UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Igor Efremov

CODERUNNER: ORODJE ZA AVTOMATSKO PREVERJANJE ZNANJA IZ PROGRAMIRANJA

Diplomska naloga

Ljubljana, 2016

ZAHVALA

Zahvaljujem se mentorju mag. Matiji Lokarju za strokovno pomoč, podporo, dosegljivost ter potrpežljivost pri pisanju diplomske naloge. Zahvaljujem se tudi somentorju dr. Gregorju Jeršetu za strokovno pomoč in moralno podporo. Največja zahvala gre družini, prijateljem in partnerki, ki so me v času študija podpirali in mi stali ob strani.

Kazalo

Program diplomske naloge

V diplomski nalogi predstavite orodje za avtomatsko preverjanje programske kode CodeRunner. Opišite način namestitve in kako s tem orodjem sestavljamo ustrezne naloge za programski jezik Python. V sklopu diplomske naloge sestavite tudi osnovno banko vprašanj, ki jo lahko uporabimo za sestavljanje testov, ki preverjajo osnovno poznavanje programskega jezika Python.

mentor: viš. pred. mag. Matija Lokar

somentor: dr.Gregor Jerše

Povzetek

V praksi se je izkazalo, da se programiranja najlažje naučimo tako, da napišemo čim več programov. Tako iz leta v leto narašča število spletnih učilnic in orodij, ki poleg učnega gradiva ponujajo še preverjanje znanja na praktičnih nalogah v obliki kvizov. V ta namen so razvili tudi orodje CodeRunner.

V diplomski nalogi smo predstavili uporabo orodja CodeRunner in njegovo namestitev v sistem Moodle. Podrobneje smo predstavili sestavljanje vprašanj in testnih primerov, ki se uporabljajo za avtomatsko preverjanje vprašanj. Sestavni del diplomske naloge je tudi zbirka sestavljenih vprašanj, razporejenih v različne kategorije. Vprašanja lahko uporabimo za sestavljanje kvizov, ki preverjajo osnovno znanje programiranja v programskem jeziku Python 3.Vprašanja so priložena k diplomski nalogi v obliki *xml* datotek.

Math. Subj. Class. (2010): 68N19

Ključne besede: CodeRunner, programiranje, predloge, testni primeri, programiranje, avtomatsko preverjanje znanja iz programiranja, spletno izobraževanje

Keywords: CodeRunner, programming, templates, test cases, automatic assessing of computer programming skills, online education

1 Uvod

CodeRunner je orodje za avtomatsko preverjanje znanja iz programiranja, ki deluje kot vtičnik v sistemu Moodle. V diplomski nalogi se bomo osredotočili na orodje CodeRunner s predpostavko, da sistem Moodle že poznamo. Več o sistemu Moodle si lahko preberemo v diplomski nalogi *Kvizi v spletni učilnici Moodle*(Gerenčer, 2008).

1.1 Orodja za avtomatsko preverjanje znanja iz programiranja

Programiranje je veščina, ki se je ni lahko naučiti. Gregor Jerše, Sonja Jerše, Matija Lokar in Matija Pretnar so avtorji članka (Preparing programming exercises with efficient automated validation tests, 2015), v katerem trdijo, da se je v praksi izkazalo, da se programiranja najlažje naučimo tako, da sami napišemo čim več programov oziroma rešimo čim več nalog iz programiranja. Dober in hiter odziv pri oddaji nalog je ključnega pomena za hitro napredovanje. V prizadevanju, da učitelji prihranijo čas, je bilo razvitih veliko orodij za avtomatsko preverjanje znanja iz programiranja.

Nekaj teh orodij deluje s pomočjo spletnih strežnikov, ki so postavljeni v ta namen. Primer takega orodja sta Codechef (Codechef, 2009) ali Putka (Putka, 2004). Študent pošlje svojo kodo na strežnik, le-ta jo nato požene, preveri njeno delovanje in takoj poroča o morebitnih napakah brez učiteljevega posega. S takšnim odzivom študenti hitreje napredujejo, saj so nemudoma seznanjeni z morebitnimi napakami v njihovih kodah.

Tak pristop zahteva precej močan strežnik, saj mora izvajati in preverjati poslane programe vseh študentov, ki pa svoje rešitve pogosto pošiljajo ob istem času. Obenem je izrednega pomena previdnost glede varnosti, saj pošiljamo na strežnik pošiljamo zunanjo kodo. Ti strežniki imajo tudi pomanjkljivost, in sicer učiteljem običajno ne dopuščajo vpogleda v študentove odgovore. Študenti so tako izpostavljeni visoki ravni samostojnega učenja in eksperimentiranja, čeprav je večina takih nalog na osnovni ravni. Morebitna težava se kaže tudi v tem, da študenti za pripravo odgovora (programa) ne morejo uporabiti svojega programskega okolja, ker strežnik zahteva drugega.

Vse zgoraj omenjene težave privedejo do tega, da se večina teh orodij uporablja v okvirjih ustanov, kjer so bila ustvarjena. Zaradi različnega pristopa učenja in nezmožnosti prilagajanja pogosto niso uporabna na drugih lokacijah.

Primer dobrega učnega orodja, ki skorajda nima zgoraj naštetih pomanjkljivosti, je sistem Moodle. Učitelj ima vpogled v vso študentovo aktivnost v učilnici. V svoji bazi ima več vtičnikov za različne tipe vprašanj. Moodlovo bazo lahko nadgradimo z vtičnikom **CodeRunner,** orodjem za avtomatsko preverjanje znanja iz programiranja. Orodje CodeRunner uporablja za izvedbo nalog samostojen strežnik Jobe, ki premore izvajanje številnih nalog hkrati, prav tako pa je varen za uporabo, saj ga lahko zaščitimo.

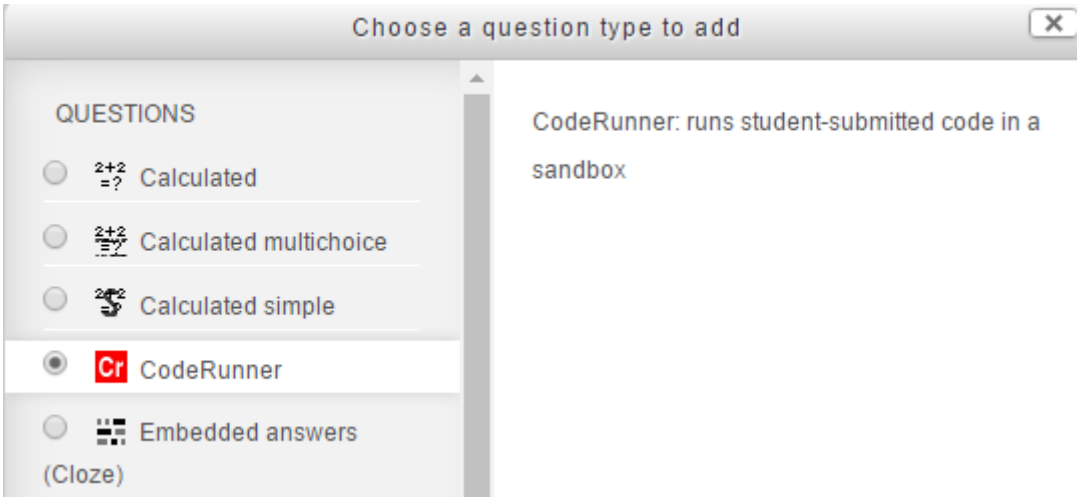
V diplomski nalogi bomo predstavili uporabo orodja CodeRunner, ki ga je zasnoval Richard Lobb s pomočjo Jenny Harlow z Univerze Canterbury na Novi Zelandiji (CodeRunner). Ogledali si bomo osnovne značilnosti tega vtičnika in postopek njegove namestitve. Prav tako bomo predstavili, kako s strani študenta poteka reševanje testa, ki ga sestavljajo naloge, pripravljene s sistemom CodeRunner. Glavnina opisovanja bo posvečena opisu, kako se sestavijo različna vprašanja, ki se kasneje lahko uporabijo pri sestavljanju testa. Sestavni del diplomske naloge je tudi banka vprašanj, pripravljena v okolju Moodle. S pomočjo teh vprašanj lahko sestavimo različne kvize, ki pokrivajo osnovni tečaj iz programskega jezika Python (Python, 2016).

Zaradi težav z nepopolnim prevodom tako sistema Moodle kot tudi samega orodja CodeRunner v slovenščino smo uporabljali nastavitev sistema Moodle v angleškem jeziku. Vse slike so tako povzete iz sistema, kjer je Moodle nastavljen na uporabo angleščine.

2 CodeRunner

2.1 Kaj je CodeRunner

CodeRunner je brezplačni odprtokodni vtičnik, ki se uporablja za razširitev sistema Moodle. Ko ga namestimo, dobimo v Moodlu poleg ostalih vgrajenih tipov vprašanj (več izbire, kratek odgovor, drži/ne drži...)še en dodaten tip (Slika 1).



Slika 1: CodeRunner je nov tip vprašanja

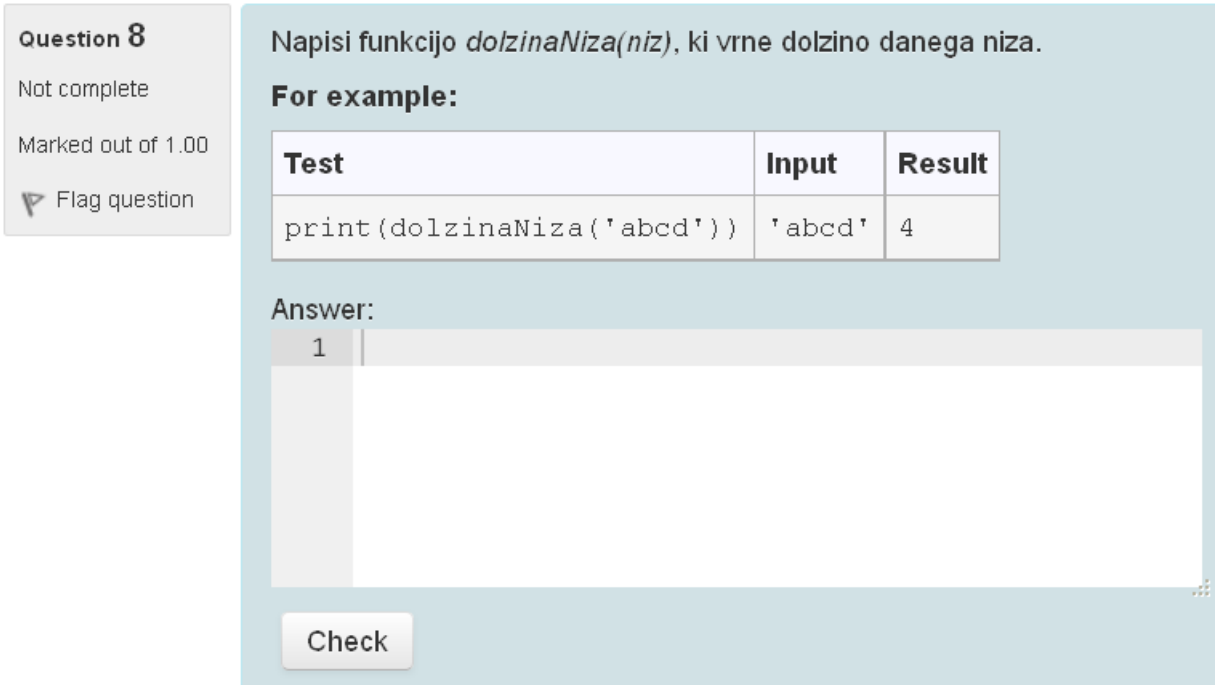
Če si izberemo tak tip vprašanja, lahko sestavljavec vprašanja (v nadaljevanju učitelj) sestavi vprašanje, katerega odgovor je ustrezna programska koda. Trenutno podpira naslednje programske jezike:

- Python 2 in 3
- C
- JavaScript
- PHP5
- Octave(Matlab)

V nadaljevanju si bomo podrobneje pogledali sestavljanje vprašanj v programskem jeziku **Python3.**

Omeniti je potrebno, da je možna nadgradnja vtičnika z načinom podpiranja drugih programskih jezikov. Ker je koda odprta, lahko to za želeni programski jezik naredimo sami. To je toliko lažje, saj je sama koda dobro dokumentirana in ima pregledno ter preprosto strukturo.

Slika 2 prikazuje primer vprašanja, sestavljenega v CodeRunnerju.

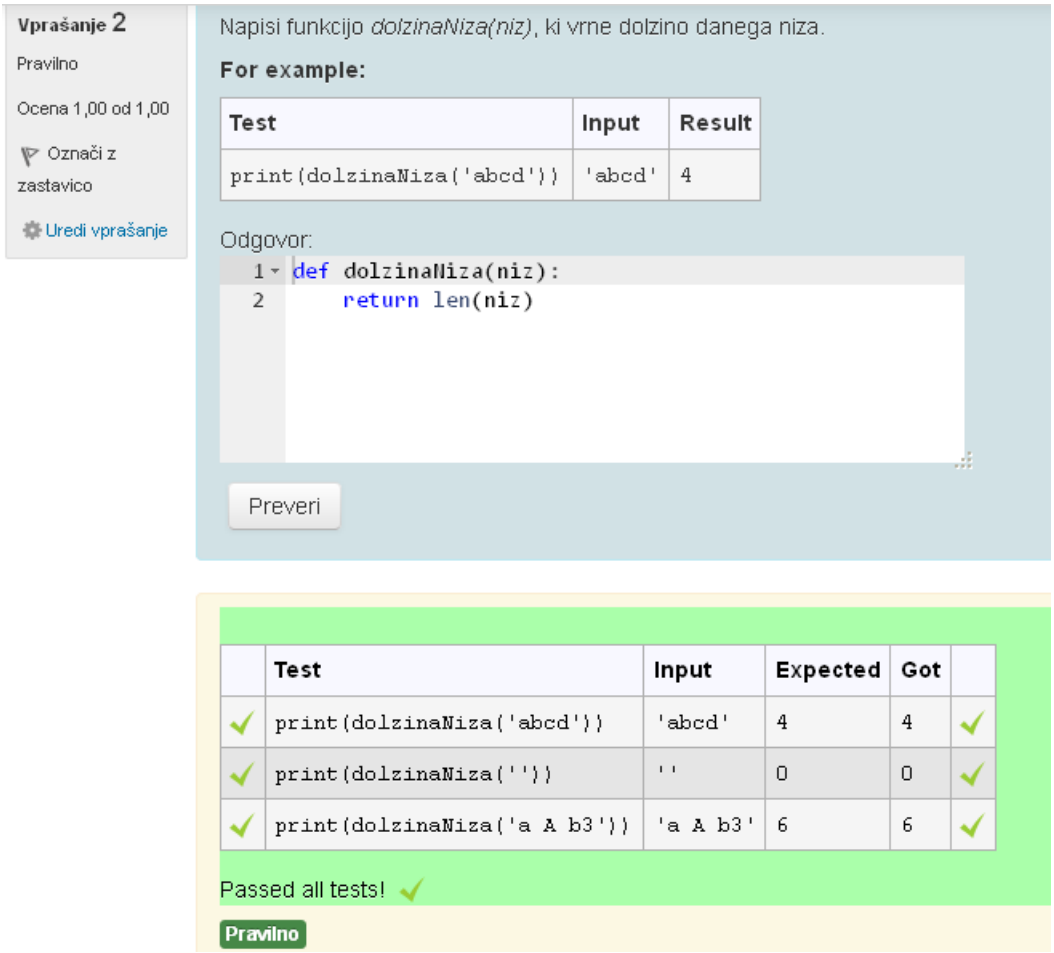


Slika 2: Primer vprašanja tipa CodeRunner v sistemu Moodle

Vprašanje je sestavljeno iz navodil in okvirčka za odgovor. V večini primerov učitelj poda tudi primer, kaj in kako se vprašanje testira. Preden študent napiše odgovor, testira kodo v svojem programskem okolju. Ko je prepričan, da njegova koda deluje pravilno, le-to kopira v okvirček za odgovor in klikne na gumb ″Preveri″.

Študentov odgovor se takoj preveri, povratna informacija se prikaže v obliki tabele s testnimi primeri. Če je odgovor pravilen, se tabela obarva zeleno (

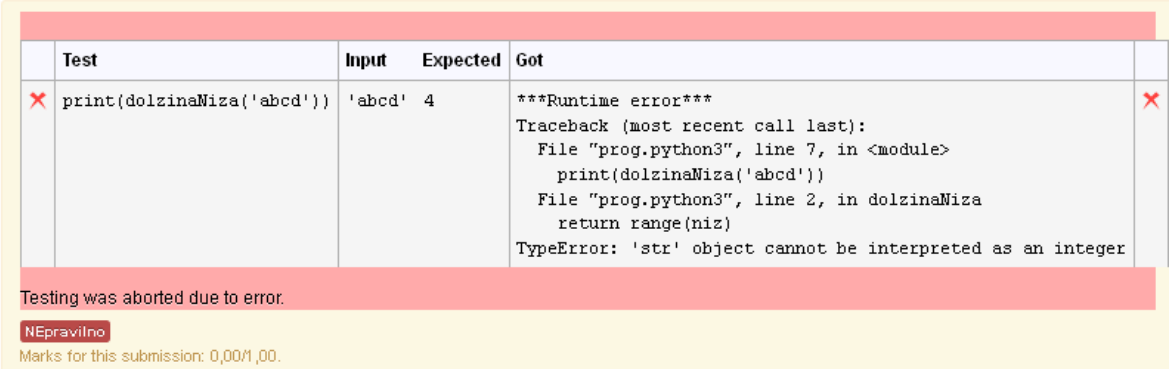
Slika 3).



Slika 3: Primer pravilno rešenega vprašanja

Če je študent rešitev sestavil tako, da je bodisi en bodisi več testnih primerov napačnih, se odgovor s povratno informacijo obarva rdeče (

Slika 4).

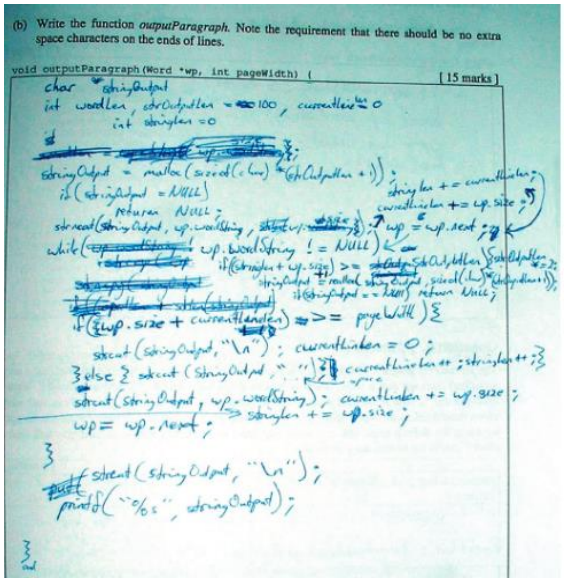


Slika 4: Tabela pri napačnem odgovoru

Možno je preverjati tudi pravilnost stila pisanja programov, zahtevati ali prepovedati uporabo določenih programskih konstrukcij ali funkcij iz programskega jezika in podobno. Seveda lahko pri samem vprašanju uporabljamo nastavitve, kot smo jih vajeni pri kvizih v Moodlu. Učitelj lahko tako dovoli, da študent svojo kodo ustrezno popravi in odgovor ponovno odda. Prav tako je lahko število študentovih poskusov omejeno ali pa ne. Kot lahko preberemo v diplomski nalogi *Kvizi v spletni učilnici Moodle* (Gerenčer, 2008), se večkratne poskuse reševanja lahko kaznuje z odbitkom. Ta se običajno izraža v odstotkih (na primer 10% za vsak napačni poskus). Več o samem načinu preverjanja in ustreznih nastavitvah bo predstavljeno v poglavju 6.

CodeRunner se aktivno uporablja na Univerzi Canterbury (Nova Zelandija) za učenje programiranja v različnih programskih jezikih. Povzemimo, kako sta avtorja tega okolja Richard Lobb in Jenny Harlow v njunem članku (CodeRunner: A Tool for Assessing Computer Programming Skills, 2016) razložila, zakaj so se lotili priprave tega orodja.

Težav pri ocenjevanju ročno napisane kode je več, in sicer od nepreglednosti zapisa do sintaktičnih napak, učitelj pa obenem težko oceni študentovo znanje programiranja (
Slika 5). V praksi se je namreč izkazalo, da redkokateri programer napiše pravilno kodo v prvem poskusu, če je le-ta kompleksnejša.



Slika 5: Problem ročno napisane kode[1]

Testiranje in razhroščevanje kode je torej sestavni del programiranja in tega je študent pri klasičnih izpitih prikrajšan. CodeRunner omogoča, da lahko študent najprej testira svojo kodo v svojem programskem okolju ter odpravi morebitne napake, šele nato pa jo odda preko spletnega brskalnika. Učitelju tako ni treba ročno ocenjevati naloge, kar je še posebej koristno pri velikih skupinah študentov.

[1] Slika iz članka (CodeRunner: A Tool for Assessing Computer Programming Skills, 2016)

2.1.1 Uporaba CodeRunnerja

CodeRunner se uporablja na Univerzi Canterbury (Nova Zelandija) za učenje programiranja v programskih jezikih Python, C, Octave in Matlab. Koda CodeRunnerja je dobro dokumentirana in ima preprosto strukturo, zato ga je mogoče nadgraditi tudi za uporabo v drugih programskih jezikih. V kombinaciji s sistemom Moodle je zelo uporaben pri izvajanju laboratorijskih vaj, sestavljanju domačih nalog, kvizov in tudi zaključnih izpitov. Izredno uporaben je pri začetnih tečajih programiranja, kjer potrebujejo študenti veliko vaje za reševanje osnovnega problema in se tako najlažje naučijo sintaktičnih pravil ter tehnik programiranja v novem programskem okolju.

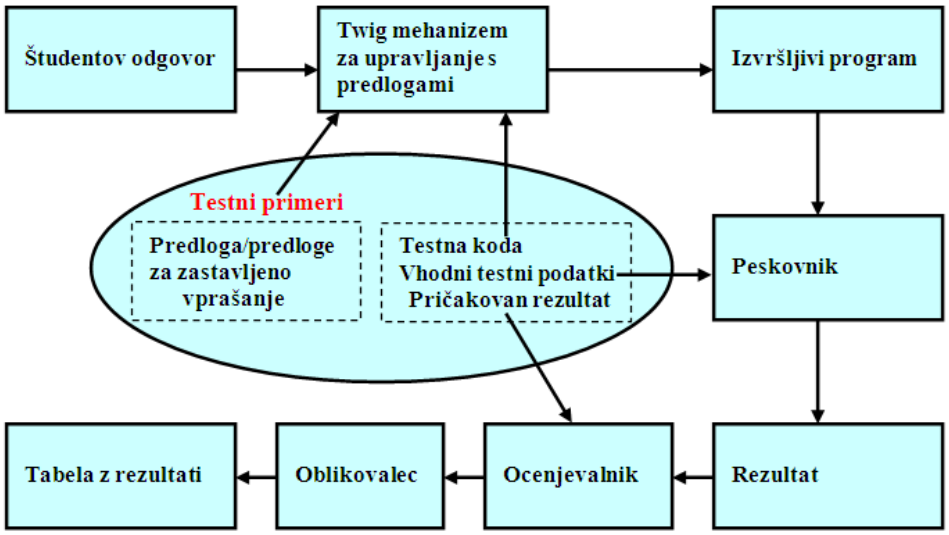
Za sistem CodeRunner imamo možnost postavitve lokalnega strežnika Jobe (Jobe, 2016). Smiselno ga je postaviti, če bomo orodje CodeRunner uporabljali v širše namene. Za osnovno preverjanje lahko uporabimo tudi strežnik, postavljen na Univerzi Canterbury. Vendar pa ima taka nastavitev določene omejitve. Lokalni strežnik nam omogoča izvajanje več nalog hkrati (več kot 60 nalog v minuti), lahko ga tudi zaščitimo z API-ključem (API-key, 2015) in tako poskrbimo za varnost, saj strežnik ne bo sprejel nobene kode iz zunanjega okolja brez ustreznega API-ključa. Več o namestitvi Jobe strežnika in zaščiti z API-ključem bo zapisanega v nadaljevanju diplomske naloge, natančneje v poglavju 3.

Ker se kvizi vse več rešujejo kar preko spleta, jih lahko študenti rešujejo tudi kasneje doma. Pri oddaji kvizov dobijo tako takojšen odziv glede uspešnosti rešenih nalog, in sicer brez fizičnega posega učitelja pri vrednotenju naloge. To je velika prednost, saj se lahko učitelj na laboratorijskih vajah posveti reševanju problematičnih nalog, obenem pa več časa nameni študentom, ki potrebujejo dodatno pomoč. Izkazalo se je, da je študentom takšen način učenja všeč, ker dobijo takojšne povratne informacije, ali so nalogo rešili pravilno ali ne. V primeru napačnega odgovora stremijo k pravilnim rešitvam in le redko nerešijo nalog, kar jih še dodatno motivira.

2.2 Delovanje CodeRunnerja

Če želimo sestavljati vprašanja v CodeRunnerju, je smiselno najprej pogledati, na kakšen način le-ta sploh deluje.

Slika 6 prikazuje shemo delovanja CodeRunnerja po korakih, ko študent odda svoj odgovor.

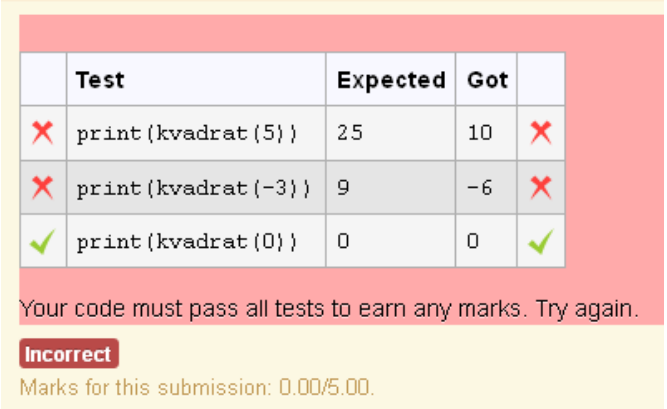


Slika 6: Shema delovanja CodeRunnerja

Študent odda odgovor na vprašanje v obliki ustrezne programske kode. Ta se v TWIG (TWIG, 2016) mehanizmu za upravljanje s predlogami združi v celoto skupaj s predlogo in testno kodo prvega testnega primera. TWIG celoto prevede v izvršilni program. V peskovniku (sandboxu) kodo programa prevedemo in jo zaženemo s testnimi podatki, podanimi v testnih primerih. Rezultat posredujemo v ocenjevalnik. Tam ga primerjamo s pričakovanim rezultatom na način, kot je določen v ocenjevalniku. Najpogosteje uporabljen način je ″popolno ujemanje″, možni pa so tudi drugi načini. Rezultat iz ocenjevalnika je objekt ″rezultat testa″, ki ima atributa ″pričakovano″ in ″dobljeno″.

Opisan postopek ponovimo za vse testne primere. Vsi dobljeni objekti iz ocenjevalnika so nato združeni v tabelo, ki jo v oblikovalcu oblikujemo v končno tabelo z rezultati. Ta se kot povratna informacija izpiše študentu. Uspešno opravljeni testni primeri so v tabeli označeni z zeleno kljukico, neuspešni pa z rdečim križcem. Tabela je v celoti obarvana zeleno, če so vsi testni primeri pravilni, rdeče obarvana tabela pa naznanja, da je vsaj eden izmed njih nepravilen (

Slika 7).

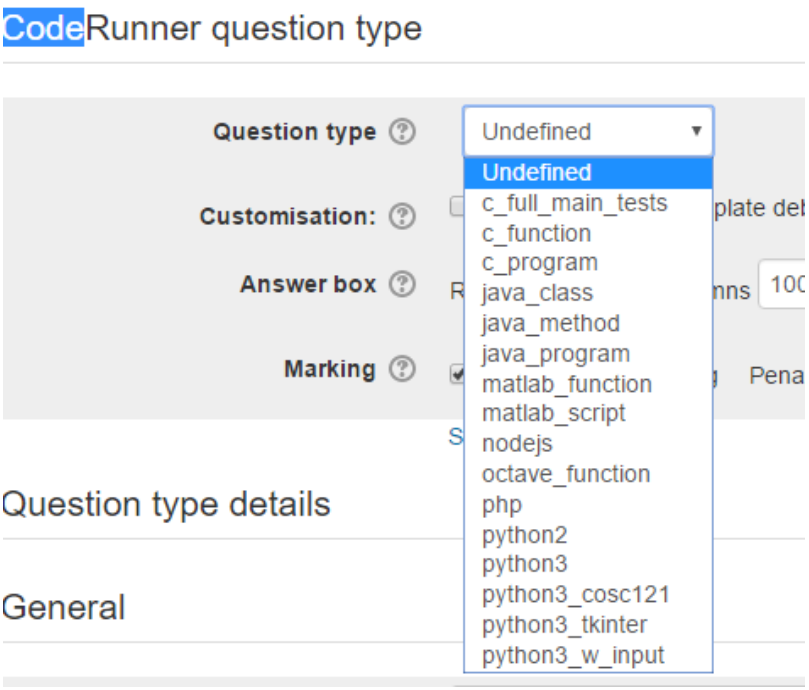


Slika 7: Tabela s pravilnimi in z nepravilnimi testnimi primeri

2.2.1 Tipi vprašanj v CodeRunnerju

Vsako vprašanje, ki ga sestavimo, je primerek tako imenovanega prototipnega vprašanja. Kot bomo videli v poglavju 5, ki govori o sestavljanju vprašanj, je potrebno najprej izbrati tip vprašanja (*QuestionType*,

Slika 8).



Slika 8: Tipi vprašanj

Prototip opredeljuje tip vprašanja. V CodeRunnerju imamo vgrajenih več osnovnih prototipnih vprašanj (prototip), ki določajo, ali bo vprašanje vrste ‘napiši program’ , 'napiši funkcijo' ali pa 'napiši razred'... Če uporabljamo programska jezika Java in C, imamo na voljo vse tri omenjene različice, v Matlabu pa sta na voljo prototipa 'napiši funkcijo' in 'napiši skripto'. Prototip lahko razširimo, mu nastavimo dodatne parametre, če seveda ustrezno prilagodimo njegovo predlogo. Za programski jezik Python 3 imamo vgrajen samo osnovni prototip 'napiši funkcijo'. Zato takrat, kadar uporabljamo Python 3, pogosto naredimo nove prototipe. Tako so na Univerzi Canterbury na Novi Zelandiji ustvarili svoj prototip **python3_cosc121**, ki ga uporabljajo za ocenjevanje kvizov in izpitov. Ta njihova predloga ima dodane številne uporabne parametre, kot so:

- *Je funkcija*
- *Pylint nastavitve*
- *Prepovedani konstruktorji*
- *Obvezni konstruktorji*
- *Dovoli globalne spremenljivke*
- *Maksimalno število konstant*
- *Brez zagona*
- *Zaženi dodatno*

Kako ti parametri delujejo, si bomo ogledali v nadaljevanju.

Predloga, ki nam omogoča sestavljanje vprašanj tipa **python3_cosc121,** ni sestavni del namestitve vtičnika CodeRunner, zato tega tipa vprašanja ni v meniju med vgrajenimi tipi vprašanj. Lahko pa to predlogo uvozimo naknadno. Pri vprašanjih, ki jih bomo sestavili za našo bazo vprašanj, bomo uporabljali poleg vgrajenega tipa *python3* še *python_w_input* in *python3_cosc121*, zato priporočamo vsem uporabnikom, da namestijo to predlogo.

Če poznamo delovanje predlog dobro, lahko sami ustvarimo poljubno število tipov vprašanj. Shranimo jih lahko kot prototip in jih kasneje najdemo v meniju z vgrajenimi vprašanji. Jezik, uporabljen v prototipih, je lahko različen od tistega, ki se uporablja za izvršitev študentovega odgovora. Tako lahko na primer za preverjanje ustreznosti sloga kodiranja v Octave-u uporabimo predlogo iz Pythona, ki preveri, če je koda napisana v ustreznem slogu, preden jo dejansko preda Octave-u za izvršitev. O tem, kako sestavljamo in spreminjamo prototipe, bomo govorili v poglavju 5.1.2.

3 Namestitev sistema CodeRunner

V tem poglavju bomo prikazali postopek namestitve celotnega sistema CodeRunner, vključno s postavitvijo in zaščito strežnika Jobe. Predvidevamo, da imamo že nameščen delujoči sistem Moodle (različico 2.6 ali novejšo), in sicer na operacijskem sistemu Linux. Vsa navodila so jasno napisana v spletni dokumentaciji:
http://coderunner.org.nz/mod/book/view.php?id=179.

S sledenjem tem navodilom smo uspešno namestili program.

3.1 Namestitev vtičnika CodeRunner v sistem Moodle

Tip vprašanja CodeRunner je trenutno dostopen v dveh github skladiščih na spletnih straneh:

- https://github.com/trampgeek/moodle-qbehaviour_adaptive_adapted_for_coderunner
- https://github.com/trampgeek/moodle-qtype_coderunner

V naš sistem Moodle moramo namestiti oba. Postopek namestitve podatkov je dostopen na https://github.com/trampgeek/moodle-qtype_coderunner.

Namestitev vtičnika lahko izvedemo na dva načina:

- Lahko prenesemo potrebno zip datoteko v ustrezen imenik *moodle/question/type* in spremenimo ime novo nastalega imenika iz *moodle-qtype_coderunner-<imeskupine>* samo v coderunner.
- V konzolnem oknu operacijskega sistema Linux poženemo ukaz *root@moodle_lokar:/var/www/moodle# git clone git://github.com/trampgeek/moodle-qtype_coderunner.git question/type/coderunner*

kjer je *root* uporabnik operacijskega sistema Linux, *moodle_lokar* pa ime navideznega računalnika, */var/www/* privzet imenik za streženje spletnih strani, */var/www/moodle* pa imenik sistema moodle.

Ker smo vse namestitve opravljali v operacijskem sistemu Linux, na katerem smo imeli nameščen program za delo z git skladišči, smo se odločili za drugo možnost. Zgoraj modro obarvan ukaz nam prenese kodo iz spletnega git skladišča v naš računalnik – iz skladišča na naslovu

git://github.com/trampgeek/moodle-qtype_coderunner.git

nam prenese kodo v naš računalnik v podimenik

question/type/coderunner

našega trenutnega imenika. Ker se nahajamo v *root@moodle_lokar:/var/www/moodle*, nam kodo prenese v imenik

/var/www/moodle/question/type/coderunner,

Naslednji ukaz, ki ga vnesemo v konzolno okno je

root@moodle_lokar:/var/www/moodle# chown root.www-data -R question/type/coderunner

Vsaka datoteka ima 2 podatka o lastniku: uporabniško ime lastnika in ime skupine lastnika. Z zgornjim ukazom *chown* nastavimo uporabniško ime lastnika na *root* (root ima podobno vlogo kot Administrator v operacijskem sistemu Windows), skupino lastnika pa na *www-data*. To je skupina, v katero spada tudi spletni strežnik. Ukaz *–R* (rekurzivno stikalo) poskrbi, da se pravice nastavijo tudi v vseh podimenikih podanega imenika (v našem primeru ukaz rekurzivno izvedemo v imeniku */var/www/moodle/question/type/coderunner).*

Z ukazom *root@moodle_lokar:/var/www/moodle# chmod g+r –R question/type/coderunner*

v istem imeniku */var/www/moodle/question/type/coderunner* rekurzivno (rekurzivno stikalo *–R*) dodamo pravice za branje skupini lastnika. To pove ukaz *g+r*, kjer *g* predstavlja skupino (group), *+r* pa dodajanje pravice za branje (read).

Primer: ukaz *g+rw* bi dodal skupini lastnika pravice za branje in pisanje po teh datotekah (r = read, w = write).

Celoten postopek ponovimo še za git skladišče na spletni strani:
https://github.com/trampgeek/moodle-qbehaviour_adaptive_adapted_for_coderunner

V konzolnem oknu operacijskega sistema Linux poženemo ukaz:
root@moodle_lokar:/var/www/moodle# git clone git://github.com/trampgeek/moodle-qbehaviour_adaptive_adapted_for_coderunner.git question/behaviour/adaptive_adapted_for_coderunner

Torej iz git skladišča na naslovu

git://github.com/trampgeek/moodle-qbehaviour_adaptive_adapted_for_coderunner.git

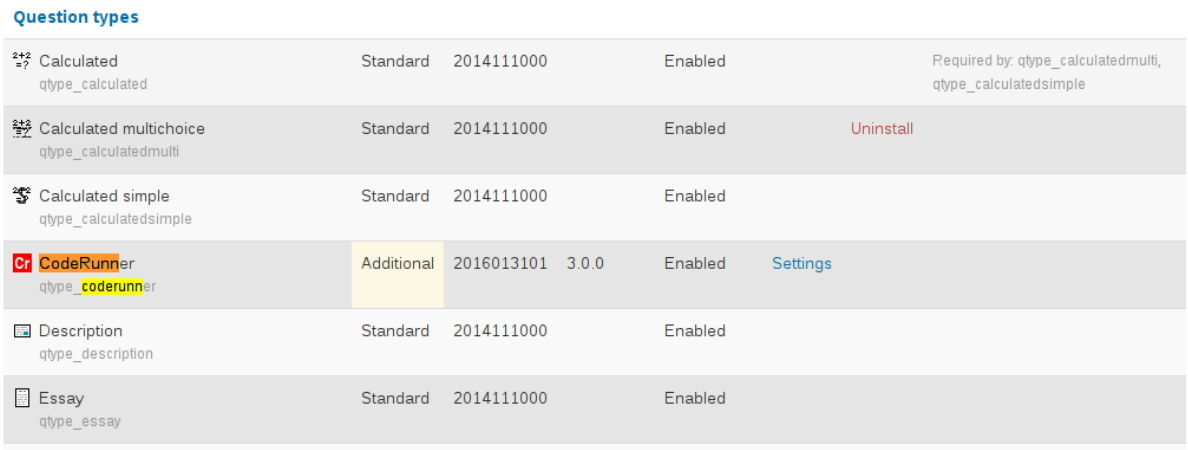
prenesemo kodo v naš računalnik v podimenik

question/behaviour/adaptive_adapted_for_coderunner

Podobno kot zgoraj vnesemo še ostala dva ukaza:

- *root@moodle_lokar:/var/www/moodle# chown root.www-data -R question/behaviour/adaptive_adapted_for_coderunner*
- *root@moodle_lokar:/var/www/moodle# chmod g+r -Rquestion/behaviour/adaptive_adapted_for_coderunner*

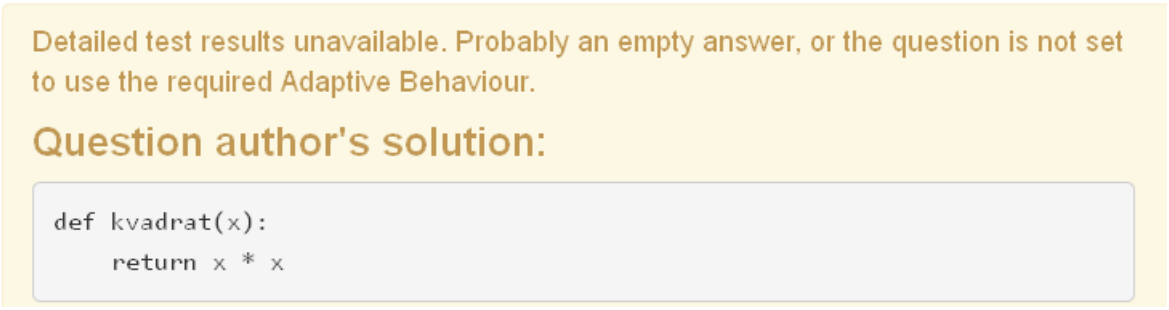
Da se namestitev dokončno izvrši, se je potrebno prijaviti na strežnik preko spletnega portala kot administrator. Tu potem sledimo navodilom za nadgradnjo baze podatkov. Ko je CodeRunner uspešno nameščen, lahko v meniju *Administration / Site administration / Plugins / Plugins Overview* pogledamo, da je tip vprašanja CodeRunner res na seznamu med ostalimi vgrajenimi tipi(
Slika 9).



