# Sustavi za automatsko ocjenjivanje koda u Pythonu

**Miloš, Klara**

Sveučilište u Splitu

Prirodoslovno – matematički fakultet

Odjel za informatiku

Klara Miloš

# SUSTAVI ZA AUTOMATSKO OCJENJIVANJE KODA U PYTHONU

Diplomski rad

Split, 2024.

## TEMELJNA DOKUMENTACIJSKA KARTICA

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

Diplomski rad

# SUSTAVI ZA AUTOMATSKO OCJENJIVANJE KODA U PYTHONU

Klara Miloš

## SAŽETAK

Ovaj rad bavi se razvojem i primjenom automatiziranog sustava za ocjenjivanje programskih zadataka, poznatog kao autograder. Autograder je implementiran korištenjem Python programskog jezika i Flask web okvira, te pruža platformu za automatsku evaluaciju studentskih rješenja putem niza definiranih zadataka. Rad započinje uvodom u koncept autograderskih sustava, njihovom funkcionalnošću i značajem u obrazovanju, s posebnim naglaskom na prednosti i izazove u usporedbi s tradicionalnim metodama ocjenjivanja.

Jedan od ključnih dijelova rada posvećen je tehničkoj implementaciji autogradera. Detaljno su opisane komponente sustava, uključujući korisničko sučelje, mehanizme za testiranje rješenja te načini na koje se rezultati prikazuju studentima

Pored same implementacije, rad se osvrće na važnost odabira tehnologija, gdje su istaknute prednosti korištenja Python programskog jezika i Flask okvira. Navedene su detaljne upute za instalaciju i korištenje sustava, kako bi čitatelji mogli replicirati ili dalje razviti autograder.

U zaključku, diskutira se o budućem razvoju autogradera, mogućnostima njegove nadogradnje i uvođenju novih funkcionalnosti poput prepoznavanja stilskih grešaka u kodu. Također su istaknute prednosti sustava, poput brzine i preciznosti ocjenjivanja, ali i slabosti, među kojima je ograničena fleksibilnost u prepoznavanju alternativnih rješenja. Unatoč tim izazovima, autograderi predstavljaju važan korak naprijed u modernizaciji obrazovnog procesa u području računalnih znanosti.

**Ključne riječi:** autograder, automatsko ocjenjivanje, Python, Flask, obrazovni sustav, programiranje, evaluacija zadataka, računalne znanosti

**Rad sadrži:** 40 stranica, 25 grafička prikaza, 1 tablicu i literaturnih navoda. Izvornik je na hrvatskom jeziku.

**Mentor:** **izv. prof. dr. sc. Ani Grubišić,** *Prirodoslovno-matematički fakultet u Splitu*

**Ocjenjivači:** **izv. prof. dr. sc. Ani Grubišić,** *Prirodoslovno-matematički fakultet u Splitu*

**Izv. prof.dr.sc. Branku Žitku,** *Prirodoslovno-matematički fakultet u Splitu*

**dipl.ing. Ines Šarić-Grgić**, *Prirodoslovno-matematički fakultet u Splitu*

Rad prihvaćen: rujan 2024. g.

## BASIC DOCUMENTATION CARD

University of Split
Faculty of Science
Department of informatics
Ruđera Boškovića 33, 21000 Split, Croatia

Graduation thesis

# AUTOGRADERS FOR PYTHON

Klara Miloš

## SUMMARY

This thesis focuses on the development and application of an automated grading system for programming tasks, commonly known as an autograder. The autograder is implemented using the Python programming language and the Flask web framework, providing a platform for the automatic evaluation of student solutions through a series of predefined tasks. The thesis begins by introducing the concept of autograding systems, their functionality, and their significance in education, with particular emphasis on the advantages and challenges compared to traditional grading methods.

A key section of the thesis is dedicated to the technical implementation of the autograder. It provides a detailed explanation of the system's components, including the user interface, mechanisms for testing solutions, and how the results are presented to students

In addition to the implementation, the thesis discusses the importance of technology selection, highlighting the benefits of using the Python programming language and the Flask framework. Detailed instructions for installing and using the system are provided, allowing readers to replicate or further develop the autograder.

In the conclusion, the future development of the autograder is discussed, exploring possibilities for upgrades and the introduction of new features, such as recognizing coding style errors. The advantages of the system, such as grading speed and accuracy, are highlighted, as well as its weaknesses, including limited flexibility in recognizing alternative solutions. Despite these challenges, autograders represent a significant step forward in modernizing the educational process in the field of computer science.

**Key Words:** autograder, automated grading, Python, Flask, educational system, programming, task evaluation, computer science

**Thesis consists of:** 40 page, 25 figures, 1 table and 22 references

Original language: Croatian

**Mentor:** **Ani Grubišić, Ph.D.** *associate professor of Faculty of Science, University of Split*

**Reviewers:** **Ani Grubišić, Ph.D.** *associate professor of Faculty of Science, University of Split,*
**Branku Žitku,** *associate professor of Faculty of Science, University of Split,*
**Ines Šarić-Grgić**, *associate assistent of Faculty of Science, University of Split,*

Thesis accepted: September 2024.

# Sadržaj

Uvod ..... 2
1 Arhitektura autogradera ..... 3
1.1 Sučelje za korisnike ..... 3
1.2 Sustav za testiranje ..... 3
1.3 Sustav za ocjenjivanje ..... 4
1.4 Baza podataka ..... 5
1.5 Sustavi za obradu pogrešaka i iznimki ..... 5
1.6 Sigurnosni aspekti autogradera ..... 6
1.7 Integracija s obrazovnim platformama ..... 7
2 Primjeri autogradera u praksi ..... 10
2.1 Autograderi u MOOC platformama ..... 10
2.2 Korištenje u visokim obrazovnim institucijama ..... 10
2.3 Prilagodba za različite jezike ..... 11
2.4 Primjeri u srednjoškolskom obrazovanju ..... 11
2.5 Izazovi i budućnost autogradera ..... 12
2.6 Prednosti i nedostatci autogradera ..... 13
3 Python: Jezik po izboru za autogradere ..... 16
4 Oblikovanje autogradera za Python ..... 19
4.1 Odabir tehnologija za razvoj aplikacije ..... 19
4.1.1 Instalacija i korištenje Flask-a ..... 21
4.2 Modeliranje zahtjeva aplikacije ..... 26
4.3 Modeliranje tijeka aplikacije ..... 26
5 Implementacija autogradera za Python ..... 27
5.1 Struktura i funkcionalnost početnog zaslona aplikacije ..... 28
5.2 Struktura i funkcionalnost stranice zadataka ..... 29
5.3 Sustav za testiranje aplikacije ..... 30
Zaključak ..... 36
Bibliografija ..... 37
Popis slika ..... 39
Tablice ..... 40

# Uvod

Razvojem računalnih znanosti i porastom interesa za programiranje, sve više obrazovnih institucija i online platformi koristi tehnologije koje omogućuju automatizirano ocjenjivanje programskih zadataka. Automatski sustavi za ocjenjivanje, tzv. autograderi, postali su ključni alat u modernom obrazovanju, posebno u kontekstu sve većeg broja studenata i potrebe za brzim i točnim ocjenjivanjem. Oni omogućuju nastavnicima da efikasnije upravljaju velikim brojem zadataka, a studentima pružaju trenutnu povratnu informaciju, što potiče brže učenje i ispravljanje grešaka.

