Journal of Global Business Management, 9*(1), 22-34. Pridobljeno iz http://www.jgbm.org/page/3%20Ahmed%20Khaled%20Ahmed.pdf

learning theories. (2005-2019). *ARCS model of motivational design theories (Keller).* Pridobljeno iz Learning theories: https://www.learning-theories.com/kellers-arcs-model-of-motivational-design.html

makeblock. (n. d.). *mBot.* Pridobljeno iz makeblok: https://www.makeblock.com/steam-kits/mbot

Massachusetts Institute of Technology. (n. d.). *Introduction to problem solving skills.* Pridobljeno iz CC/MIT: https://ccmit.mit.edu/problem-solving/

McCormack, J. (2005). *Topic 25: Software Maintenance and Re-engineering.* Pridobljeno iz CSE2305 Object-Oriented Software Engineering: http://users.monash.edu/~jonmc/CSE2305/Topics/13.25.SWEng4/html/text.html

Mechado, L. (9. 12 2017). *What is the best age to start learning programming?* Pridobljeno iz Quora: https://www.quora.com/What-is-the-best-age-to-start-learning-programming-1

Michael, D., & Chen, S. (2006). *Serious Games: Games That Educate, Train, and Inform.* Boston, MA: Thomson Course Technology PTR. Pridobljeno iz https://anagroudeva.files.wordpress.com/2013/06/serious_games__games_that_educate__train__and_inform.pdf

micro:bit. (n. d.). *Power your imagination with code.* Pridobljeno iz micro:bit: https://microbit.org/code/

Mills, A. (4. 10 2017). *What is STEM education?* Pridobljeno iz Phys.org: https://phys.org/news/2017-10-stem.html

Mishra, P., & Koehler, M. J. (2006). Technological Pedagogical Content Knowledge: A Framework for Teacher Knowledge. *Teachers College Record, 108*(6), 1017-1054.

MIT App inventor. (n. d.). *MIT App inventor*. Pridobljeno iz MIT App inventor: http://appinventor.mit.edu/explore/

Moos, C. M., & Brookhart, S. M. (2009). *Advancing Formative Assessment in Every Classroom: A Guide for Instructional Leaders.* Alexandria, VA: ASCD.

Moursund, D. (2007). *Introduction to Problem Solving in the Information Age.* Oregon: College of Education, University of Oregon. Pridobljeno iz https://pages.uoregon.edu/moursund/Books/IAE-PS/PS-in-IA.pdf

Nishida, T., Kanemune, S., Idosaka, Y., Namiki, M., Bell, T. C., & Kuno, Y. (2009). A CS unplugged design pattern. *40th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2009* (str. 231-235). Chattanooga, TN: ACM.

Nuryan Issayan, L. (2011). *Teacher-Centered Vs Learner-Centered Approach: Teacher Behavior.* LAP Lambert Academic Publishing.

Ohanesian, S. (31. 7 2018). *What is the difference between coding & programming?* Pridobljeno iz Julian Krinsky Camps & Programs: https://info.jkcp.com/blog/coding-vs-programming

Open Culture. (n. d.). *Free Textbooks: Computer Science.* Pridobljeno iz Open Culture: The best free cultural & educational media on the web: http://www.openculture.com/free-computer-science-textbooks

p.org. (2017). *What is Plagiarism?* Pridobljeno iz p.org: https://www.plagiarism.org/article/what-is-plagiarism

Packt. (n. d.). *Free programming ebooks and videos.* Pridobljeno iz Packt publishing: https://www.packtpub.com/packt/offers/free-learning

Peterson, S. K. (07. 11 2017). *What's the difference between open source software and free software?* Pridobljeno iz opensource.com: https://opensource.com/article/17/11/open-source-or-free-software

Petrík, J., Chudá, D., & Steinmüller, B. (2017). Source code plagiarism detection: The Unix way. *IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI).* Herl'any, Slovakia: IEEE.

Prottsman, K. (4. 12 2015). *Coding vs. Programming — Battle of the Terms!* Pridobljeno iz WebCite: http://www.webcitation.org/77YHuuIqu

Punie, Y. (2017). *DigComp 2.1: The Digital Competence Framework for Citizens.* Luxembourg: Publications Office of the European Union. Pridobljeno iz https://ec.europa.eu/jrc/en/publication/eur-scientific-and-technical-research-reports/digcomp-21-digital-competence-framework-citizens-eight-proficiency-levels-and-examples-use

Quora. (8. 4 2017). *Can anyone become a good programmer?* Pridobljeno iz Quora: https://www.quora.com/Can-anyone-become-a-good-programmer

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137–172.

Robogarden. (13. 06 2018). *How coding improve collaboration and cooperation?* Pridobljeno iz Robogarden: https://robogarden.ca/blog/how-coding-improve-collaboration-and-cooperation

Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using “Scratch” in five schools. *Computers & Education, 97*, 129-141.

Santrock, J. W. (2009). *Educational Psychology (4th edition).* New York: McGraw-Hill Education.

sape. (n. d.). *Accuracy.* Pridobljeno iz sape - Software and Programmer Efficiency Research Group: http://sape.inf.usi.ch/accuracy

Seifert, K., & Sutton, R. (2009). *Educational Psychology (Second Edition).* Zurich, Switzerland: The Saylor Foundation. Pridobljeno iz https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2012/06/Educational-Psychology.pdf

Sentance, S., & Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teacher’s perspective. *Education and Information Technologies, 22*(2), 469–495.