Slika 9: V seznamu *Question types* zdaj najdemo tudi CodeRunner

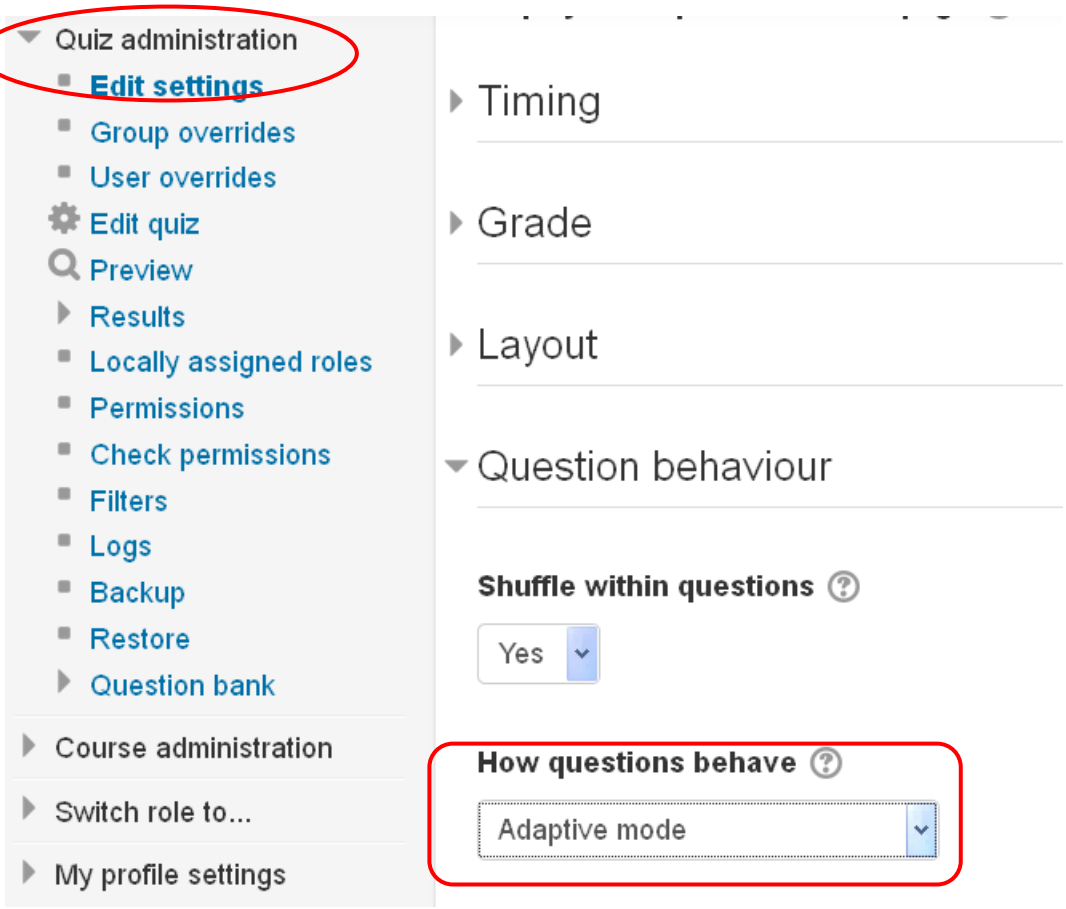
Opozorilo: CodeRunner deluje samo vprilagojenem načinu (*Adaptive Mode*). Potrebno je spremeniti nastavitve kviza, v nasprotnem primeru bomo pri odgovoru na vprašanje namesto tabele s testnimi primeri dobili sporočilo (

Slika 10).



Slika 10: Sporočilo o potrebni prilagoditvi na *Adaptive Mode*

Slika 11 prikazuje omenjeni postopek.



Slika 11: Nastavitve kviza

V meniju *Quiz administration* kliknemo na *Edit settings* in poiščemo razdelek *Question behaviour*. V meniju *How questions behave* izberemo možnost *Adaptive mode*.

3.2 Postavitev Jobe strežnika

Čeprav ima CodeRunner preprosto in fleksibilno strukturo, ki omogoča najrazličnejše načine preverjanja študentove kode, ga trenutno podpira samo eno zaščiteno okolje (v nadaljevanju peskovnik ali sandbox) – Jobe sandbox. Ta peskovnik uporablja ločen strežnik, razvit posebej za uporabo CodeRunnerja, ki se imenuje Jobe. Privzeto se uporablja strežnik Jobe z Univerze v Canterburyju z Nove Zelandije, ki je primeren za prvotno testiranje in spoznavanje CodeRunnerja. Omejitev na privzetem strežniku je zaradi možnosti preobremenitve postavljena na 100 nalog na uro. Zato je za samostojno uporabo sistema smiselno postaviti svoj strežnik Jobe.

Strežnik Jobe deluje samo v operacijskem sistemu Linux, na katerem mora biti nameščen strežnik Apache. Predhodno moramo imeti nameščene tudi programske jezike Python3 in C.

Na povezavi https://github.com/trampgeek/jobe poglavje (*Instalation)* sledimo navodilom za postavitev strežnika Jobe. Navodila so napisana v angleškem jeziku. Vsi ukazi (če ni eksplicitno napisano drugače) se izvajajo v konzolnem oknu operacijskega sistema Linux Debian. Označeni so z modro barvo.

V konzolnem oknu poženemo ukaz

```
# apt-get install php libapache2-mod-php php-mcrypt mysql-server
    php-mysql php-cli octave nodejs
    git python3 build-essential openjdk-8-jre openjdk-8-jdk python3-pip
    fp-compiler pylint3
```

S tem ukazom namestimo na naš računalnik vse potrebne pakete (Java, Python3, Octave, Pylint...).

Za začetno konfiguracijo Pylinta izvršimo ukaz

```
/var/www/html# pylint --reports=no --generate-rcfile > /etc/pylintrc
```

Ta ukazje potreben le, če bomo testirali kakovost kode s sistemom Pylint.

V privzetem imeniku za streženje spletnih strani */var/www/html* namestimo Jobe strežnik s spodnjim ukazom:

```
/var/www/html#git clone https://github.com/trampgeek/jobe.git
```

Ta ukaz v imenik */var/www/html* prenese datoteke, potrebne za delovanje strežnika Jobe. Prenese jih v podimenik *jobe*. V tem podimeniku potem poženemo skripto *install*.

```
/var/www/html/jobe# ./install
```

Ta ukaz uredi vse potrebno za namestitev.

V vrstici *JOBE_SERVER = 'localhost'* zamenjamo vrednost *'localhost'* z našim IP naslovom (v našem primeru *10.0.0.3*).

Nato testiramo namestitev. To storimo z zagonom ukaza

/var/www/html/jobe# python3 testsubmit.py

V tem koraku imamo Jobe strežnik pripravljen za uporabo.

3.2.1 Zaščita strežnika z API-ključem

Na Univerzi Canterbury dovolijo uporabo njihovega Jobe strežnika, vendar je avtentikacija ter avtorizacija urejena preko API-ključa. Zaradi varnosti in preobremenitve svojega strežnika so z API ključem, ki je priložen pri sami namestitvi CodeRunnerja, določili omejitev izvajanj na 100 ukazov na uro. Pametno je zaščititi tudi naš Jobe strežnik, sicer ga lahko uporablja vsak, kar lahko vodi do zlorabe in preobremenitve.

Za nastavitev API-ključa sledimo navodilom na strani https://github.com/trampgeek/jobe, poglavje *Securing with API keys*.

Najprej moramo namestiti Mysql strežnik. V terminalu poženemo ukaz

 # apt-get install mysql-server

Potem ustvarimo bazo podatkov z imenom "jobe". Da si bomo lažje zapomnili uporabniške podatke, ki jih bomo vnesli, bomo vse poimenovali z "jobe". Torej ustvarimo uporabnika "jobe" z geslom "jobe", ki ima vse pravice na bazi podatkov z imenom "jobe". Ukazi se izvršijo v Mysql konzoli.

```
CREATE USER 'jobe'@'localhost' IDENTIFIED BY 'jobe';
CREATE DATABASE jobe;
GRANT ALL PRIVILEGES ON jobe . * TO 'jobe'@'localhost';
```

Nato uredimo datoteko *application/config/database.php* v skladu z nastavitvami v prejšnji točki. Uredimo še datoteko *application/config/rest.php* v skladu z zgornjimi nastavitvami in nastavimo parameter *rest_enable_keys na 1.*

Potem ustvarimo tabeli "keys" in "limits", kot je opisano v datoteki rest.php.

```
CREATE TABLE `keys` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `key` varchar(40) NOT NULL,
  `level` int(2) NOT NULL,
  `ignore_limits` tinyint(1) NOT NULL DEFAULT '0',
  `is_private_key` tinyint(1)  NOT NULL DEFAULT '0',
  `ip_addresses` TEXT NULL DEFAULT NULL,
  `date_created` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `limits` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `uri` varchar(255) NOT NULL,
  `count` int(10) NOT NULL,
  `hour_started` int(11) NOT NULL,
  `api_key` varchar(40) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Nato v tabelo "keys" v polje "*key*" shranimo vrednost našega API-ključa. Za geslo smo si izbrali *nasesupervarnogeslo.*

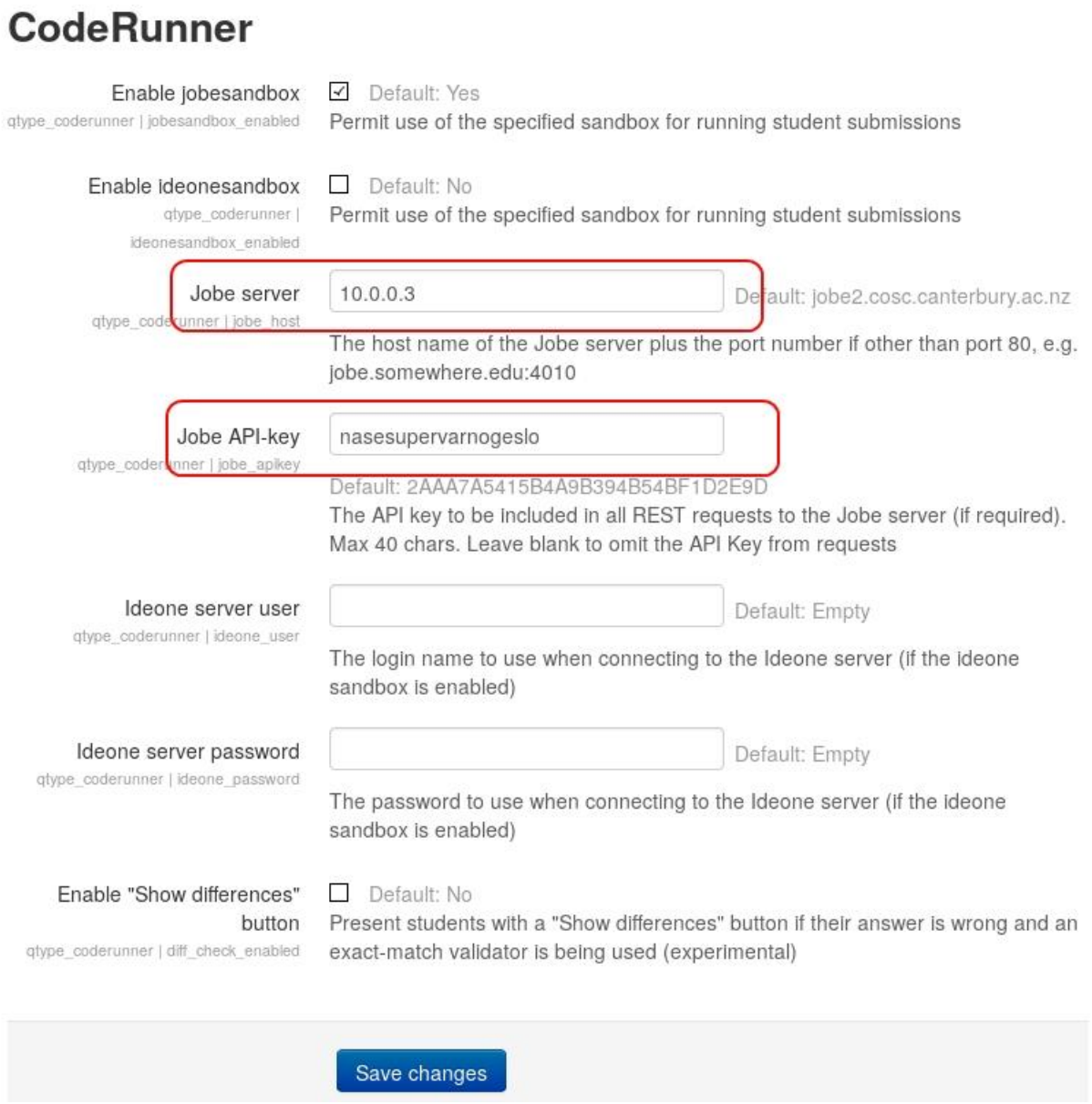
```
mysql> INSERT INTO  `keys` ( `key` ) VALUES ( 'nasesupervarnogeslo' );
```

Ponovno zaženemo spletni strežnik Apache, da se upoštevajo vse novo nastale nastavitve. Strežnik ponovno zaženemo z ukazom

```
# service apache2 restart
```

Sedaj bi moral biti naš API-ključ ustrezno nastavljen. Potrebna je še nastavitev Jobe strežnika in API-ključa v sistemu Moodle. Za urejanje teh nastavitev moramo imeti administratorske pravice.

V meniju *Site administration* kliknemo na *Plugins* / *Question type* / *CodeRunner.* Slika 12 prikazuje nastavitve vtičnika CodeRunner.



Slika 12: Nastavitve vtičnika CodeRunner

Spremeniti moramo vrednost v okencu *Jobe server*. Vnesemo naslov našega Jobe strežnika (iz **jobe2.cosc.canterbury.ac.nz** nastavimo na **10.0.0.3**). Spremeniti moramo še API-ključ, ki je privzeto nastavljen na *2AAA7A5415B4A9B394B54BF1D2E9D*.

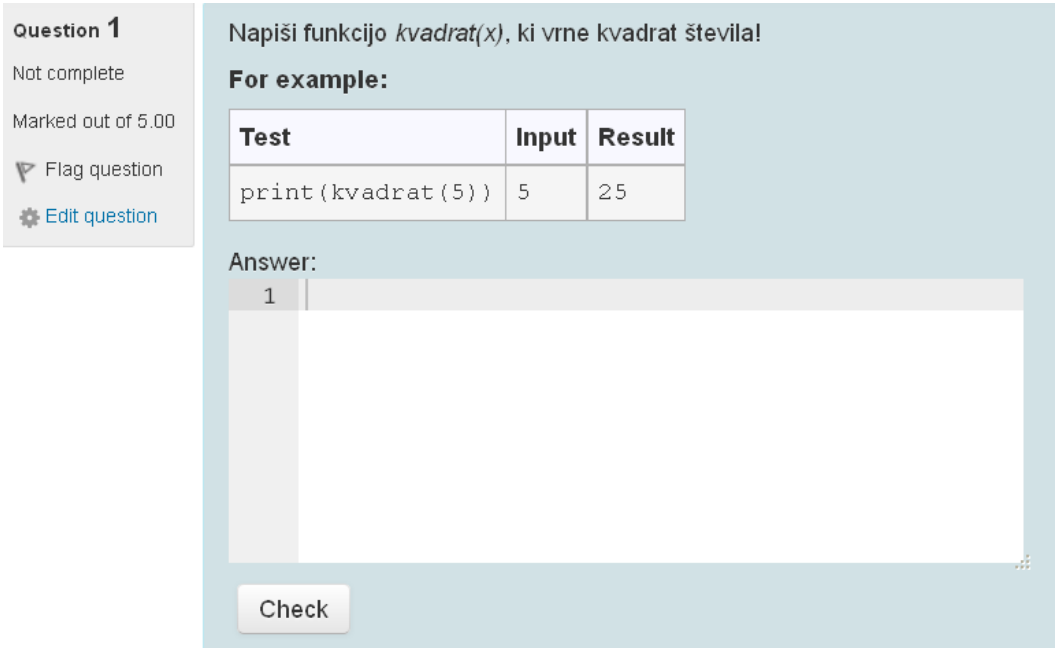
V okencu *Jobe API-key*: spremenimo vrednost **2AAA7A5415B4A9B394B54BF1D2E9D** na **nasesupervarnogeslo**. Okenca sta na Sliki 12 označena z rdečim okvirjem.

4 Uporaba CodeRunnerja s strani študenta

V tem poglavju si bomo ogledali, kako vidi študent v kvizu vprašanja, sestavljena v CodeRunnerju. Kot študenta mislimo na uporabnika, ki nima pravic za urejanje gradiva na spletni strani, torej lahko samo sodeluje pri predmetih in rešuje kvize, sestavljati, ali spreminjati vprašanj pa ne more.

4.1 Reševanje kviza

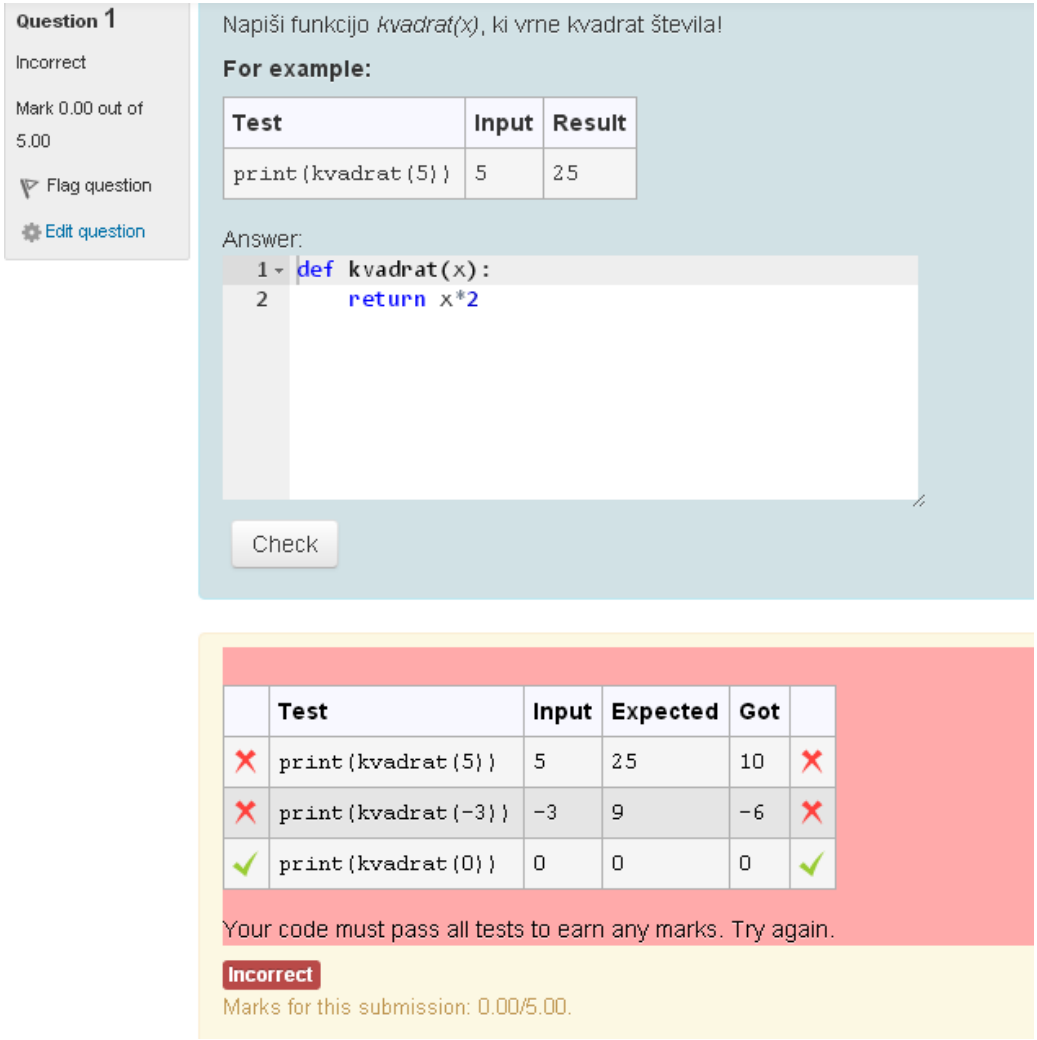
Na praktičnem primeru si bomo ogledali, kaj lahko študent pričakuje, ko se v kvizu sreča z vprašanjem, sestavljenem v CodeRunnerju.



Slika 13: Vprašanje sestavljeno v CodeRunnerju

Poglejmo si prvo vprašanje (Slika 13). Ta zahteva, da napišemo preprosto funkcijo, ki vrne kvadrat poljubnega števila x. Za lažje razumevanje je učitelj dodal še primer obnašanja te funkcije. Pogosto so ti testni primeri ključni, da lahko napišemo pravilen odgovor. Odgovor moramo podati v obliki programske kode, ki jo vpišemo v okence *Answer*. Kot smo omenili že prej, običajno kodo preverimo lokalno na svojem računalniku in jo s pomočjo ukaza *Copy/Paste* le prepišemo v okence *Answer*. Seveda pa lahko kodo vpišemo neposredno v okence. Ko smo prepričani, da je koda v okencu *Answer* napisana prav, kliknemo na gumb *Check*.

Kadar študent napiše napačen odgovor, dobimo prizor, kot ga prikazuje Slika 14.



Slika 14: Primer napačnega odgovora

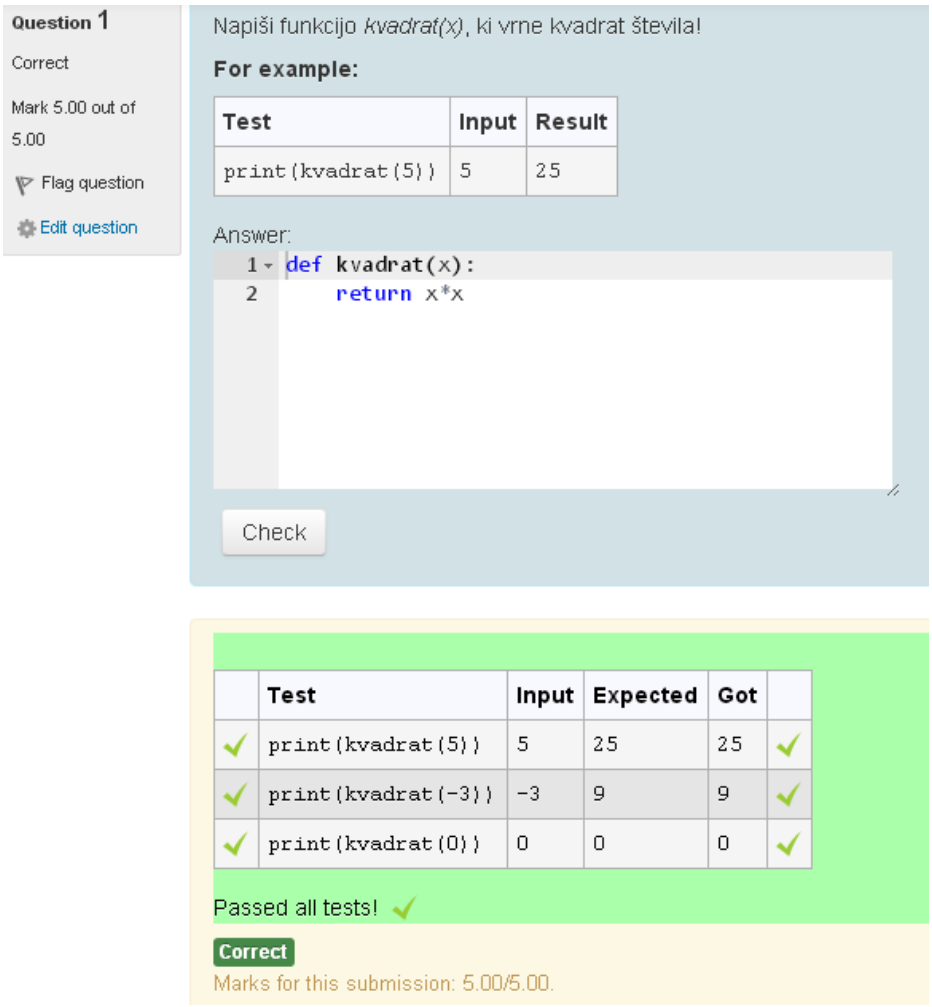
V tabeli opazimo dva testna primera, označena z rdečim križcem, zadnji pa ima zeleno kljukico. To pomeni, da koda ni uspešno prestala prvih dveh testnih primerov. Ker imamo način ocenjevanja nastavljen na *All-or-nothing grading*, se tabela v celoti obarva rdeče. To pomeni, da za odgovor nismo prejeli točk.

Pozornost moramo posvetiti tudi zadnjemu testnemu primeru s Slika 14, označenim s kljukico. Kljub napačnemu odgovoru je študentova koda uspešno prestala zadnji testni primer. Zato moramo biti pri sestavljanju testnih primerov previdni. Zgodi se lahko, da študent svojo kodo prilagodi tako, da le-ta deluje pravilno za vse testne primere, ki smo jih sestavili, kljub temu da je koda napačna. Več o sestavljanju testnih primerov in ocenjevanju nalog si bomo ogledali v poglavju 6.

V tabeli imamo v stolpcih predstavljene naslednje podatke:

- **Test**: kjer je napisano, kako je bila testirana pravilnost vprašanja
- **Input**: podatki, ki so vneseni v testne primere
- **Expected**: pričakovani rezultat testnih primerov
- **Got**: rezultat testnih primerov za našo kodo

Ko kodo popravimo in ponovno oddamo odgovor, dobimo rezultat, kot je prikazan na Slika 15**Error! Reference source not found.**.



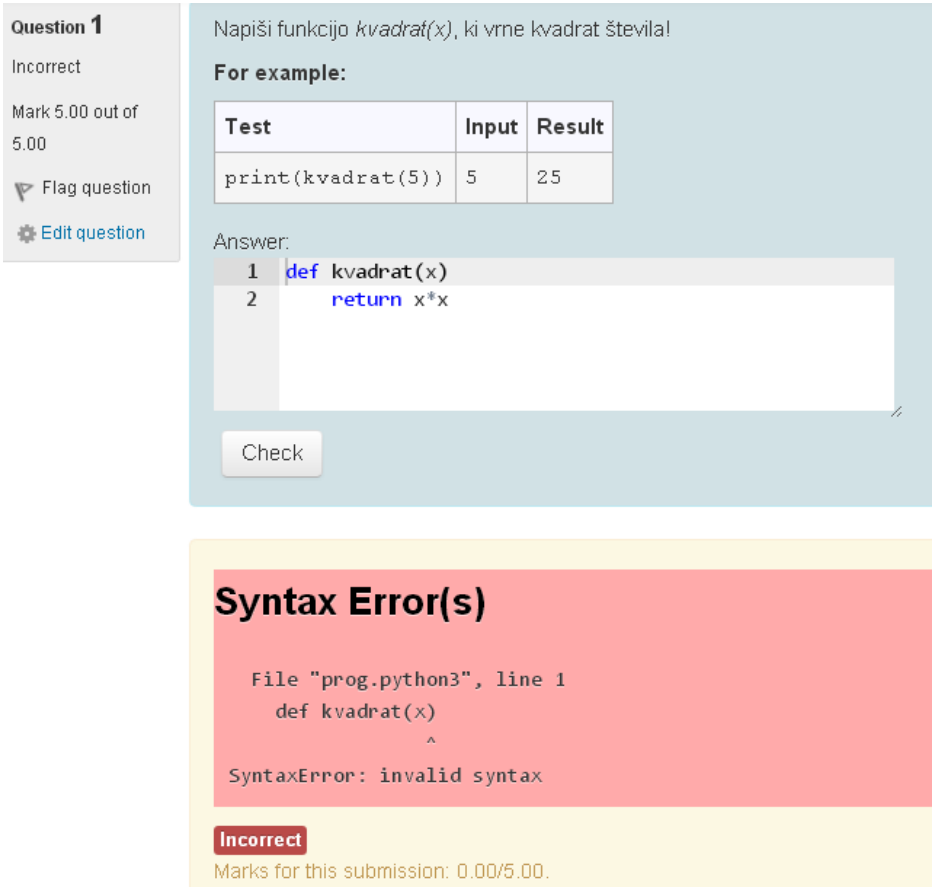
Slika 15: Primer pravilnega odgovora

Tabela s testi se obarva zeleno, testni primeri pa so obkljukani z zeleno kljukico, kar nakazuje pravilnost kode.

Če pri sestavljanju vprašanja odkljukamo možnost *All-or-nothing grading*, se nam bo ocenjeval vsak testni primer posebej. Takrat obstaja možnost, da kot rezultat dobimo tabelo z delno pravilnimi odgovori, ki je obarvana rumeno. To se zgodi takrat, kadar v tabeli nastopijo pravilni in nepravilni testni primeri. Primer delnega ocenjevanja si bomo podrobneje ogledali v poglavju 6.2.

Obstaja tudi možnost, da naredi študent v kodi različne napake. Ogledali si bomo nekaj primerov:

- **Napaka je sintaktična**. Študent pozabi na dvopičje, nepravilno gnezdi oklepaje... V tem primeru se študentu namesto tabele z napačnimi testnimi primeri prikaže sporočilo *Syntax Error(s)* (Slika 16)

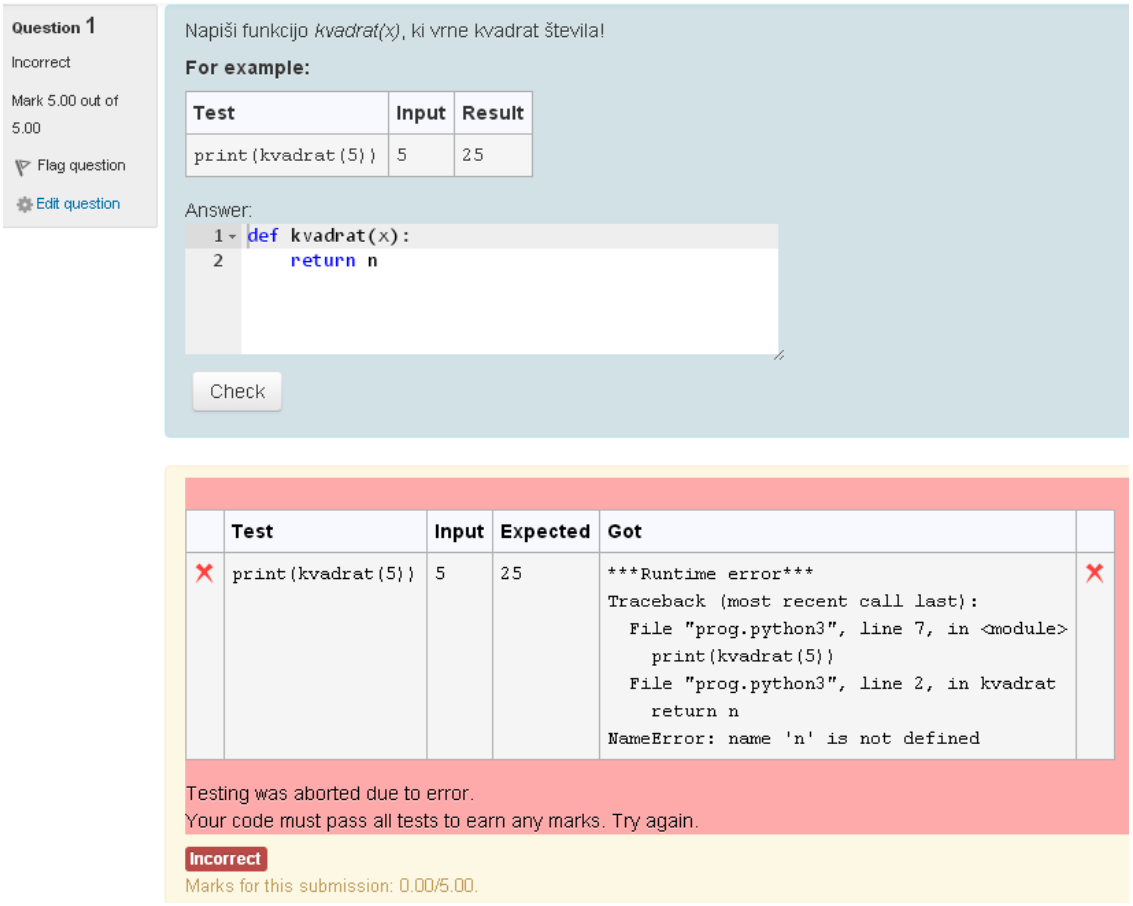


Slika 16: Sintaktično napačen odgovor

V sporočilu se prikaže vrstica kode, kjer je ugotovljena sintaktična napaka. Testiranje se pravzaprav sploh ni izvedlo.

Od zdaj naprej bomo predpostavili, da je koda odgovorov vsaj sintaktično pravilna.

- ***Runtime Error***. Do te napake lahko pride med samim izvajanjem programa, na primer, če študent pozabi definirati kakšno spremenljivko. V tabeli s testnimi rezultati se nam prikaže sporočilo (Slika 17).



Slika 17: Runtime error

Tudi v tem primeru se nam prikaže sporočilo z ugotovljeno napako, tokrat v tabeli s testnimi primeri. Vendar pa je tu izveden le prvi test, ostali pa ne. Izpisana je vrstica, kjer sta ugotovljeni napaka in vrsta napake. V tem primeru gre za napako vrste *NameError*, spremenljivka 'n' ni definirana.

Podobno napako dobimo, če študent bodisi uporabi napačno ime funkcije (Slika 18). Tudi tu CodeRunner odneha že ob prvem testu.

	Test	Expected	Got	

✗	`print(kvadrat(5))`	25	`***Runtime error***`
			`Traceback (most recent call last):`
			`  File "prog.python3", line 7, in <module>`
			`  print(kvadrat(5))`
			`NameError: name 'kvadrat' is not defined`
				✗

Testing was aborted due to error.
Your code must pass all tests to earn any marks. Try again.

Incorrect
Marks for this submission: 0.00/4.00.

Slika 18: Študent napačno uporabi ime funkcije

Če je študent svojo rešitev prej preizkusil, praviloma tudi do takih napak ne bi smelo priti.

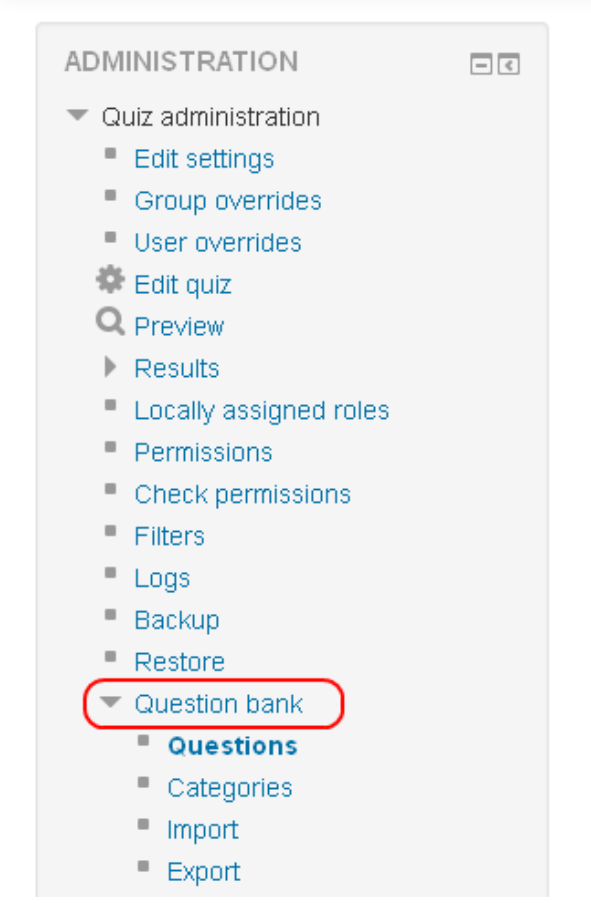
5 Sestavljanje vprašanj v CodeRunnerju

V tem poglavju si bomo ogledali, kako sestavljamo vprašanja s pomočjo CodeRunnerja. S pomočjo teh vprašanj bomo sestavili nekaj kvizov, ki jih študenti lahko uporabljajo za preverjanje svojega znanja programiranja v Pythonu. V okviru praktičnega dela diplomske naloge smo tako pripravili osnovno banko vprašanj, ki vsebuje več kot 60 različnih nalog, razporejenih v ustrezne kategorije.

Da lahko sestavljamo naloge (vprašanja) kot tudi kvize, moramo imeti ustrezne pravice. Predpostavljamo torej, da ima uporabnik vlogo učitelja. Kot učitelja torej mislimo uporabnika, ki ima pravico do sestavljanja in spreminjanja gradiva v okviru predmeta.

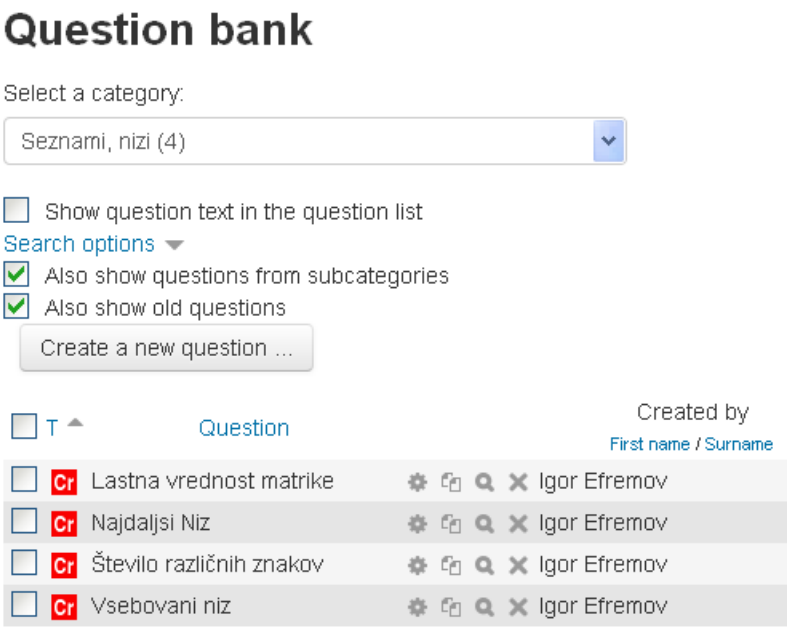
5.1 Sestavljanje vprašanja

Najprej se bomo splošno seznanili z vsemi polji in možnostmi pri sestavljanju vprašanja tipa CodeRunner. Nato bomo sestavili preprosto nalogo, kjer si bomo podrobneje ogledali izpolnjena polja. Na spletni učilnici Moodle v meniju *Administration* izberemo razdelek *Question bank* (Slika 19).



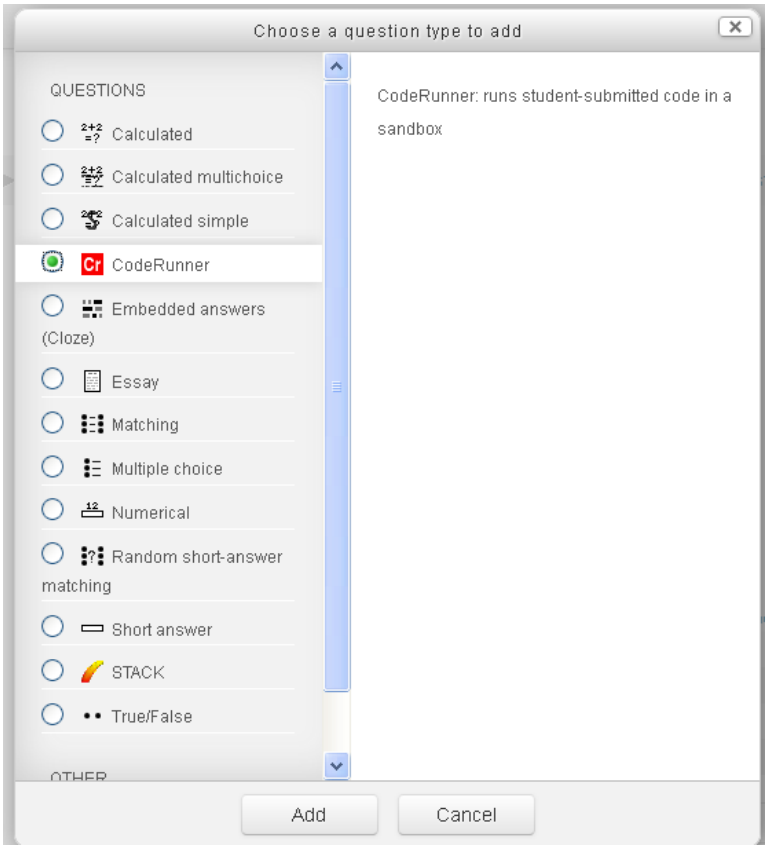
Slika 19: Meni *Administration*

Zagledamo zbirko vseh sestavljenih vprašanj, ki so razporejena v kategorije (Slika 20).



Slika 20: Zbirka vprašanj

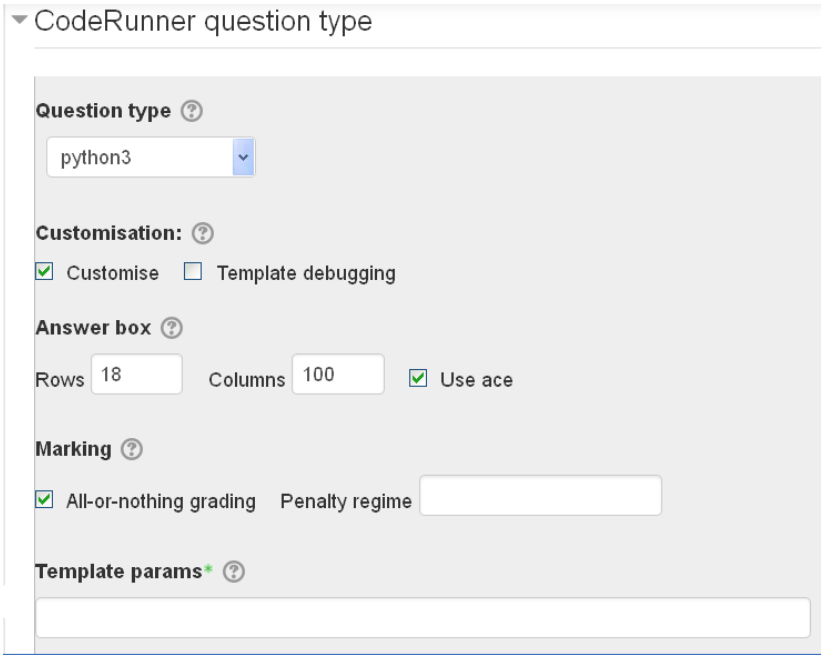
Če želimo ustvariti novo vprašanje, kliknemo na gumb *Create a new question...*V okencu izberemo tip vprašanja, ki ga želimo ustvariti. Mi bomo izbrali *CodeRunner* (**Slika 21**).

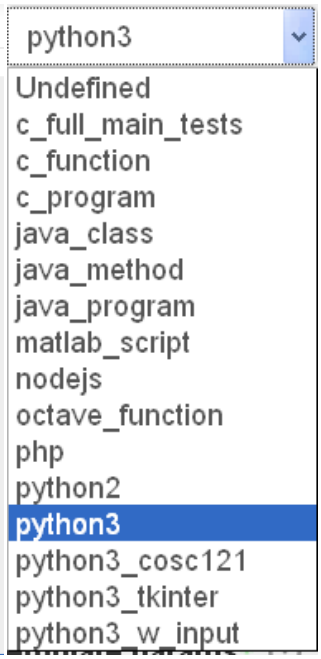


Slika 21: Izberemo tip vprašanja CodeRunner

Odpre se nam glavna stran za urejanje in sestavljanje vprašanja tipa CodeRunner (Slika 22). Ogledali si bomo vse možnosti in parametre, ki so na voljo za sestavljanje vprašanja.

Najprej si oglejmo razdelek *CodeRunner question type.* Tu si najprej izberemo ustrezen programski jezik in tip vprašanja (Slika 23).





Slika 22: Sestavljanje vprašanja tipa CodeRunner **Slika 23: Možnosti *Question type***

Pri ukazu *Customisation* lahko izberemo, ali bomo tip vprašanja ustrezno prilagodili. Za programski jezik Python3 je privzeti tip vprašanja 'napiši funkcijo'. Z ustrezno prilagoditvijo lahko ta tip vprašanja spremenimo v drugo obliko, na primer ‘napiši program’. To naredimo tako, da s kljukico označimo polje *Customize*. Odpreta se nam nova razdelka (Slika 24); *Customisation*, kjer lahko predlogo urejamo sami, in *Advanced Customisation*, kjer urejamo napredne nastavitve predloge. Če obkljukamo še polje *Template debugging*, se nam pri odgovoru na vprašanje izpiše ustrezna koda predloge. Iz kode je razvidno, kako predloga obravnava testne primere. Koda se izpiše za vsak testni primer posebej. Več o polju *Customisation* si bomo pogledali v nadaljevanju.

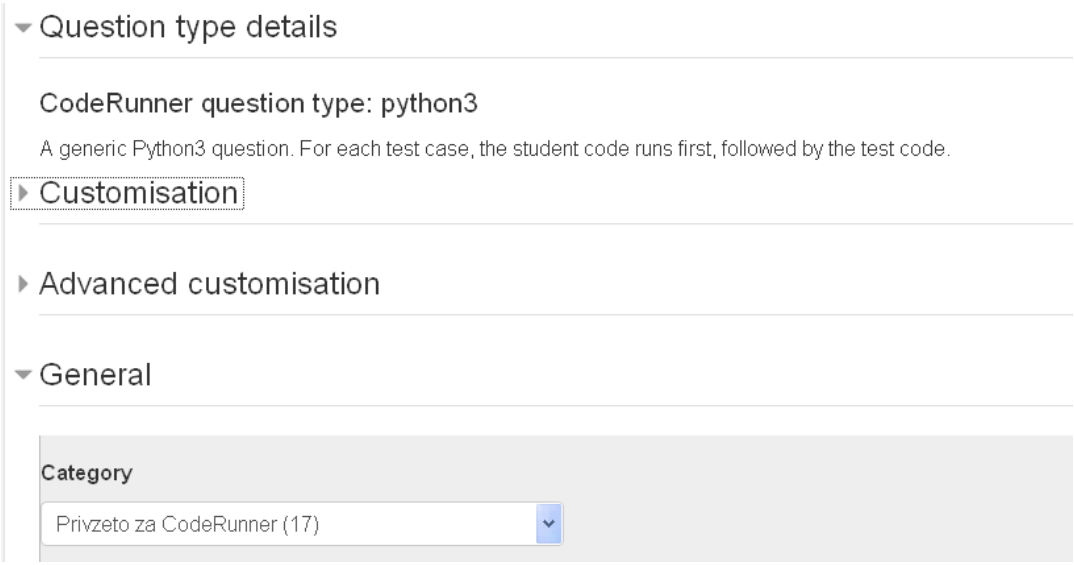
V polju *Answer box* določimo, kako veliko bo okence za odgovor. Širino izberemo v okencu *columns*, višino pa v okencu *rows*. Če obkljukamo ukaz *Use ace*, bo polje za odgovor (ustrezno kodo) uredil urejevalnik kode *ACE JavaScript* (ACE, 2010). ACE v kodi zazna ugnezdene stavke, konstruktorje, komentarje... Pravilno zapisano sintakso ustrezno obarva, med pisanjem kode poskrbi tudi za ustrezen zamik. Koda z uporabo urejevalnika kode ACE je zato preglednejša.

V nastavitvah *Marking* nastavimo, kako bo vprašanje ocenjeno. Če pustimo obkljukano izbiro *All-or-nothing grading*, dobi reševalec točke za nalogo le, če so vsi testni primeri rešeni pravilno. Če odkljukamo to možnost, se vsak testni primer vrednoti posebej. Koliko točk je posamezni testni primer vreden, nastavimo v *Test cases.*

V okencu *Penalty regime* (Slika 22**Error! Reference source not found.**) nastavimo odbitek za vsak napačni poizkus. Če na primer odbitek nastavimo kot 0,0,5,10,20,40… bomo za prva poizkusa ostali še nekaznovani, za vsak nadaljnji napačni odgovor pa nato sledi odbitek, izražen v odstotkih. Znak "*...*" pomeni, da se kazen stopnjuje za razliko med zadnjima dvema številoma, torej v našem primeru za 20 %.

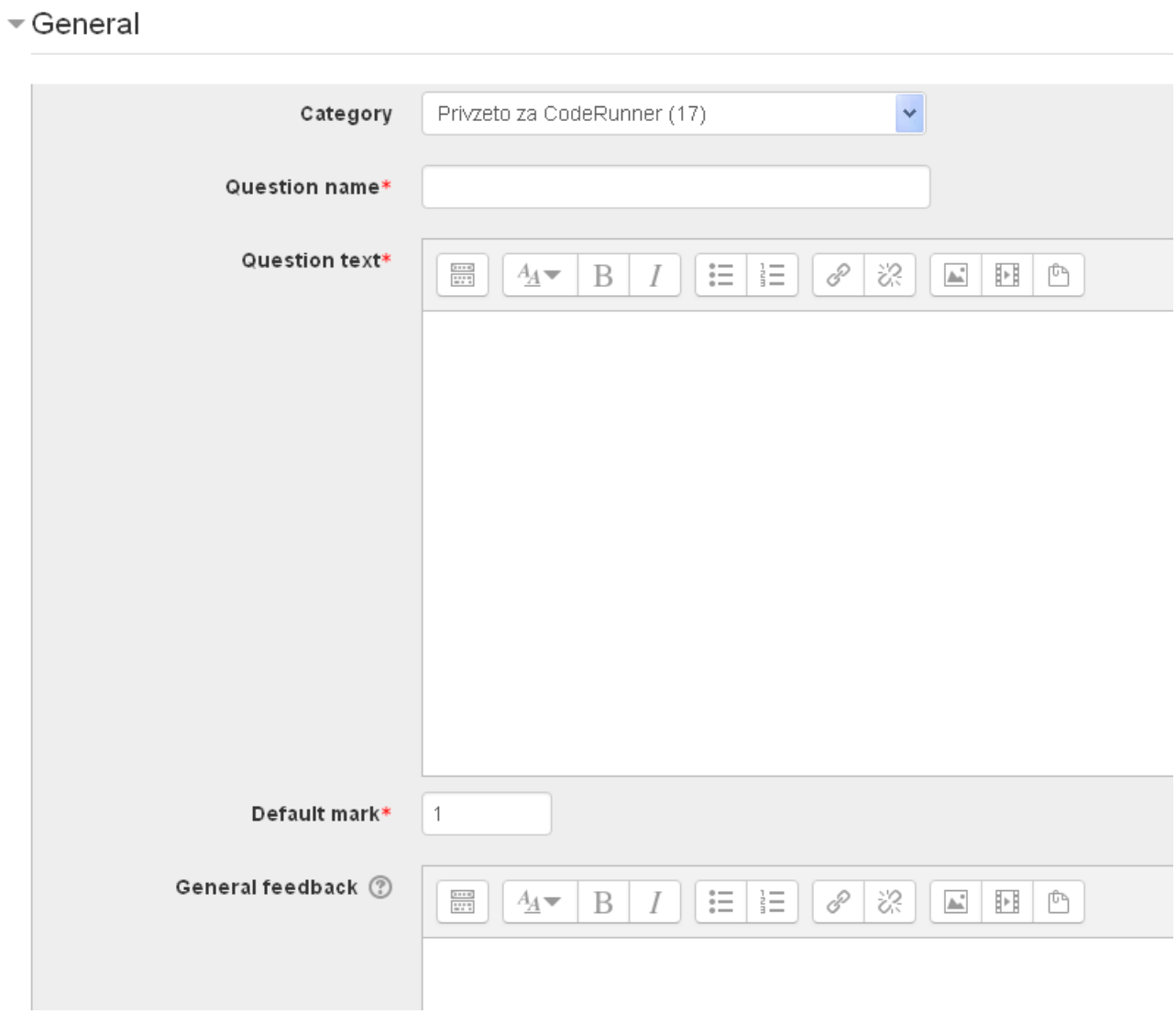
V okencu *Template params** lahko nastavimo posebne parametre, ki so definirani v predlogi. Tako lahko na primer zahtevamo, da mora biti koda napisana z zahtevanimi konstruktorji ali pa obratno, torej da jih ne sme vsebovati. Več o tem si bomo ogledali v nadaljevanju.

V zavihku *Question type details* (Slika 24) dobimo osnovne informacije o izbranem tipu vprašanja. Opisani so tudi posebni parametri, če jih predloga vsebuje. Ker tu običajno ne opravljamo nobenih nastavitev, je ta razdelek "zaprt".



Slika 24: Razdelek *Question type details,* ter dodatna razdelka *Customisation* in *Advanced customisation*

Nato sledi osrednji del, označen z *General*. Tu si izberemo, v katero od vnaprej pripravljenih kategorij (*Category*) bomo shranili naše vprašanje (Slika 25). Za vprašanje moramo izbrati ustrezno ime (*Question name*). Smiselno je podati ime, ki ustrezno opisuje vprašanje, da ga lahko kasneje lažje najdemo v večji zbirki vprašanj.



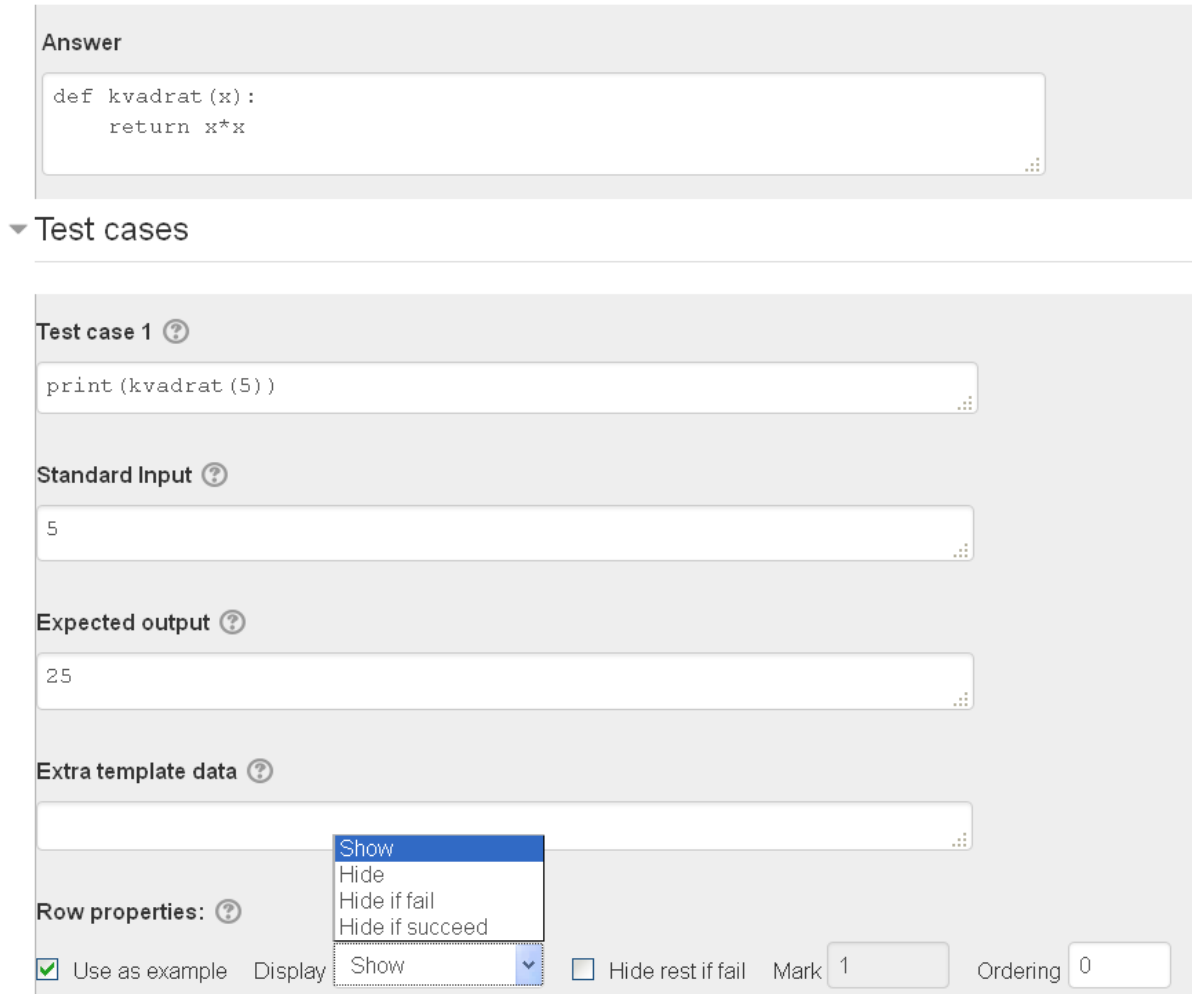
Slika 25: Sestavljanje vprašanja – razdelek *General*

V polje *Question text* napišemo jasna navodila za reševanje naloge. Poudarek je na natančnosti navodil, ker se morajo študentove rešitve običajno natančno ujemati z rešitvami v testnih primerih. V okvirček *Default mark* napišemo, s koliko točkami bomo nalogo vrednotili.

V polje *General feedback* napišemo sporočilo, namig za reševanje naloge ali pripnemo internetno povezavo z razlago snovi. Ker CodeRunner deluje v prilagojenem načinu, bo sporočilo prikazano šele, ko študent zaključi celoten kviz. Zato to polje ni preveč koristno in ga običajno pustimo prazno.

V polje *Answer* napišemo rešitev naloge, ki je običajno v obliki programske kode (Slika 26), ni pa nujno. Lahko je napisano tudi drugo sporočilo, na primer razlaga naloge ali podobno. Rešitev, napisana v polju *Answer,* ne vpliva na testiranje naloge. Za to so odgovorni izključno testni primeri, ki jih sestavimo v razdelku *Test cases*.

V razdelku *Test cases* nastavimo ustrezne testne primere, ki bodo preverjali pravilnost študentovega odgovora. Kako se izvajajo, določimo v predlogi. Smiselno je nastaviti več testnih primerov z različnimi podatki, ki bodo preverjali delovanje študentove kode. V okvirček *Test case 1* vnesemo del kode, ki jo želimo testirati. Običajno je to izpis vrednosti, ki jo vrne študentova funkcija.



Slika 26: Razdelka *Answer* in *Test cases*

V okvirček *Standard Input* vnesemo vhodne podatke za naš testni primer. S temi podatki bo zagnana študentova koda. Njegova koda mora vrniti rešitev, ki je enaka zapisu v okvirčku *Expected output* (pričakovani rezultat).

Polje *Extra template data* se uporablja za napredno testiranje študentovega odgovora. Najpogosteje se uporablja, kadar želimo testiranje kode razdeliti na dva dela. V predlogo je potrebnovstaviti ukaz *{{ TEST.extra }}*. Primer takšne predloge je na Slika 27.

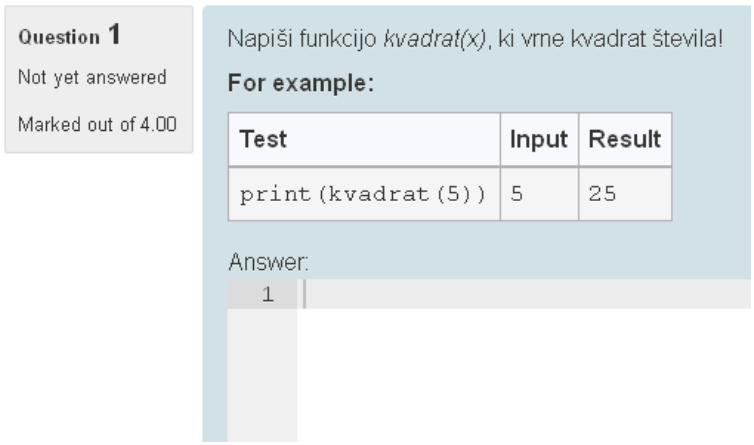
Template