Autograderi se najčešće koriste u predmetima vezanim uz programiranje, gdje je ispravnost i učinkovitost koda ključna. Posebno su važni u jeziku Python, koji je jedan od najpopularnijih jezika za učenje programiranja zbog svoje jednostavnosti i široke primjene. U Pythonu, autograderi omogućuju automatsko testiranje koda na temelju unaprijed definiranih kriterija, što može uključivati testove funkcionalnosti, testove performansi, pa čak i analizu stilskih pravila.

U ovom radu fokus će biti na analizi autogradera za Python, na njihovoj ulozi u obrazovanju, na prednostima i izazovima koje donose te mogućnosti za daljnji razvoj i unaprjeđenje. Istražit će se kako autograderi utječu na proces učenja, koje tehnologije i alati se koriste za njihovu izradu, te kako se mogu integrirati u moderne obrazovne platforme. Kroz usporedbu različitih pristupa i tehnologija, cilj je pružiti sveobuhvatan pregled autogradera u Pythonu, s posebnim naglaskom na njihovu primjenu u obrazovnom kontekstu

# 1 Arhitektura autogradera

Autograderi su postali integralni dio suvremenih sustava za učenje programiranja, omogućujući automatizaciju procesa ocjenjivanja programskih zadataka. Njihova arhitektura je višeslojna i kompleksna, sastavljena od nekoliko ključnih komponenti koje zajedno omogućuju efikasno, točno i brzo ocjenjivanje studentskih rješenja. Svaka komponenta ima specifičnu ulogu i funkciju, a njihova međusobna interakcija osigurava uspješno funkcioniranje sustava kao cjeline.

## 1.1 Sučelje za korisnike

Sučelje za korisnike (eng. frontend) predstavlja prvi kontakt između studenta i sustava za automatsko ocjenjivanje. Ovaj sloj sustava mora biti intuitivan, pristupačan i jednostavan za korištenje, omogućujući studentima laku predaju rješenja, praćenje povratne informacije i analizu rezultata. U obrazovnim institucijama, sučelje autogradera često se integrira sa sustavima za upravljanje učenjem (LMS), kao što su Moodle, Canvas ili Blackboard, čime se studentima omogućuje neprekinuto iskustvo između predavanja, vježbi i ocjenjivanja.

Za optimalno korisničko iskustvo, frontend autogradera mora podržavati različite uređaje i preglednike, osiguravajući responzivnost i kompatibilnost. Na primjer, studenti mogu predavati svoje zadatke putem web sučelja, gdje mogu vidjeti svoje rezultate u realnom vremenu, primiti povratne informacije o pogreškama i usporediti svoje rješenje s primjerima. Sučelja često uključuju jednostavne editore koda ili čak integrirane razvojne okoline (IDE), posebno prilagođene za obrazovne svrhe. [1]

Implementacija frontend dijela autogradera može biti izazovna, posebno kada je riječ o integraciji s različitim platformama i alatima. Programeri frontenda moraju osigurati da su svi elementi sučelja intuitivni, te da sustav može učinkovito upravljati velikim brojem korisnika, uz minimiziranje latencije i optimizaciju performansi.

## 1.2 Sustav za testiranje

Backend sustav (eng. backend) autogradera odgovoran je za izvršavanje koda koji studenti predaju i evaluaciju rezultata na temelju unaprijed definiranih kriterija. Ovaj dio sustava obavlja

ključni posao - automatsko pokretanje studentskih programa u kontroliranom okruženju, testiranje njihovih rezultata protiv zadanih očekivanja, te bilježenje svih potrebnih informacija za kasniju analizu.

Backend sustav obično uključuje virtualizacijske ili sandbox tehnologije koje osiguravaju da se studentski kod može sigurno pokrenuti bez rizika za sigurnost sustava. Na primjer, Docker ili slični kontejnerski alati često se koriste za izolaciju studentskih okruženja, omogućujući da svaki studentski program bude izvršen u svom zasebnom okruženju, čime se smanjuje mogućnost da jedan program ometa rad drugog. [2]

Osim izolacije, backend sustav mora biti sposoban rukovati velikim brojem istovremenih zahtjeva, posebno u situacijama kada veliki broj studenata predaje svoje zadatke u isto vrijeme. To zahtijeva efikasnu distribuciju resursa, optimizaciju procesa i skalabilnost sustava. U tu svrhu često se koriste distribuirani sustavi i računarstvo u oblaku, koji omogućuju dinamičko prilagođavanje kapaciteta sustava u realnom vremenu.

Primjer implementacije backenda može uključivati sustave kao što su unittest ili pytest u Pythonu, koji omogućuju definiranje i pokretanje testnih slučajeva. Testni slučajevi mogu biti jednostavni, poput provjere ispravnog izlaza za određeni ulaz, ili kompleksni, uključujući mjerenje vremena izvršavanja, korištenje memorije, ili čak detekciju specifičnih obrazaca u studentskom kodu. [3]

## 1.3 Sustav za ocjenjivanje

Nakon što su testovi izvršeni, sustav za ocjenjivanje generira konačnu ocjenu za svaki zadatak. Ocjenjivanje može biti vrlo jednostavno, temeljeno samo na broju ispravnih testova, ili složeno, uključujući dodatne kriterije poput učinkovitosti, stila, i upotrebe određenih algoritamskih tehnika. Sustavi za ocjenjivanje mogu uključivati i različite vrste povratnih informacija, pružajući studentima detaljan pregled onoga što je dobro napravljeno i gdje su pogriješili.

Ocjene se često pohranjuju u bazu podataka zajedno s detaljima o izvedbi svakog studenta, što omogućuje praćenje napretka kroz vrijeme. U naprednim sustavima, ocjenjivanje može biti dinamično, prilagođavajući težinu zadataka i testova na temelju prethodnih performansi studenta. Na primjer, sustav može ponuditi teže zadatke studentima koji redovito uspješno rješavaju sve testove, ili dodatne savjete i jednostavnije zadatke onima koji imaju poteškoća. [4]

## 1.4 Baza podataka

Baza podataka u autograder sustavu je ključna komponenta koja pohranjuje sve relevantne informacije, uključujući studentske predaje, testne slučajeve, rezultate testiranja, i povratne informacije. Ovaj sloj sustava mora biti dizajniran tako da osigura sigurnost, integritet i konzistentnost podataka, posebno u okruženjima gdje veliki broj studenata istovremeno koristi sustav.

