2.9 Programiranje: pomoč pri izbranih nalogah

Učiti se programirati, ne da bi kodo pisala UI

Predmet: programiranje · **Razred:** 9. razred (14–15 let) · **Trajanje:** ena dvojna ura (90 min) · **UI:** pomoč **Z** UI

Kompetence, povezane z UI (okvir DUAL.AI.TEACHer) – raven učitelja	Predmetno specifični učni cilji – raven učenca
AF-TL-2b *(Temelji UI in njihova uporaba × Poučevanje in učenje, raven 2 – reflektirano izvajanje)*: »Učitelji lahko organizirajo elemente učne ure, ki vključujejo uporabo UI z jasnimi pedagoškimi vlogami in strategijami za preverjanje elementov izhoda.« **AF-FC-2b** *(Temelji UI in njihova uporaba × Spodbujanje digitalne kompetence učencev na področju UI, raven 2 – reflektirano izvajanje)*: »Učitelji lahko izvajajo usmeritve, ki učencem pomagajo razvijati praktične veščine UI, pri čemer naloge prilagajajo predznanju učencev in razpoložljivim orodjem UI.«	Ob koncu ure učenci znajo: • uporabiti UI za ponavljajoče se ali podporne naloge pri programiranju, kot sta zamikanje kode ali pisanje komentarjev; • od modela UI zahtevati predloge za izboljšave kode, ne da bi jim ta dal samo kodo, tako da lahko sami preverijo svoje znanje; • kritično ovrednotiti model UI, ko ga vprašajo, kako bi optimiziral dano kodo.

> **Ključno sporočilo:** Učenci morajo poznati osnove programiranja, če želijo vrednotiti rezultate, ki jih da model. Zato naj učenci v prvih korakih učenja programiranja modele UI uporabljajo za podporne in ponavljajoče se naloge, povezane s programiranjem, ne pa za ustvarjanje kode. Model lahko uporabijo tudi za iskanje možnih izboljšav kode in tako preizkusijo svoje programerske sposobnosti.



Slika 1: »Učiti se programirati, ne da bi kodo pisala UI« (slika, ustvarjena z UI, Google Gemini 3.6 Thinking)

Vsebina

Poučevanje programiranja se spreminja v okolju, v katerem lahko sistemi UI ustvarjajo, razlagajo in pregledujejo kodo. Ta orodja sicer odpirajo nove možnosti za učenje, hkrati pa prinašajo pomemben izobraževalni izziv: učenci potrebujejo dovolj znanja programiranja, da lahko presodijo, ali je predlog, ki ga ustvari UI, pravilen, učinkovit ali sploh relevanten za nalogo. Brez osnovnega razumevanja programerskih pojmov lahko učenci sprejmejo napačne rešitve, spregledajo napake ali ne prepoznajo boljših možnosti. Zato ostaja učenje temeljev programiranja bistveno, tudi ko so orodja UI na voljo.

Pri uvodnih predmetih programiranja naj ima UI vlogo podpornega učnega orodja in ne nadomestnega programerja. Učenci lahko UI učinkovito uporabljajo za ponavljajoče se ali preprostejše naloge, ki ne zmanjšujejo miselnega napora, potrebnega za učenje programiranja. Primeri so oblikovanje in zamikanje kode, ustvarjanje dokumentacijskih komentarjev, razlaga sporočil o napakah, povzemanje pravil sintakse ali pomoč pri razumevanju namena posameznih programskih konstruktov. Takšna uporaba učencem omogoča, da se osredotočijo na logiko, algoritme in procese reševanja problemov, ki so jedro programerske kompetence.

Druga koristna uporaba UI je pregled kode. Namesto da bi model UI prosili, naj napiše rešitev, lahko učenci najprej napišejo svojo kodo in nato model prosijo, naj prepozna možne izboljšave. UI lahko predlaga jasnejša imena spremenljivk, boljšo organizacijo kode, manj podvajanja, boljšo berljivost ali drugačne pristope, ki jih učenci lahko samostojno ovrednotijo. Tako UI postane partner za povratno informacijo: učenci lahko svoje razmišljanje primerjajo z zunanjimi predlogi ter presodijo prednosti in slabosti svojih rešitev.

Povratne informacije, ki jih ustvari UI, pa ne smemo sprejemati nekritično. Modeli lahko predlagajo spremembe, ki so nepotrebne, neučinkovite ali temeljijo na napačnih predpostavkah o namenu kode. Osrednji cilj te učne ure je zato razvijati sposobnost učencev, da predloge UI kritično ovrednotijo. Učenci se naučijo model UI vprašati ne le, katere izboljšave predlaga, temveč tudi, zakaj meni, da so te spremembe koristne. S primerjavo utemeljitve modela z uveljavljenimi načeli programiranja lahko presodijo, ali predlagana optimizacija kodo dejansko izboljša ali pa gre le za drugačno oblikovalsko odločitev.

Ta učna ura UI obravnava kot orodje, ki podpira refleksijo, samovrednotenje in izboljševanje kakovosti kode, ne pa ustvarjanja kode. Učenci vadijo pisanje lastnih programov, uporabo UI za podporne naloge, pridobivanje povratnih informacij o svojem delu in kritično analizo kakovosti prejetih odgovorov. Končni cilj je, da učenci postanejo boljši programerji in hkrati bolj ozaveščeni uporabniki sistemov UI.