```
1  {{TEST.extra}}
2
3  {{STUDENT_ANSWER}}
4
5  __student_answer__ = """{{ STUDENT_ANSWER | e('py') }}"""
6
7  {{TEST.testcode}}
```

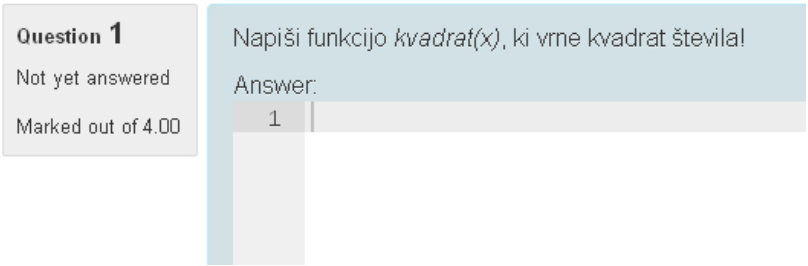
Slika 27: Prilagojena predloga za dodatno testiranje študentovega odgovora

S takšno predlogo lahko kodo testiramo pred izvedbo študentovega odgovora in po izvedbi. Za takšno testiranje moramo dobro poznati delovanje predloge, kar presega okvirje te diplomske naloge. To polje bomo pri sestavljanju svojih vprašanj pustili prazno.

Če obkljukamo izbiro *Use as example,* se nam bo ta testni primer prikazal kot zgled v navodilih naloge. Denimo, da je testni primer sestavljen tako, kot kaže Slika 26. Na Slika 28 vidimo, kako vidi študent nalogo, če je *Use as example* obkljukan, na Slika 29 pa kako, če izbira nima kljukice.



Slika 28: Vprašanje z izbiro *Use as example*



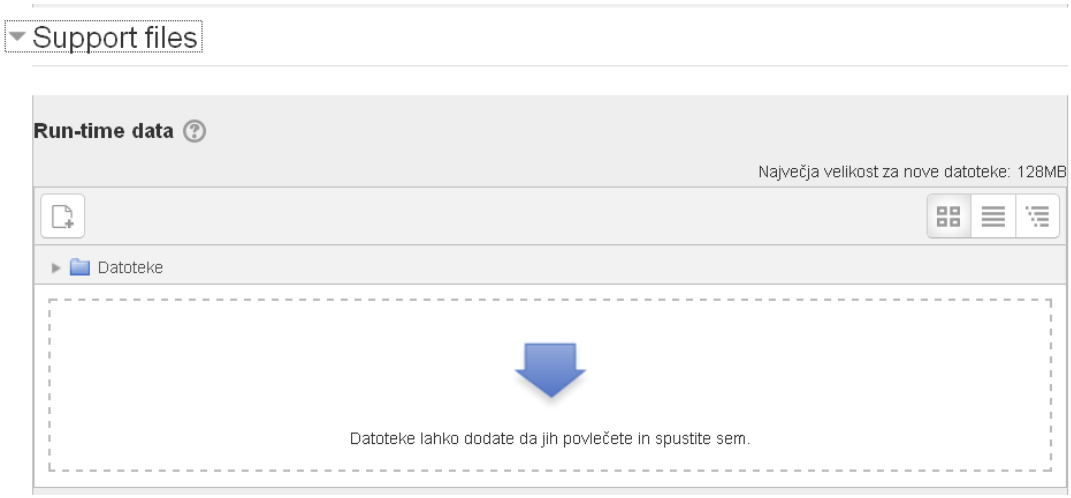
Slika 29: Vprašanje brez izbire *Use as example*

V meniju *Display* (Slika 26) si lahko izberemo, ali naj bo testni primer prikazan v končni tabeli ali ne. Možnost *Show* nam prikaže testni primer v končni tabeli z rezultati, medtem ko možnost *Hide* skrije testni primer. Skriti testni primer se kljub temu izvrši, vendar študentu ni viden. Pri izbiri *Hide if fail* bo testni primer skrit le, če ne uspe. Pri izbiri *Hide if succeed* bo testni primer skrit le, če bo študentova koda uspešno rešila testni primer.

Desno od menija *Display* imamo okvirček *Hide rest if fail.* Če obkljukamo to možnost in testni primer ne uspe, potem bodo vsi nadaljnji testni primeri študentu skriti. Primer: Imamo sestavljenih 5 testnih primerov in v 2. testnem primeru obkljukamo možnost *Hide rest if fail*. Če bo 2. testni primer napačen, se 3.,4. in 5. testni primeri študentu ne bodo prikazali.

V okvirčku *Mark* (Slika 26) nastavimovrednost(oceno) posameznega testnega primera. Ta možnost je na voljo le, kadar nimamo obkljukane možnosti *All-or-nothing grading.* V nasprotnem primeru morajo vsi testni primeri uspešno opraviti test.

Okvirček *Ordering* (Slika 26) služi za spreminjanje vrstnega reda testnih primerov. Vrstni red določamo s števili 0, 10, 20… tako lahko določimo, da se bo testni primer 3 izvedel prvi (če mu nastavimo vrednost 0). Ko bomo vprašanje shranili, bo sistem Moodle testne primere uredil tako, da bo v tabelo na prvo mesto postavil tistega s številom 0, nato tistega s številom 10, tretji bo s številom 20... S tem lažje spremenimo vrstni red testnih primerov, če smo pri sestavljanju vprašanja uporabili napačnega.



Slika 30: Razdelek *Support files*

V razdelku *Support files* imamo možnost dodajanja datoteke, če je to za izvajanje določene naloge potrebno. Če sestavljamo nalogo, v kateri bomo morali spreminjati podatke v določeni datoteki, potem to datoteko preprosto povlečemo v prikazan okvirček (Slika 30).

5.1.1 Predloge

Če želimo razumeti, kako se izvedejo testiranja odgovora, moramo razumeti delovanje predlog. Vsak tip vprašanja (prototip) vsebuje dve predlogi:

per_test_template: Predloga, ki definira, kako je program zgrajen s kombinacijo študentovega odgovora in enega testnega primera. Ta predloga ima običajno dve spremenljivki:

- *{{STUDENT_ANSWER}}* Tu se shrani besedilo, ki ga študent napiše v polje *Answer* (njegova koda)
- *{{TEST.testcode}}* delček kode, ki ga navedemo v testnem primeru za preverjanje odgovora

Ta predloga poskrbi, da se program prevede in zažene skupaj z vhodnimi podatki. Celotni izhodni podatki tega programa se morajo nato ujemati s tem, kar napišemo v polje *Expected output* v tem testnem primeru.

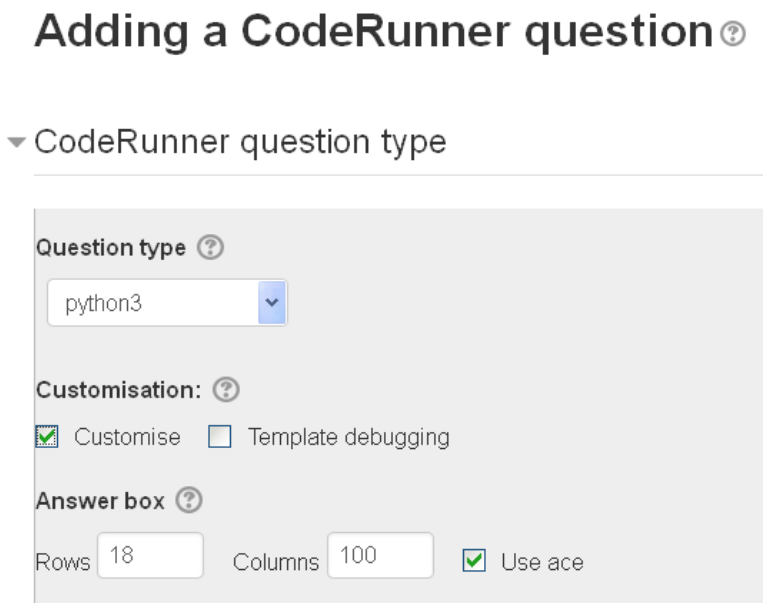
Ujemanje je določeno z izbranim načinom ujemanja. Lahko izbiramo med *'Exact match', 'Nearly exact match'* in *'Regular expression'. 'Exact match*' zahteva, da se izhodni podatki programa in testnega primera ujemajo v popolnosti, medtem ko *'Nearly exact match*' dopušča rahle razlike. Dovoljene so prazne vrstice, dodatni presledki, razlika v mali in veliki začetnici...

Primer:

Sporočili 'Vpisi Prvo Naravno Stevilo: 23' in 'vpisi prvo naravno stevilo:23' bosta v tabeli s testnimi primeri obravnavani enakovredno, če bo za način ujemanja izbrana možnost *'Nearly exact match'*.

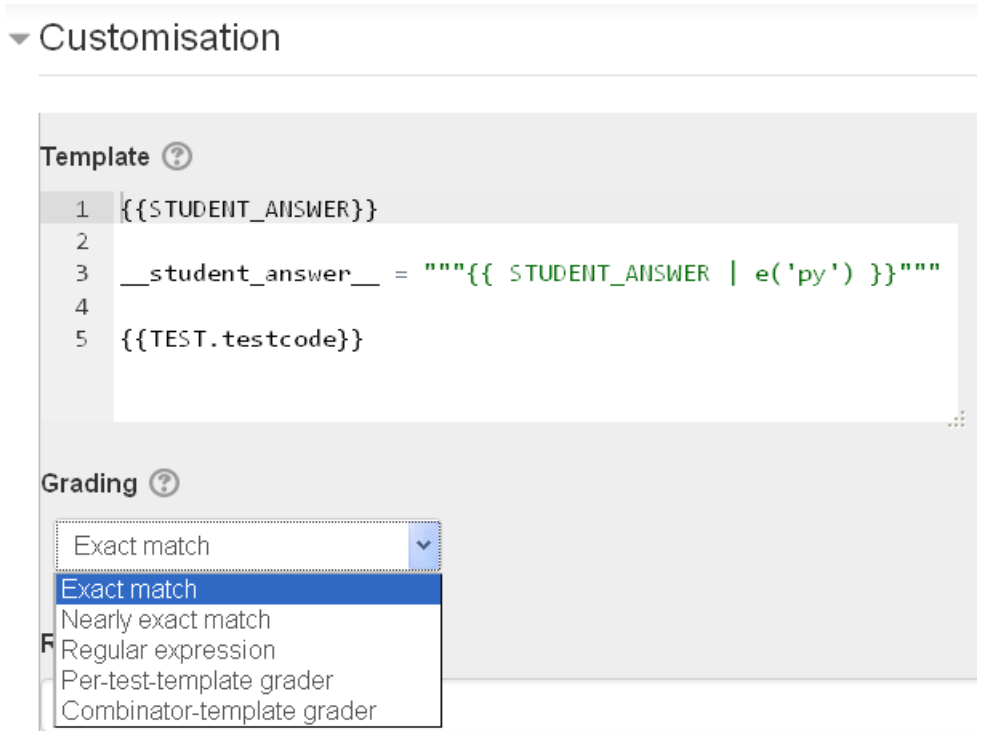
V praksi uporabljamo največkrat način ujemanja *'Exact match'*.

Da bomo razumeli, kako testiramo vprašanje s pomočjo predloge, si oglejmo privzeto predlogo tipa vprašanja *python3.* Na strani, kjer sestavljamo vprašanje v razdelku *Question type,* obkljukamo možnost *Customise* (Slika 31).



Slika 31: Obkljukamo možnost *Customise*

S klikom na možnost *Customise* se nam pojavita dva nova razdelka, *Customisation* in *Advanced Customisation.* V razdelku *Customisation* se nahaja *per_test* predloga (Slika 32).



Slika 32: *per_test* predloga

Kot vidimo na Slika 32, predloga združi študentovo kodo (*{{STUDENT_ANSWER}}*) in kodo v testnem primeru (*{{TEST.testcode}}*) v izvršilni program. Izhodni podatek tega programa se nato primerja s podatkom v polju *Expected output* v določenem testnem primeru. V polju *Grading* nastavimo način ujemanja, opisanega zgoraj. Poglejmo si že prej uporabljen primer, kjer naloga zahteva, da napišemo funkcijo *kvadrat(x)*, ki vrne kvadrat števila *x.* Ta naloga ima rešitev v obliki:

```
def kvadrat(x):
    return x * x
```

Slika 33: Rešitev preprostega problema

Če pogledamo, kako je sestavljen testni primerza to nalogo (Slika 34), vidimo, da imamo v polju *Test case 1* delček kode. Ta delček kode se bo v predlogi izpisal v vrstici namesto *{{TEST.testcode}}*.

Slika 34: Testni primer

Opazimo, da se v predlogi (Slika 35) namesto *{{STUDENT_ANSWER}}* pojavi koda, ki jo študent napiše v polje *Answer*, ko odgovarja na vprašanje. Namesto *{{TEST.testcode}}* pa se pojavi delček kode iz polja *Test case 1*. Taka predloga se zdaj obravnava kot en izvršilni program. Postopek se ponovi z vsemi testnimi primeri, ki so podani v nalogi. V predlogi se bo spreminjala samo zadnja vrstica, kjer bodo podani drugi testni primeri.

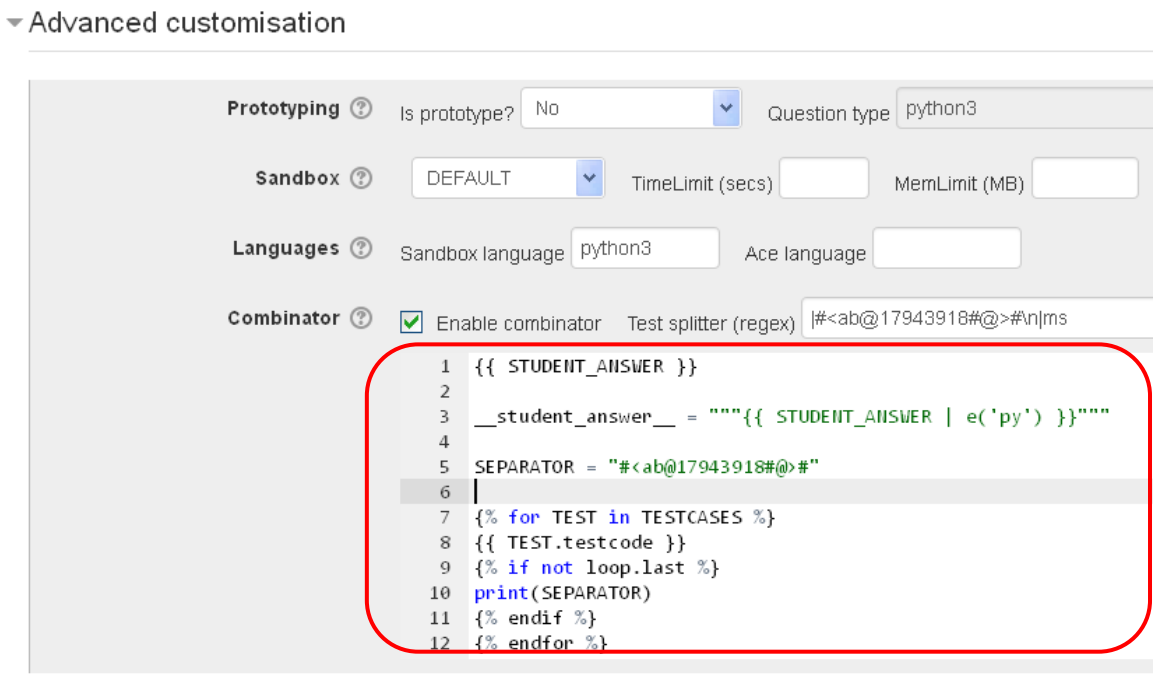
```
def kvadrat(x):
    return x*x

__student_answer__ = """def kvadrat(x):
    return x*x"""

print(kvadrat(5))
```

Slika 35: Primer *per_test* predloge za določen testni primer

Combinator_template je kombinirana predloga, ki jo poskuša CodeRunner uporabiti najprej. Ta predloga poskuša združiti vse testne primere skupaj s študentovim odgovorom v en izvršilni program. Z določenim separatorjem poskrbi, da so v končni tabeli vsi testni primeri obravnavani ločeno.

Ta predloga se ne uporablja, kadar imamo v testnih primerih določene vhodne podatke. Vsak testni primer se potem jemlje kot neodvisen od drugih. Prav tako ni uporabna, ko se v kodi srečamo z izredno napako ali izjemo. Takrat se vsi testni primeri ponovno izvršijo s pomočjo predloge *per_test*.

Poglejmo si privzeto kombinirano predlogo. Kot smo že omenili, se nam s klikom na možnost *Customise* v razdelku *Question type* (Slika 31) odpreta dva nova razdelka, *Customisation* in *Advanced customisation.* Kombinirana predloga se nahaja v razdelku *Advanced customisation.* Razdelek *Advanced customisation* si bomo pogledali v naslednjem poglavju, zaenkrat si poglejmo samo kodo kombinirane predloge, ki je navidez skrita. Njena koda se prikaže, ko se s kurzorjem približamo polju pod možnostjo *Enable combinator* (Slika 36).



Slika 36: Prikaz kombinirane predloge

Opazimo, da je kombinirana predloga sestavljena podobno kot predloga *per_test*. V tej predlogi je definiran še ločilni niz (*SEPARATOR*), potem pa si sledijo testni primeri z različnimi vhodnimi podatki. Med posameznimi tesnimi primeri z ukazom '*print(SEPARATOR)'* izpišemo ločilni niz. Ta ukaz bo poskrbel, da se bodo testni primeri v tabeli izpisali ločeno.

Primer sestavljanja vprašanja s prilagoditvijo predloge si bomo ogledali v nadaljevanju.
Če želimo spreminjati ali ustvariti svojo kombinirano predlogo, moramo dobro razumeti delovanje zanke v predlogi in TWIG mehanizma, kar presega okvirje te diplomske naloge.

Poglejmo si primer, kako kombinirana predloga združi študentov odgovor z vsemi testnimi primeri. Za zgled si ponovno izberimo zgoraj omenjeno nalogo *kvadrat(x)*. Predvidevajmo, da imamo za to nalogo sestavljene 3 testne primerez različnimi vhodnimi podatki (*5,-3 in 0*). Predloga bo združila vse testne primere skupaj z odgovorom iz polja *Answer*, ki ga poda študent (Slika 37).