Baze podataka se obično implementiraju koristeći relacijske baze poput MySQL ili PostgreSQL, iako se u nekim slučajevima mogu koristiti NoSQL baze poput MongoDB, ovisno o specifičnim potrebama sustava. Podaci su organizirani u tablice koje omogućuju jednostavno pretraživanje i filtriranje, što omogućava nastavnicima da brzo dobiju uvid u napredak studenata, te da prepoznaju potencijalne probleme ili potrebe za dodatnim instrukcijama. [5]

Baza podataka također može uključivati sustave za autentifikaciju i autorizaciju, osiguravajući da samo ovlašteni korisnici imaju pristup određenim podacima ili funkcijama sustava. Dodatno, sigurnosni mehanizmi poput enkripcije i sigurnosnih kopija podataka osiguravaju da su podaci zaštićeni od neovlaštenog pristupa i gubitka.

## 1.5 Sustavi za obradu pogrešaka i iznimki

Jedan od ključnih elemenata svakog autograder sustava je mehanizam za obradu pogrešaka i iznimki. U kontekstu automatskog ocjenjivanja programskih zadataka, pogreške u studentskom kodu su neizbježne, bilo da se radi o sintaksnim greškama, iznimkama koje se pojavljuju tijekom izvođenja, ili pogrešnim logičkim rješenjima. Autograder mora biti sposoban prepoznati i adekvatno reagirati na takve situacije, osiguravajući da sustav ostane stabilan i da studenti dobiju relevantne povratne informacije.

### Sintaksne greške

Sintaksne greške su najosnovniji tip pogrešaka koje se javljaju u programima, obično zbog nepravilne uporabe jezika. Kada se pojavi sintaksna greška, Python interpreter ne može izvršiti kod, te se generira `SyntaxError`. Autograder sustavi obično koriste unaprijed definirane alate koji analiziraju kod prije njegovog izvršavanja kako bi identificirali takve pogreške. Na primjer, funkcija poput `ast.parse()` može se koristiti za parsiranje Python koda i hvatanje sintaksnih pogrešaka prije nego što se kod izvrši. [6]

U takvim slučajevima, autograder može pružiti studentima povratnu informaciju koja uključuje točnu lokaciju greške u kodu, opis greške, te savjet o tome kako ju ispraviti. Na ovaj način,

studenti dobivaju brze povratne informacije koje im pomažu da isprave svoje rješenje i predaju ispravan kod u što kraćem vremenu.

### Iznimke tijekom izvođenja

Osim sintaksnih grešaka, studenti se često susreću s iznimkama koje se javljaju tijekom izvođenja koda. Ove iznimke mogu uključivati `ZeroDivisionError`, `IndexError`, `KeyError`, i mnoge druge koje se mogu dogoditi zbog nepravilne logike ili nepredviđenih uvjeta u kodu. Autograder sustavi moraju biti dizajnirani tako da pravilno obrađuju ove iznimke, hvataju ih i pružaju korisne povratne informacije studentima.

Jedan od pristupa je korištenje `try-except` blokova unutar okvira za testiranje, koji omogućuju autograderu da nastavi s izvršavanjem ostalih testova čak i ako neki od njih generira iznimku. [6] Na taj način, sustav može dati cjelovit izvještaj o svim aspektima studentskog koda, a ne samo o prvoj grešci na koju naiđe. Ovo je posebno važno u situacijama kada studenti rade na kompleksnijim zadacima koji uključuju više međusobno ovisnih funkcija ili modula.

### Logičke pogreške

Logičke pogreške su najteže za otkrivanje i ispravljanje jer se program može uspješno izvršiti, ali ne daje očekivane rezultate. Autograderi za Python često koriste unaprijed definirane testne slučajeve koji provjeravaju ispravnost izlaza programa na različite ulazne podatke. Ako izlaz ne odgovara očekivanom rezultatu, sustav može pružiti povratne informacije koje objašnjavaju gdje je došlo do odstupanja, te eventualno sugerirati što je moglo uzrokovati pogrešku.

Osim toga, autograderi mogu koristiti tehnike kao što su analize pokrivenosti koda (eng. code coverage) kako bi identificirali dijelove koda koje studenti možda nisu dovoljno testirali. Alati poput `coverage.py` omogućuju autograderima da generiraju izvještaje koji pokazuju koji dijelovi studentskog koda nisu izvršeni tijekom testiranja, što može pomoći studentima u prepoznavanju potencijalnih problema u njihovom rješenju. [6]

## 1.6 Sigurnosni aspekti autogradera

Sigurnost je ključni aspekt svakog sustava koji se koristi u obrazovnom kontekstu, a autograderi nisu iznimka. Budući da autograderi često izvršavaju nepoznat i neprovjeren kod, postoji niz sigurnosnih izazova koje treba adresirati kako bi se osigurao integritet sustava i zaštitili podaci korisnika.

### Izolacija izvršenja koda

Jedan od najvažnijih sigurnosnih aspekata je izolacija izvršenja koda. Budući da studentski kod može sadržavati potencijalno zlonamjerne ili pogrešne upute, autograderi moraju osigurati da svaki kod bude izvršen u izoliranom okruženju, čime se sprečava bilo kakav utjecaj na sustav domaćin ili druge programe.

Za postizanje ove izolacije, često se koriste tehnologije kao što su kontejneri ili virtualne mašine, koje omogućuju izvršavanje studentskog koda u potpuno odvojenim okruženjima. Ovo osigurava da čak i ako studentski kod sadrži zlonamjerne ili pogrešne upute, one ne mogu utjecati na integritet sustava ili podataka. [2]

### Ograničavanje resursa

Drugi važan sigurnosni aspekt je ograničavanje resursa koje studentski kod može koristiti. Autograderi moraju biti dizajnirani tako da spriječe studentske programe od prekomjernog korištenja resursa, poput CPU-a, memorije ili diskovnog prostora, što bi moglo dovesti do degradacije performansi sustava ili čak do njegovog pada.

Jedan od pristupa je implementacija vremenskih ograničenja za izvršenje koda (engl. timeout), kao i ograničenja memorije (engl. memory limits). Na taj način, sustav može prekinuti izvršavanje programa koji prelazi zadane granice, te obavijestiti studenta da je njihov kod možda suboptimalan ili sadrži beskonačnu petlju. [1]

### Sigurnosne provjere koda

Dodatno, autograderi mogu uključivati sigurnosne provjere koda kako bi se identificirali potencijalni sigurnosni problemi u studentskom kodu. Na primjer, alati za statičku analizu koda mogu se koristiti za prepoznavanje potencijalno opasnih operacija, poput upotrebe neosiguranih funkcija, ili za identificiranje obrazaca koji bi mogli dovesti do sigurnosnih propusta, poput SQL injekcija ili ranjivosti na XSS napade. [1]

## 1.7 Integracija s obrazovnim platformama

Autograderi se često integriraju s obrazovnim platformama kao što su sustavi za upravljanje učenjem (eng. Learning Management Systems, LMS) ili platforme za online učenje, čime se osigurava besprijekoran tijek rada između učenja, vježbanja i ocjenjivanja. Ova integracija omogućava nastavnicima da centralizirano upravljaju svim aspektima učenja, dok studenti imaju jedno mjesto za pristup svim resursima i zadacima.

