Didaktične metode pri poučevanju programiranja

Jan Robas

Šolski center Kranj
Slovenska cesta 27
4207 Cerklje na Gorenjskem

janrobas@gmail.com

**POVZETEK**

V tem prispevku obravnavamo nekaj didaktičnih metod s konkretnimi primeri iz poučevanja programiranja. Pri tem smo izbrali didaktične metode, ki so po našem mnenju za poučevanje programiranja najbolj primerne. S tem postavimo smernice za načrtovanje učnih ur pri programiranju. Pri tem še posebej izpostavimo individualno delo, s katerim si študentje skonstruirajo znanje.

**Ključne besede**

Didaktične metode, osnove programiranja, poučevanje.

**ABSTRACT**

In this paper, we discuss didactic methods with concrete examples from teaching basic programming skills. We have chosen didactic methods that we think are best suited for teaching programming. Thus guidelines for lesson planning have been set. Individual work which allows students to construct their own knowledge has been emphasized.

**Keywords**

Didactic methods, basic programming, teaching.

# UVOD

Računalniki so stroji, ki nam pomagajo razumeti svet in reševati probleme. Učenje programiranja pomeni tako učenje načina razmišljanja kot učenje sintakse programskega jezika. Čeprav je prejšnje bolj pomembno kot slednje, je oboje pomemben del znanja programiranja.

Kot velja za vsako organizirano učenje, je tudi k učenju programiranja potrebno pristopiti didaktično smiselno in strokovno. Didaktične metode, ki so se nam izkustveno izkazale za najbolj pomembne in učinkovite, so: razgovor, razlaga, laboratorijsko-eksperimentalne metode, diskusija in povzemanje.

# DIDAKTIČNE METODE

## Razgovor

Pri prvem srečanju z vsakim študentom naredimo kratek razgovor, s pomočjo katerega ugotovimo njegovo predznanje. Tako ugotovimo, kakšen je najmanjši skupni imenovalec predznanja in temu prilagodimo razlago osnovi. Označimo si študente, ki imajo nadpovprečno znanje in študente, ki predznanja nimajo. Študentom z več predznanja tako ponudimo dodatne izzive, hkrati pa se individualno več posvečamo študentom brez predznanja in s tem poskrbimo, da usvojijo osnove. Izkazalo se je tudi, da študentje z več predznanja radi priskočijo na pomoč kolegom, ki jo potrebujejo in jim je to še dodatna vzpodbuda, da bodo tudi oni lahko zmogli usvojiti zahtevane vsebine.

Kasneje pri študentih večkrat vrednotimo pridobljeno znanje in rezultate dela pri predmetu. S tem se izognemo zanašanju na morebitno napačno ocenitev predznanja na začetku. S povratnimi informacijami ustrezno prilagodimo razlago oziroma se individualno posvetujemo s študenti, ki izstopajo od povprečja.

## Razlaga

Smiselno je, da pri obravnavanju nove snovi s področja programiranja začnemo z razlago. Uvodoma gre za pripovedovanje, pri čemer študentom predstavimo probleme, ki jih bomo reševali s pomočjo na novo pridobljenega znanja. Pri tem poskušamo na čimbolj zanimiv način predstaviti uporabnost in konkretno aplikacijo znanja, ki ga bodo usvojili. Nato se osredotočimo na razlago novih pojmov in načinov programiranja.

Sledi obrazložitev, pri kateri hkrati uporabimo metodo prikazovanja. Pri tem razložimo nova orodja, procese in pojme ter jih obenem predstavimo s projeciranjem na platno. Študentom pokažemo nekaj primerov, ki jih rešimo skupaj. Na platno projeciramo predvsem dejansko pisanje kode, s katero rešujemo kratke naloge, na podoben način kot učitelj pri matematiki na tabli rešuje enačbe. Vmes povemo, s kakšnim načinom razmišljanja smo prišli do rešitve. Gre za enostavnejše primere, ki podprejo razumevanje razloženih pojmov. S tem poskušamo študente navaditi na pravilen pristop k programerskim problemom. Študentje delajo zapiske s prepisovanjem kode, ki jo vmes poganjajo na svojih računalnikih in s tem preverjajo njen rezultat. Spodbujamo jih, da začetne spremenljivke nastavijo poljubno in preverijo, če program še deluje in vrne pravilen rezultat. S tem študentje pridobijo ustrezno znanje za samostojno reševanje problemov.