Potek učne ure

Faza	Čas	Dejavnost
Uvod	10 min	Učitelj pokaže dve kratki rešitvi istega problema v obliki kode: eno je napisal učenec, drugo je ustvaril model UI. Učenci razpravljajo, katera rešitev se zdi jasnejša, učinkovitejša ali lažje razumljiva.
Uvodna razlaga	10 min	Učitelj predstavi ustrezne načine uporabe UI pri poučevanju programiranja in poudari razliko med podpornimi nalogami, pregledom kode in ustvarjanjem kode.
Razisko vanje	15 min	Učenci v parih rešujejo kratko programersko vajo in ugotavljajo, katere naloge bi lahko ustrezno prepustili pomočniku UI in katere morajo ostati njihova odgovornost.
Krog izboljšav	35 min	Učenci napišejo lastno rešitev in izvedejo 2 do 3 kroge napiši → vprašaj → ovrednoti → popravi, pri čemer UI uporabljajo le za podporne naloge in predloge izboljšav.
Refleksij a	20 min	Razprava s celotnim razredom o uporabnosti in omejitvah povratnih informacij UI, s poudarkom na tem, kako so učenci presodili, ali so bili predlogi koristni ali ne.

Vodilna vprašanja: dodajanje plasti kompleksnosti

Učenci v 35-minutnem krogu izboljšav izvedejo toliko krogov, kot dopušča čas. Vsak krog sledi istemu postopku v štirih korakih: napiši → vprašaj → ovrednoti → popravi.

1. **1. krog – uporaba UI za podporne naloge**
   - Napiši: Samostojno rešite kratko programersko vajo.
   - Vprašaj: UI prosite za pomoč pri neprogramerski nalogi, povezani s kodo, na primer:
     - izboljšanje zamikov,
     - ustvarjanje komentarjev,
     - razlaga sporočila o napaki,
     - opis tega, kaj počne funkcija.
   - Ovrednoti: Ali razlaga UI natančno opisuje kodo? Ali so ustvarjeni komentarji uporabni in razumljivi?

- Popravi: Vključite koristne spremembe, pri tem pa poskrbite, da logika programa v celoti ostane vaše lastno delo.

2. **2. krog – pridobivanje predlogov za izboljšave**
   - Napiši: Preglejte svoj dokončani program in poiščite dele, ki bi jih bilo mogoče izboljšati.
   - Vprašaj: Prosite za predloge izboljšav, ne da bi UI dovolili, da kodo prepiše. Na primer:

»Predlagaj tri načine, kako izboljšati berljivost in vzdrževanje tega programa, vendar ne podaj nadomestne kode.«

1. - Ovrednoti: Kateri predlogi se zdijo koristni? Kateri bi na program le malo vplivali?
   - Popravi: Uvedite izboljšave, s katerimi se strinjate, in preverite, ali program še vedno deluje pravilno.
2. **3. krog – kritično vrednotenje nasvetov za optimizacijo**
   - Napiši: Izberite del svoje kode, ki izvaja ponavljajočo se nalogo ali uporablja zanke.
   - Vprašaj: Prosite za predloge optimizacije in od UI zahtevajte, da svoja priporočila utemelji.

Primer poziva:

»Pojasni, kako bi lahko to kodo optimizirali in zakaj bi posamezna predlagana optimizacija izboljšala delovanje ali berljivost. Ne podaj optimizirane kode.«

- ◦ Ovrednoti: Ali so predlagane optimizacije res koristne? Katere so podprte z jasno utemeljitvijo? Katere se zdijo nepotrebne ali težko upravičljive?
  ◦ Popravi: Odločite se, katera priporočila boste uvedli, in pojasnite svojo odločitev.

Vprašanja za refleksijo:

- Kateri predlogi UI so bili najbolj koristni za izboljšanje vaše kode?
- Ali je UI podala predloge, s katerimi se niste strinjali? Zakaj?
- Kako ste presodili, ali je bila optimizacija smiselna?
- Kdaj je UI v procesu programiranja najbolj v pomoč?
- Katere programerske naloge naj še vedno opravlja učenec?
- Zakaj je za vrednotenje povratnih informacij, ki jih ustvari UI, potrebno znanje programiranja?
- Kako se lahko učenci pri učenju programiranja izognejo prevelikemu zanašanju na orodja UI?

Zaključno vprašanje (za fazo refleksije):

Koliko znanja programiranja je potrebnega, da lahko presodimo, ali predlog UI program dejansko izboljša, in kakšna tveganja nastanejo, ko se uporabniki zanašajo na nasvete UI, ne da bi sami razumeli kodo?

Gradiva

- En prenosnik na par učencev s programskim okoljem
- Dostop do klepetalnega pomočnika UI ali lokalno nameščenega jezikovnega modela
- Kratka programerska vaja, primerna ravni izkušenj učencev
- Delovni list, ki sledi ciklu napiši → vprašaj → ovrednoti → popravi
- Vprašanja za refleksijo, osredotočena na pismenost na področju UI in kritično vrednotenje
- Neobvezni primeri kode, ki ponazarjajo dobre in slabe programerske prakse

---

Revision #7
Created 2026-10-20 15:28:41 UTC by Alexander
Updated 2026-10-29 14:15:49 UTC by Jannik