### API integracije

Jedan od najčešćih načina integracije autogradera s obrazovnim platformama je korištenje API-a (eng. Application Programming Interface). API omogućuje da autograder komunicira s LMS-om, šalje rezultate ocjenjivanja, povratne informacije, i ažurira evidenciju o studentskom napretku. Ovo smanjuje potrebu za ručnim unosom podataka i omogućava nastavnicima da se fokusiraju na pedagoške aspekte nastave.

Na primjer, platforme kao što su Moodle ili Canvas često nude API-e koji omogućuju jednostavnu integraciju s vanjskim alatima poput autogradera. Ovo omogućuje da se rezultati testova automatski unesu u ocjensku knjigu sustava, a studenti dobiju povratne informacije odmah nakon predaje zadatka.

### Prilagodba sučelja

Integracija s obrazovnim platformama često zahtijeva prilagodbu sučelja autogradera kako bi se uklopila u izgled i osjećaj LMS-a. Ovo uključuje prilagodbu tema, stilova, pa čak i funkcionalnosti kako bi se osiguralo da studenti imaju konzistentno korisničko iskustvo.

Prilagodba može također uključivati lokalizaciju sučelja na različite jezike, prilagodbu za specifične obrazovne norme ili standarde, te integraciju s drugim alatima koje obrazovne institucije koriste, poput sustava za autentifikaciju, sustava za analitiku učenja, i slično.

### Praćenje napretka i analitika

Jedna od ključnih prednosti integracije autogradera s obrazovnim platformama je mogućnost praćenja napretka studenata i analize njihovih performansi. Autograderi generiraju veliku količinu podataka tijekom ocjenjivanja studentskih zadataka, uključujući broj predaja, vremena izvršenja, postotak uspješno riješenih testova, i drugo. Kroz integraciju s analitičkim alatima, ovi podaci mogu se analizirati kako bi se identificirali obrasci u učenju studenata, područja gdje se često javljaju pogreške, i opći napredak grupe.

### Personalizacija učenja

Podaci dobiveni analizom također omogućuju personalizaciju učenja. Na primjer, ako autograder primijeti da student često griješi u određenom tipu zadataka, sustav može automatski preporučiti dodatne resurse ili vježbe koje ciljaju te specifične slabosti. Ovaj pristup omogućuje

studentima da dobivaju personalizirane smjernice koje su usmjerene na njihove individualne potrebe, što može značajno poboljšati njihovo učenje i uspjeh.

### Povratne informacije i refleksija

Autograderi mogu pružiti detaljne povratne informacije koje studentima omogućuju da reflektiraju na svoje rješenje i uče iz vlastitih pogrešaka. Kroz automatske komentare, sugestije za poboljšanje, i linkove na dodatne resurse, studenti mogu dobiti uvid u to gdje su pogriješili i kako mogu poboljšati svoje rješenje. Ovo ne samo da poboljšava kvalitetu učenja, nego također potiče studente na samostalno istraživanje i rješavanje problema.

### Praćenje akademskog integriteta

Kroz analitičke alate, autograderi mogu pomoći u praćenju akademskog integriteta. Na primjer, usporedbom predanih rješenja među studentima, sustav može identificirati potencijalne slučajeve plagijata ili prepisivanja. Ove informacije mogu se proslijediti nastavnicima, koji mogu dodatno istražiti situaciju i poduzeti odgovarajuće mjere.

Ovaj sustav ne samo da pomaže u očuvanju integriteta ocjenjivanja, nego također educira studente o važnosti originalnog rada i poštenja u akademskom kontekstu.

# 2 Primjeri autogradera u praksi

Korištenje autogradera postaje sve raširenije u obrazovnim institucijama diljem svijeta. Postoji mnogo primjera autogradera koji su uspješno implementirani i koriste se u raznim kontekstima, od osnovnih tečajeva programiranja do naprednih kolegija o računalnim znanostima.

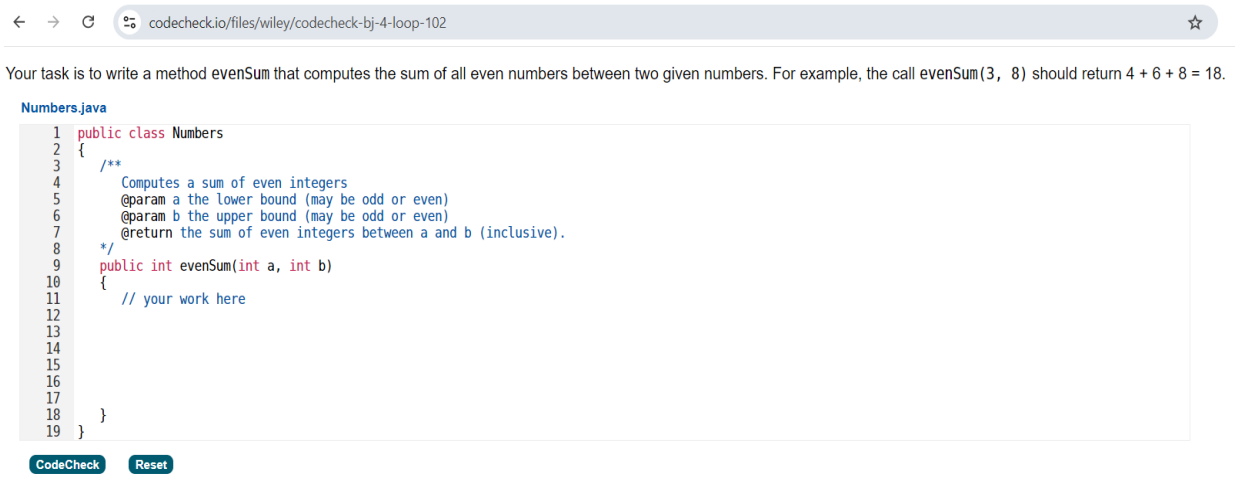
## 2.1 Autograderi u MOOC platformama

Masovni otvoreni online tečajevi (eng. Massive Open Online Courses, MOOC) su jedan od najčešćih primjera gdje se autograderi koriste. Platforme kao što su Coursera, edX, i Udacity koriste autogradere za automatsko ocjenjivanje zadataka milijuna studenata diljem svijeta. Ovo omogućava skaliranje obrazovanja na globalnoj razini, gdje bi ručno ocjenjivanje bilo jednostavno neizvedivo. [7]


*Slika 1 Autograderi u MOOC platformama*

## 2.2 Korištenje u visokim obrazovnim institucijama

Mnoge visokoobrazovne institucije, poput MIT-a, Stanforda, i Sveučilišta u Zagrebu, koriste autogradere za ocjenjivanje studentskih zadataka u kolegijima vezanim uz programiranje. Autograderi kao što su Web-CAT, CloudCoder, CodeWorkout i CodeCheck se koriste za različite razine zadataka, od osnovnih vježbi programiranja do kompleksnih projekata koji uključuju više modula i baza podataka. [8]