Na sliki 1 je primer kode, ki je projecirana. Koda v tekstovnem načinu izriše kvadrat s pomočjo zvezdic (znak “\*”), diagonalno pa z znakom “#”. Študente nato spodbudimo, da s spreminjanjem pogojev v zankah (i < 10, j < 10) spremenijo velikost kvadrata oziroma pravokotnika. Poleg tega lahko omenjena znaka zamenjajo s poljubnima drugima znaka in spremenijo pogoj v vejitvi (i == j). Študentje po poljubni spremembi poženejo program in, če se program prevede, dobijo drugačen kvadrat. Če se program zaradi sintaktičnih napak ne požene ali pa pride do nepričakovanega rezultata, študentom pomagamo razumeti in popraviti napako.

Slika 1: Primer kode, ki je projecirana.



## Laboratorijsko–eksperimentalne metode

Ideja konstruktivistične teorije učenja je, da študentje sami razvijejo (skonstruirajo) svoje znanje [1]. Pri stilu učenja, ki sledi temu pogledu, ima posamezen študent več odgovornosti, da se nauči, kot pri klasičnem pristopu. To pa ne pomeni, da je vloga učitelja lažja, saj mora le-ta študente voditi v pravo smer in jih motivirati. Klasičen pristop s pasivnim učenjem ponuja malo možnosti za sodelovanje študentov s pomočjo eksperimentalnih metod in diskusije [2].

Učenje programiranja ni zgolj pomnenje dejstev in njihova interpretacija. Pri programiranju ima kompleksen problem več rešitev, ki so si lahko med sabo zelo različne. Da pridemo do rešitve, potrebujemo pravi način razmišljanja. Do tega najbolj učinkovito pridemo s samostojnim reševanjem problemov. Študentje pri vajah samostojno rešujejo probleme in s tem usvajajo način razmišljanja ter utrjujejo znanje, ki so ga pridobili ob razlagi. Vloga učitelja pri tem je, da jim pomaga, ko se jim zatakne in razloži nejasnosti, ki se tičejo razumevanja problemov in novih pojmov.

Zelo preprost primer naloge je izpis seštevka vseh števil, ki so shranjena v določeni spremenljivki. Pri tem študentom pripravimo primer deklaracije in inicializacije spremenljivke, kot je naprimer spremenljivka tabela na sliki 2 (int [] tabela = { 1,2,3 }) in jim povemo, da želimo, da program izpiše seštevek teh števil. Na sliki 2 je primer ustrezne rešitve naloge. Program najprej nastavi spremenljivko seštevek na 0, nato pa ji po vrsti prišteje vsa števila, ki so shranjena v spremenljivki tabela. Seštevanje opravimo znotraj zanke for, ki gre po vrsti po vseh indeksih, ki naslavljajo vrednosti v tabeli. Študentje morajo pri tem znati pravilno sintakso. Če pozabijo zaklepaj ali podpičje, se program ne bo prevedel in pognal, temveč bo računalnik javil napako. Gre za sintaktične napake. Študentje morda razmišljajo pravilno, vendar še niso usvojili sintakse. Druga vrsta napak so semantične napake, torej gre za napačno napisan postopek oziroma algoritem, ki se sicer uspešno požene, vendar javi napačen rezultat. Pri tem gre lahko za nerazumevanje navodil, manjšo napako ali pa študent še ni usvojil osnovnega programerskega načina razmišljanja. Bolj zanimivi primeri so, ko program izpiše pravi rezultat, vendar ima rešitev pomanjkljivosti. Trije taki primeri so prikazani v tabeli 1.

Slika 2: Primer kode, ki sešteje števila v tabeli.