```
def kvadrat(x):
    return x*x

__student_answer__ = """def kvadrat(x):
    return x*x"""

SEPARATOR = "#<ab@17943918#@>#"

print(kvadrat(5))
print(SEPARATOR)
print(kvadrat(-3))
print(SEPARATOR)
print(kvadrat(0))
```

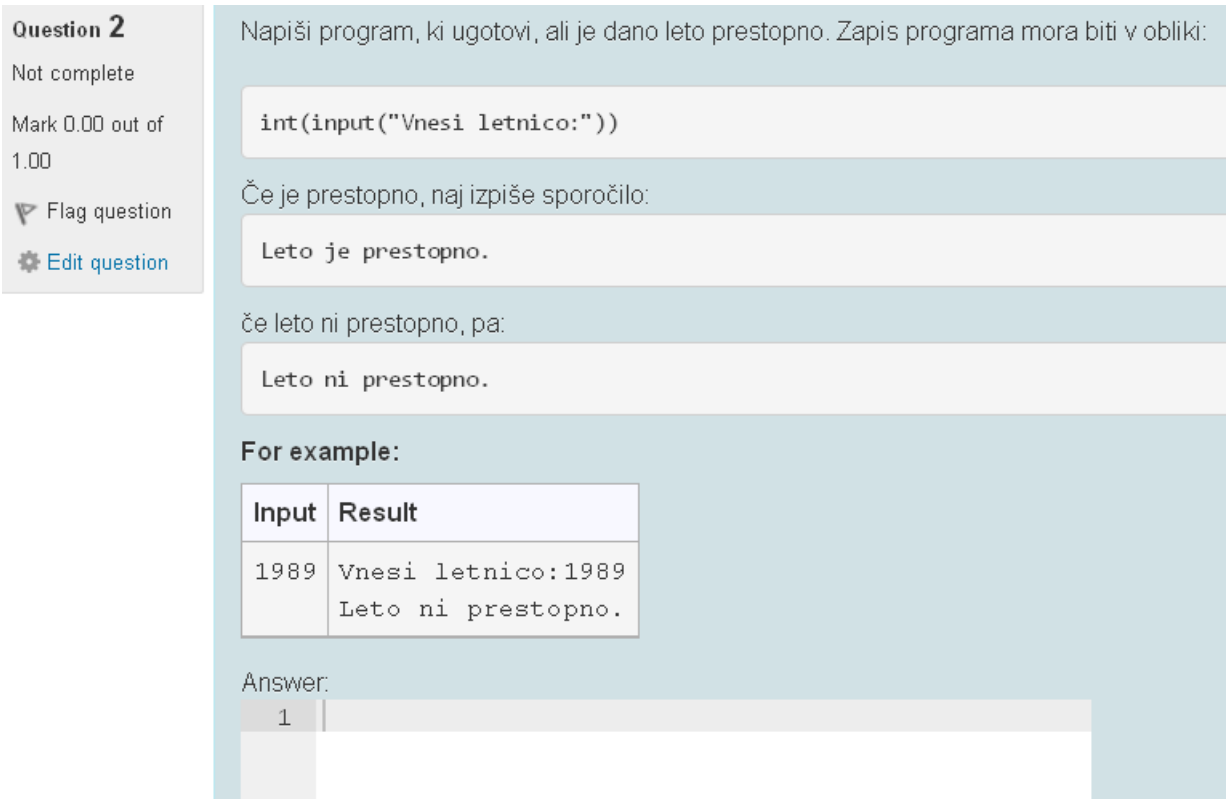
Slika 37: Kombinirana predloga

Opazimo, da je prvi del kode enak kot v *per_test* predlogi. Definiran je še separator, potem pa si sledijo vsi trije testni primeri z različnimi vhodnimi podatki, ločeni z ukazom '*print(SEPARATOR)',* ki poskrbi, da se bodotestni primeri v tabeli izpisali ločeno.

5.1.2 Napredna uporaba predlog

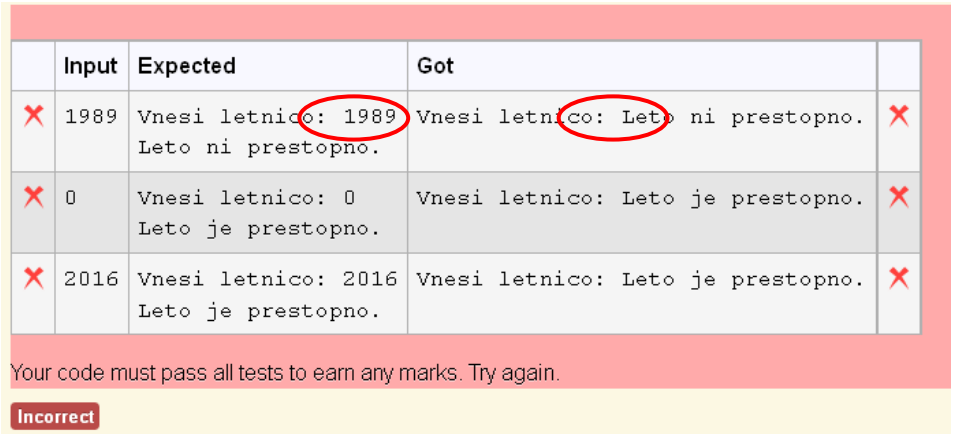
Pogosto se srečamo s težavo, ko vgrajeni tip vprašanja (prototip) ne ustreza za sestavljanje naloge. Težava se pojavi, ko se v predlogi neuspešno združita odgovor študenta in testni primer. Takrat je treba predlogo ustrezno prilagoditi. Če sestavljamo vprašanja v programskem jeziku Python3, imamo med vgrajenimi tipi vprašanj na voljo samo prototip 'napiši funkcijo'.

Oglejmo si primer, kjer naloga zahteva, da v programskem jeziku Python3 napišemo program, ki ugotovi, ali je dano leto prestopno (Slika 38).



Slika 38: Naloga tipa 'napiši program'

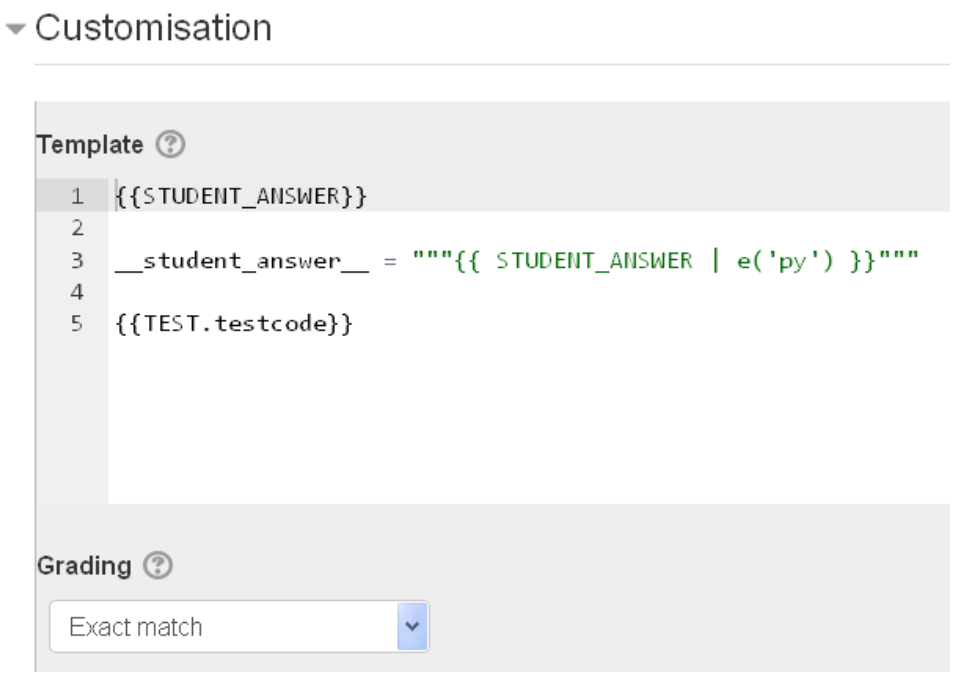
Take naloge zahtevajo od študenta, da iz danih vhodnih podatkov, ki jih vnesemo neposredno v program, vrne želene izhodne podatke. Testiranje takšnih nalog je zelo podobno testiranju nalog tipa 'napiši funkcijo'. Podati moramo vhodne in pričakovane izhodne podatke. Težava nastane, ko se vhodni podatki (ki jih je lahko več) ne izpišejo na zaslon.



Slika 39: Vhodni podatek se ne izpiše

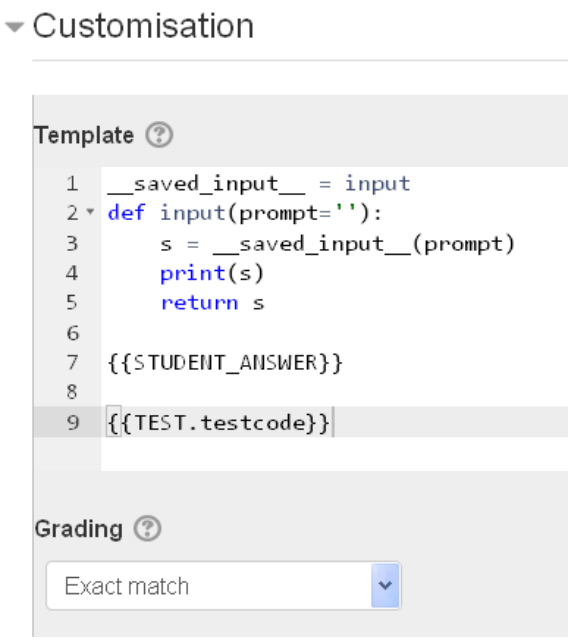
Slika 39 prikazuje pravilno delovanje študentovega programa, manjka le izpis letnice, ki je podana kot vhodni podatek.

Če želimo, da se vhodni podatek izpiše, bomo morali predlogo prilagoditi. Kot smo omenili v poglavju 5.1.1, to storimo tako, da na strani, kjer sestavljamo vprašanje v razdelku *Question type,* obkljukamo možnost *Customise* (Slika 31). Pojavita se dva nova razdelka, *Customisation* in *Advanced Customisation.* Za prikaz predloge odpremo razdelek *Customisation* (Slika 40).



Slika 40: Standardna predloga *per_test*

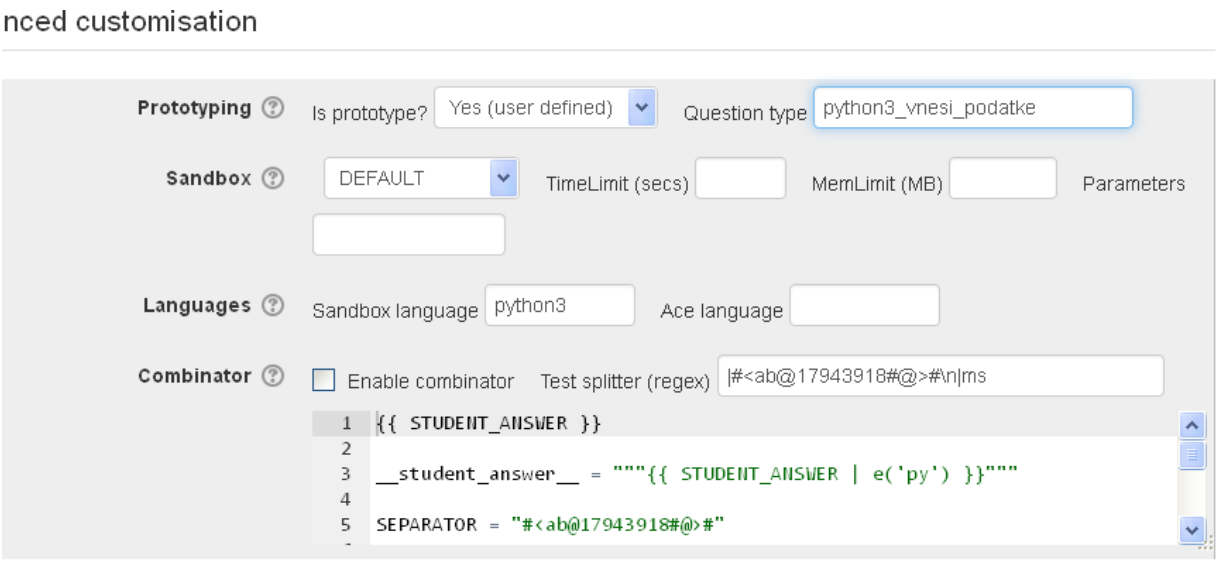
V polju *Template* je napisana standardna koda predloge, ki deluje za tipe nalog 'napiši funkcijo'. Kodo bomo prilagodili tako, da bo podatek iz Pythonove vgrajene funkcije *input()* izpisala tudi na zaslon (Slika 41).



Slika 41: Prilagojena predloga

Ker imamo v testnih primerih določene vhodne podatke, bo CodeRunner testne primere obravnaval s pomočjo *per_test* predloge. V tem primeru kombinirane predloge ni treba spreminjati.

Ker bomo večkrat sestavljali naloge tipa 'napiši program', si je pametno to predlogo shraniti kot nov prototip. To naredimo tako, da odpremo razdelek *Advance customisation,* kjer zagledamo naslednje možnosti (Slika 42).



Slika 42: Razdelek *Advanced customisation*

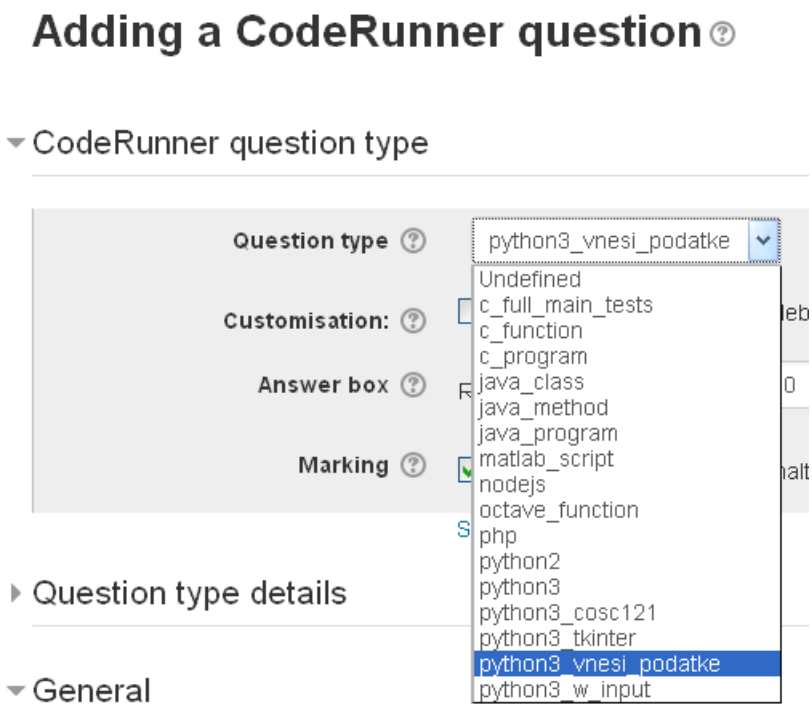
V okvirčku *Is prototype?* izberemo možnost *Yes (user defined).* V polje *Question type* napišemo ime našega prototipa, na primer *python3_vnesi_podatke*.

Nastavitve peskovnika (*Sandbox*) pustimo privzete (okenca pustimo prazna). Če želimo, lahko omejimo čas izvajanja enega testnega primera, izraženega v sekundah (okence *TimeLimit*). Prav tako lahko omejimo porabo strežnikovega delovnega spomina, izraženega v MB (okence *MemLimit*). V okence *Parameters* vpišemo dodatne možnosti za peskovnik (na primer dodatne nastavitve pri prevajanju programa, API-ključ...).

V okvirček *Sandbox language* vnesemo programski jezik, v katerem se naloga sestavlja. Vpisali bomo *python3.* Okence *Ace language* pustimo prazno. Po privzetih nastavitvah se bo upošteval enak programski jezik, kot ga uporablja peskovnik, torej *Python3*.

Ker bomo ta tip vprašanja uporabljali samo za naloge tipa 'napiši program', bomo možnost *Enable combinator* odkljukali. V tem prototipu se bo torej uporabljala samo predloga *per_test*.

Ko bomo sestavljeno nalogo shranili, se samodejno shrani tudi prototip. Ko bomo sestavljali naslednjo nalogo, bo naš novi prototip na voljo v meniju *Question type* (Slika 43).



Slika 43: Novi prototip se prikaže v meniju *Question type*

Med tipi vprašanj že imamo prototip z enako prilagojeno predlogo, kot jo ima *python3_vnesi_podatke.* To je prototip *python3_w_input*, ki ima prilagojeno tudi kombinirano predlogo. Namen ustvarjanja našega prototipa je bil zgolj ta, da smo pokazali možnost ustvarjanja lastnega prototipa in njegovo kasnejšo uporabo pri sestavljanju vprašanj.

Na univerzi Canterbury na Novi Zelandiji so ustvarili svoj prototip ***python3_cosc121***, ki ga uporabljajo za ocenjevanje kvizov in izpitov. Ta prototip preveri, če je koda napisana v skladu s Pylint sistemom, preden se le-ta izvrši.

Pylint je sistem za statično preverjanje kode. To pomeni, da lahko analizira kodo, preden se ta izvrši. Pylint poišče možne napake v kodi, poskuša uveljaviti standard kodiranja in slog kodiranja. Za slog kodiranja uporablja slogovni priročnik *PEP 8*.

Poleg preverjanja s Pylint sistemom ima prototip ***python3_cosc121*** vgrajene še številne druge uporabne parametre, kot so:

- *Je funkcija* (*isfunction*): Ta parameter poskrbi, da je koda definirana kot funkcija. Če bomo sestavljali vprašanje tipa 'napiši program', moramo ta parameter nastaviti na *False*
- *Pylint nastavitve* (*pylintoptions*): dodatne nastavitve pri preverjanju s sistemom *Pylint*. Primer: ukaz *{"pylintoptions":["--max-statements=20", "--max-args=3"]}* bo vsako definirano funkcijo v kodi omejil z uporabo največ 20 ukazov (*"--max-statements=20"*) in s 3 argumenti (*"--max-args=3"*)
- *Prepovedani konstruktorji* (*proscribedconstructs*): določi, katere konstruktorje je prepovedano uporabiti v kodi
- *Obvezni konstruktorji* (*requiredconstructs*): določi, katere konstruktorje je treba uporabiti v kodi
- *Dovoli globalne spremenljivke* (*allowglobals*): dovoli uporabo globalnih spremenljivk
- *Maksimalno število konstant* (*maxnumconstants*): parameter določi maksimalno število konstant, ki jih smemo uporabljati v kodi
- *Brez zagona* (*norun*): študentova koda se ne izvrši, izvršila se bo samo testna koda, ki smo jo podali
- *Zaženi dodatno* (*runextra*): s tem ukazom dobimo v predlogi dodatno polje *{{ TEST.extra | e('py') }}*, za dodatno testiranje kode (glej razlago v poglavju 5.1)

V poglavju 6 bomo pri sestavljanju nalog uporabili tudi ta prototip. Pokazali bomo, kako lahko uporabljamo določene parametre, opisane zgoraj.

5.2 Primeri sestavljanja vprašanja

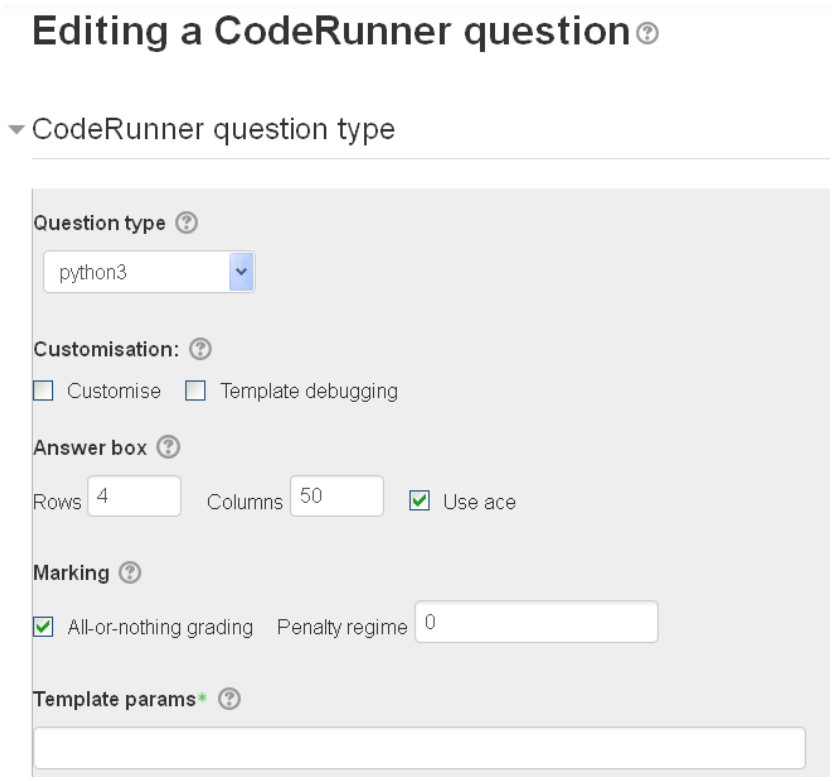
V tem poglavju si bomo ogledali sestavljanje preprostih vprašanj. Prvo vprašanje bo tipa 'napiši funkcijo', kjer bomo uporabili tip vprašanja *python3* s klasično predlogo. Drugo vprašanje bo tipa 'napiši program', kjer bomo uporabili naš novi tip vprašanja *python3_vnesi_podatke* s prilagojeno predlogo. Poudarek bo na razlagi vseh možnosti pri sestavljanju vprašanj. Načini preverjanja vprašanja in smiselno sestavljanje testnih primerov bomo podrobneje predstavili v poglavju 6 na praktičnih primerih.

5.2.1 Vprašanje tipa 'napiši funkcijo'

Za zgled si oglejmo postopek sestavljanja preprostega vprašanja v CodeRunnerju.
Ostanimo pri svojem zgledu, ki od učenca zahteva, da napiše funkcijo *kvadrat(x),* ki vrne kvadrat števila *x*. To nalogo bomo sestavili s pomočjo tipa vprašanja *python3*.

Kot smo opisali v prejšnjem poglavju 5.1, kliknemo za ustvarjanje novega vprašanja v meniju *administration* (Slika 19) na razdelek *Question bank*. Ko se odpre zbirka vprašanj, kliknemo na *Create a new question* (Slika 20), za tip vprašanja pa si izberemo CodeRunner (**Slika 21**).

Prikaže se glavna stran za sestavljanje vprašanja (Slika 44).

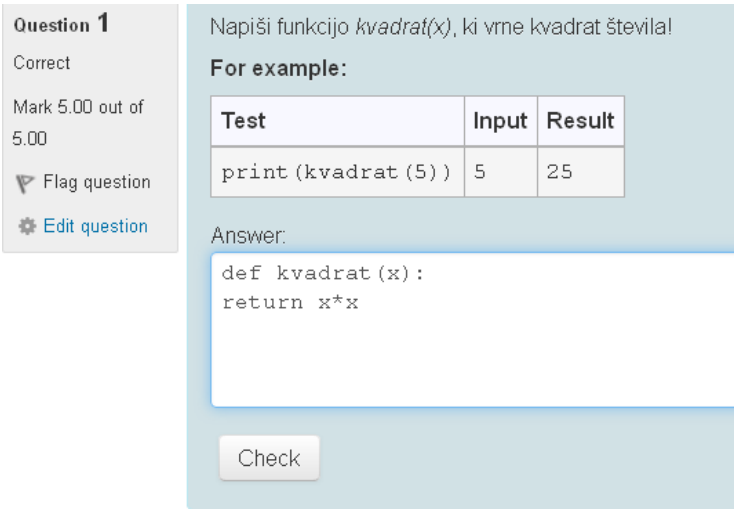


Slika 44: Sestavljanje vprašanja

Ker sestavljamo nalogo v programskem jeziku Python3, v polju *Question type* izberemo python3. Naloga je tipa 'napiši funkcijo', spreminjanje predloge ne bo potrebno. Zato pustimo polje *customize* neobkljukano.

Velikost okenca za odgovor zmanjšamo, saj je odgovor kratka koda. Nastavimo ga recimo na 4 vrstice in 50 stolpcev. Izbiro *Use ace* pustimo obkljukano, da študentu olajšamo pisanje odgovora v okvirček. Urejevalnik kode ACE pomaga študentu pri oblikovanju kode, obarva konstruktorje, samodejno naredi ustrezne zamike... Taka koda je preglednejša, zato je manj možnosti, da bo študent naredil sintaktično napako, tudi če bo kodo pisal neposredno v okence.

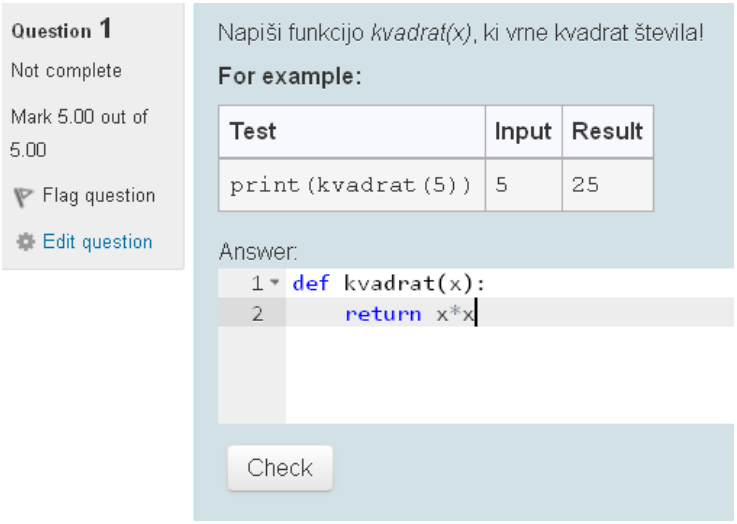
Slika 45 prikazuje polje *Answer* brez uporabe urejevalnika ACE.



Slika 45: Polje *Answer* brez ACE urejevalnika kode

Opazimo, da je koda v polju *Answer* enobarvna, prav tako pa ni ustreznega zamika v drugi vrstici kode. Študent mora biti pazljiv in tu narediti ustrezen zamik kode, da bodo stavki pravilno gnezdeni.

Sedaj pa si poglejmo polje *Answer* z uporabo urejevalnika kode ACE (Slika 46).



Slika 46: Polje *Answer* z ACE urejevalnikom kode

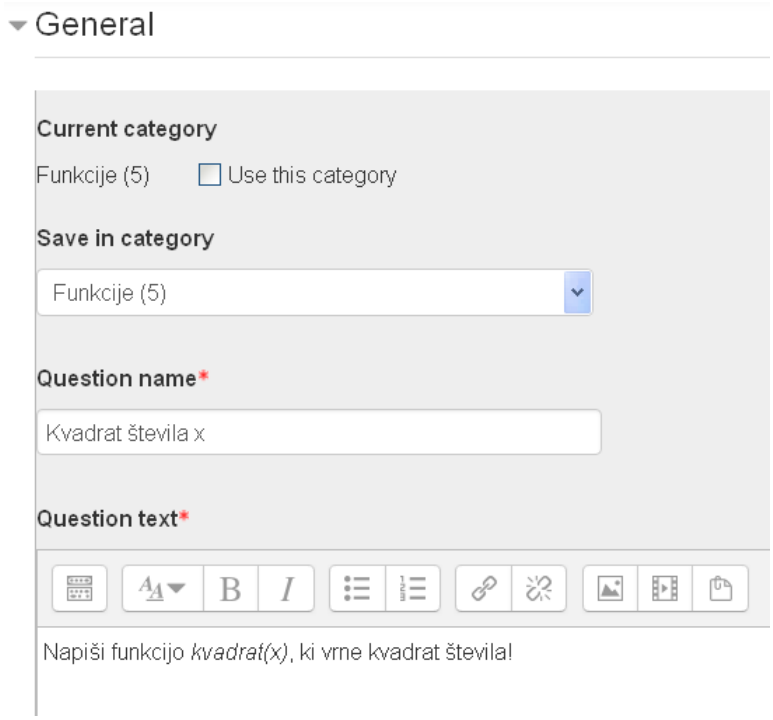
Opazimo, da je koda v tem primeru drugačna. Koda je oštevilčena po vrsticah, ukaza 'def' in 'return' sta obarvana modro. Zamik v 2. vrstici se je naredil samodejno. Taka koda je za študenta preglednejša.

V razdelku *Marking* si izberemo način ocenjevanja naloge. Obkljukajmo možnost *All-or-nothing grading*. To pomeni, da zahtevamo pravilnost vseh testnih primerov za pridobitev točk. Če program ne prestane enega ali več testov, točk ne dobimo. *Penalty regime* nastavimo na 0, ker študenta v primeru napačnega odgovora ne želimo kaznovati z odbitkom. Ker smo v to polje vnesli vrednost 0, nam kasneje v razdelku *Multiple tries* ne bo potrebno spreminjati nastavitev. To bi bilo potrebno le, če bi polje *Penalty regime* pustili prazno.

Tudi polje *Template params** pustimo prazno, ker ne bomo uporabili nobenih posebnih parametrov predloge.

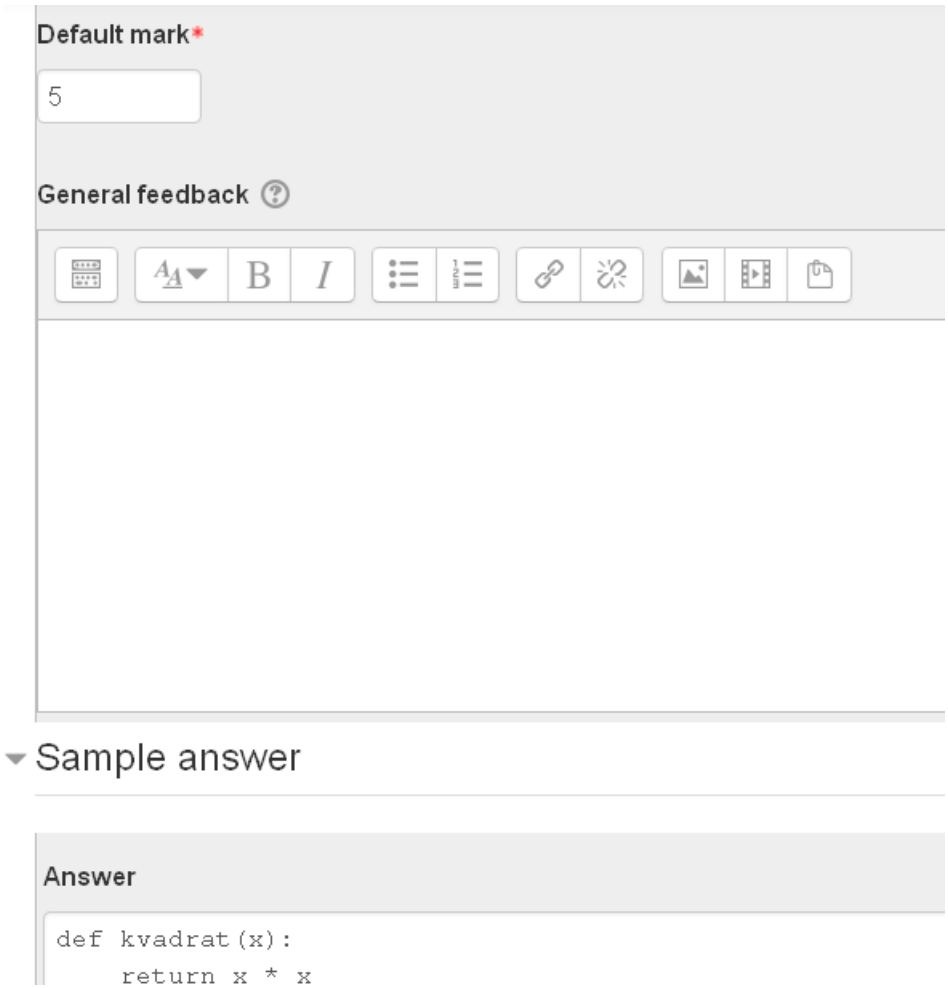
Nadaljujemo s sestavljanjem vprašanja. V razdelku *General* (Slika 47) nastavimo ime vprašanja v polju *Question name.* Za ime vprašanja smo si izbrali *Kvadrat števila x*. V meniju *Save in category* si izberemo kategorijo, v katero shranimo naše vprašanje. Shranili smo ga v kategorijo *Funkcije.*

V polje *Question text* napišemo navodila za reševanje naloge. Predloga ima privzeto nastavljeno ocenjevanje na *'Exact match*'. To pomeni, da se morajo izhodni podatki testnih primerov natančno ujemati z izhodnimi podatki študentove kode. Upoštevati moramo tudi morebitne prazne vrstice, odvečne presledke...



Slika 47: Sestavljanje vprašanja – razdelek *General*

V polju *Default mark* izberemo vrednost točk naloge. Recimo, da bo vredna 5 točk (Slika 48). Polje *General feedback* pustimo prazno (glej razlago v poglavju 5.1), v *Sample answer* pa napišemo rešitev naše naloge, ki bo študentu prikazana na koncu, ko kviz odda in zaključi.

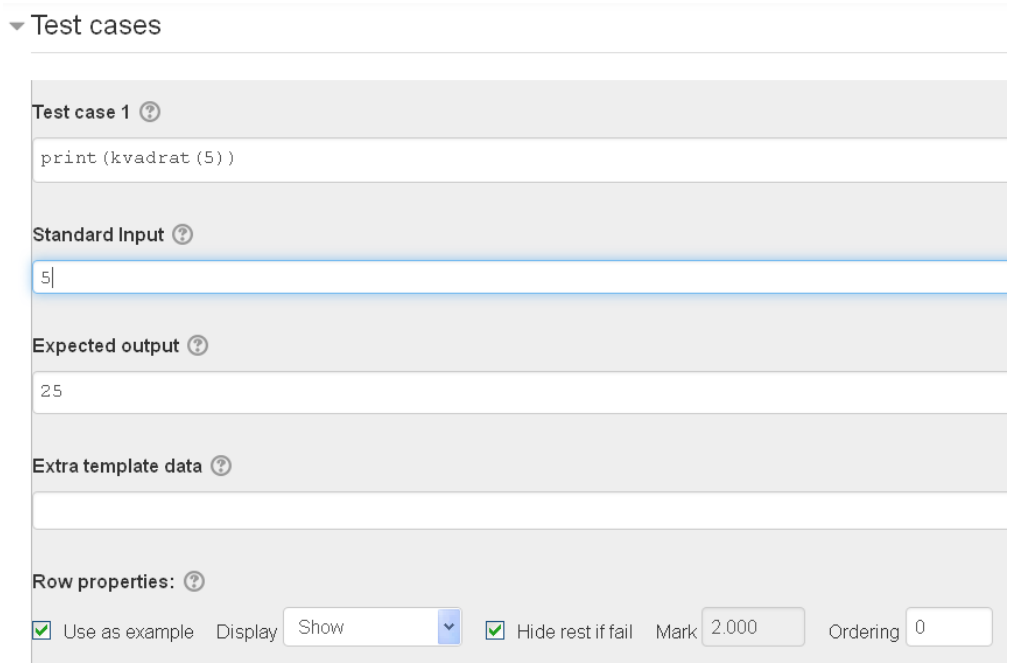


Slika 48: Sestavljanje vprašanja – razdelek *Sample answer*

Zdaj moramo sestaviti še testne primere. Postopek ter kakšne testne primere in na kaj moramo biti pozorni pri sestavljanju testnih primerov si bomo podrobneje ogledali v poglavju 6.

V razdelku *Test cases* (Slika 49) v polje *Test case 1* vpišemo delček kode, s katerim bomo testirali študentovo kodo. V polje *Standard Input* vpišemo vhodni podatek za testiranje kode. Ker je naloga tipa 'napiši funkcijo in nam predloge ni potrebno spreminjati, bi lahko polje *Standard Input* pustili prazno. Razlika bi bila le v tabeli s testnimi primeri, v kateri ne bi bilo stolpca *Input*.

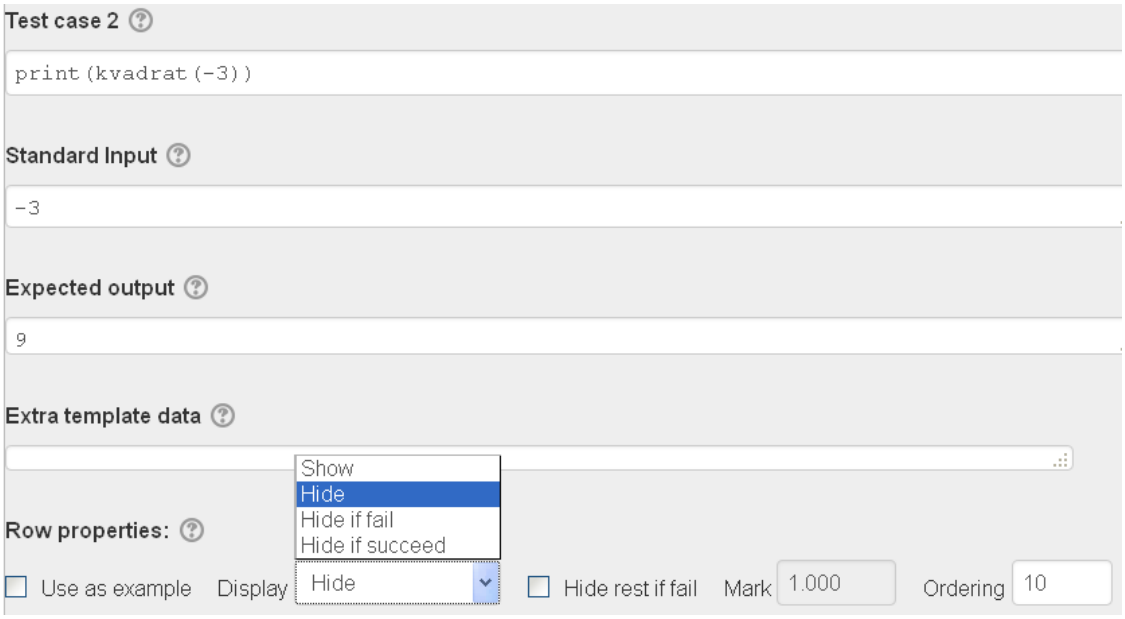
V polje *Expected output* vnesemo podatek, ki ga pričakujemo glede na vnesen vhodni podatek. Ker je naš vhodni podatek število 5 in ker naloga pravi, da mora funkcija vrniti kvadrat našega števila, pričakujemo, da bo funkcija vrnila število 25.



Slika 49: Sestavljanje testnega primera 1

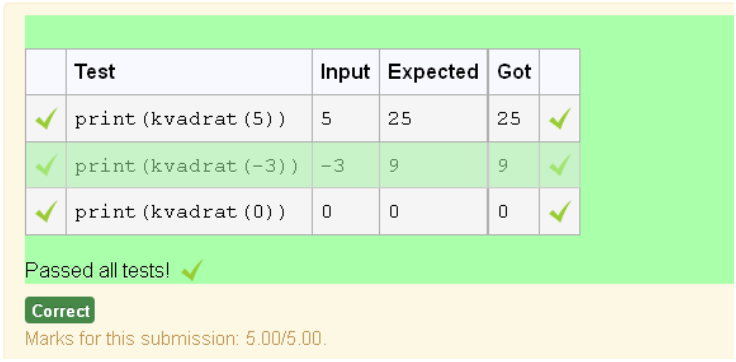
V razdelku *Row properties* imamo obkljukano možnost *Use as example.* To pomeni, da bo ta testni primer prikazan kot zgled v navodilih naloge v obliki tabele. Možnost *Use as example* si bomo podrobneje ogledali v poglavju 6.1. Opazimo, da vrednosti v polju *Mark* ne moremo spreminjati. Razlog za to je obkljukana možnost *All-or-nothing grading* v razdelku *Question type.* Ta nam pove, da mora študentova koda uspešno prestati vse testne primere, da dobi študent točke. Podatek v polju *Ordering* nam pove, da se bo ta testni primer nahajal v prvi vrstici končne tabele s testnimi primeri.

Podobno sestavimo še ostale testne primere. Vsa polja izpolnimo na enak način kot v testnem primeru 1, vendar z različnimi vhodnimi in izhodnimi podatki. V meniju *Display* bomo izbrali možnost *Hide* (Slika 50).



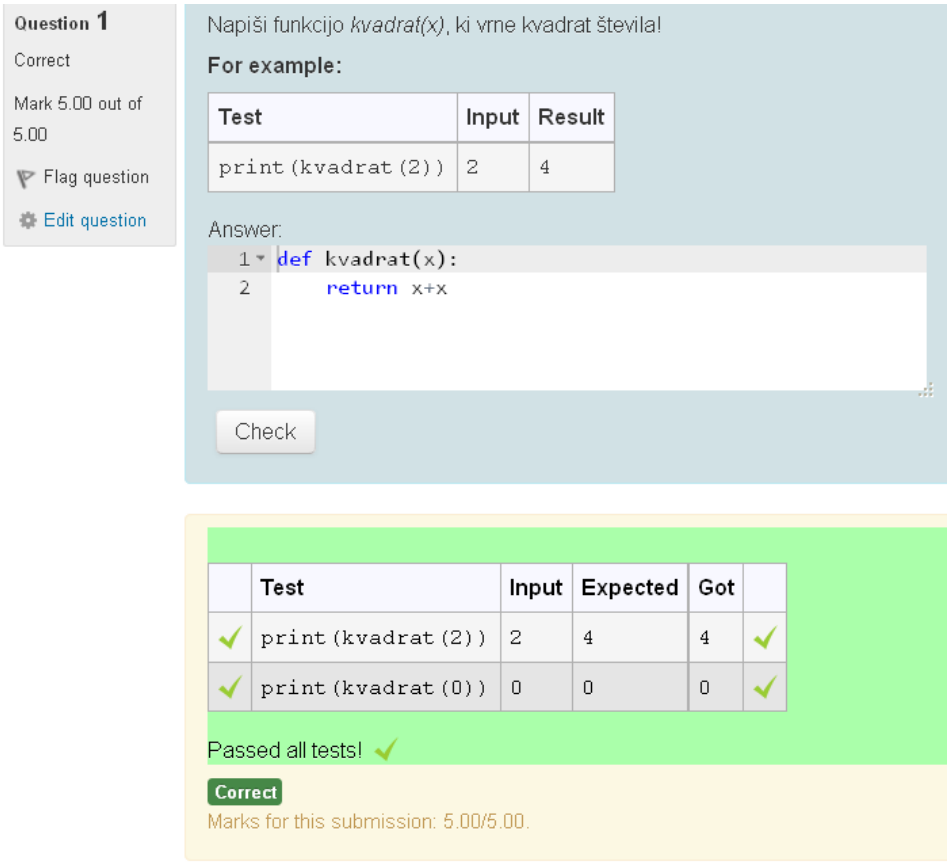
Slika 50: Testni primer 2

Ko študent odda odgovor, opazimo, da je drugi testni primer v tabeli drugače obarvan. To pomeni, da je ta testni primer viden le nam, študentom pa ostaja skrit (Slika 51).



Slika 51: Skriti testni primer 2

Prikrivanje testnih primerov je uporabno, saj lahko preprečimo, da bi študent delovanje kode prilagodil samo za določene testne primere. Če si za testne primere izberemo slabe vhodne podatke, se lahko hitro zgodi, da bodo študenti zlorabili slabo postavljene testne primere. Recimo, da v svoji nalogi nimamo testnega primera z negativnim številom in za testni primer 1 uporabimo vhodni podatek *2*, za testni primer 2 pa vhodni podatek *0*. Študent lahko z napačno kodo (*x+x* namesto *x*x*) zadovolji zahteve testnih primerov (Slika 52).



Slika 52: Slabo izbrani testni primeri

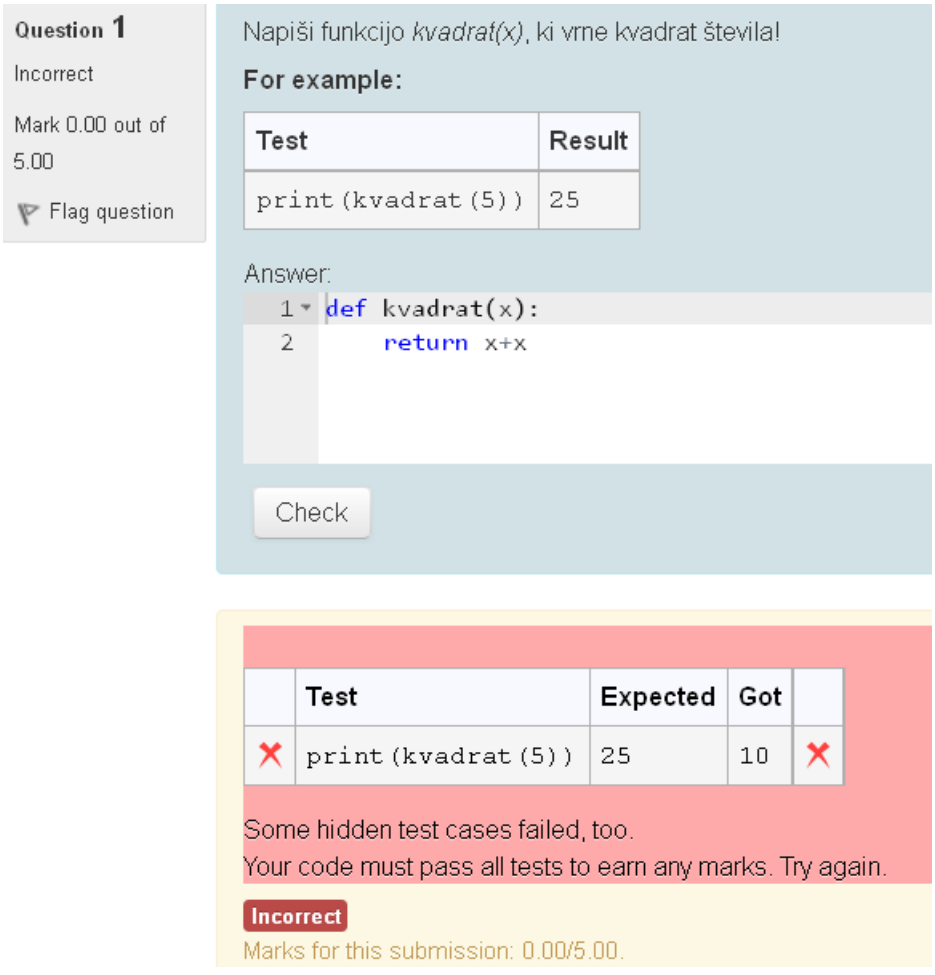
Prav tako se lahko s tem izognemo rešitvi v obliki programa, ki pove le, kakšno vrednost mora program vrniti v primeru določenega vhodnega podatka:

```
def kvadrat(n):
  if n == 5:
     return 25
  if n == -3:
    return 9
  if n == 0:
    return 0
```

Zato je prikrivanje testnih primerov koristno. Študent tako ne vidi vseh načinov, kako preverjamo pravilnost njegove kode.

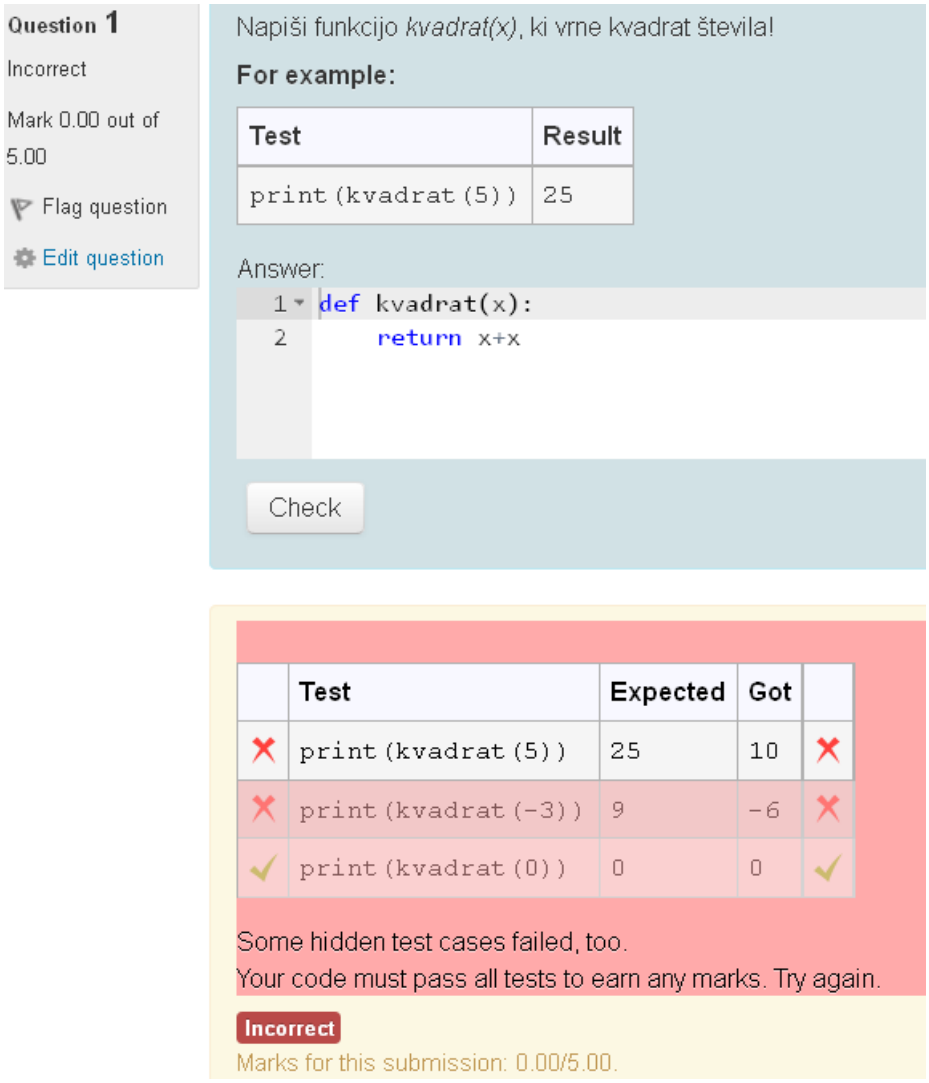
Poglejmo si še možnost *Hide rest if fail*. Če obkljukamo to možnost (Slika 53**Error! Reference source not found.**) in testni primer ne uspe, potem bodo vsi nadaljnji testni primeri študentu skriti (Slika 54).

Slika 53: Izberemo možnost *Hide rest if fail*



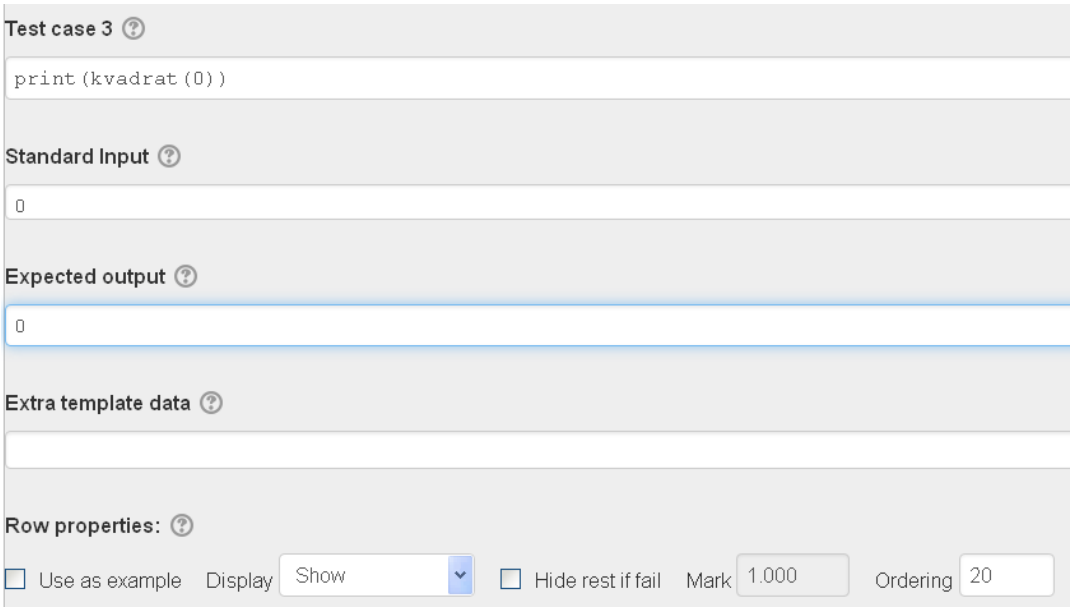
Slika 54: Ostali testni primeri so študentu skriti

Študentu je viden samo prvi testni primer, ostala dva sta prikrita. Poglejmo si, kako mi (kot učitelj) vidimo študentov napačen odgovor (Slika 55).



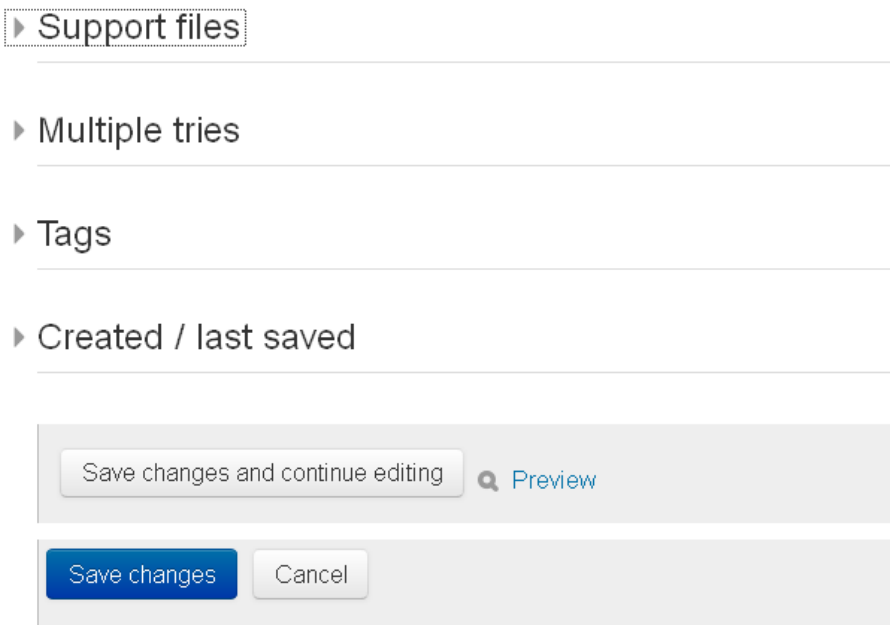
Slika 55: Učitelj vidi skrite testne primere

Podobno sestavimo še testni primer 3, ki bo imel za vhodni podatek 0 (Slika 56).



Slika 56: Testni primer 3

Ker naloga za izvedbo ne potrebuje nobene dodatne datoteke, lahko razdelek *Support files* spustimo. V razdelku *Multiple tries* nam kazni za napačen odgovor ni treba nastavljati, ker smo to storili že na začetku sestavljanja vprašanja v razdelku *Question type,* polje *Penalty regime.* Da shranimo sestavljeno vprašanje, kliknemo na gumb *Save changes* (Slika 57). Naše novo sestavljeno vprašanje se bo prikazalo v kategoriji, v katero smo ga shranili.

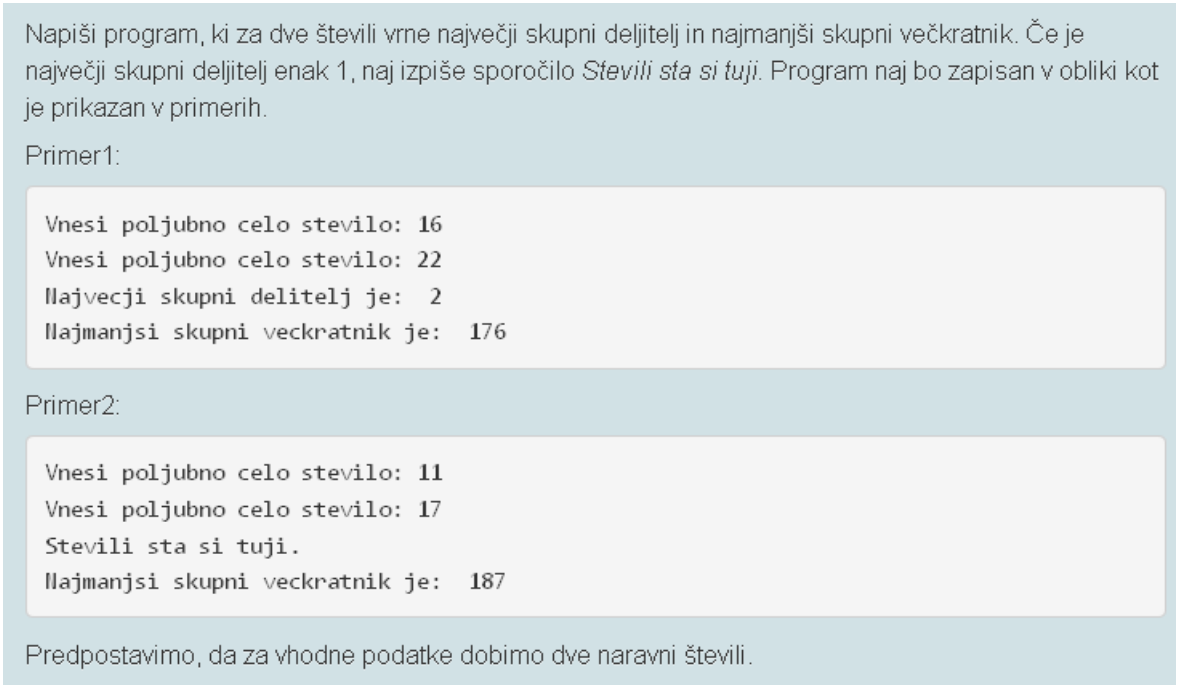


Slika 57: Shranimo sestavljeno vprašanje

5.2.2 Vprašanje tipa 'napiši program'

Naloga od študenta zahteva, da napiše program, ki za vhodni podatek dobi 2 naravni števili, vrne pa njun največji skupni delitelj in najmanjši skupni večkratnik. To nalogo bomo sestavili s pomočjo tipa vprašanja *python3_vnesi_podatke*. Pri sestavljanju vprašanja bo veliko nastavitev podobnih oz. enakih kot pri sestavljanju vprašanja v predhodnem poglavju, zato se bomo osredotočili le na nastavitve, ki se razlikujejo od nastavitev prejšnjega vprašanja.

Poglejmo si njegova natančna navodila (Slika 58).


Napiši program, ki za dve števili vrne največji skupni deljitelj in najmanjši skupni večkratnik. Če je največji skupni deljitelj enak 1, naj izpiše sporočilo *Stevili sta si tuji*. Program naj bo zapisan v obliki kot je prikazan v primerih.

Primer1:

```
Vnesi poljubno celo stevilo: 16
Vnesi poljubno celo stevilo: 22
Najvecji skupni delitelj je:  2
Najmanjsi skupni veckratnik je:  176
```

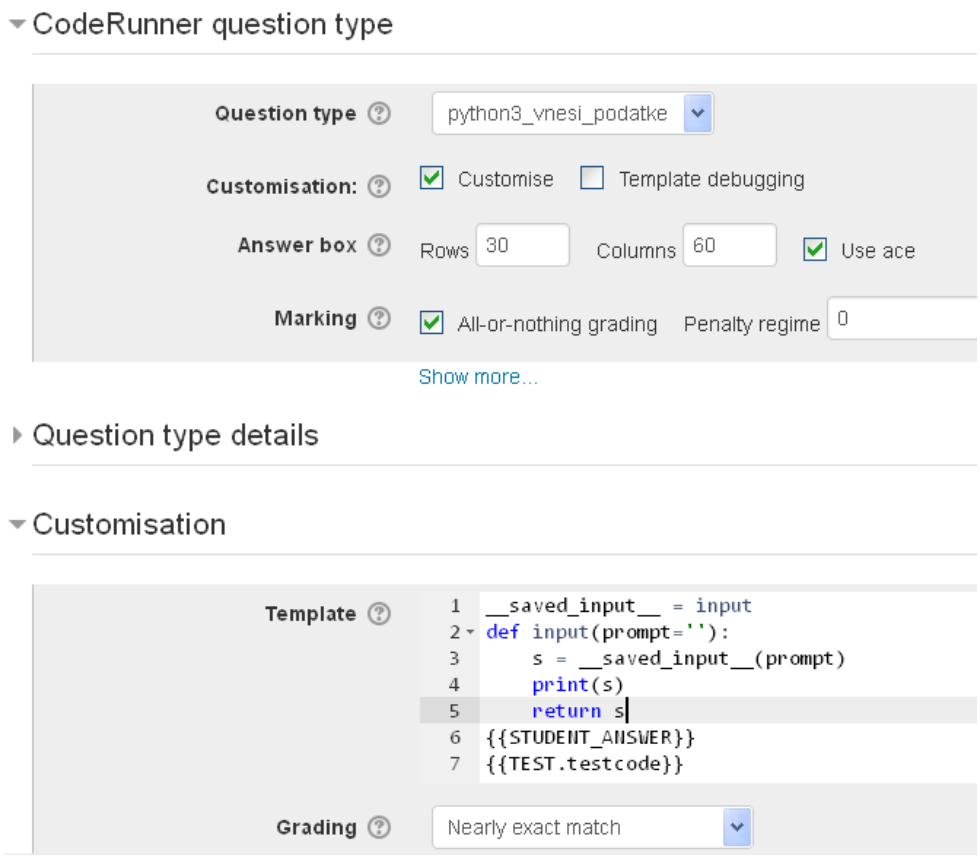
Primer2:

```
Vnesi poljubno celo stevilo: 11
Vnesi poljubno celo stevilo: 17
Stevili sta si tuji.
Najmanjsi skupni veckratnik je:  187
```

Predpostavimo, da za vhodne podatke dobimo dve naravni števili.

Slika 58: Navodila vprašanja

Ponovimo postopek za ustvarjanje novega vprašanja (glej poglavje 5.2.1). V meniju *Administration* kliknemo na razdelek *Question bank*. Ko se odpre zbirka vprašanj, kliknemo na *Create a new question*, za tip vprašanja pa izberemo *CodeRunner*. Nato se prikaže glavna stran za sestavljanje vprašanja (Slika 59).



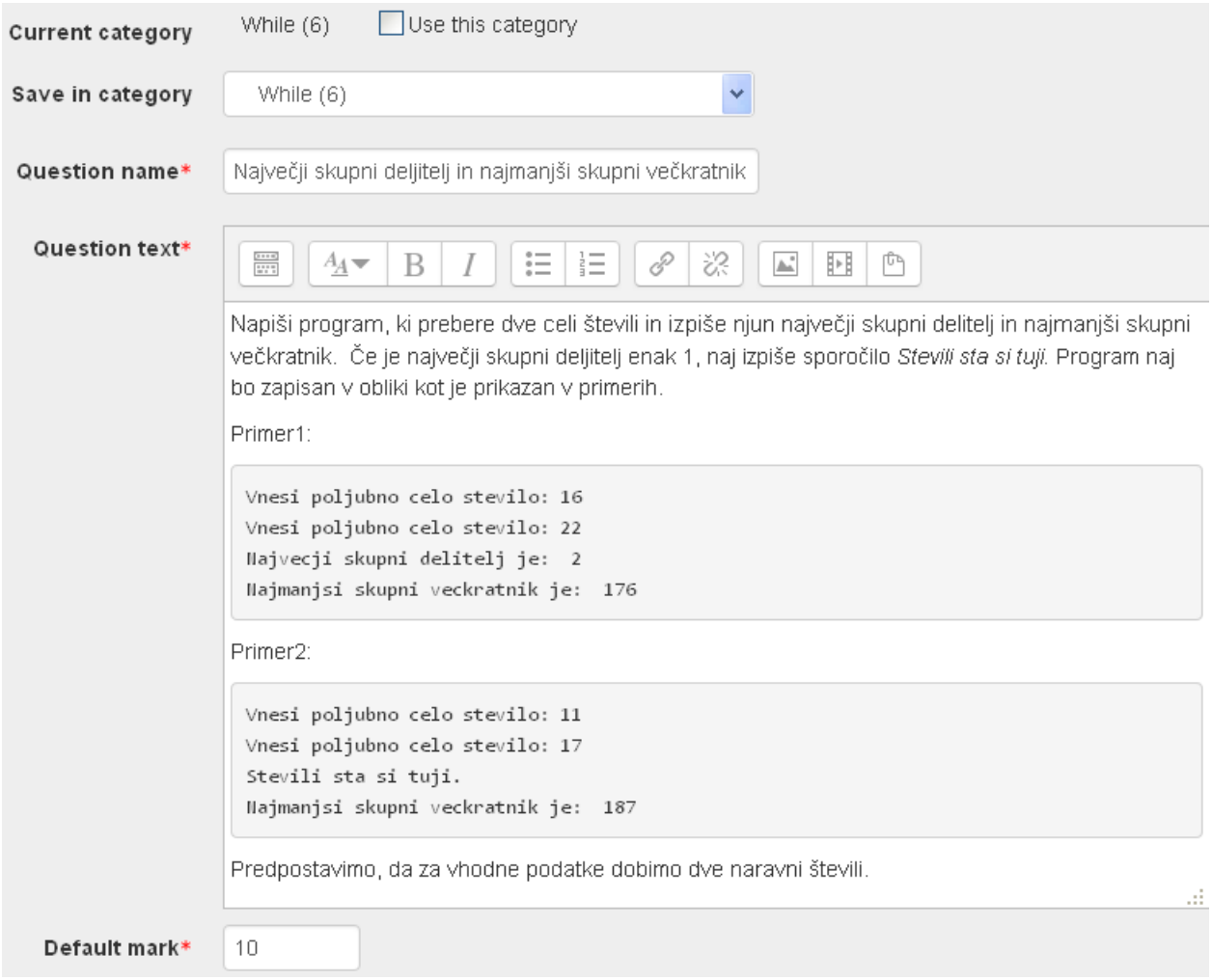
Slika 59: Sestavljanje vprašanja

V polju *Question type* si bomo tokrat izbrali novi tip vprašanja *python3_vnesi_podatke* s prilagojeno predlogo. Obkljukamo možnost *Customise,* da se prikaže razdelek *Customisation.* V polju *Answer box* nastavimo velikost okenca širine 30 (*rows*) in višine 60 (*columns*), ker pričakujemo za odgovor daljšo kodo. Možnosti *Use ace* in *All-or-nothing grading* pustimo obkljukani iz istih razlogov kot v prejšnjem primeru (glej poglavje 5.2.1). Študenta v primeru napačnega odgovora ne bomo kaznovali, zato v okence *Penalty regime* napišemo *0*.

V razdelku *Customisation* opazimo, da je koda predloge že prilagojena. Način ujemanja (možnost *Grading*) nastavimo na '*Nearly exact match*'. Dovoljene bodo manjše razlike v končni tabeli z rezultati (glej razlago v poglavju 5.1.2).

Sledi razdelek *General* (Slika 60). Nalogo uvrstimo v ustrezno kategorijo. Ker bomo nalogo rešili s pomočjo zanke *While*, bomo nalogo shranili v kategorijo *While*. Izberemo si ustrezno ime za nalogo in ga napišemo v okvirček *Question name*.

Potem v polje *Question text* napišemo natančna navodila za reševanje naloge. Podali smo tudi 2 primera, prvi primer s poljubnimi števili, ki si nista tuji, in drugi primer s tujimi si števili. S tem smo opozorili študenta na drugačen izpis sporočila. Opomnimo tudi, da za vhodni podatek dobimo dve naravni števili. V okvirčku *Default mark* ovrednotimo našo nalogo. Recimo, da bo vredna 10 točk.



Slika 60: Razdelek *General*

V razdelku *Sample answer* v polje *Answer* vpišemo možno rešitev naloge, ni pa nujno.

Sledi sestavljanje testnih primerov. Slika 61 prikazuje sestavljanje testnega primera 1. Opazimo, da smo v polje *Standard Input* vnesli vhodne podatke, torej dve naravni števili. Vsako število napišemo v svojo vrstico, saj program zahteva, da najprej vnesemo prvo naravno število in nato še drugo.

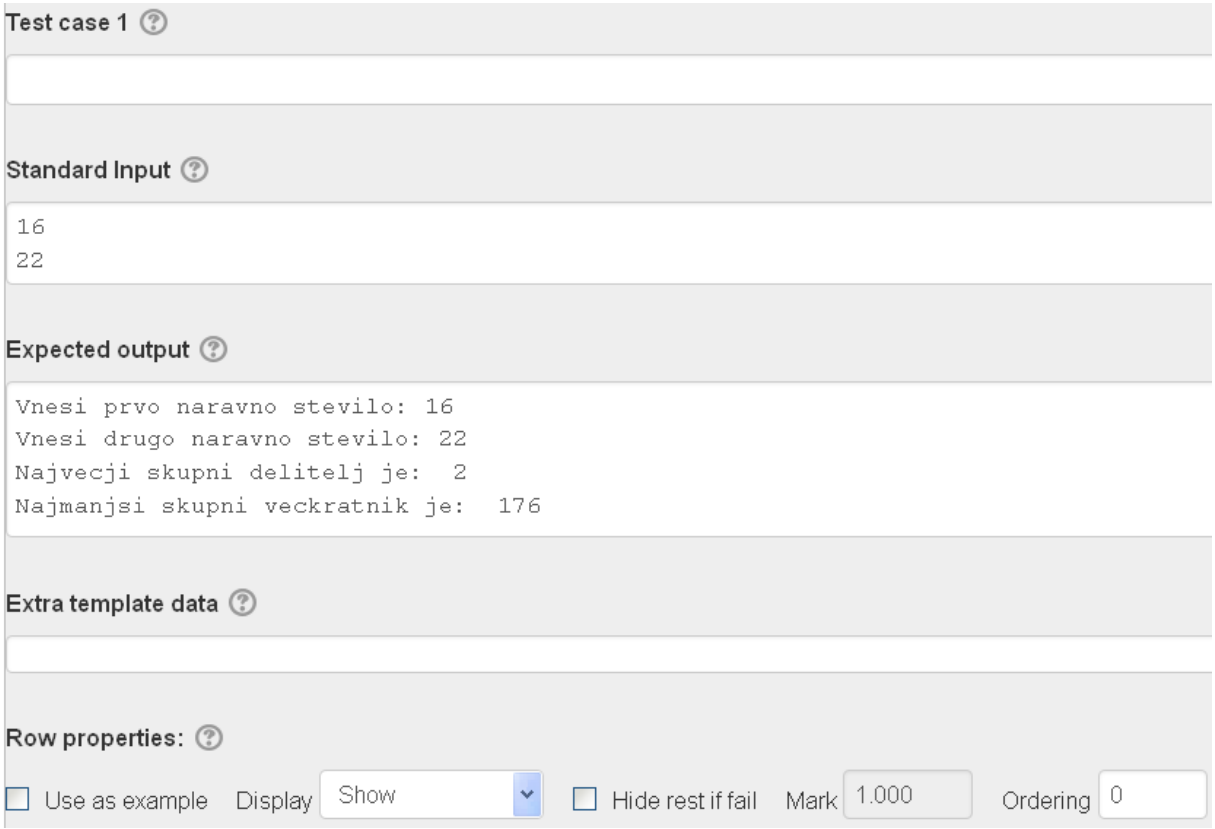
V polje *Expected output* pa tokrat napišemo celoten izpis programa. Napišemo torej besedilo skupaj z vhodnimi podatki in celotno besedilo, ki ga program izpiše po izvršitvi.

Ker sestavljamo vprašanje tipa 'napiši program', pustimo polje *Test case 1* prazno.Testiranje se bo izvršilo s pomočjo vhodnih podatkov, ki so napisani v polju *Standard Input*. Te podatke bo naša prilagojena predloga obdelala skupaj s študentovim odgovorom, za rezultat pa bomo dobili izpis celotnega programa.

Polje *Extra template data* bomo pustili prazno (glej razlago v poglavju 5.1).

V razdelku *Row properties* bomo pustili privzete nastavitve. To pomeni da:

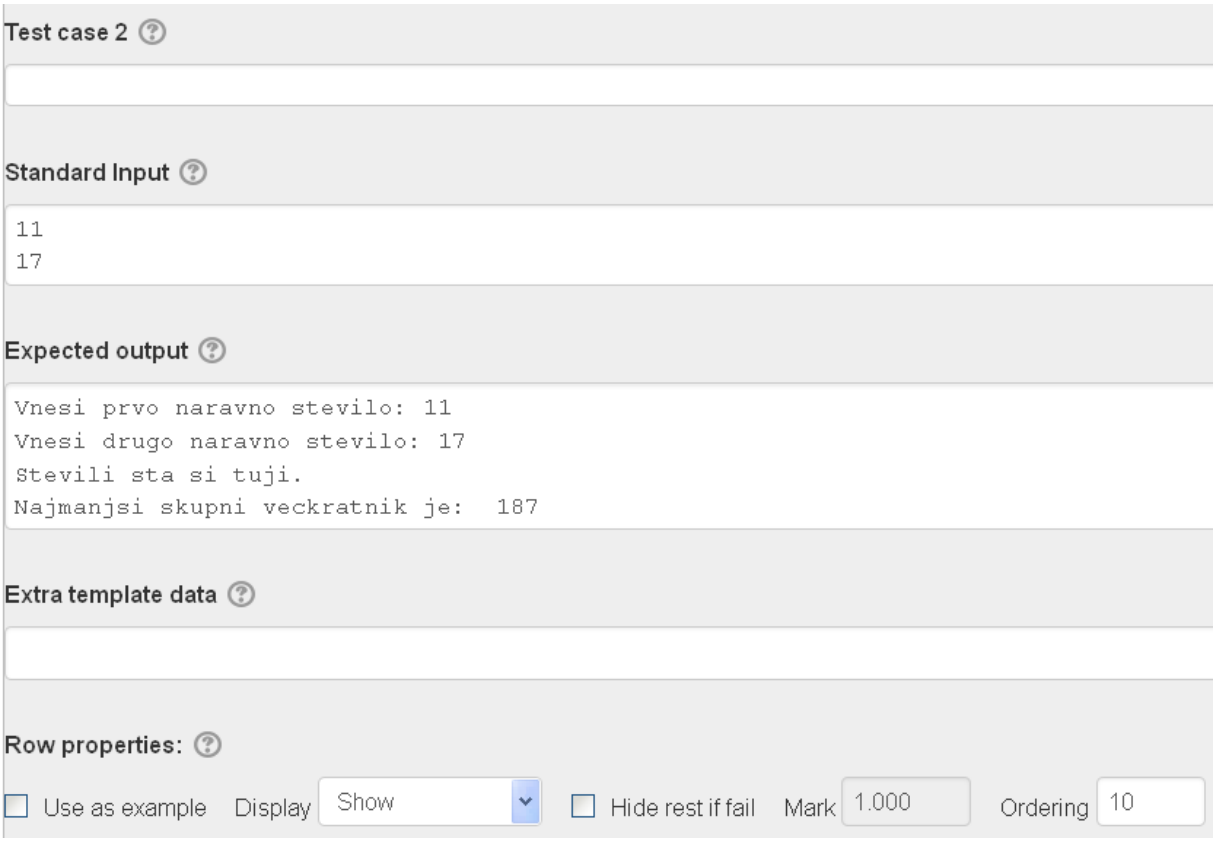
- testni primer ne bo prikazan kot zgled v navodilih (odkljukana možnost *Use as example*),
- testni primer bo prikazan v končni tabeli s testnimi primeri (Izbira *Show*v meniju *Display*)
- prikazani bodo vsi nadaljnji testni primeri, ne glede na uspešnost tega testnega primera (odkljukana možnost *Hide rest if fail*)
- v okvirčku *Ordering* ne spreminjamo vrednosti, torej ne bomo menjali vrstnega reda testnih primerov



Slika 61: Testni primer 1

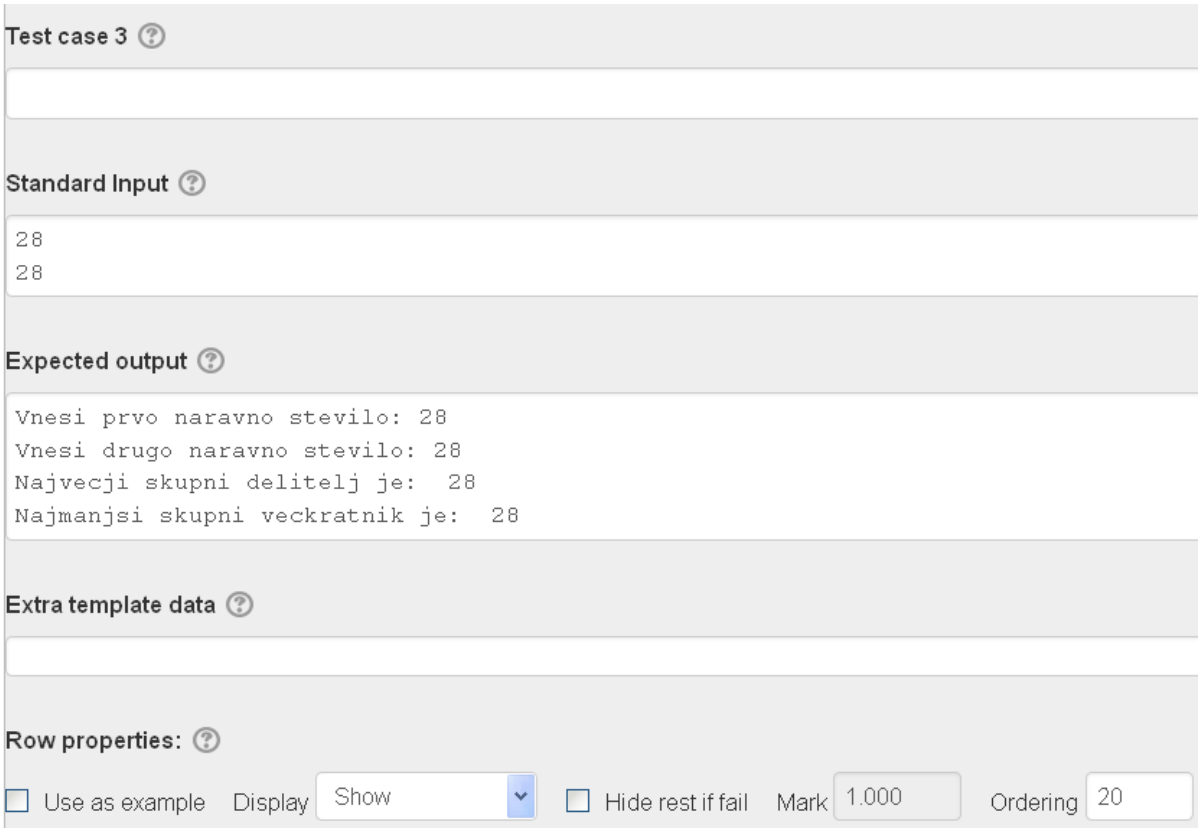
Podobno sestavimo še ostale testne primere.

V drugem testnem primeru bomo vzeli za vhodni podatek dve praštevili. Ker je njun največji skupni delitelj 1, pričakujemo v polju *Expected output* namesto sporočila '*Najvecji skupni delitelj je:'* sporočilo '*Stevili sta si tuji.'* Ostale nastavitve so enake kot v prvem testnem primeru (Slika 62).



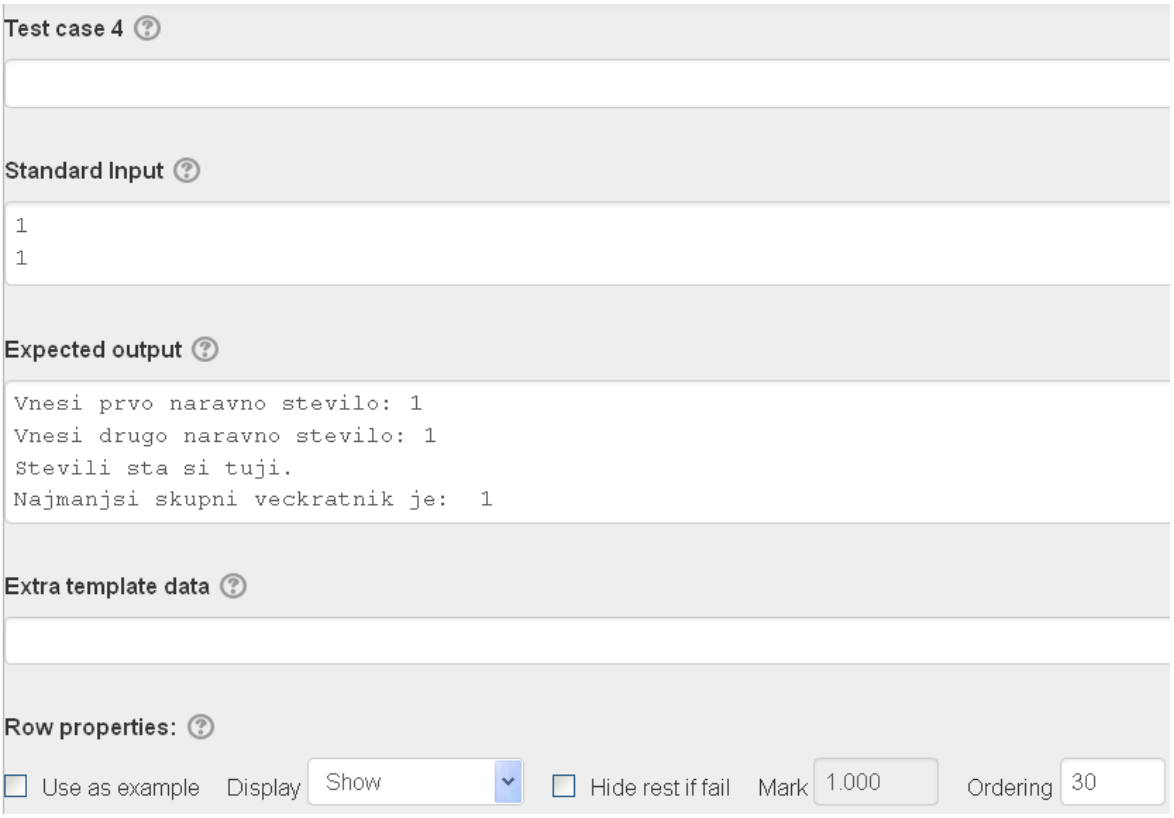
Slika 62: Testni primer 2

V testnem primeru 3 si za vhodni podatek izberemo dve enaki števili (Slika 63).



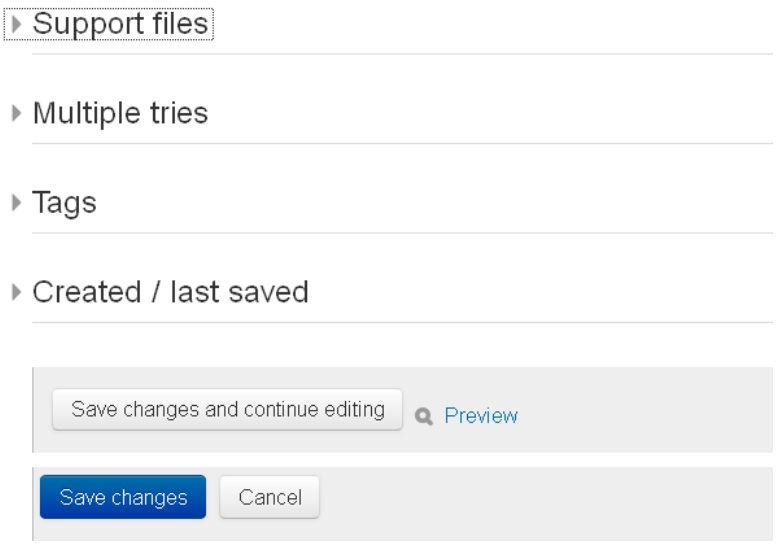
Slika 63: Testni primer 3

V testnem primeru 4 bomo iskali največji skupni delitelj in najmanjši skupni večkratnik števil 1 in 1. Namesto največjega skupnega delitelja mora program izpisati sporočilo *'Stevili sta si tuji.'* (Slika 64).



Slika 64: Testni primer 4

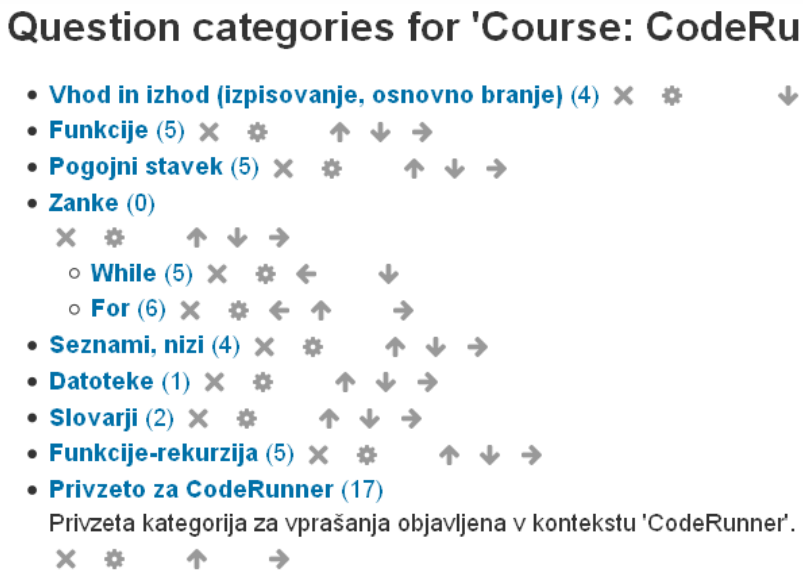
Ker tudi ta naloga za izvedbo ne potrebuje nobene dodatne datoteke, smo razdelek *Support files* spustili. Tudi v ostalih razdelkih (*Multiple tries, Tags, Created / last saved*) nam ni potrebno ničesar spreminjati. Sestavljeno vprašanje shranimo s klikom na gumb *Save changes* (Slika 65).



Slika 65: Shranimo vprašanje s klikom na *Save changes*

6. Naloge na praktičnih primerih

V sklopu diplomske naloge je bila pripravljena tudi osnovna zbirka vprašanj. Vprašanja smo razvrstili v kategorije, kot so prikazane na Slika 66: Prikaz kategorij v učilniciSlika 66.



Slika 66: Prikaz kategorij v učilnici

Vrstni red kategorij je pomemben, saj so razvrščene tako, da v prvi kategoriji pridobimo najosnovnejše znanje (branje in izpisovanje podatkov). Znanje iz predhodnih kategorij se nato navezuje na naslednjo kategorijo. Torej v kategoriji Slovar lahko predvidevamo, da že znamo uporabiti znanje iz predhodnih kategorij (Zanka While, Datoteke, Seznami...).

Opišimo na kratko razloge za tak vrstni red. Prvi prijem, ki ga moramo osvojiti, je, da znamo prebrati in izpisati podatke. Zato se kategorija *Vhod in izhod (izpisovanje, osnovno branje)* nahaja na prvem mestu. Druga stvar, ki je zelo pomembna, je, da znamo sami sestaviti funkcijo, ki jo lahko kasneje uporabimo. Zato bo naša druga kategorija *Funkcije*. Ko obvladamo stvari iz prvih dveh kategorij, lahko začnemo s pisanjem zahtevnejših programov. Takrat nam velikokrat pride prav uporaba pogojnih stavkov, ter zank *while* and *for*. Zato sta naslednji dve kategoriji *Pogojni stavek* in *Zanke*. V kategoriji *Zanke* imamo dve podrejeni kategoriji *While* in *For.* Svoje znanje programiranja potem nadgrajujemo z uporabo seznamov in nizov, branju in zapisovanju priloženih datotek, uporabo slovarjev, rekurzije... Temu primerno so razporejene še ostale kategorije, vidne na Slika 66.

Zdaj si bomo ogledali nekaj tipičnih vprašanj, sestavljenih v CodeRunnerju.

6.1 Delitelji števila

Na primeru te naloge bomo pokazali, kako sestavimo primer naloge, ki zahteva sestavo programa. Za preverjanje vprašanja tipa ‘napiši program’ bomo uporabili tip vprašanja *python3_vnesi_podatke*. Pokazali bomo tudi, kako ocenimo nalogo z načinom *All-or-nothing grading*, torej bomo zahtevali, da so vsi testni primeri rešeni pravilno. Napačnega odgovora ne bomo kaznovali z odbitki in bomo uporabili možnost '*Nearly exact match*'.

Ta metoda dopušča, da se študentov rezultat ne ujema popolnoma z rešitvijo. Dovoljeni so razni presledki in prazne vrstice v rešitvi, vse črke pa prebere kot male (glej poglavje 5.1.1). V primeru bomo obenem prikazali, kako se testni primer uporabi kot zgled v navodilih naloge, ki jih vidi študent.

Besedilo naloge: *Napiši program, ki izpiše vse pozitivne delitelje podanega celega števila. Program mora delovati tudi za negativna števila, za število 0 pa naj izpiše stavek 'Deliteljev je neskoncno'. Vnos podatkov naj bo v obliki:*