*Slika 2 Sučelje autogradera CodeCheck [9]*

## 2.3 Prilagodba za različite jezike

Iako su autograderi popularni za ocjenjivanje zadataka u Pythonu, postoje verzije za mnoge druge programske jezike, uključujući Javu, C++ kao na primjer autograderi CloudCoder i Codeheck. Prilagodba autogradera za različite jezike zahtijeva razumijevanje specifičnih značajki i izazova svakog jezika, kao i dizajn testova koji će obuhvatiti sve aspekte jezika koji se ocjenjuje. [10]

## 2.4 Primjeri u srednjoškolskom obrazovanju

Autograderi se također sve više koriste u srednjoškolskom obrazovanju, gdje se programiranje uvodi kao dio kurikuluma. Jedan od primjera je CS50, autograder razvijen za srednjoškolske tečajeve programiranja koji omogućuje učiteljima da automatski ocjenjuju jednostavne Python zadatke i pružaju brze povratne informacije učenicima. [11]


*Slika 3 Sučelje autogradera CS50*

## 2.5 Izazovi i budućnost autogradera

Iako autograderi donose mnoge prednosti, njihova implementacija i korištenje također donose niz izazova. Neki od tih izazova uključuju tehnička ograničenja, problem skalabilnosti, kao i etička pitanja povezana s automatiziranim ocjenjivanjem.

### Tehnička ograničenja

Jedno od glavnih tehničkih ograničenja autogradera je njihova sposobnost da ocijene kreativnost i kompleksnost rješenja. Iako autograderi mogu učinkovito ocijeniti ispravnost koda, teško im je prepoznati inovativna rješenja ili ocijeniti kod koji je funkcionalan, ali netipičan. Ovo zahtijeva daljnji razvoj algoritama koji bi mogli analizirati kod na dubljoj razini i razumjeti namjeru iza njega. [12]

### Skalabilnost i resursi

Autograderi moraju biti skalabilni kako bi podržali veliki broj istovremenih korisnika, posebno na platformama poput MOOC-ova. Održavanje performansi sustava dok se istovremeno ocjenjuje tisuće zadataka može biti izazovno, posebno ako sustav nema dovoljno resursa ili nije adekvatno optimiziran. [7]

### Etička pitanja

Automatsko ocjenjivanje također otvara pitanje etike. Na primjer, postoje zabrinutosti oko toga kako automatizirano ocjenjivanje može utjecati na studentsku motivaciju ili kako se nositi s greškama koje sustav može napraviti. Također, postoji pitanje transparentnosti u ocjenjivanju - studenti trebaju razumjeti kako su njihove ocjene generirane i trebaju imati mogućnost žalbe na ocjenu ako smatraju da je neadekvatna.

### Budući razvoj

Budućnost autogradera uključuje daljnji razvoj u smjeru umjetne inteligencije (AI) i strojskog učenja. Korištenje AI-a moglo bi omogućiti autograderima da prepoznaju i ocijene kompleksnije zadatke, uključujući analizu stilskog aspekta koda, optimizaciju performansi, i ocjenjivanje kreativnosti. Također, AI bi mogao omogućiti personalizaciju povratnih informacija na temelju individualnog napretka studenata. [12]

Kroz daljnji razvoj i integraciju, autograderi će igrati sve značajniju ulogu u obrazovanju, omogućujući efikasnije, personaliziranije, i inkluzivnije učenje.

## 2.6 Prednosti i nedostatci autogradera

Već su spomenute neke prednosti i nedostaci autogradera u prethodnim dijelovima, no može biti korisno dodati poseban odjeljak koji sistematično sažima sve prednosti i nedostatke.

### Prednosti autogradera

- Automatizacija procesa ocjenjivanja: Autograderi omogućuju brzo i efikasno ocjenjivanje velikog broja zadataka, što smanjuje potrebu za ručnim radom i omogućava nastavnicima da se više fokusiraju na druge aspekte poučavanja.
- Objektivnost ocjenjivanja: Budući da autograderi slijede precizno definirane kriterije, osiguravaju objektivno ocjenjivanje bez subjektivnih čimbenika koje bi moglo unijeti ljudsko ocjenjivanje.
- Brze povratne informacije: Studenti dobivaju povratne informacije odmah nakon predaje zadatka, što im omogućuje brzu korekciju pogrešaka i učenje na temelju vlastitih grešaka.

- Poboljšana skalabilnost: Autograderi omogućuju skaliranje obrazovnih aktivnosti na veliki broj studenata, što je posebno korisno u kontekstu online obrazovanja i MOOC platformi.
- Smanjenje opterećenja nastavnika: Automatsko ocjenjivanje zadataka značajno smanjuje opterećenje nastavnika, omogućujući im da se usmjere na interakciju sa studentima i rješavanje kompleksnijih problema.
- Praćenje napretka i personalizacija: Autograderi omogućuju praćenje napretka svakog studenta i pružaju personalizirane povratne informacije na temelju individualnih potreba, što može značajno poboljšati proces učenja. [13]

### Nedostatci autogradera

- Ograničena sposobnost prepoznavanja kreativnosti: Iako su autograderi izvrsni u ocjenjivanju ispravnosti koda, često imaju problema s prepoznavanjem inovativnih ili netradicionalnih rješenja.
- Moguće tehničke pogreške: Kao i svaki softverski sustav, autograderi mogu imati bugove ili tehničke greške koje mogu dovesti do netočnog ocjenjivanja.
- Potencijal za površno učenje: Ako se studenti previše oslanjaju na autogradere, postoji rizik da će se fokusirati samo na prolazak testova, a ne na dublje razumijevanje problema i koncepata.
- Izazovi u ocjenjivanju kompleksnih zadataka: Kompleksni zadaci koji zahtijevaju višestruke korake ili kreativna rješenja mogu biti teže za automatizirano ocjenjivanje, što može zahtijevati dodatni rad nastavnika.
- Etički izazovi: Automatizirano ocjenjivanje može izazvati etičke dileme, posebno u vezi s transparentnošću ocjenjivanja i mogućnošću žalbe na ocjenu.
- Ovisnost o tehničkoj infrastrukturi: Korištenje autogradera zahtijeva odgovarajuću tehničku infrastrukturu, uključujući moćne poslužitelje i stabilnu mrežnu povezanost, što može predstavljati izazov u nekim okruženjima. [13]

| Autograder | Kontekst korištenja | Programski jezici | Vrsta ocjenjivanja | Cijena | Dodatne značajke |
|---|---|---|---|---|---|
| **Coursera Autograder** | MOOC platforme | Python, Java, C++, JavaScript | Automatsko ocjenjivanje zadataka | Besplatno uz ograničenja, plaćene opcije | Skalabilnost, široka primjena |
| **edX Autograder** | MOOC platforme | Python, Java, C++, JavaScript | Automatsko ocjenjivanje zadataka | Besplatno uz ograničenja, plaćene opcije | Brze povratne informacije |
| **Udacity Autograder** | MOOC platforme | Python, Java, C++, JavaScript | Automatsko ocjenjivanje zadataka | Plaćeni certifikati | Personalizirane povratne informacije |
| **Web-CAT** | Visoko obrazovanje | Java, C++, Python | Ocjenjivanje testova jedinica | Open-source | Detaljna analiza koda |
| **CloudCoder** | Visoko obrazovanje | Python, Java, C++ | Ocjenjivanje programskih isječaka | Open-source | Jednostavno sučelje za nastavnike |
| **CodeWorkout** | Visoko obrazovanje | Python, Java, C++ | Automatsko ocjenjivanje zadataka | Besplatno | Interaktivne vježbe i zadaci |
| **CS50 IDE** | Srednjoškolsko obrazovanje | Python, C, SQL, HTML/CSS, JavaScript | Automatsko ocjenjivanje zadataka | Besplatno | Prilagođeno za interaktivno učenje, bogata okruženja |