Tabela 1: Primer slabih rešitev, ki izpišejo pravi rezultat.

|  |  |
| --- | --- |
| Rešitev | Razložitev napake |
|  | Ročen izpis rezultata. Če spremenimo elemente v tabeli, bo rezultat napačen. Študent zagotovo ne razume navodil. |
|  | Ročno naslavljanje indeksov elementov v tabeli. Če spremenimo število elementov v tabeli, rešitev ne bo več pravilna. Če imamo v tabeli 1000 elementov, bo koda zelo dolga, saj ne uporablja zank, ki so temu namenjene. Študent ne zna uporabljati zank oziroma se ne zaveda, kdaj je uporaba zank primerna. |
|  | Ročno nastavljena meja za najvišji indeks v tabeli. Če spremenimo število elementov v tabeli, rešitev ne bo več pravilna. Študent razume nalogo in zna uporabljati zanke, vendar ni pomislil na primernost rešitve za drugačne testne primere. |

## Diskusija in povzemanje

Ko študentje zaključijo svoje delo, je zelo pomembna diskusija. Za vsak problem, ki ga študenti rešujejo, namreč obstaja več pravilnih rešitev. Kljub temu je lahko ena pravilna rešitev boljša od druge, prav tako pravilne rešitve.

Ena od bolj pomembnih lastnosti programske kode je njena berljivost oziroma razumljivost. Pri tem je pomembno tudi poimenovanje konstruktov v programskem jeziku, kot so na primer spremenljivke in imena metod. Dobro razumljiva in logično strukturirana programska koda omogoča lažjo kasnejšo nadgradnjo in lažje sodelovanje z drugimi programerji v primeru, ko jih več dela na istem projektu.

Pomembna lastnost programske rešitve je tudi njena performanca. Pri tem se predvsem osredotočamo na časovno in prostorsko kompleksnost algoritma. Dobro napisana rešitev lahko dela hitreje in porabi manj pomnilnika. Te razlike so lahko zelo velike in lahko pomenijo, da bo v enem primeru algoritem terjal nekaj sto let, v drugem primeru pa nekaj milisekund, da se izvede. V prejšnjem primeru je rešitev neuporabna. Lahko pa gre za manjše razlike, ki se odražajo na primer v porabi energije, kar je predvsem pomembno za mobilne naprave.

Na koncu je vse odvisno od konteksta. Velikokrat moramo narediti kompromis med časovno in prostorsko zahtevnostjo ali pa med performanco in berljivostjo kode. Pri tem so lahko različni načini razmišljanja pravilni. Če bi bile rešitve enostavne in matematično izpeljive, se gotovo z učenjem programiranja ne bi ukvarjali. S študenti zato na koncu zberemo skupaj različne zanimive rešitve istih problemov, jih projeciramo na projektor in diskutiramo o njihovih prednostih in slabostih. S tem študente spodbujamo h kritičnemu razmišljanju glede algoritmov in programskih rešitev, ki so v literaturi in na spletu, ter jih motiviramo, da pri reševanju programerskih problemov razmišljajo o alternativnih in inovativnih pristopih.

# ZAKLJUČEK

Programiranje si lahko predstavljamo kot umetnost [3]. Problemi v resničnem svetu so raznoliki ter terjajo različne pristope in programske rešitve. Vloga učitelja programiranja je, da študente nauči osnovnih pristopov, hkrati pa jih spodbuja h kritičnemu razmišljanju in iskanju alternativnih rešitev. Pri zelo preprostih primerih je sicer prostora za različne rešitve manj. Kljub temu pa je pomembno, da vzpodbujamo tudi manj optimalne rešitve, pri katerih so se študentje znašli po svoje. Cilj je, da so študentje na koncu zmožni samostojno reševati probleme. Pri tem je zelo pomembno individualno delo. Pomembno je tudi, da pri slabših rešitvah izpostavimo pomanjkljivosti in s tem študentom pokažemo pravo smer. Pomembno je, da študentom pripravimo dobro podlago za razvoj znanja, s pomočjo katerega bodo prišli do ustreznih rešitev za prihodnost.

# REFERENCE

1. Bada, S.O. and Olusegun, S., 2015. Constructivism learning theory: A paradigm for teaching and learning. Journal of Research & Method in Education, 5(6), pp.66-70.
2. Michel, N., Cater III, J.J. and Varela, O., 2009. Active versus passive teaching styles: An empirical study of student learning outcomes. Human resource development quarterly, 20(4), 397-418.
3. Knuth, D.E., 1974. Computer programming as an art. Communications of the ACM, 17(12), pp.667-673.