```
input('Vpisi stevilo: ')
```

Poglejmo, kako to vprašanje sestavimo v CodeRunnerju. V ta namen najprej premislimo, kakšen bo odgovor (pravilen program) in kakšni bi bili smiselni testni programi. V tem primeru bomo pripravili naslednje teste:

- **Testni primer 1** - Vhodni podatek je število z več delitelji. Zato bomo vzeli število 998, ki ima delitelje 1, 2, 499 in 998. S tem testnim primerom preverimo, če program pravilno deluje za tipični primer. Ta testni primer bomo uporabili tudi kot zgled v navodilih naloge.
- **Testni primer 2** -Vhodni podatek je negativno število. Če je podatek negativno število (uporabili bomo -10),mora program izpisati delitelje 1, 2, 5 in 10.
- **Testni primer 3** - Vhodni podatek je praštevilo. Za število 13 mora program izpisati 1 in 13.
- **Testni primer 4** - Vhodni podatek je število 0. Program mora izpisati sporočilo 'Deliteljev je neskoncno', saj vemo, da lahko število 0 delimo z vsemi števili.
- **Testni primer 5** - Vhodni podatek je edino pozitivno celo število, ki ima le en delitelj. To je 1.
- **Testni primer 6** – Tudi pri -1 moramo kot odgovor dobiti le 1.

Pravilen program za to nalogo je na primer tak, kot ga prikazuje Slika 67.

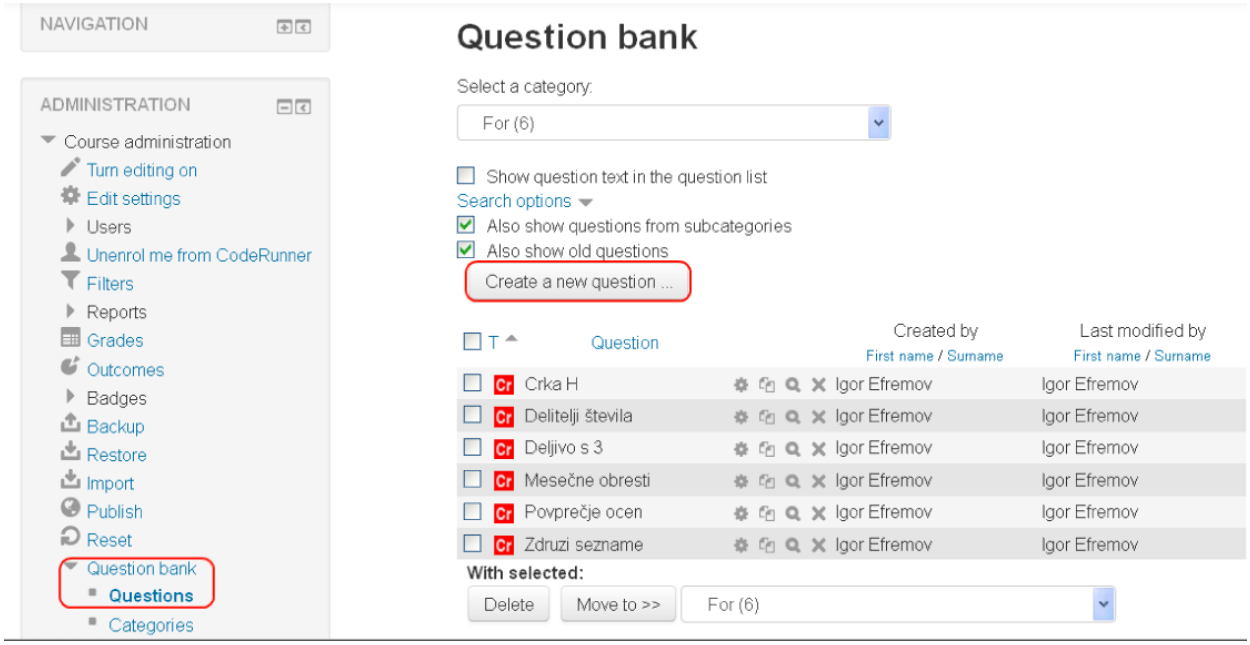
Answer:

```
1  stevilo = int(input('Vpisi stevilo: '))
2  if stevilo == 0:                    #za število 0 izpišemo sporočilo
3      print('Deliteljev je neskoncno')
4  for i in range(1, abs(stevilo) + 1):
5      if stevilo % i == 0:    #če je ostanek pri deljenju števila enak 0,
6          print(i)            #je i delitelj števila
```

Slika 67: Koda, ki reši primer

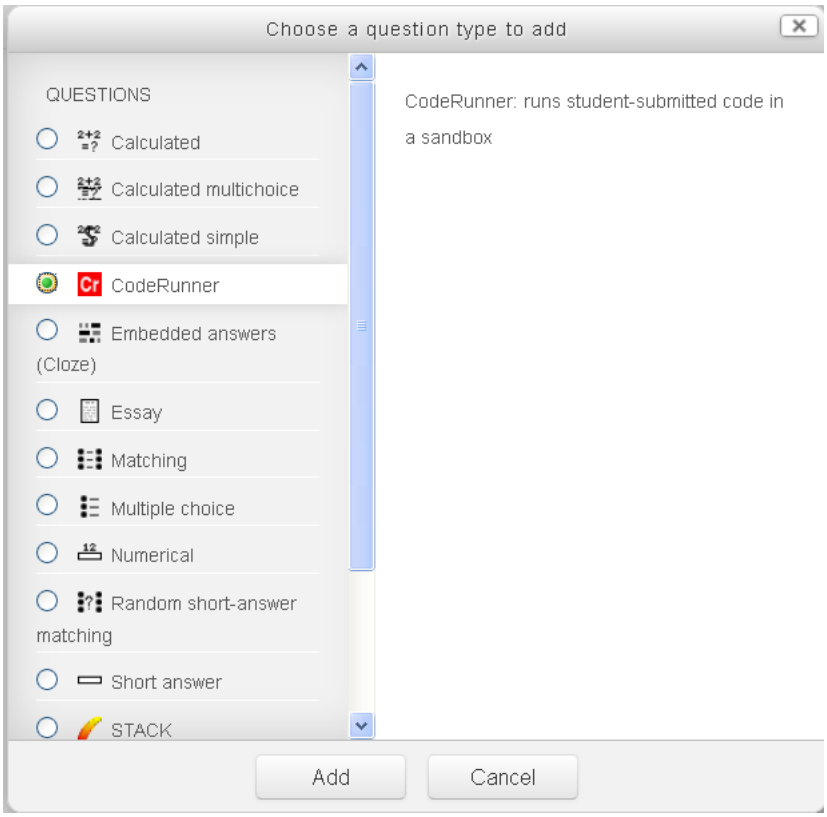
Najprej poskrbimo za ustrezno obravnavo števila 0. Nato v zanki preverimo vse potencialne delitelje in jih izpišemo. Za negativna števila poskrbimo z ukazom *abs().*

Obnovimo začetne korake pri sestavljanju vprašanj. V razdelku *Administration* kliknemo najprej na *Question bank– Questions* in nato na *Create a new question* (Slika 68).



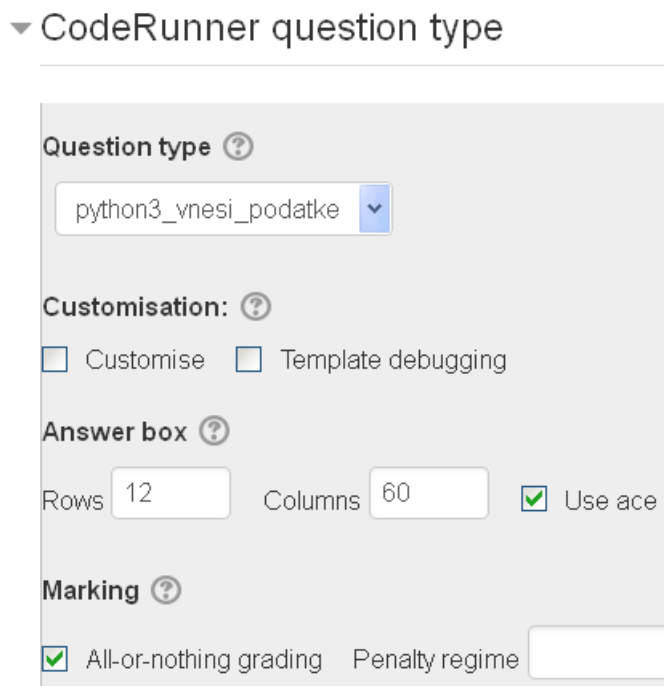
Slika 68: Ustvarjanje novega vprašanja

Ker naloga zahteva, da napišemo programsko kodo, izberemo tip vprašanja CodeRunner (Slika 69).

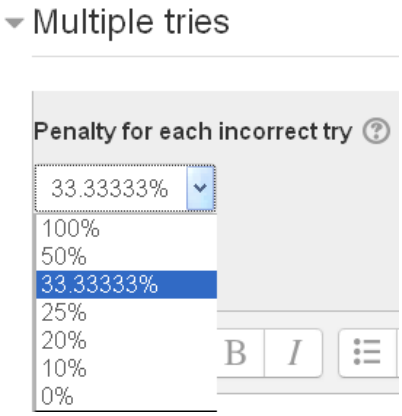


Slika 69: Izberemo tip vprašanja CodeRunner

Kot smo povedali v poglavju 5.1, dobimo obrazec, razdeljen v več delov. V *CodeRunner question type* si za tip vprašanja izberemo *python3_vnesi_podatke* (Slika 70). V polju *Marking* pustimo privzeto nastavitev *All-or-nothing grading* obkljukano, ker želimo, da so vsi testni primeri rešeni pravilno (glej poglavje 5.1). V *Penalty regime* vnesemo število 0, ker ne želimo kaznovati napačnega odgovora. To lahko storimo tudi tako, da okence *Penalty regime* pustimo prazno, vendar moramo potem spremeniti nastavitev v razdelku *Multiple tries*. Kazen iz privzetih *33,33333%* spremenimo na *0%* (Slika 71).

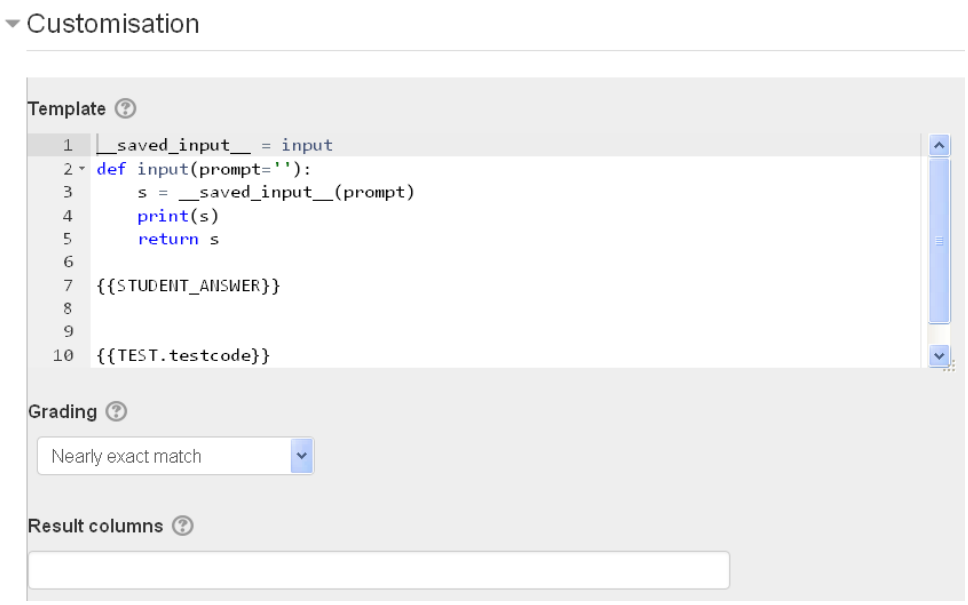


Slika 70: Razdelek *CodeRunner question type*



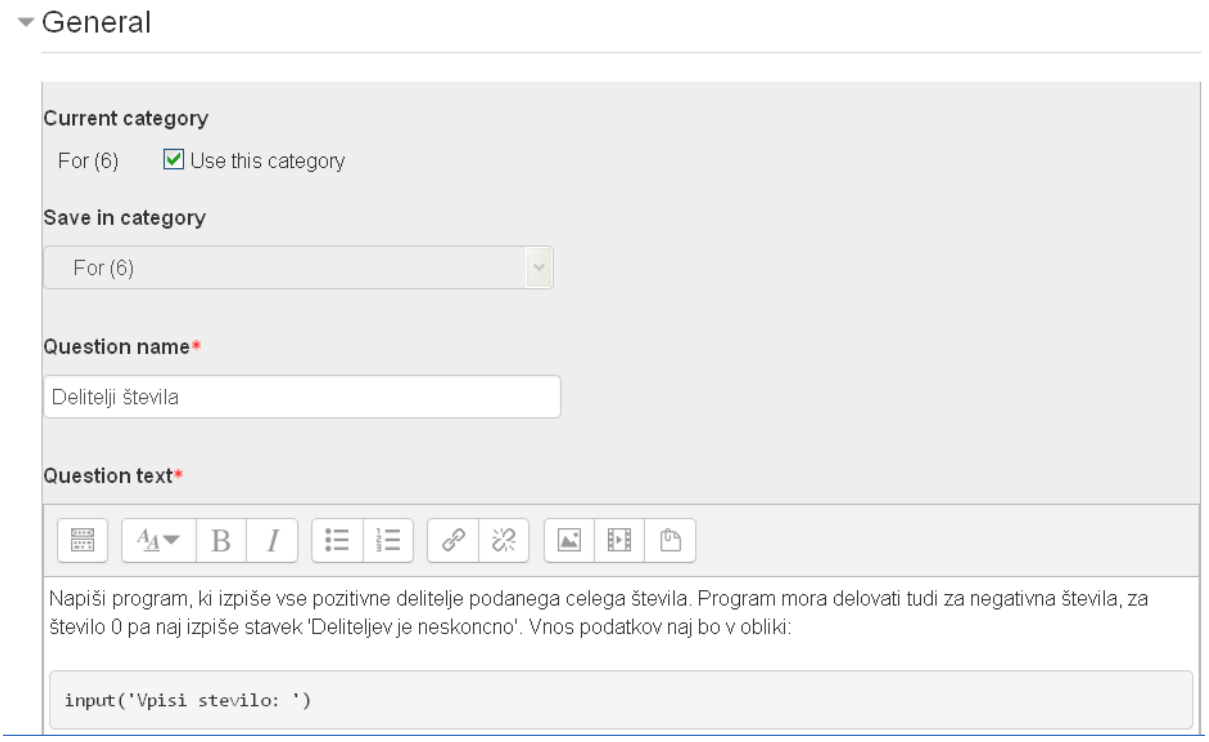
Slika 71: Kazen nastavimo na 0%

Ko obkljukamo okence *Customize*, se pojavi dodatni razdelek *Customisation* (Slika 72). Možnost *Grading* nastavimo na '*Nearly exact match'.* S tem dopustimo minimalne razlike v tabeli pri rešitvi naloge (glej poglavje 5.1.1).



Slika 72: Razdelek *Customisation*

V razdelku *General* izberemo kategorijo, v katero bomo shranili vprašanje. Izberemo si kategorijo *For*, ker bomo v nalogi uporabili omenjeni konstruktor. V okvirček *Question name* napišemo smiselno ime vprašanja, na primer *Delitelji števila*, v okvirček *Question text pa* napišemo jasna navodila za reševanje naloge (Slika 73).



Slika 73:Razdelek *General*

V *Sample Answer* vpišemo rešitev (Slika 74).

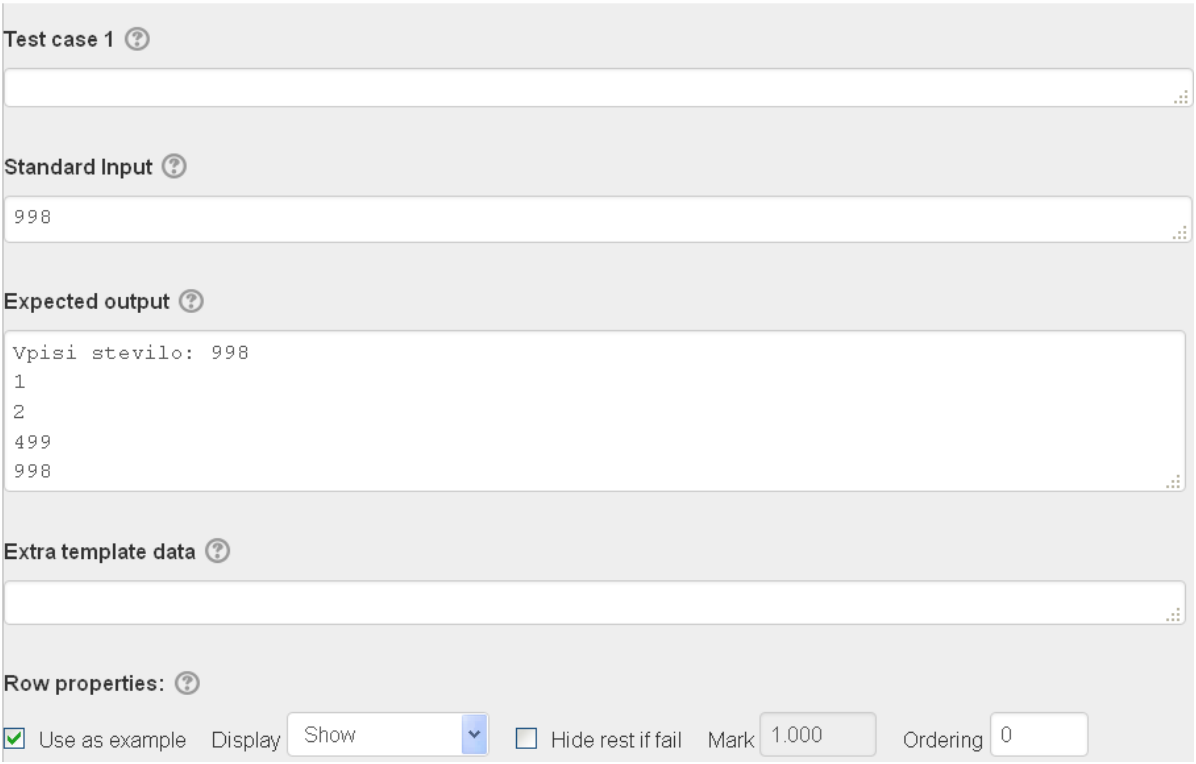
Sample answer

Answer

```
stevilo = int(input('Vpisi stevilo: '))
if stevilo == 0:                        #za število 0 izpišemo sporočilo
    print('Deliteljev je neskoncno')
for i in range(1, abs(stevilo) + 1):
    if stevilo % i == 0:     #če je ostanek pri deljenju števila enak 0,
        print(i)             #je i delitelj števila
```

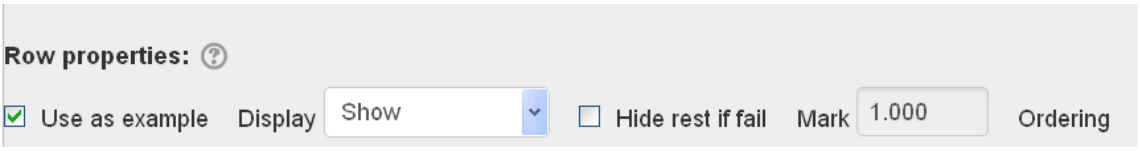
Slika 74: Rešitev naloge

V razdelku *Test cases* sestavimo testne primere. Ker je naloga tipa ‘napiši program’, pustimo polje *Test case 1* prazno (glej poglavje 5.2.2). V polje *Input* vpišemo število, ki ga bomo testirali, v *Expected output* pa pričakovani rezultat. Predloga ne zahteva nobenih posebnih podatkov, zato polje *Extra template data* ostane prazno. V polja bomo torej vpisali to, kar prikazuje Slika 75.



Slika 75: Sestavljanje testnega primera 1

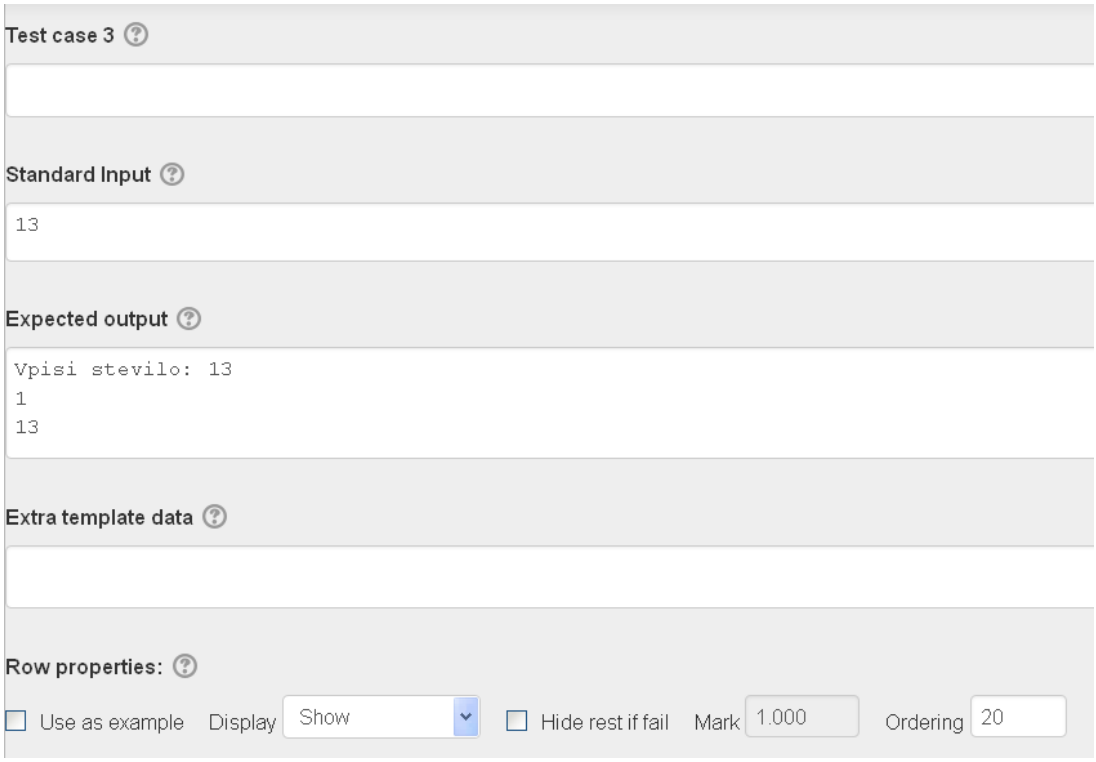
Pogosto je koristno, da kot del naloge prikažemo tudi en ali več tipičnih primerov izvajanja. Zato bomo v razdelku *Test cases – Row properties*, kjer sestavljamo testne primere, pri določenih testih obkljukali možnost *Use as example* (Slika 76).



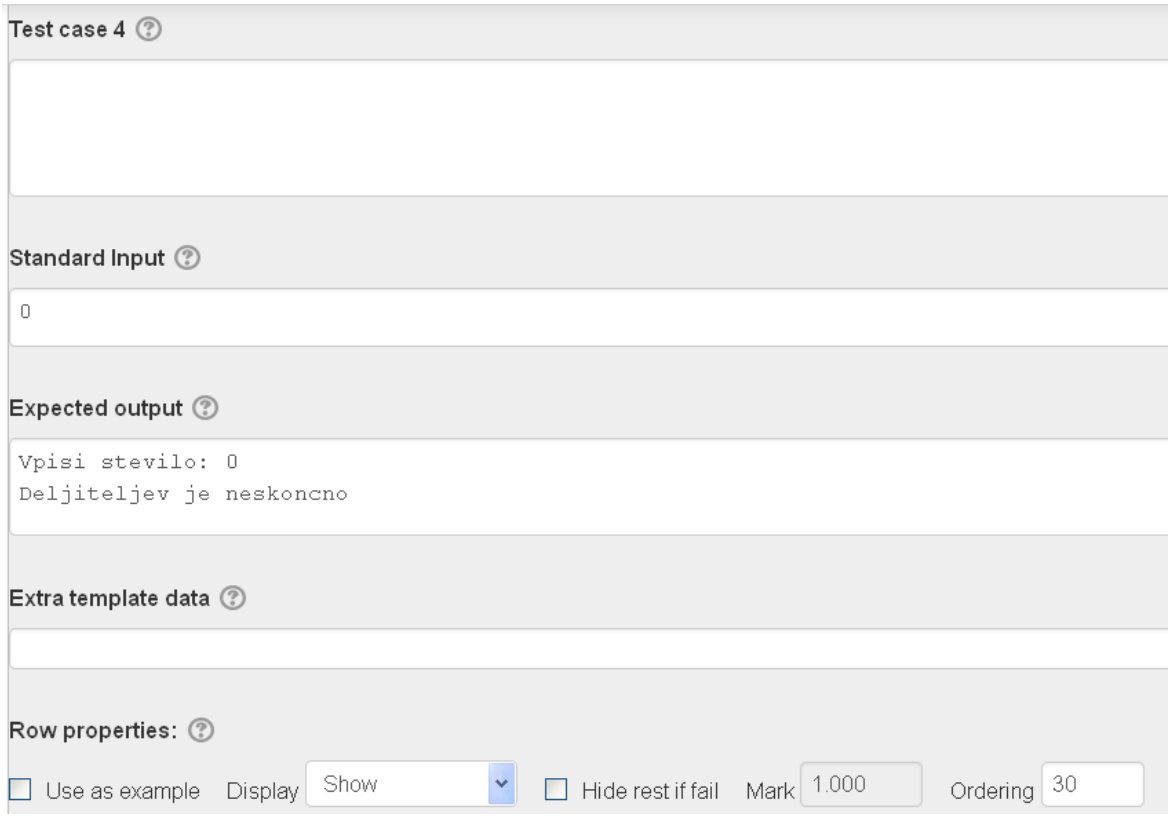
Slika 76:Testni primer bo prikazan v navodilih

V razdelku *Test cases* moramo sestaviti še ostale testne primere, opisane na začetku poglavja. Sestavimo jih na podoben način, kot smo sestavili Testni primer 1 (Slika 77, Slika 78, Slika 79, Slika 80 in Slika 81).

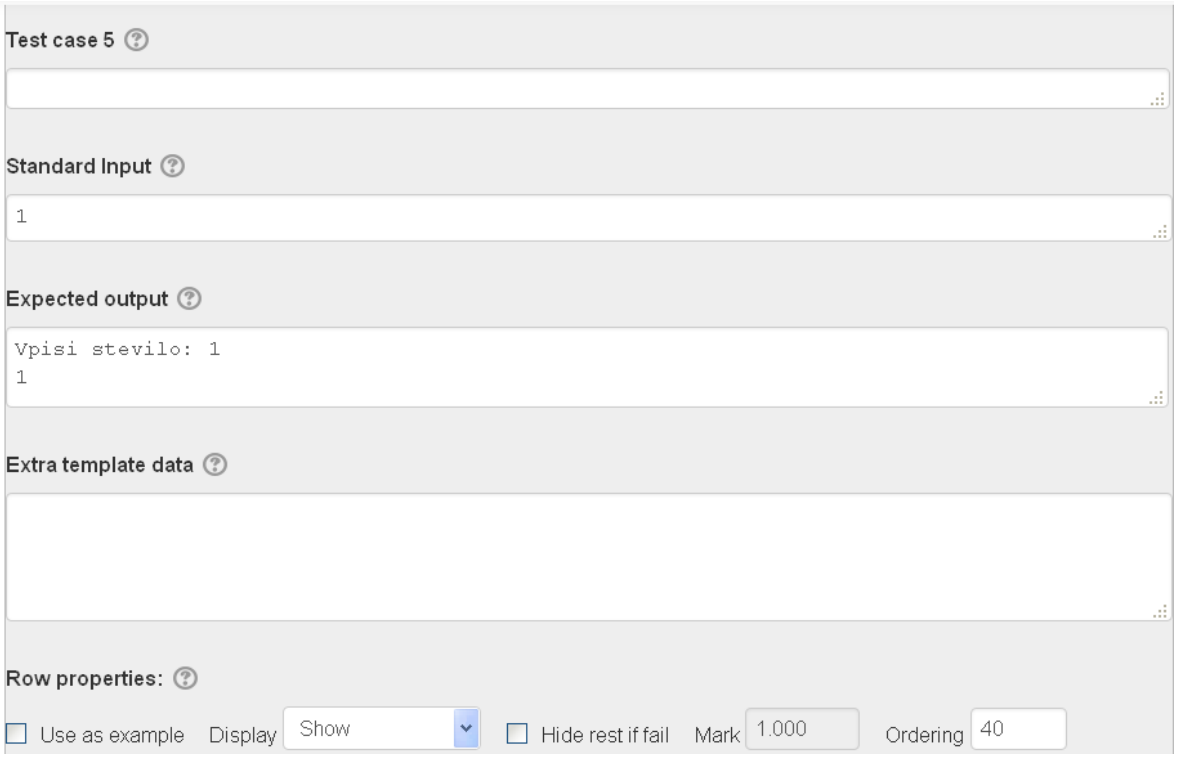
Slika 77: Testni primer z negativnim številom



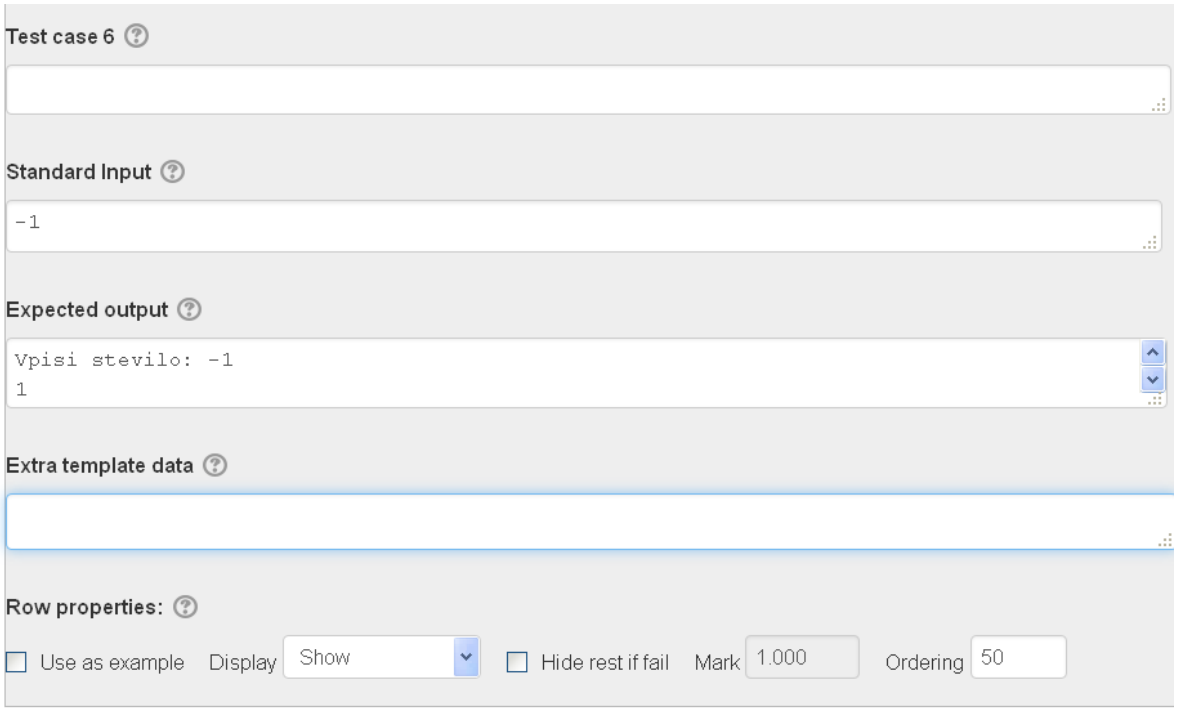
Slika 78: Testni primer za praštevilo



Slika 79: Testni primer za število 0

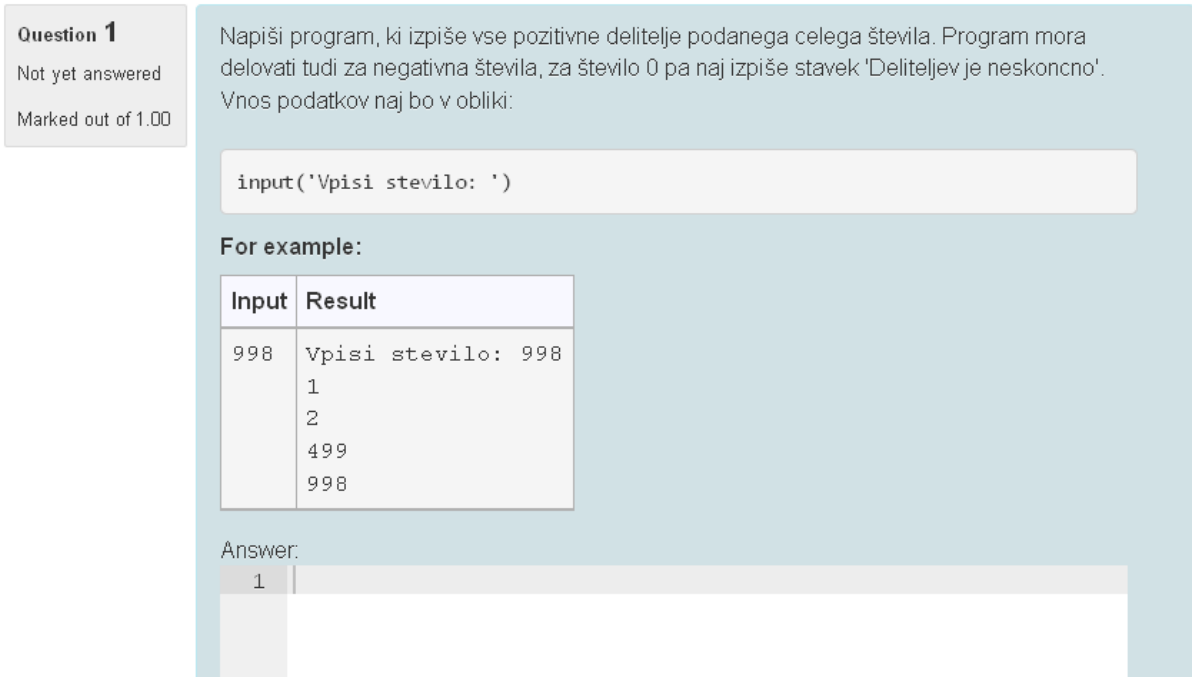


Slika 80: Testni primer za število 1



Slika 81: Testni primer za število -1

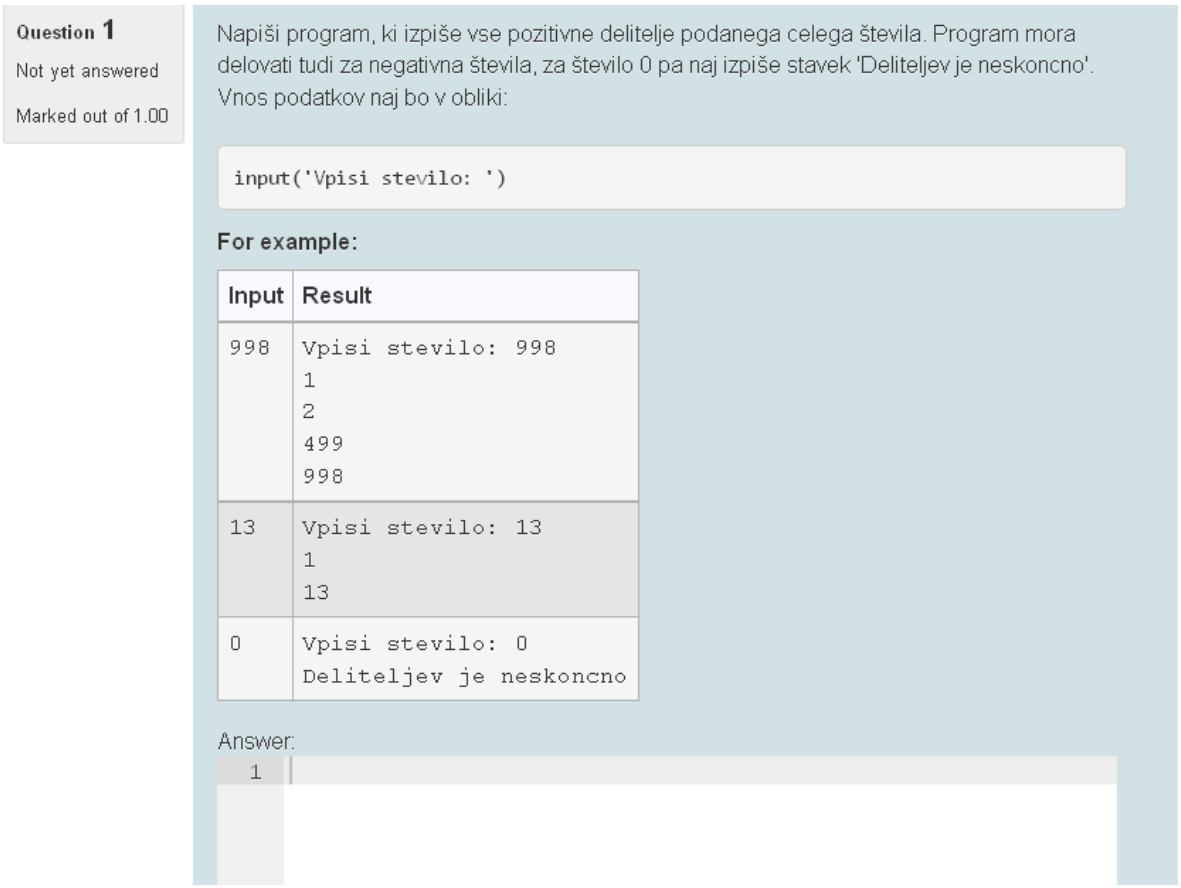
Ko je vprašanje sestavljeno, vidi študent navodila takole (Slika 82).



Slika 82: Sestavljeno vprašanje z zgledom

V navodilih se prikaže tabela s tistimi testnimi primeri, ki smo jih označili za prikaz. V našem primeru je to le prvi testni primer. Tabela ima dva stolpca, *Input,* kjer je zapisan vhodni podatek, in *Result*, v katerem je zapisan pričakovan odgovor. Torej za število 998 nam mora program izpisati delitelje števila 998 v obliki, kot je prikazano v tabeli v tej celici.

Oglejmo si še primer, ko dodamo več testnih primerov. Obkljukajmo sedaj še *Use as example* pri 3. in 4. testnem primeru. Takrat bo vprašanje prikazano kot na Slika 83.



Question 1

Not yet answered

Marked out of 1.00

Napiši program, ki izpiše vse pozitivne delitelje podanega celega števila. Program mora delovati tudi za negativna števila, za število 0 pa naj izpiše stavek 'Deliteljev je neskoncno'. Vnos podatkov naj bo v obliki:

```
input('Vpisi stevilo: ')
```

For example:

Input	Result
998	Vpisi stevilo: 998 1 2 499 998
13	Vpisi stevilo: 13 1 13
0	Vpisi stevilo: 0 Deliteljev je neskoncno

Answer:

```
1
```

Slika 83: Navodilo naloge z dodatnimi zgledi v tabeli

Opazimo, da je dobila tabela s testnimi primeri dve dodatni vrstici. V vsaki vrstici je napisan posamezni testni primer, ki smo ga predhodno obkljukali v *Test cases* za zgled.

Poglejmo primer, ko je študent to nalogo rešil pravilno. Na Slika 84 opazimo, da je odgovor drugačen od naše uradne rešitve. Študent je uporabil drugačen pristop (zanko *while*), kljub temu pa je odgovor pravilen.

Question 2

Correct

Mark 1.00 out of 1.00

Flag question

Edit question

Napiši program, ki izpiše vse pozitivne delitelje podanega celega števila. Program mora delovati tudi za negativna števila, za število 0 pa naj izpiše stavek 'Deliteljev je neskoncno'. Vnos podatkov naj bo v obliki:

```
input('Vpisi stevilo: ')
```

For example:

Input	Result
998	Vpisi stevilo: 998 1 2 499 998

Answer:

```
1  stevilo = int(input('Vpisi stevilo: '))
2  if stevilo == 0:                    #za število 0 izpišemo sporočilo
3      print('Deliteljev je neskoncno')
4  i=1
5  while i < abs(stevilo)+1:
6      if stevilo % i == 0:    #če je ostanek pri deljenju števila enak 0,
7          print(i)            #je i delitelj števila
8      i=i+1
```

Check

	Input	Expected	Got	
✓	998	Vpisi stevilo: 998 1 2 499 998	Vpisi stevilo: 998 1 2 499 998	✓
✓	-10	Vpisi stevilo: -10 1 2 5 10	Vpisi stevilo: -10 1 2 5 10	✓
✓	13	Vpisi stevilo: 13 1 13	Vpisi stevilo: 13 1 13	✓
✓	0	Vpisi stevilo: 0 Deliteljev je neskoncno	Vpisi stevilo: 0 Deliteljev je neskoncno	✓
✓	1	Vpisi stevilo: 1 1	Vpisi stevilo: 1 1	✓
✓	-1	Vpisi stevilo: -1 1	Vpisi stevilo: -1 1	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Slika 84: Primer pravilnega odgovora

Slika 85 pa prikazuje primer napačnega odgovora.

Question 2

Incorrect

Mark 1.00 out of 1.00

Flag question

Edit question

Napiši program, ki izpiše vse pozitivne delitelje podanega celega števila. Program mora delovati tudi za negativna števila, za število 0 pa naj izpiše stavek 'Deliteljev je neskoncno'. Vnos podatkov naj bo v obliki:

```
input('Vpisi stevilo: ')
```

For example:

Input	Result
998	Vpisi stevilo: 998 1 2 499 998

Answer:

```
1 stevilo = int(input('Vpisi stevilo: '))
2 i=1
3 while i < stevilo+1:
4     if stevilo % i == 0:    #če je ostanek pri deljenju števila enak 0,
5         print(i)            #je i delitelj števila
6     i=i+1
```

Check

	Input	Expected	Got	
✓	998	Vpisi stevilo: 998 1 2 499 998	Vpisi stevilo: 998 1 2 499 998	✓
✗	-10	Vpisi stevilo: -10 1 2 5 10	Vpisi stevilo: -10	✗
✓	13	Vpisi stevilo: 13 1 13	Vpisi stevilo: 13 1 13	✓
✗	0	Vpisi stevilo: 0 Deliteljev je neskoncno	Vpisi stevilo: 0	✗
✓	1	Vpisi stevilo: 1 1	Vpisi stevilo: 1 1	✓
✗	-1	Vpisi stevilo: -1 1	Vpisi stevilo: -1	✗

Your code must pass all tests to earn any marks. Try again.

Incorrect

Marks for this submission: 0.00/1.00.

Slika 85: Primer napačnega odgovora

Rešitev študenta, prikazana na Slika 85, je delno prestala testne primere. Pravilno deluje na 1., 3. in 5. testnem primeru, napačno pa za negativna števila in 0. Študent je očitno pozabil, da so podatki lahko tudi negativni ali pa 0.

6.2 Produkt števil v seznamu

Sestavili bomo klasično nalogo tipa 'napiši funkcijo' z uporabo prototipa *python3_cosc121*. Od študenta bomo zahtevali, da kodo napiše v skladu s sistemom za preverjanje kakovosti kode *Pylint*. Pokazali bomo tudi, kako se naloga delno oceni, torej bo študent nagrajen za vsak pravilno rešen testni primer.

Preden se lotimo sestave vprašanja, moramo biti seznanjeni z načinom preverjanja sloga kodiranja. *Pylint* za preverjanje sintakse uporablja slogovni priročnik kodiranja *PEP 8,* ki obsega bogato knjižnico z navodili in s priporočili sloga kodiranja. Pri pisanju naših kod so potrebna naslednja:

- Zamik se določi s 4 presledki
- Število znakov v vrstici je omejeno na 100. S tem izboljšamo preglednost kode
- Ko definiramo funkcijo, je obvezen dokumentacijski niz (*docstring*), v katerem opišemo, kaj počne naša funkcija. Primer:

```
def funkcija(spremenljivka):
    ''' Funkcija naredi nekaj s spremenljivko'''
```

- Imena funkcij, podatkovnih struktur, nizov. itd. morajo biti sestavljena iz najmanj 3 znakov. V imenu je prepovedana uporaba velikih črk. Besede najlepše ločimo s podčrtajem. Primer:

```
def kvadriraj(n):
```
prekratko ime spremenljivke

```
mojSeznam = []
```
ne smemo uporabljati velikih črk

```
def vsota(seznam_stevil):
vsota = 0
```
paziti moramo tudi, da ne poimenujemo glavnih in lokalnih spremenljivk enako

```
def kvadriraj(stevilo):
```
primer ustrezno izbranega imena spremenljivke

```
moj_seznam = []
```
ustrezno ime seznama

- Opozori na nekoristne ukaze. Primer je definirana spremenljivka oziroma konstanta, ki se ne uporabi v nadaljevanju kode.
- Potreben je natanko en presledek okoli znakov: `==` , `<`, `>`, `!=`,`<>`, `<=` , `>=` , `in` , `not in` , `is` , `is not`, `and` , `or` , `not` ...

Sistem *Pylint* pri preverjanju sloga kodiranja upošteva vsa zgoraj našteta pravila (in še ostala), zato moramo biti pri pisanju kode dosledni.

Besedilo naloge: Sestavi funkcijo *produkt(seznam)*, ki izračuna produkt vseh števil v seznamu, zaokrožen na celo število. Koda bo najprej preverjena s sistemom za preverjanje kakovosti kode *Pylint*, zato pazi na slog kodiranja. Funkcija mora delovati tudi, če je podatek(število) v seznamu zapisan v obliki niza. Če je seznam prazen, naj vrne 0.

Za seznam `[1, 2 ,3.1 ,'-4']` nam funkcija vrne rezultat `-25`, za seznam `[]` pa 0.

Kot pri prejšnji nalogi najprej premislimo, kakšen bo odgovor (pravilen program) in kakšne testne primere bi bilo pametno sestaviti. Pripravili bomo naslednje teste:

- **Testni primer 1** –podali bomo seznam z naravnimi števili, da preverimo delovanje funkcije za tipični primer.
- **Testni primer 2** – podali bomo prazen seznam. Funkcija mora vrniti število 0.
- **Testni primer 3** – podali bomo seznam, ki bo vseboval negativno število in decimalno število. S tem testnim primerom bomo preverili, če funkcija rezultat zaokroži na celo število. Ta primer bomo podali tudi za zgled v navodilih naloge.
- **Testni primer 4** – podali bomo seznam, ki vsebuje nize in števila. Preverili bomo, če funkcija pravilno razbere podatek (število) iz niza.
- **Testni primer 5**– V seznamu bo med drugimi tudi število 0. Funkcija mora vrniti rezultat 0.

Pravilen odgovor, ki ustreza zahtevam sistema *Pylint,* je na primer sledeč (Slika 86):

```
1  def produkt(seznam):
2      '''Funkcija zmnozi vsa stevila v
3      danem seznamu in vrne zmnozek
4      zaokrozen na celo stevilo'''
5      if seznam == []:
6          return 0
7      rezultat = 1                        #na zacetku postavimo rezultat na 1
8      for i in seznam:                    #rezultat pomnozimo z i
9          rezultat = rezultat * float(i)  #niz spremenimo v stevilo
10     return round(rezultat)              #zaokrozimo na celo stevilo
```

Slika 86: Odgovor, ki ustreza sistemu *Pylint*

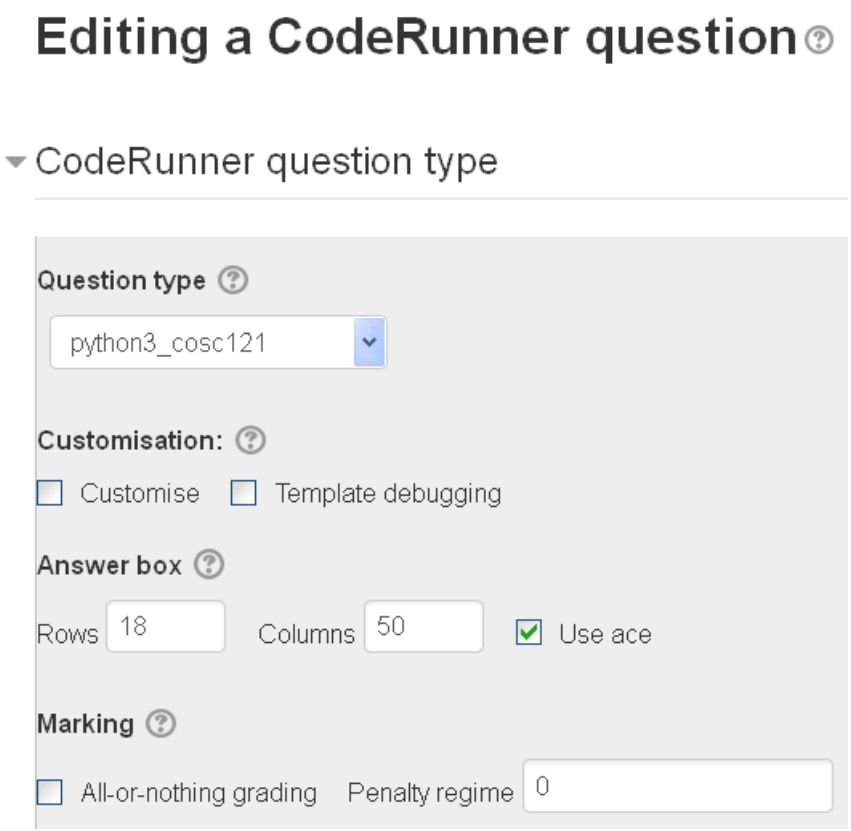
Najprej poskrbimo za ustrezno obravnavo praznega seznama. Funkcija v tem primeru vrne rezultat 0. Nato z zanko poskrbimo, da zmnožimo vsak element seznama s prejšnjim. Z ukazom *round()* zaokrožimo rezultat na celo število.

Poglejmo si postopek sestavljanja vprašanja. Polja, ki so nam znana iz poglavja 5.2, ne bomo podrobneje opisovali, omenili bomo samo, kaj smo si izbrali. Podrobneje si bomo ogledali sestavljanje testnih primerov in kasnejše preverjanje študentovega odgovora.

Obnovimo začetne korake pri sestavljanju vprašanj. V razdelku *Administration* kliknemo najprej na *Question bank* – *Questions* in nato na *Create a new question*. Izberemo tip vprašanja *CodeRunner.*

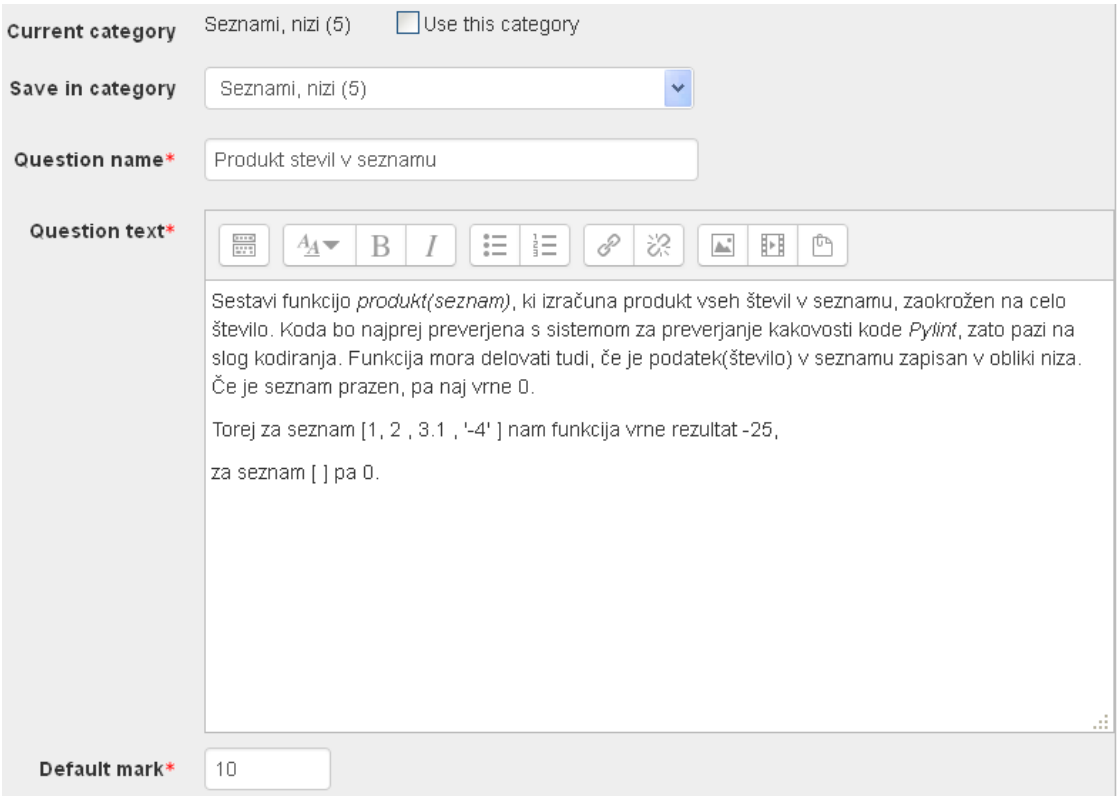
Odpre se nam znana stran za sestavljanje vprašanja (Slika 87). V razdelku *Question type* si tokrat izberemo tip vprašanja *python3_cosc121.* Velikost okenca za odgovor (*Answer box*) bomo nastavili na 18 vrstic (*rows*) in 50 stolpcev (*columns*). Možnost *Use ace* pustimo obkljukano.

Tokrat bomo v polju *Marking* odkljukali možnost *All-or-nothing grading*. To pomeni, da bomo vsak testni primer točkovali posebej. Študent bo ocenjen s toliko točkami, kolikor pravilnih testnih primerov bo imel. V okence *Penalty regime* bomo vnesli vrednost *0*, ker ne želimo odbitka pri napačnem odgovoru.



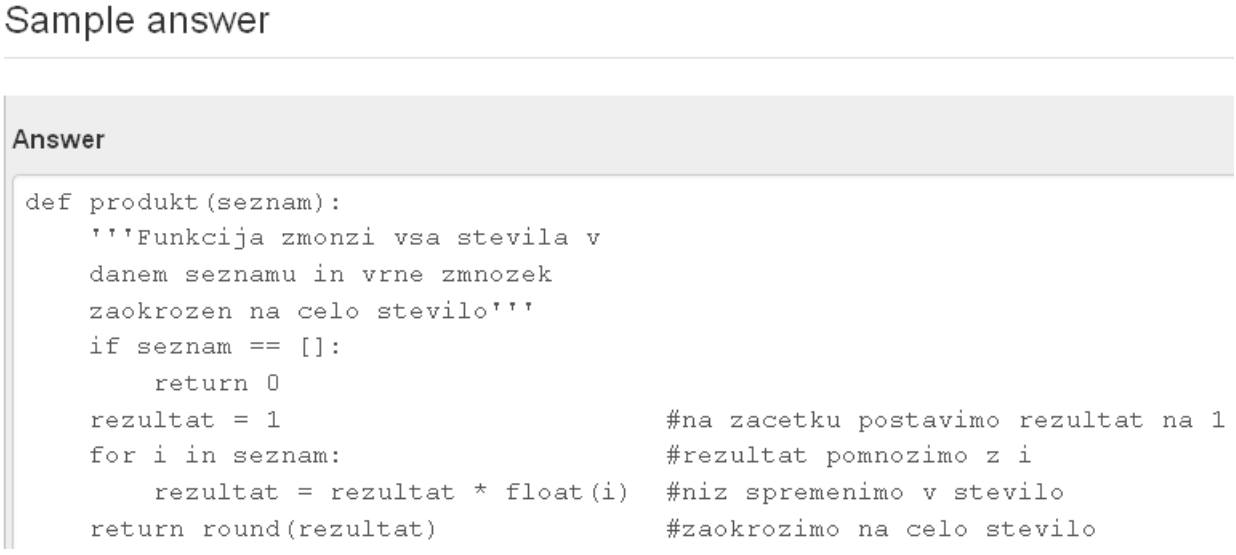
Slika 87: Razdelek *CodeRunner question type*

Sledi razdelek *General* (Slika 88). Vprašanje bomo shranili v kategorijo *Seznami, nizi* (meni *Save in category*). V *Question name* vpišemo ime našega vprašanja, na primer *Produkt stevil v seznamu*. V polje *Question text* napišemo natančna navodila. V okvirček *Default mark* vpišemo 10. Vprašanje bomo torej vrednotili z 10 točkami. Teh 10 točk bomo morali razporediti v 5 testnih primerov, ki jih bomo sestavili.



Slika 88: Razdelek *General*

Polje *General feedback* pustimo prazno, v polje *Answer* pa napišemo rešitev, ki bo vidna študentu po oddaji kviza (Slika 89).

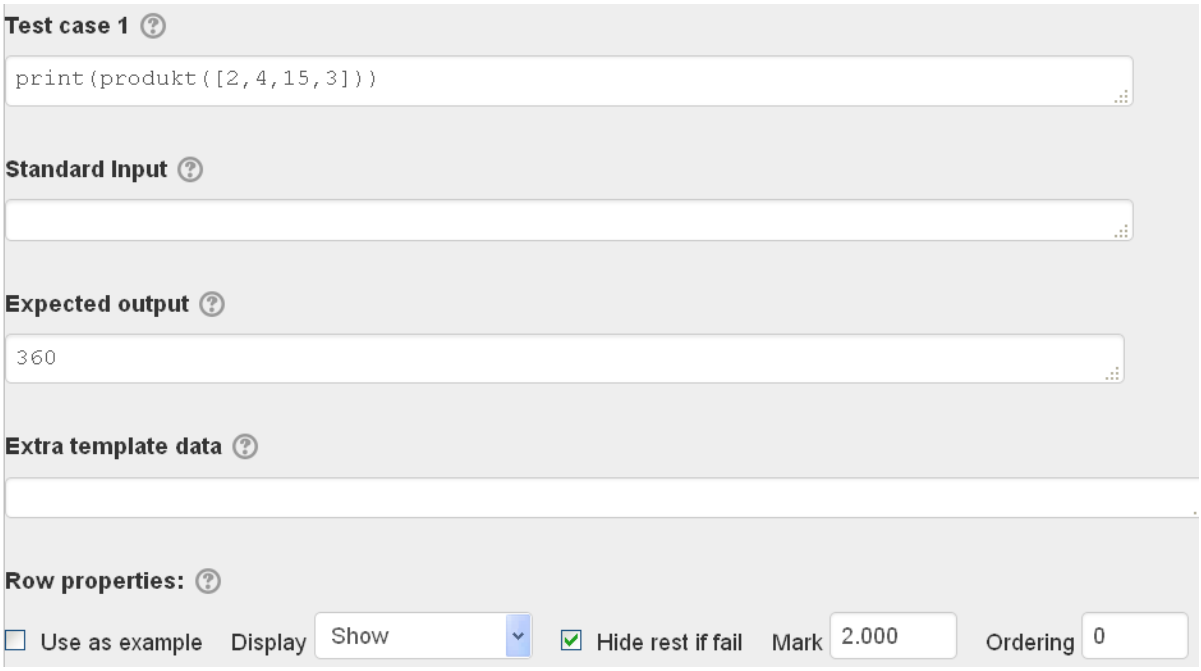


Slika 89: Razdelek *Sample answer*

Sledi sestavljanje testnih primerov. Ker je vprašanje tipa 'napiši funkcijo', se bodo testni primeri sestavljali na podoben način kot pri vprašanju *kvadrat števila x* v poglavju 5.2.1. Polji *Standard Input* in *Extra template data* pustimo prazni v vseh testnih primerih. Tudi v okencu *Ordering* bomo pustili privzete vrednosti.

V testnem primeru 1 (Slika 90) bomo testirali funkcijo s seznamom, ki vsebuje cela števila. V polje *Test case 1* vpišemo del kode, s katero bomo testirali funkcijo. Testirali jo bomo tako, da bomo zahtevali izpis funkcije pri določenih vhodnih podatkih. Za izpis funkcije uporabimo ukaz *print()*. V polje *Expected output* vpišemo produkt števil iz seznama.

V razdelku *Row properties* bomo obkljukali možnost *Hide rest if fail*, v okencu *Mark* pa bomo nastavili vrednost na 2. V kolikor bo študent uspešno rešil prvi testni primer, bo nagrajen z 2 točkama .



Slika 90: Testni primer 1

V testnem primeru 1 vidimo, da je vhodni podatek seznam [2, 4, 15, 3]. Funkcija mora vrniti produkt teh števil, to je 360.

V testnem primeru 2 (Slika 91) bo vhodni podatek prazen seznam. Funkcija mora vrniti vrednost 0. V razdelku *Row properties* bomo v meniju *Display* izbrali možnost *Hide*, ker ne želimo prikazati testnega primera študentu. Obkljukali bomo tudi možnost *Hide rest if fail*, vrednost testnega primera pa bomo nastavili na 2 točki (okence *Mark*).

Slika 91: Testni primer 2

Testni primer 3 (Slika 92) bo v seznamu vseboval negativno in decimalno število. S tem testnim primerom bomo preverili, če funkcija produkt števil na koncu zaokroži na celo število in če upošteva predznak.

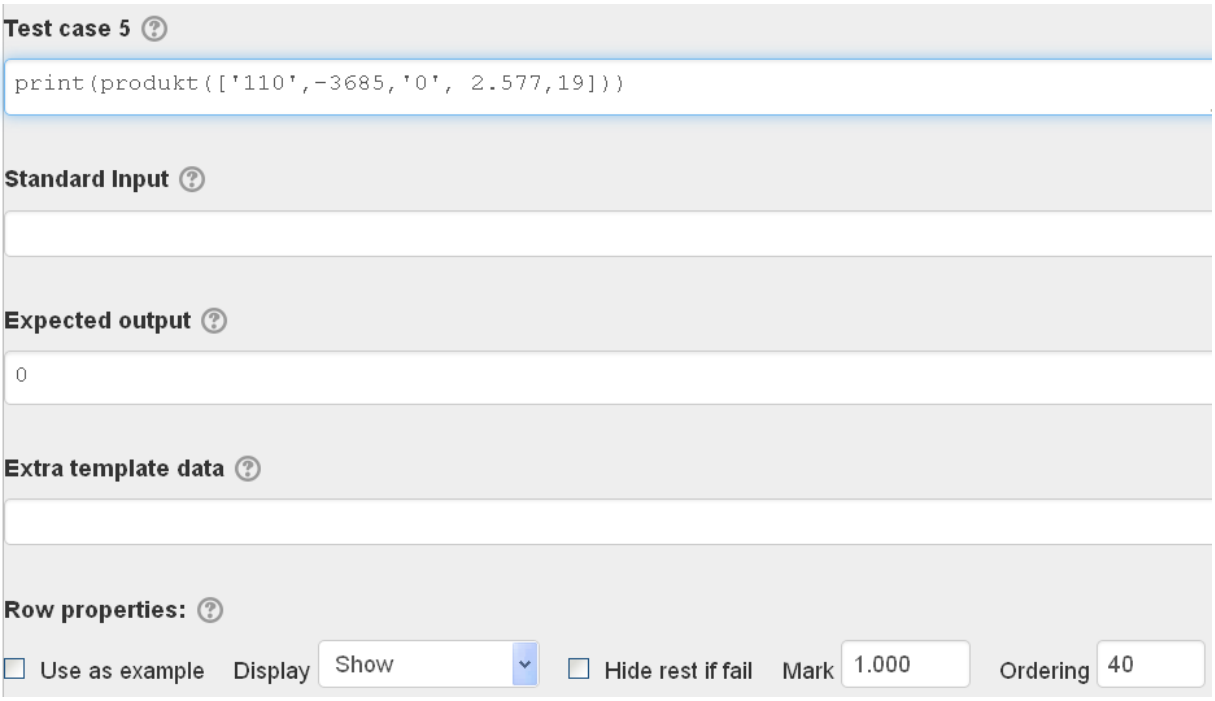
Ta testni primer bomo uporabili tudi kot zgled, zato bomo v razdelku *Row properties* obkljukali možnost *Use as example*. Obkljukali bomo tudi možnost *Hide rest if fail*. V polje *Mark* vpišemo vrednost *3*, torej bo ta testni primer vreden 3 točke.

Slika 92: Testni primer 3

V testnem primeru 4 (Slika 93) bomo preverili, če funkcija prebere število, ki je podano v nizu. Pametno je v niz vstaviti decimalno število. S tem preverimo, če koda iz niza prebere tudi decimalna števila in ne samo cela. V razdelku *Row properties* si v meniju *Display* izberemo možnost *Hide.* Obkljukali bomo tudi možnost *Hide rest if fail.* Testni primer 4 ovrednotimo z 2 točkama (polje *Mark*).

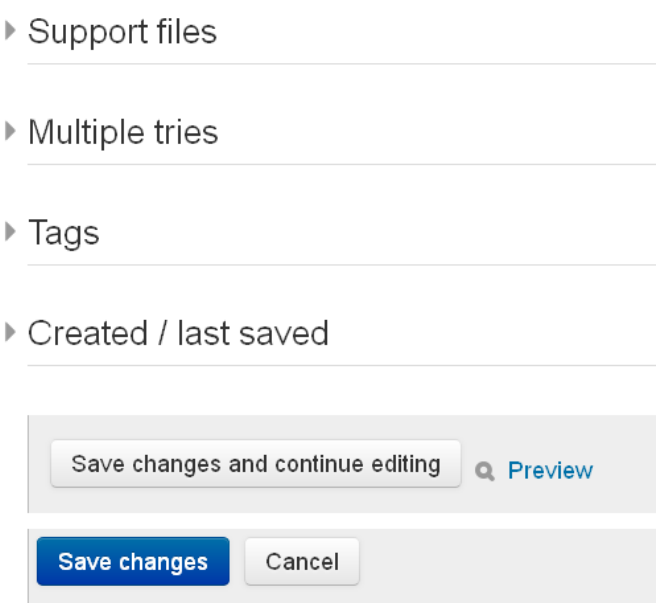
Slika 93: Testni primer 4

Testni primer 5 (Slika 94) bo v seznamu poleg vseh ostalih možnosti zapisa števil (število v nizu, decimalno število, negativno število) vseboval tudi število 0. Od funkcije pričakujemo, da bo vrnila vrednost 0. V polju *Mark ga* ovrednotimo z 1 točko.



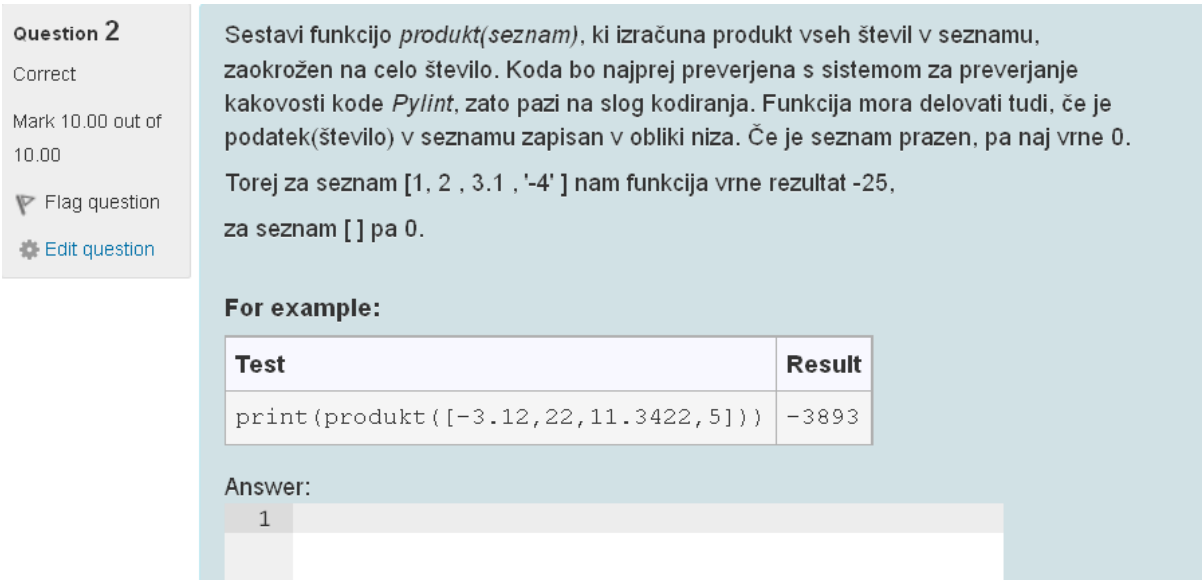
Slika 94 : Testni primer 5

Vse razdelke, ki sledijo za razdelkom *Test cases* (*Support files, Multiple tries, Tags, Created/last saved*), lahko spustimo, ker tam ne bomo spreminjali nastavitev. Da shranimo novo nastalo vprašanje, kliknemo na gumb *Save changes* na dnu strani (Slika 95).



Slika 95: Shranimo vprašanje

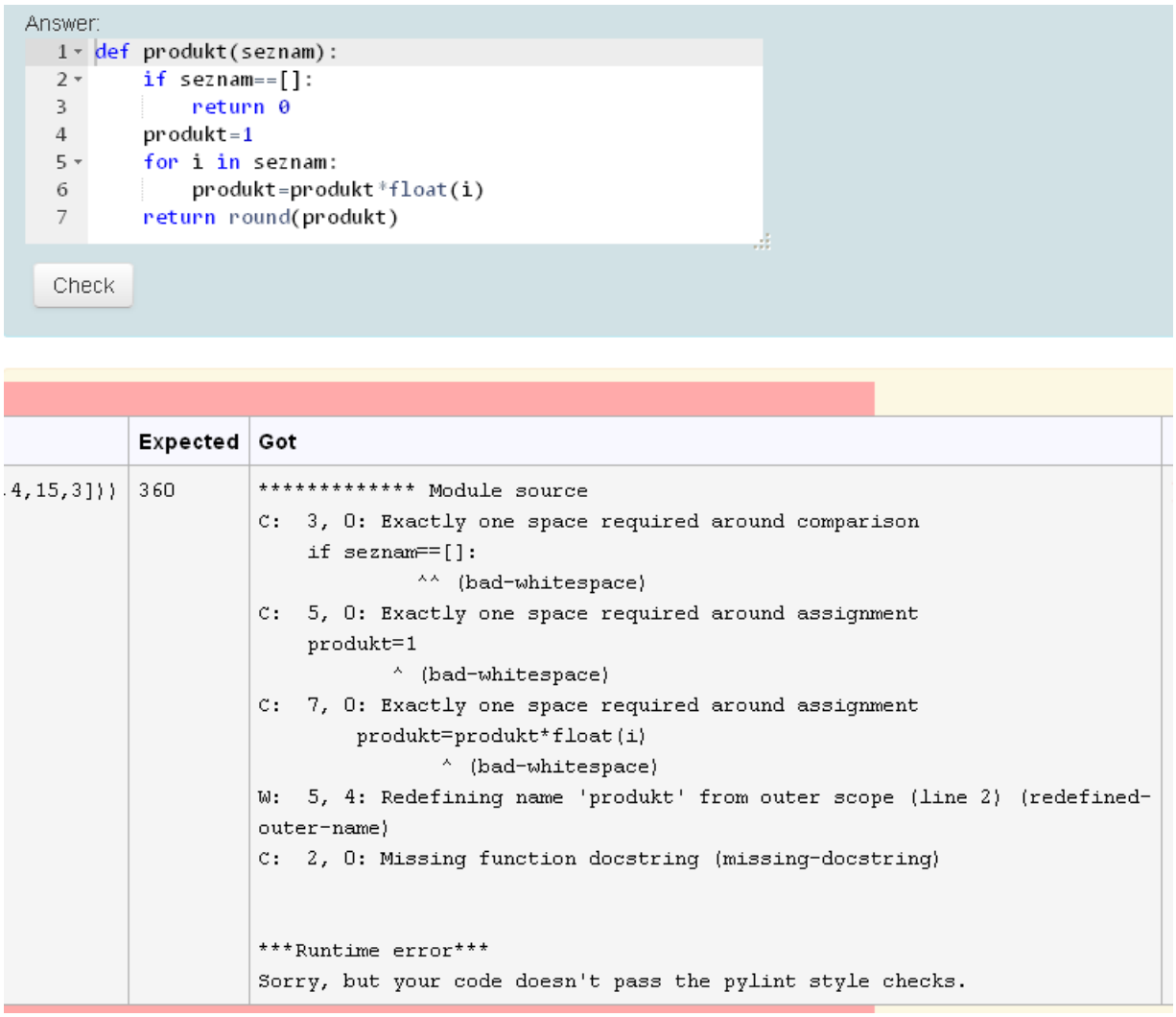
Naše novo vprašanje študent vidi takole (Slika 96):



Slika 96: Kako vidi študent vprašanje

Na sliki opazimo, da je za zgled podan testni primer 3.

Poglejmo si najprej, kaj se zgodi, če študent odda kodo, ki ni v skladu s sintakso, katero preverja sistem *Pylint* (Slika 97).



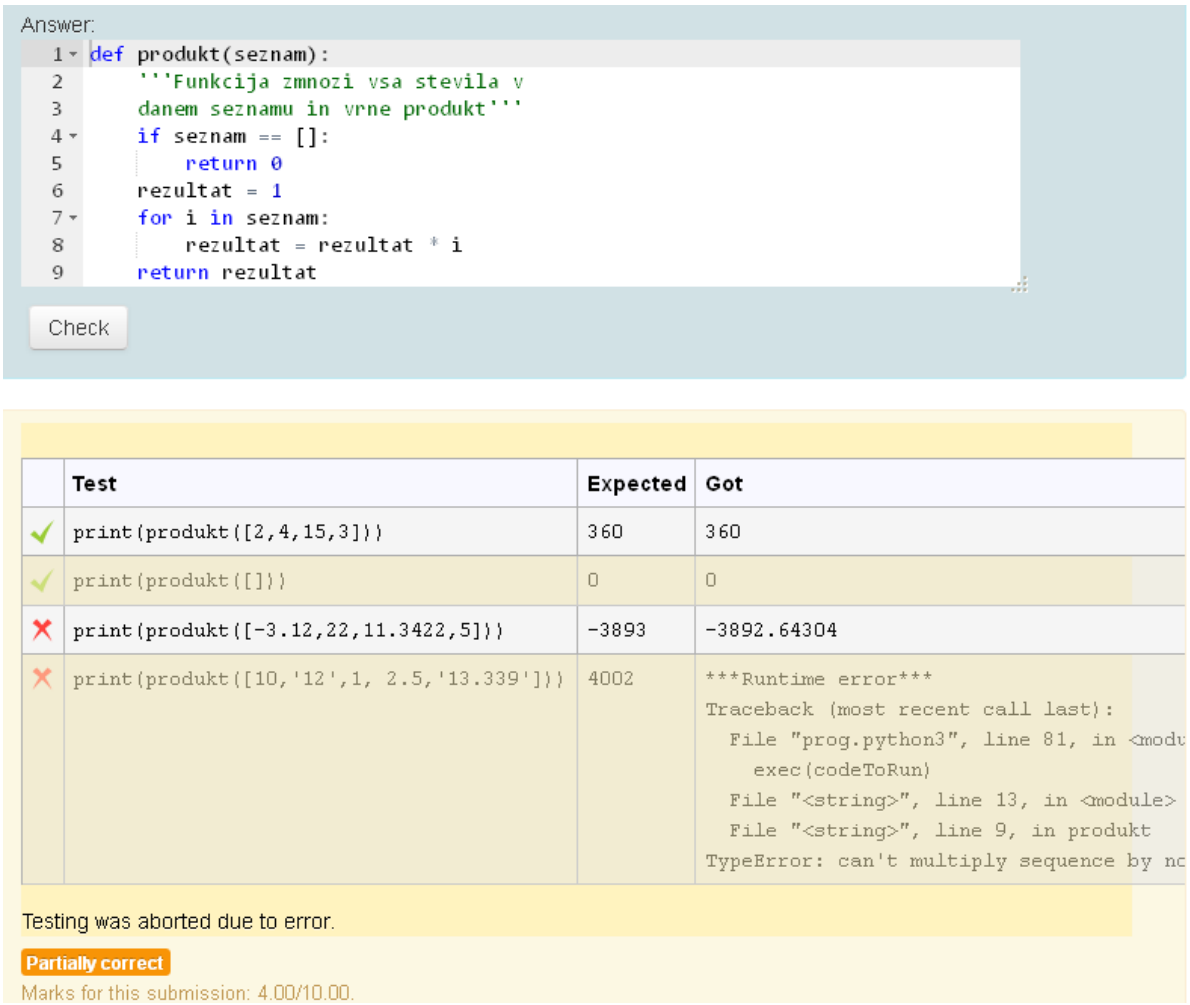
Slika 97: Koda ni v skladu s sintakso

V tabeli z rezultati opazimo nekaj opozoril, na koncu pa je izpisano sporočilo *Sorry, but your code doesn't pass the pylint checks.* To pomeni, da naša koda ni prestala testiranja sintakse. Poglejmo si razlago zgoraj prikazanih opozoril v tabeli:

- *`Exactly one space required around assignment`* – natanko en presledek je potreben okoli operatorja '='
- *`Redefining name 'produkt' from outer scope`* – potrebno je spremeniti ime spremenljivke 'produkt'
- *`Missing function docstring`* – manjka dokumentacijski niz funkcije

Študent mora kodo dopolniti in preurediti tako, kot opisujejo opozorila.

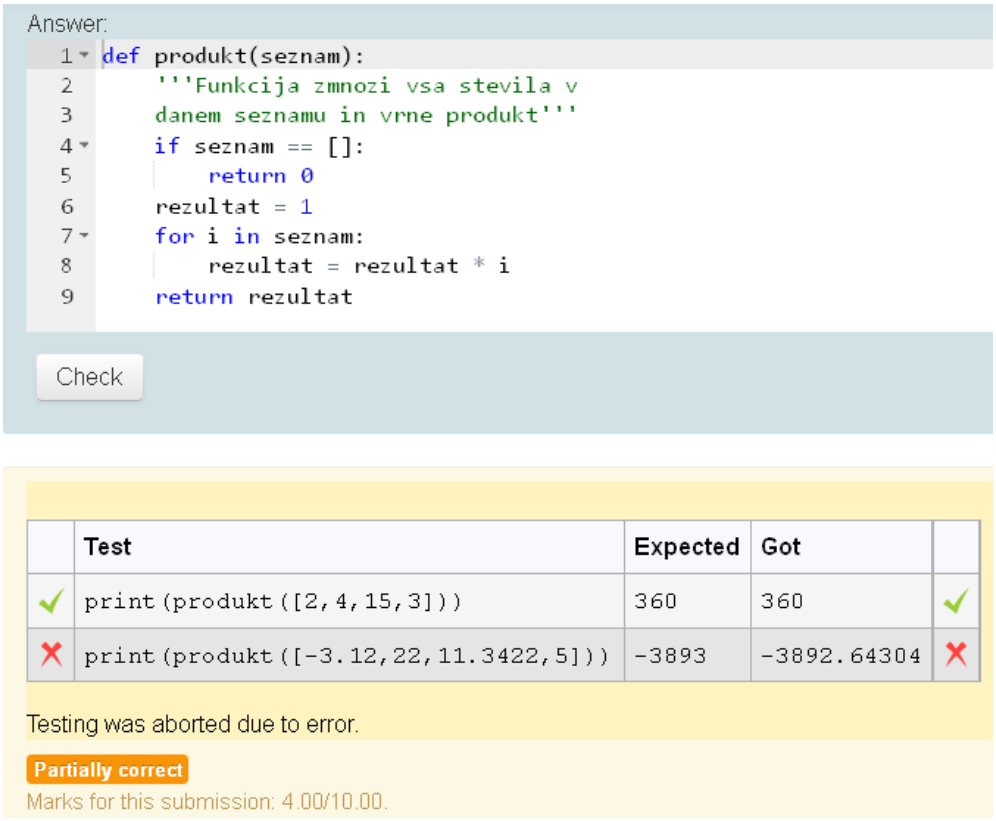
Slika 98 prikazuje primer, ko je študentova koda napisana v skladu s sintakso, vendar ne deluje popolnoma pravilno.



Slika 98: Delno pravilna koda

Opazimo, da je tabela tokrat obarvana rumeno. Študentova koda deluje pravilno samo za cela števila in za prazen seznam. Koda ni prestala testnega primera 3, ker rezultat ni zaokrožen na celo število. Ker smo pri sestavljanju testnega primera 3 obkljukali možnost *Hide rest if fail*, se 4. in 5. testni primer študentu ne prikažeta.

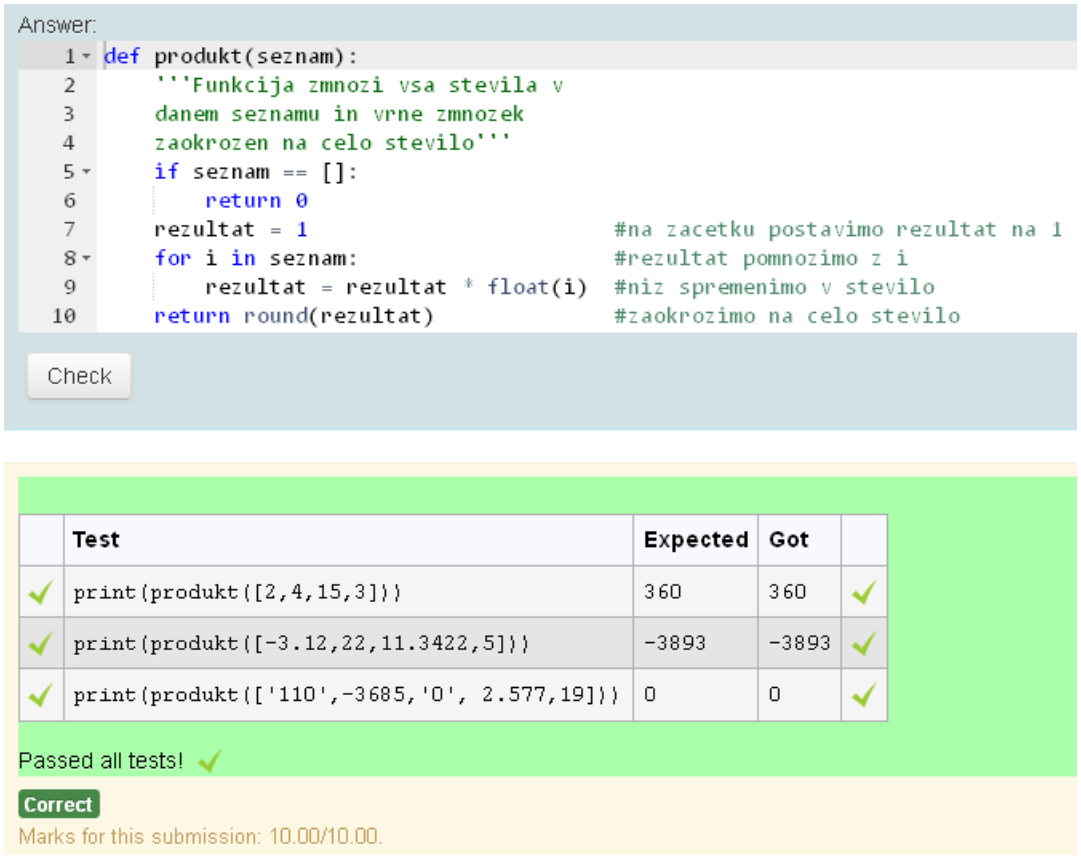
Slika 99 prikazuje, kako vidi študent tabelo z rezultati.



Slika 99: Kako vidi študent odgovor

Študentu se prikažeta samo testna primera 1 in 3. Testni primer 2 je skrit, kljub temu da je pravilen. Testna primera 4 in 5 nista prikazana, ker smo v testnem primeru 3 obkljukali možnost *Hide rest if fail*. Študent dobi za odgovor 4 točke, ker je koda uspešno rešila 1. in 2. testni primer (vsak vreden 2 točki).

Slika 100 prikazuje primer, kako vidi študent vprašanje, ko napiše pravilen odgovor.



Slika 100: Kako vidi študent pravilen odgovor

Tudi pri pravilnem odgovoru študent ne vidi vseh testnih primerov. Pri sestavljanju testnih primerov smo skrili 2. in 4. testni primer, ostali trije so vidni.

6.3 Izpis kvadratov števil od 1 do n

Pri sestavljanju te naloge bo poudarek na prepovedani uporabi določenega konstruktorja in obvezni uporabi drugega konstruktorja. Tudi to nalogo bomo sestavili s pomočjo prototipa *python3_cosc121*, zato bomo morali paziti na slog kodiranja. Tokrat bomo napačen odgovor študenta kaznovali z določenim odbitkom. Zahtevali bomo tudi popolno ujemanje v testnih primerih, način ocenjevanja pa bo *All-or-nothing grading*, torej bomo zahtevali, da so vsi testni primeri rešeni pravilno. Pokazali bomo tudi možnost, ko so testni primeri skriti le, kadar so napačni (z izjemo prvega testnega primera, ki je predstavljen kot zgled). Ko bo študent napisal pravilen odgovor, bodo vidni vsi testni primeri.

Besedilo naloge: *Napiši funkcijo kvadrati_1_do_n(stevilo_n), ki za vhodni podatek dobi poljubno celo število, izpiše pa tabelo vseh celih števil od 1 do vključno n. Za n< 1 izpiše sporočilo 'Stevilo je manjse od 1!'. Pri sestavljanju funkcije je prepovedana uporaba zanke For, nujno potrebna pa je uporaba zanke While.Tabela se mora izpisati v obliki, kot je prikazana v zgledu* (Slika 101).

Test	Result
kvadrati_1_do_n(4)	1 * 1 = 1 2 * 2 = 4 3 * 3 = 9 4 * 4 = 16

Slika 101: Zgled

Za pravilni odgovor tokrat vemo, da bomo morali uporabiti *zanko while*. Pri sestavljanju testnih primerov bomo preverili naslednje:

- **Testni primer 1** – Vhodni podatek je število 4. S tem testnim primerom preverimo, če funkcija pravilno deluje za tipični primer. Ta testni primer bomo uporabili tudi kot zgled v navodilih naloge.
- **Testni primer 2** – Vhodni podatek je število 1. Tu bomo preverili, če funkcija izpiše samo eno vrstico.
- **Testni primer 3** – Preverimo izpis za število 0. Funkcija mora izpisati sporočilo *Stevilo je manjse od 1!*
- **Testni primer 4** – Preverimo izpis za negativno število -5. Tudi tu mora funkcija izpisati sporočilo *Stevilo je manjse od 1!*

Pravilno sestavljena funkcija z uporabo *zanke while* je na primer taka, kot jo prikazuje Slika 102.