*Tablica 1 Usporedba različitih autogradera s glavnim karakteristikama*

# 3 Python: Jezik po izboru za autogradere

Python je jedan od najrasprostranjenijih programskih jezika u modernom softverskom razvoju. Njegova jednostavna sintaksa, velika zajednica korisnika i širok raspon primjena čine ga idealnim za različite vrste zadataka, uključujući izradu autogradera. U ovom poglavlju analizirat ćemo zašto je Python toliko popularan, zašto je dobar za početnike, te njegove prednosti i nedostatke u usporedbi s drugim jezicima.

Python je razvio nizozemski programer Guido van Rossum 1991. godine. Njegova vizija bila je stvoriti jezik koji je jednostavan za korištenje, s naglaskom na čitljivost koda i fleksibilnost. Ime "Python" potječe iz njegove ljubavi prema britanskoj humorističnoj seriji "Monty Python's Flying Circus", a ne iz asocijacije na zmiju. Od svog početka, Python je brzo stekao popularnost u akademskim i istraživačkim krugovima zbog svoje jednostavnosti, a ubrzo je postao omiljeni jezik u industriji zbog svoje svestranosti i širokog spektra primjena. [14]

Python je postao osobito popularan krajem 2000-ih s porastom zanimanja za znanost o podacima i strojnim učenjem, gdje je njegova bogata biblioteka alata omogućila brz i efikasan rad. Danas je Python jezik izbora za mnoge aplikacije, od web razvoja do znanosti o podacima, automatizacije i obrazovnih sustava. [15]

Python se ističe kao jedan od najjednostavnijih programskih jezika za početnike, zbog svoje čitljivosti i sintakse koja je slična prirodnom jeziku. Za razliku od jezika poput C++ ili Jave, Python ne zahtijeva složene strukture i deklaracije koje mogu zbuniti nove programere.

Evo nekoliko razloga zašto je Python dobar izbor za one koji tek počinju učiti programiranje:

- Jednostavna sintaksa: Pythonov kod je lak za razumjeti i pisati jer koristi manje znakova i složenih izraza. Na primjer, za ispisivanje jednostavne poruke u Pythonu koristi se samo jedan redak. Za usporedbu, u nekim drugim jezicima poput Jave ili C++, potrebno je više linija koda samo za istu funkcionalnost.
- Dinamičko tipiziranje: Python ne zahtijeva da unaprijed definiramo tip varijabli (npr. int, float, string), već se tipovi automatski dodjeljuju prilikom izvršavanja. To olakšava brzo testiranje i eksperimentiranje s kodom.
- Velika zajednica i podrška: Budući da je Python tako popularan, postoji ogromna količina besplatnih resursa, tutorijala, vodiča i foruma na kojima početnici mogu postavljati pitanja i dobiti pomoć. To čini učenje bržim i manje stresnim.

- Interaktivno okruženje: Pythonov interaktivni interpreter omogućuje korisnicima da odmah testiraju kod i vide rezultate. Ovaj način rada, poznat kao REPL (Read-Eval-Print Loop), omogućuje brzu iteraciju i eksperimentiranje s kodom, što je iznimno korisno za one koji uče programiranje.
- Široka upotreba u obrazovanju: Python je često jezik izbora u školama i na fakultetima jer omogućuje studentima da se fokusiraju na osnovne koncepte programiranja bez prevelikog opterećenja složenom sintaksom. Ovo je ključna prednost jer pomaže studentima da brže nauče osnove algoritama, struktura podataka i problem-solving tehnika. [16]

### Prednosti Pythona

Osim što je jednostavan za početnike, Python ima brojne prednosti koje ga čine odličnim izborom za profesionalni razvoj, uključujući izradu autogradera:

- Čitljivost koda: Pythonova sintaksa ohrabruje pisanje koda koji je lako razumljiv. Razvijatelji mogu brzo shvatiti što kod radi, čak i ako ga nisu osobno pisali. Ova značajka je osobito korisna u timovima gdje više ljudi radi na istom projektu, kao i kod dugoročnog održavanja softvera.
- Velika standardna knjižnica: Python dolazi s bogatom standardnom knjižnicom koja pokriva širok raspon funkcionalnosti – od manipulacije datotekama do mrežnog programiranja. Ovo smanjuje potrebu za dodatnim alatima i omogućuje brži razvoj aplikacija. [17]
- Podrška za više paradigmi programiranja: Python podržava različite paradigme programiranja, uključujući proceduralno, objektno-orijentirano i funkcionalno programiranje. Ova fleksibilnost omogućuje programerima da koriste stil koji najbolje odgovara specifičnom problemu.
- Svestranost: Python je izuzetno svestran jezik, pogodan za širok raspon primjena. Korišten je u razvoju web aplikacija (Flask, Django), znanosti o podacima (NumPy, Pandas), strojnog učenja (TensorFlow, PyTorch), automatizacije sustava i mnogih drugih područja. Upravo zbog svoje svestranosti, Python je često prvi izbor za brzi razvoj prototipa.
- Integracija s drugim jezicima i tehnologijama: Python se lako integrira s kodom napisanim u drugim jezicima kao što su C, C++ ili Java, zahvaljujući alatima poput

ctypes, Cython ili Jython. Ova mogućnost omogućuje da Python koristi snagu drugih jezika u područjima gdje oni pružaju bolje performanse. [15]

### Nedostaci Pythona

Iako Python ima mnoge prednosti, postoje i određeni nedostaci koji mogu ograničiti njegovu upotrebu u nekim situacijama:

- Brzina izvršavanja: Python je interpretirani jezik, što znači da je sporiji od kompajliranih jezika poput C ili C++. U aplikacijama gdje je brzina kritična, Python možda nije najbolji izbor. Međutim, ovaj nedostatak se može ublažiti korištenjem ekstenzija poput Cython-a ili prepisivanjem ključnih dijelova aplikacije u jeziku s boljim performansama.
- Potrošnja memorije: Zbog dinamičkog tipiziranja i fleksibilnosti, Python može trošiti više memorije u usporedbi s drugim jezicima. Ovo može biti problem kod aplikacija koje zahtijevaju visoku učinkovitost u smislu memorijskih resursa.
- Ograničenja u mobilnom razvoju: Iako postoje alati poput Kivy-ja koji omogućuju razvoj mobilnih aplikacija u Pythonu, većina mobilnih aplikacija razvija se koristeći specijalizirane jezike poput Swift-a za iOS ili Java/Kotlin-a za Android. Python jednostavno nije prvi izbor za razvoj mobilnih aplikacija.
- Global Interpreter Lock (GIL): Python koristi GIL koji ograničava istovremeno izvršavanje više thread-ova unutar jednog procesa. To može biti ograničavajuće za aplikacije koje zahtijevaju intenzivnu paralelnu obradu. Iako postoje rješenja, poput multiprocessinga, GIL je i dalje tema mnogih rasprava među Python zajednicom. [17]