Shaw, M. (1992). *We Can Teach Software Better.* Pridobljeno iz Carnegie Mellon University School of Computer Science: http://www.cs.cmu.edu/~Compose/ftp/TchSWBetter.pdf

Shimic, G., & Jevremovic, A. (2010). Problem-based learning in formal and informal learning environments. *Interactive Learning Environments, 20*(4), 351-367.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review, 22*, 142-158. Pridobljeno iz http://myweb.fsu.edu/vshute/pdf/CT.pdf

Singapore Management University. (n.d.). *Lesson planning.* Pridobljeno iz Teaching@SMU: https://cte.smu.edu.sg/approach-teaching/integrated-design/lesson-planning

Sommerville, I. (2011). *Software Engineering.* Boston: Addison-Wesley.

Sookdeo, C. (26. 4 2012). *Motivational Theories In Education.* Pridobljeno iz SlideShare: https://www.slideshare.net/ChristinaSookdeo/motivational-theories-12696671

STEAM Powered Family. (19. 3 2018). *What is STEM and STEAM? A guide for parents and educators.* Pridobljeno iz STEAM Powered Family: https://www.steampoweredfamily.com/education/what-is-stem/

Taylor, E., Breed, M., Hauman, I., & Homann, A. (2013). Choosing learning methods suitable for teaching and learning in computer science. *IADIS International Conference e-Learning* (str. 74-82). Prague, Czech Republi: IADIS - International association for development of the information society.

Techno Kids. (n. d.). *TechnoCode*. Pridobljeno iz Techno Kids: K-12 Technology projects: https://www.technokids.com/store/middle-school/technocode/scratch-for-kids.aspx

technopedia. (n. d.). *Code Efficiency.* Pridobljeno iz technopedia: https://www.techopedia.com/definition/27151/code-efficiency

TechTarget network. (n. d.). *Robot.* Pridobljeno iz TechTarget network: https://searchenterpriseai.techtarget.com/definition/robot

Texas Tech University. (n. d.). *ARCS model of motivation.* Pridobljeno iz The Texas A&M University System: http://www.tamus.edu/academic/wp-content/uploads/sites/24/2017/07/ARCS-Handout-v1.0.pdf

Togyer, J., & Wing, J. M. (n. d.). Research Notebook: Computational Thinking--What and Why? *The LINK - The magazine of Carnegie Mellon University's School of Computer Science*. Pridobljeno iz https://www.cs.cmu.edu/link/research-notebook-computational-thinking-what-and-why

Turnitin. (2017). *Plagiarism and Programming: How to Code Without Plagiarizing.* Pridobljeno iz Turnitin: https://www.turnitin.com/blog/plagiarism-and-programming-how-to-code-without-plagiarizing-2

tutorialzine. (2018). *10 Free Programming Books You Should Read in 2018.* Pridobljeno iz tutorialzine: https://tutorialzine.com/2018/01/10-free-programming-books-you-should-read-in-2018

Usmani, F. (28. 1 2019). *Three Tools to Estimate Activity Duration.* Pridobljeno iz PM Study Circle: https://pmstudycircle.com/2012/06/three-tools-estimate-activity-duration/

Vero, E., & Puka, E. (2017). The Importance of Motivation in an Educational Environment. *Formazione & Insegnamento XV, 15*(1), 57-66.

Video. Hour of Code Intro. (n. d.). *Frozen - Hour of Code Introduction.* Pridobljeno iz Code.org: https://studio.code.org/s/frozen/stage/1/puzzle/1

Vitkutė Adžgauskienė, D., & Vidžiūnas, A. (2012). Problems in Choosing Tools and Methods for Teaching Programming. *Informatics in Education, 11*(2), 271–282.

w3school.com. (n. d.). *Python Tutorial.* Pridobljeno iz w3school.com: https://www.w3schools.com/python/default.asp

Wagner, N. R. (2000). *Plagiarism by Student Programmers.* Pridobljeno iz Department of Computer Science, The University of Texas at San Antonio: http://www.cs.utsa.edu/~wagner/pubs/plagiarism0.html

Wang, L.-C., & Chen, M.-P. (2010). The effects of game strategy and preference-matching on flow experience and programming performance in game-based learning. *Innovations in Education and Teaching International, 47*(1), 39-52.

Waters, A., Bassendowski, S., & Petrucka, P. (2008). Serious Games for Students in Healthcare: Engaging a Technically Inclined Generation. *Canadian Journal of Nursing Informatics, 3*(4), 16-27.

WebFX. (n. d.). *15 Free Books for People Who Code.* Pridobljeno iz WebFX Digital Marteking That DRivers Results: https://www.webfx.com/blog/web-design/free-books-code/

Weimer, M. (2002). *Learner-Centered Teaching.* New York: Jossey-Bass John Wiley & Sons.

White, D. (2014). What is STEM education and why is it important? *Florida Association of Teacher Educators Journal, 14*, 1-8.

Williams, K. C., & Williams, C. (2011). Five Key Ingredients for Improving Student Motivation. *Research in Higher Education Journal, 12*(1), 104-122.

Wilson, A., Hainey, T., & Connolly, T. (2013). Using Scratch with Primary School Children: An Evaluation of Games Constructed to Gauge Understanding of Programming Concepts. *International Journal of Game-Based Learning, 3*, 93-109.

Wilson, L. O. (2016). *Anderson and Krathwohl – Bloom's Taxonomy Revised.* Pridobljeno iz The Second Priciple: https://thesecondprinciple.com/teaching-essentials/beyond-bloom-cognitive-taxonomy-revised/

Wing, J. M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology,, 25*(2), 7-14. Pridobljeno iz http://www.cs.cmu.edu/~wing/publications/Wing17.pdf