```
1  def kvadrati_1_do_n(stevilo_n):
2      ''' Funkcija izpise kvadrate
3      stevil od 1 do n'''
4      if stevilo_n < 1:                #ce je stevilo manjse od 1
5          print('Stevilo je manjse od 1!') #izpisemo sporocilo
6      i = 1
7      while i < stevilo_n + 1:         #v zanki zajamemo tudi
8          print(i, '*', i,             #zadnje stevilo
9                '=', i * i)
10         i += 1
```

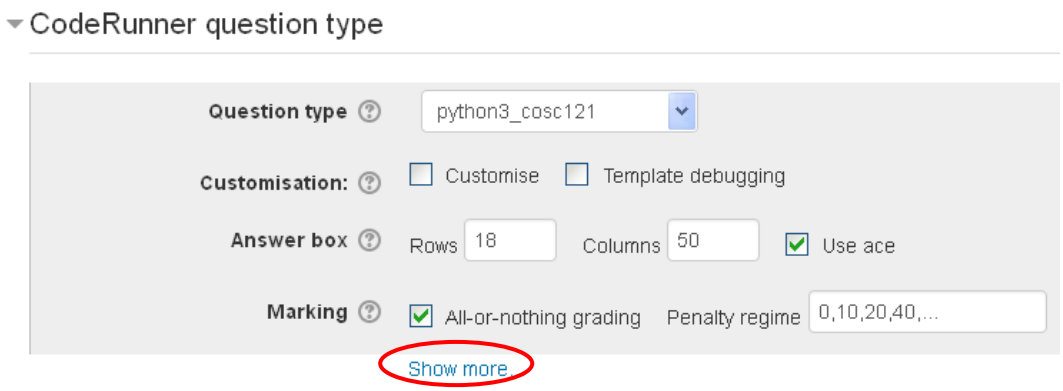
Slika 102: Rešitev naloge z uporabo zanke w*hile*

Najprej preverimo, če je podano število manjše od 1, nato pa ustvarimo zanko *while*, ki bo izpisala množenje števil in njihovih kvadratov. Pazimo na slog kodiranja. V kodi mora biti napisan dokumentacijski niz (*docstring)*, pazimo na natančne presledke, na pravilen zamik, pravilno poimenovanje funkcije...

V razdelku *Administration* kliknemo najprej na *Question bank – Questions* in nato na *Create a new question*. Izberemo si tip vprašanja *CodeRunner.* Odpre se nam že znana stran za sestavljanje vprašanja. V razdelku *CodeRunner question type* si izberemo tip vprašanja *python3_cosc121.* Velikost okenca za odgovor (*Answer box*) bomo nastavili na 18 vrstic (*rows*) in 50 stolpcev (*columns*). Možnost *Use ace* pustimo obkljukano.

Polje *All-or-nothing grading* pustimo obkljukano, v okence *Penalty regime* pa napišemo številke *0,10,20,40...*Ta zapis pomeni, da za prvi napačni poizkus študenta ne bomo kaznovali. Drugi poizkus bo kaznovan z 10% odbitka, tretji poizkus z 20% odbitka, četrti poizkus s 40% odbitka. Znak '*...'* pomeni, da se bo kazen stopnjevala za razliko zadnjih dveh števil napisanih v vrsti, torej za 20%.

V tem primeru si bomo pogledali še dodatno polje *Template params**. Do njega pridemo tako, da v razdelku *CodeRunner question type* kliknemo na *Show more* (Slika 103).

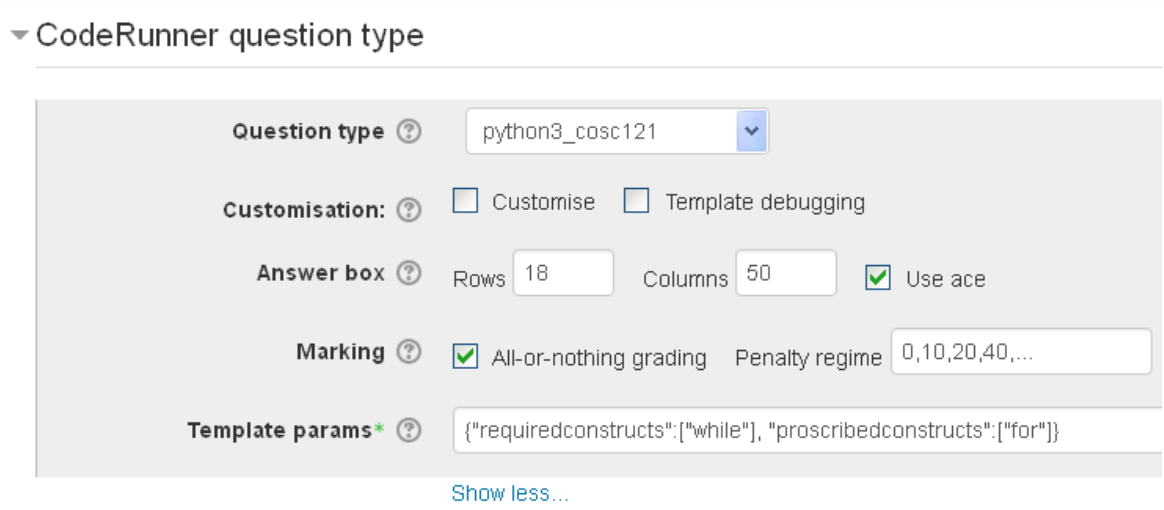


Slika 103: Kliknemo na *Show more*

Kot je bilo predstavljeno v poglavju 5.1.2, ima predloga prototipa *python3_cosc121* vgrajene številne parametre. V tej nalogi bomo uporabili parametra za obvezno uporabo konstruktorja (*requiredconstructs*) in za prepovedano uporabo konstruktorja (*proscribedconstructs*). Prepovedali bomo uporabo konstruktorja *for*, zahtevali pa bomo uporabo konstruktorja *while*. V polje *Template params** vpišemo ukaz

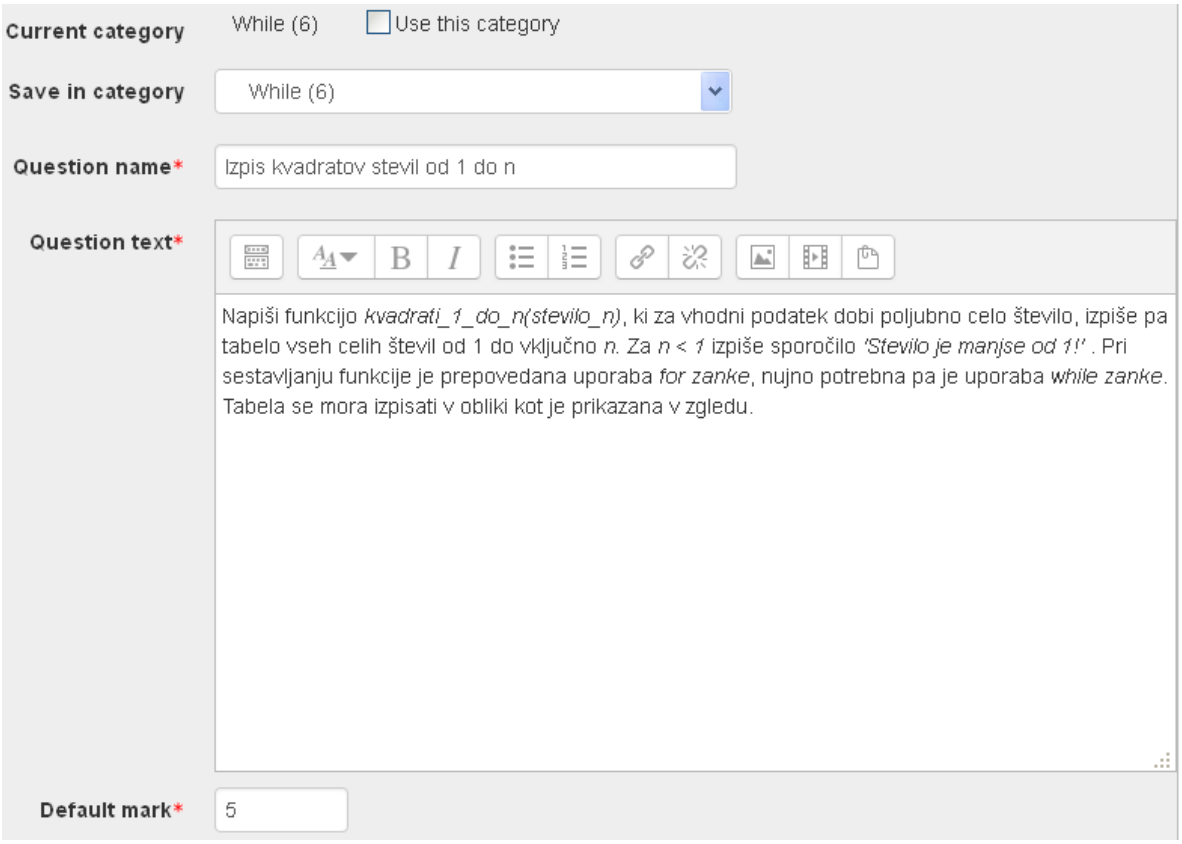
```
{"requiredconstructs":["while"], "proscribedconstructs":["for"]},
```

prikazan na Slika 104.



Slika 104: Ukaz v polju *Template params**

S sestavljanjem vprašanja nadaljujemo v razdelku *General* (Slika 105). Vprašanje bomo shranili v kategorijo *While*, poimenovali ga bomo *Izpis kvadratov stevil od 1 do n*. V polje *Answer* bomo napisali navodila naloge ter jo ovrednotili s 5 točkami (polje *Default mark*).



Slika 105: Razdelek *General*

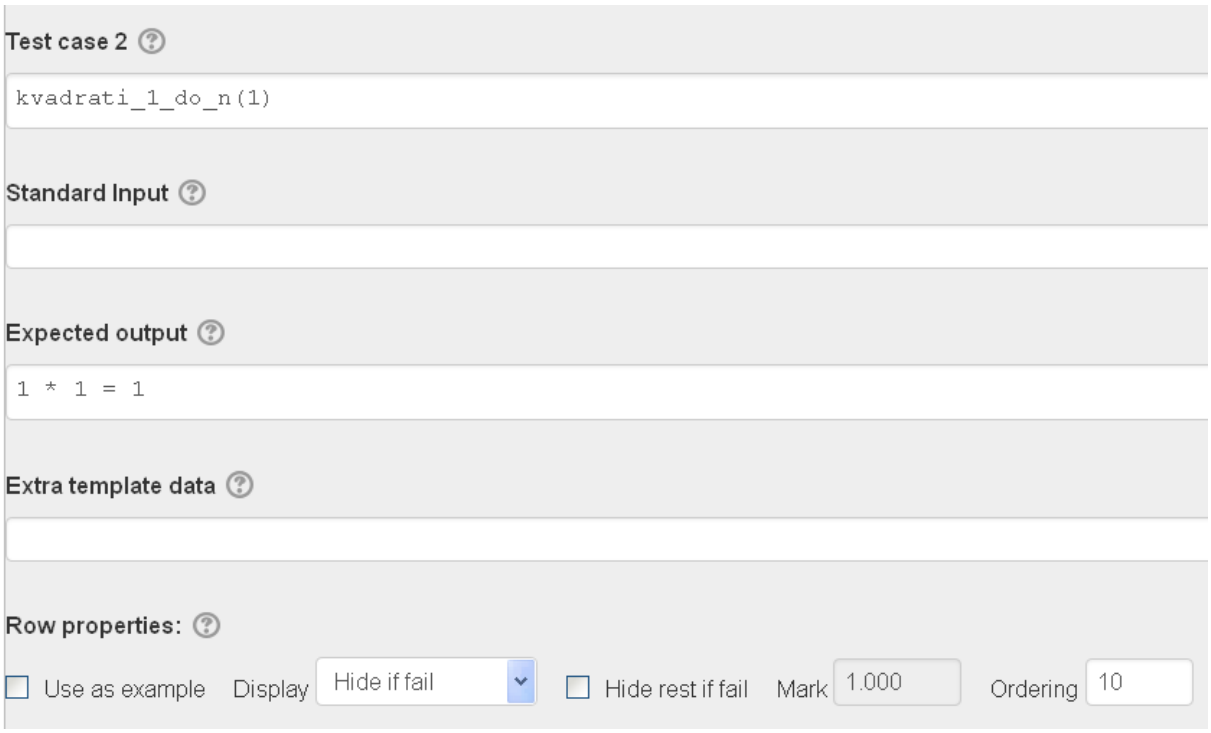
Polje *General feedback* pustimo prazno, v polje *Answer* pa napišemo rešitev naloge (Slika 102). Sledi sestavljanje testnih primerov v razdelku *Test cases*.

Naloga je tipa 'napiši funkcijo', zato bomo testne primere sestavljali podobno kot pri prejšnji nalogi. Polja *Standard Input* in *Extra template data*bomo pustili prazna. V polju *Ordering* bomo pustili privzete vrednosti.

Sestavimo testni primer 1 (Slika 106). Funkcija je v tej nalogi definirana malo drugače. Namesto da vrne vrednosti, jih izpiše (ukaz `print()`). Zato v polje *Test case 1* kot del kode za testiranje izvedemo samo klic funkcije za vhodni podatek 4. V polje *Expected output* pa napišemo celotno tabelo kvadratov od 1 do 4. V *Row properties* obkljukamo *Use as example*, da bomo testni primer prikazali kot zgled v navodilih naloge.

Slika 106: Testni primer 1

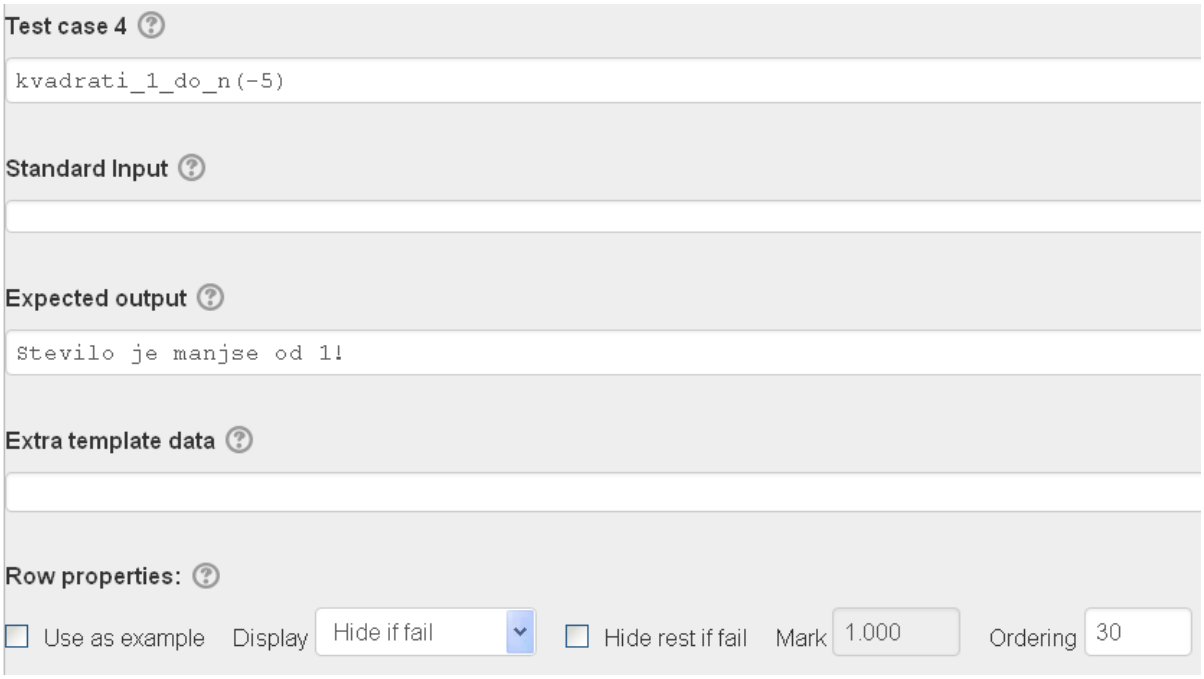
Testni primer 2 (Slika 107) sestavimo podobno kot testni primer 1. V meniju *Display* bomo izbrali možnost *Hide if fail*. Prikazan bo le, če bo pravilen.



Slika 107: Testni primer 2

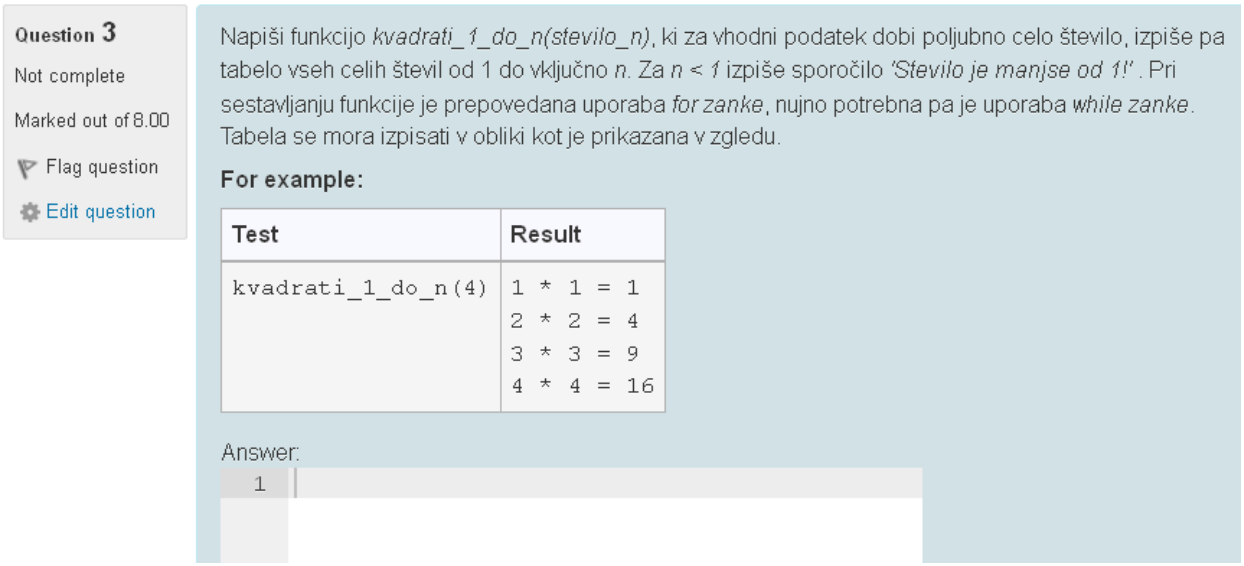
Na podoben način sestavimo še testna primera 3 in 4. V testnem primeru 3 bo vhodni podatek število 0, v testnem primeru 4 pa število -5. Funkcija mora v obeh primerih izpisati sporočilo *Stevilo je manjse od 1!*. V obeh testnih primerih bomo v meniju *Display* izbrali možnost *Hide if fail* (Slika 108 in Slika 109).

Slika 108: Testni primer 3



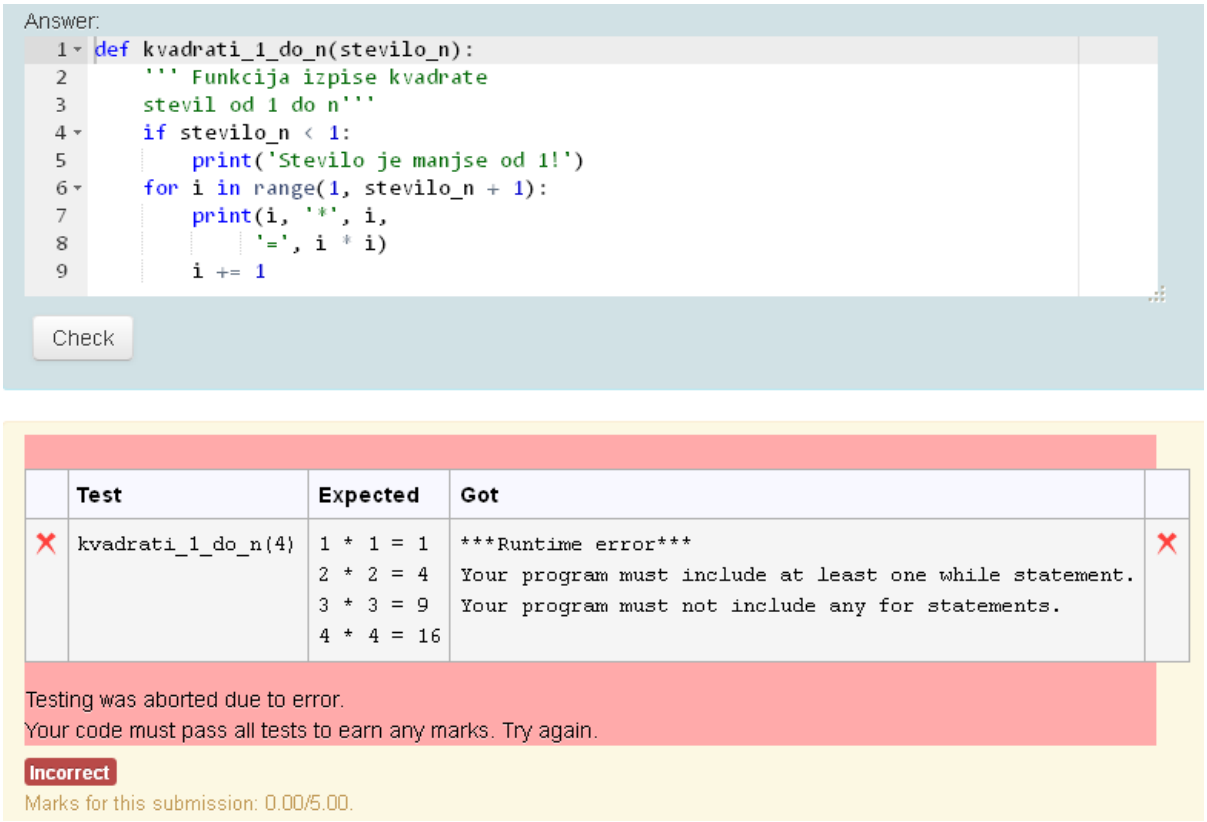
Slika 109: Testni primer 4

Ostalih razdelkov ni potrebno dodatno nastavljati, zato lahko shranimo vprašanje s klikom na gumb *Save changes* na dnu strani. Sestavljeno vprašanje je videti tako kot na Slika 110.



Slika 110: Sestavljeno vprašanje

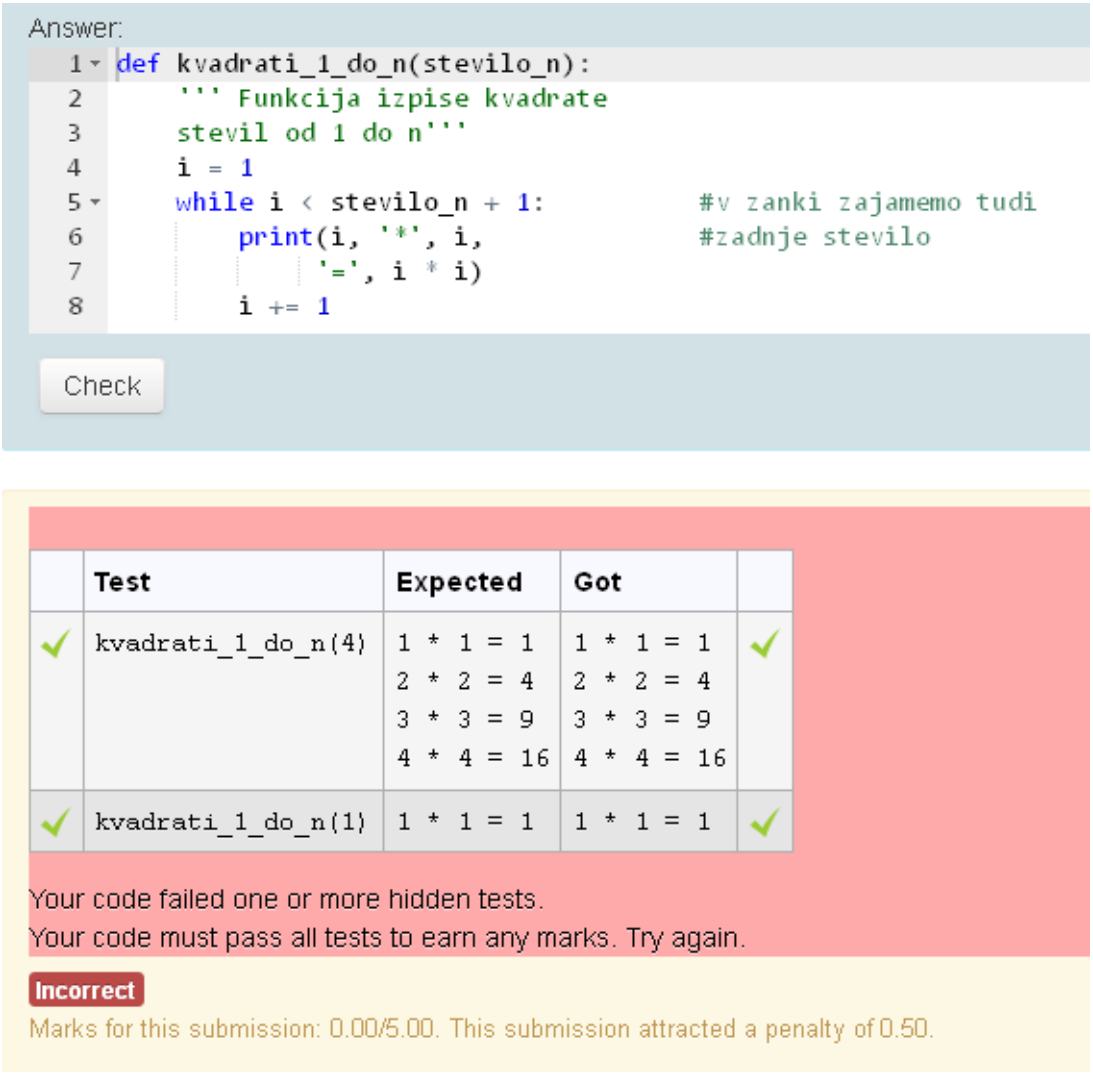
Slika 111 predstavlja primer, ko študent odda kodo, v kateri je uporabil prepovedan konstruktor, zanka *for.*



Slika 111: Odgovor s prepovedanim konstruktorjem

V tabeli s testnimi primeri smo dobili sporočilo o napaki. Opozarja nas, da mora naša funkcija vsebovati vsaj eno zanko *while*, ne sme pa vsebovati nobene zanke *for*. Opazimo, da za napačen odgovor nismo bili kaznovani. V primeru drugega napačnega odgovora sledi kazen z 10% odbitkom.

Slika 112 predstavlja primer, ko študent kodo ustrezno popravi, ampak pozabi na primer, ko je vhodni podatek manjši kot 1.



Slika 112: Študent odda nepopolni odgovor

Na sliki opazimo, da se študentu izpišejo samo pravilni testni primeri. Zadnja dva testna primera sta skrita, ker sta bila napačna. Študentu se pod tabelo izpiše sporočilo, da je njegova koda napačna za enega ali več testnih primerov. Opazimo tudi, da je v 2. poizkusu dobil odbitek 0,50 točke (kar znaša 10%).

Če odda študent v 3. poizkusu pravilen odgovor, dobi za uspešno opravljeno nalogo 4,50 točk. Prikažejo se mu vsi testni primeri (Slika 113).

Slika 113: Študent odda pravilen odgovor

7 Zaključek

Vsako vprašanje, katerega odgovor zahteva pisanje računalniškega programa, ki ga lahko avtomatsko ocenimo, lahko sestavimo tudi v CodeRunnerju. Izkaže pa se, da je sestavljanje kompleksnejših vprašanj dokaj zapleteno. Tudi ocenjevanje vprašanj, ki imajo za odgovor nek grafični prikaz, je zelo težavno.

Pa vendar se orodje konstantno nadgrajuje. Njegova uporaba se širi po vsem svetu, aktivnost na forumu na domači spletni strani se je močno povečala. V bližnji prihodnosti lahko pričakujemo, da se bo sestavljanje vprašanj poenostavilo, testirati bo možno še več stvari, verjetno tudi grafične prikaze. V bližnji prihodnosti bi bilo dobro postaviti neko vrsto spletnega skladišča, kjer bi bil možen uvoz in izvoz vprašanj. Dostop bi omogočili učiteljem po vsem svetu, saj bi si na ta način lahko izmenjali veliko število vprašanj in prihranili ogromno časa.

Temo diplomske naloge smo izbrali zaradi velike uporabnosti orodja. Možnost, da se bomo tudi v prihodnje srečevali s podobnim orodjem, je velika. S sestavljanjem praktičnih vprašanj in testnih primerov smo obnovili tudi znanje iz programiranja v programskem jeziku Python 3. Zbirka vprašanj je trenutno v spletni učilnici http://lokar.fmf.uni-lj.si/moodle/course/view.php?id=53, kot priloga diplomski nalogi pa so priložene tudi *xml* datoteke, v katerih so shranjena vprašanja po kategorijah. Zbirko vprašanj imamo namen še dodatno razširiti.

8 Spletni viri

ACE. [Online] 2010. https://ace.c9.io/#nav=about (dostop 9.9. 2016)

API-key.[Online]2015. https://en.wikipedia.org/wiki/Application_programming_interface_key (dostop 9.9. 2016)

Directi Group. *Codechef*.[Online] 2009. https://www.codechef.com/(dostop 9.9. 2016)

Richard Lobb, Jenny Harlow.*CodeRunner*. The University of Canterbury, Nova Zelandija. [Online]http://coderunner.org.nz(dostop 9.9. 2016)

Richard Lobb, Jenny Harlow. *CodeRunner: A Tool for Assessing Computer Programming Skills.*[Online]2016. http://coderunner.org.nz/pluginfile.php/1746/mod_resource/content/2/CodeRunnerArticlePublishedForm.pdf(dostop 9.9. 2016)

Gerenčer Barbara.*Kvizi v spletni učilnici Moodle* [Online]junij 2008. http://lokar.fmf.uni-lj.si/www/osebno/OpravljeneDiplome/KlaudijaGerencer_diploma-koncna.pdf(dostop 9.9. 2016)

Richard Lobb. *Jobe*. University of Canterbury, Nova Zelandija. [Online]2016https://github.com/trampgeek/jobe(dostop 9.9. 2016)

Matija Lokar, Matija Pretnar, Gregor Jerše, Sonja Jerše.*Preparing programming exercises with efficient automated validation tests,International Conference on Informatics in Schools*, str. 142[Online]2015 https://issep15.fri.uni-lj.si/files/issep2015-proceedings.pdf(dostop 9.9. 2016)

Žiga Ham, Mitja Trampuš, Jan Berčič. *Putka.*Zavod za računalniško izobraževanje Ljubljana. [Online]2004. http://www.putka.si(dostop 9.9. 2016)

Guido van Rossum. *Python.* Corporation for National Research Initiatives. [Online]2016 https://www.python.org(dostop 9.9. 2016)

TWIG. [Online]2016. http://twig.sensiolabs.org(dostop 9.9. 2016)