Ukratko, Python je zbog svoje jednostavne sintakse, fleksibilnosti i bogate zajednice postao jedan od najpopularnijih programskih jezika u svijetu. Njegova svestranost i prilagodljivost čine ga savršenim alatom za početnike, ali i za profesionalce koji rade na kompleksnim projektima. Iako ima nekoliko nedostataka, kao što su sporost i veća potrošnja memorije, prednosti koje donosi, osobito u kontekstu obrazovanja i automatizacije, nadmašuju te nedostatke. U kontekstu razvoja autogradera, Python je idealan izbor zbog svojih ugrađenih alata i podrške za različite biblioteke koje olakšavaju izradu automatiziranih sustava za ocjenjivanje. Njegova jednostavnost omogućava brzi razvoj testnih skripti, a bogata standardna biblioteka pomaže u pisanju funkcija za provjeru koda, ispitivanje različitih tipova zadataka i interakciju s drugim sustavima. [1]

Python je, dakle, ne samo idealan jezik za početnike, već i moćan alat za napredne zadatke, poput izrade autograderskih sustava, što ga čini ključnim elementom u modernom obrazovanju i industriji.

# 4 Oblikovanje autogradera za Python

## 4.1 Odabir tehnologija za razvoj aplikacije

Flask je jedan od najpopularnijih mikroframeworka za Python, koji omogućuje razvoj web aplikacija uz minimalan broj dodatnih komponenti. Kao "micro" framework, Flask ne dolazi s unaprijed definiranim alatima poput ORM-a (eng. Object Relational Mapping) ili autentifikacijskog sustava, ali pruža osnovne alate za stvaranje funkcionalne web aplikacije.

Glavna filozofija Flask-a je jednostavnost i modularnost, što znači da pruža samo osnovne funkcionalnosti koje su potrebne za izgradnju web aplikacija, dok napredne funkcionalnosti mogu biti dodane po potrebi. Ovaj fleksibilan pristup omogućuje programerima da izgrade upravo ono što im treba, bez nepotrebnog bremena složenih značajki koje dolaze s drugim frameworkovima, poput Djangoa.

Flask je prvi put razvijen 2010. godine, a njegov autor Armin Ronacher osmislio ga je kao odgovor na potrebu za laganim i fleksibilnim frameworkom za Python. Danas Flask koristi široka zajednica programera i dobro je podržan, s brojnim ekstenzijama koje omogućuju dodavanje novih funkcionalnosti, poput autentifikacije, upravljanja bazama podataka i slanja e-pošte. [18]

Za razvoj autogradera Flask je odabran zbog nekoliko ključnih razloga koji ga čine prikladnim za ovu vrstu projekta.

- Jednostavnost i brzina razvoja: Flask omogućuje brzo postavljanje i pokretanje osnovne aplikacije, što je bilo ključno za ovaj projekt. Za razliku od većih frameworkova koji zahtijevaju više vremena za konfiguraciju i postavljanje, Flask omogućavabrzu izradu i testiranje prototipa autogradera.
- Minimalizam i modularnost: Flask dolazi s minimalnim brojem unaprijed ugrađenih komponenti, što daje slobodu integriranja samo onih alata i funkcionalnosti koje su potrebne za neki projekt. To je posebno važno kod izrade aplikacija poput autogradera,

gdje je cilj fokusirati se na osnovne funkcije poput uploadanja zadataka i pokretanja testova.

- Fleksibilnost u strukturi aplikacije: Jedna od najvažnijih značajki Flask-a je ta da ne nameće rigidnu strukturu projekta. To omogućuje izradu aplikacije na način koji najviše odgovara svakom korisniku, uz mogućnost da se lako prošire funkcionalnosti prema potrebi.
- Integracija s alatima za testiranje: Za autograder je ključno automatsko pokretanje testova nad studentskim kodom. Flask omogućava jednostavnu integraciju s Python alatima za testiranje, poput unittest ili pytest, što omogućuje automatsko pokretanje testova i vraćanje rezultata korisnicima.Dokumentacija i zajednica: Flask je iznimno dobro dokumentiran, a velika zajednica korisnika pruža podršku kroz različite online forume i resurse. Iako je Flask idealan izbor za ovaj projekt, kao i svaki alat, ima svoje prednosti i nedostatke.

### Prednosti:

- Jednostavnost i lakoća korištenja: Flask je poznat po svojoj jednostavnosti, što omogućava brz razvoj prototipa i lagano uvođenje promjena.
- Fleksibilnost i proširivost: Flask ne dolazi s unaprijed definiranim komponentama poput ORM-a, autentifikacije ili administracijskog sučelja, ali zato pruža fleksibilnost da se te komponente integriraju prema potrebi, što omogućava skalabilnost aplikacije.
- Modularna arhitektura: Flask je idealan za aplikacije poput autogradera jer omogućava modularan razvoj, gdje se svaka funkcionalnost može dodati zasebno, a sam kod ostaje čist i razumljiv.
- Velika podrška za ekstenzije: Flask ima bogat ekosustav ekstenzija koje olakšavaju dodavanje dodatnih funkcionalnosti bez potrebe za ponovnim izumljivanjem kotača. Ekstenzije poput Flask-SQLAlchemy za rad s bazama podataka ili Flask-Login za autentifikaciju korisnika mogu se lako dodati aplikaciji.

### Nedostaci:

- Manjak ugrađenih značajki: Za razliku od drugih frameworkova kao što je Django, Flask ne dolazi s ugrađenim alatima za upravljanje bazama podataka, autentifikaciju korisnika ili administracijsko sučelje, što može povećati složenost kod složenijih aplikacija.

- Zahtjev za ručnim postavljanjem strukture aplikacije: Iako je fleksibilnost prednost, za složenije projekte ručno postavljanje strukture može biti izazov. Flask ne nameće unaprijed definiranu strukturu, što može dovesti do toga da programeri moraju više pažnje posvetiti organizaciji koda.
- Nedostatak ugrađene sigurnosti: Flask nema ugrađene sigurnosne značajke poput zaštite od CSRF napada, što znači da programeri moraju dodatno implementirati ove mjere.

### 4.1.1 Instalacija i korištenje Flask-a

U ovom dijelu rada prikazat će se proces instalacije „Flask-a“ te razvoj jednostavne aplikacije pomoću ovog frameworka. Flask je mikroframework pisan u Pythonu, koji je zbog svoje jednostavnosti i fleksibilnosti postao popularan alat za razvoj web aplikacija. Budući da se koristi za stvaranje laganih i brzih aplikacija, posebno je koristan za edukativne svrhe, poput autogradera za ocjenjivanje studentskih zadataka. Prikazat će se detaljni koraci za instalaciju i pokretanje Flask aplikacije na različitim operacijskim sustavima, uz dodatno objašnjenje svake faze instalacije i ključnih alata potrebnih za rad s Flask-om.

Flask je kompatibilan s različitim operacijskim sustavima, uključujući „macOS“, „Windows“ i „Linux“. Instalacija Flask-a podrazumijeva nekoliko koraka, ali postupak je relativno jednostavan zahvaljujući Pythonovom paket menadžeru „pip“ koji omogućava brzo preuzimanje i instalaciju potrebnih biblioteka. U nastavku su detaljno opisani koraci za instalaciju Flask-a.

#### Provjera instalacije Pythona:

Prije nego što se Flask može instalirati, potrebno je osigurati da je Python ispravno instaliran na računalu. Flask se oslanja na Python, pa se mora koristiti kompatibilna verzija. Preporučuje se Python verzija 3.7 ili novija zbog bolje podrške i sigurnosnih ažuriranja.

Primjer instalacije na „Windows“ uređajima koristeći „Command Prompt“:


*Slika 1 Provjera instalacije Pythona*

Ako Python nije instaliran, može se preuzeti sa službene stranice Python.org.

Na „macOS-u“ i „Linux-u“, Python bi trebao biti unaprijed instaliran, ali preporučuje se ažuriranje na najnoviju verziju. Provjeru verzije možete obaviti sličnom naredbom:

```
python3 –version
```

### Python paket menadžera (pip):

„pip“ je alat za upravljanje paketima koji je službeni paket menadžer za Python. Omogućuje jednostavno preuzimanje, instaliranje, nadogradnju i uklanjanje Python paketa i modula, koji su dostupni u PyPI (Python Package Index) repozitoriju. Python dolazi s mnoštvom ugrađenih modula, ali često je potrebno instalirati dodatne pakete iz PyPI-a za specifične projekte, a pip olakšava taj proces.

Prednosti Python paket menadžera (pip) su :

- Paket menadžer: pip je osnovni alat za instalaciju i upravljanje Python bibliotekama, neophodan za većinu Python projekata.
- Instalacija paketa: Pomoću pip-a možete jednostavno instalirati tisuće paketa dostupnih na PyPI-u jednim naredbenim redom.
- Jednostavno održavanje projekata: pip omogućava lakše održavanje koda jer možete specificirati sve ovisnosti projekta u datotekama poput requirements.txt, koje se mogu automatski instalirati na drugim računalima pomoću pip-a. [19]

„pip“ dolazi unaprijed instaliran uz Python verziju 3.4 i novije. Ako nije dostupan, može se instalirati.

### Instalacija Flask-a

Nakon što je Python uspješno instaliran i ažuriran, slijedi postavljanje Flask-a, popularnog mikroračunarskog web okvira. Flask omogućuje izradu jednostavnih, ali moćnih web

aplikacija, a posebno je koristan za projekte poput autograderskih sustava zbog svoje fleksibilnosti i lakoće upotrebe. Instalacija Flask-a provodi se putem alata za upravljanje paketima, pip, koji dolazi unaprijed instaliran s modernim verzijama Pythona. Međutim, za projekte koji uključuju različite Python biblioteke, preporučuje se kreiranje zasebnog virtualnog okruženja kako bi se spriječili konflikti između različitih verzija paketa.

## Virtualno okruženje

Virtualno okruženje (engl. virtual environment) izolira Python projekt od globalne instalacije Python biblioteka. Ova tehnika omogućuje rad s različitim verzijama istih paketa u različitim projektima bez straha da će jedan projekt utjecati na drugi. U kontekstu Flask aplikacija, virtualno okruženje osigurava da Flask i sve povezane biblioteke budu instalirane samo za taj projekt, što doprinosi boljoj organizaciji i izbjegavanju problema s verzijama prilikom instaliranja novih alata ili paketa u budućnosti. [20]

Za stvaranje virtualnog okruženja koristi se venv, Pythonov alat koji je integriran u samu Python instalaciju. Prvi korak kod kreiranja virtualnog okruženje je otvaranje Command Prompt. Na Windows sustavu, klikne se na Start i upiše se "cmd", nakon čega se pritisne Enter. Pojavit će se prozor naredbenog retka (Command Prompt). [21] Nadalje, navigira se do direktorija željenog projekta. U naredbenom retku, pomoću naredbe cd, treba se doći do direktorija gdje se želi pohraniti projekt. Na primjer:

```
PS C:\Users\emirh> cd C:\Users\emirh\OneDrive\Masaüstü\moj.diplomski
PS C:\Users\emirh\OneDrive\Masaüstü\moj.diplomski>
```

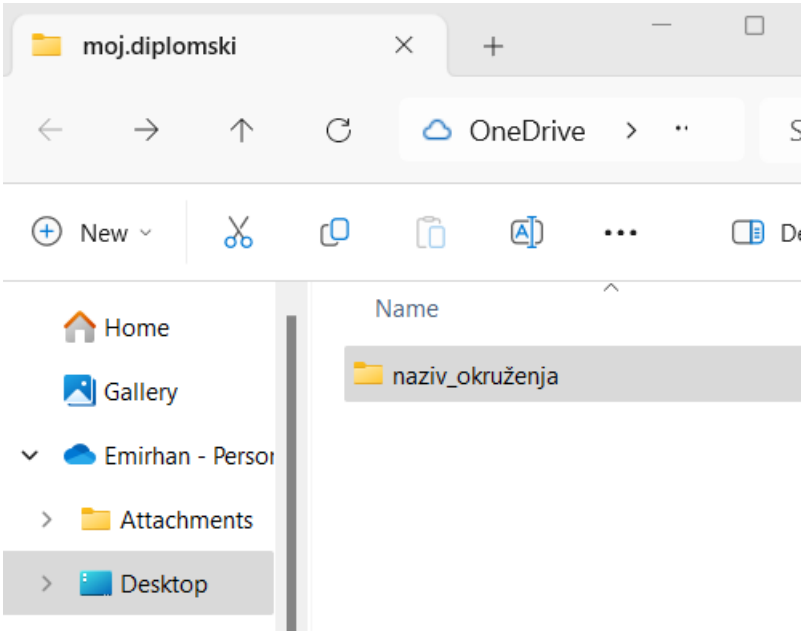
*Slika 4 Navigiranje do direktorija projekta*

Potrebno je unijeti sljedeću naredbu za kreiranje virtualnog okruženja:

```
PS C:\Users\emirh\OneDrive\Masaüstü\moj.diplomski> python -m venv naziv_okruženja
```

*Slika.3 Kreiranje virtualnog okruženja*

Tom će naredbom biti kreiran novi direktorij pod imenom 'naziv_okruženja' (koji se može imenovati po želji), gdje će biti pohranjene sve potrebne datoteke za virtualno okruženje.




*Slika 4 Naziv okruženja „naziv_okruženja“ unutar mape „moj.diplomski“*

Nakon kreiranja, potrebno je aktivirati virtualno okruženje. To se na Windows sustavu radi sljedećom naredbom: `pip install Flask`.

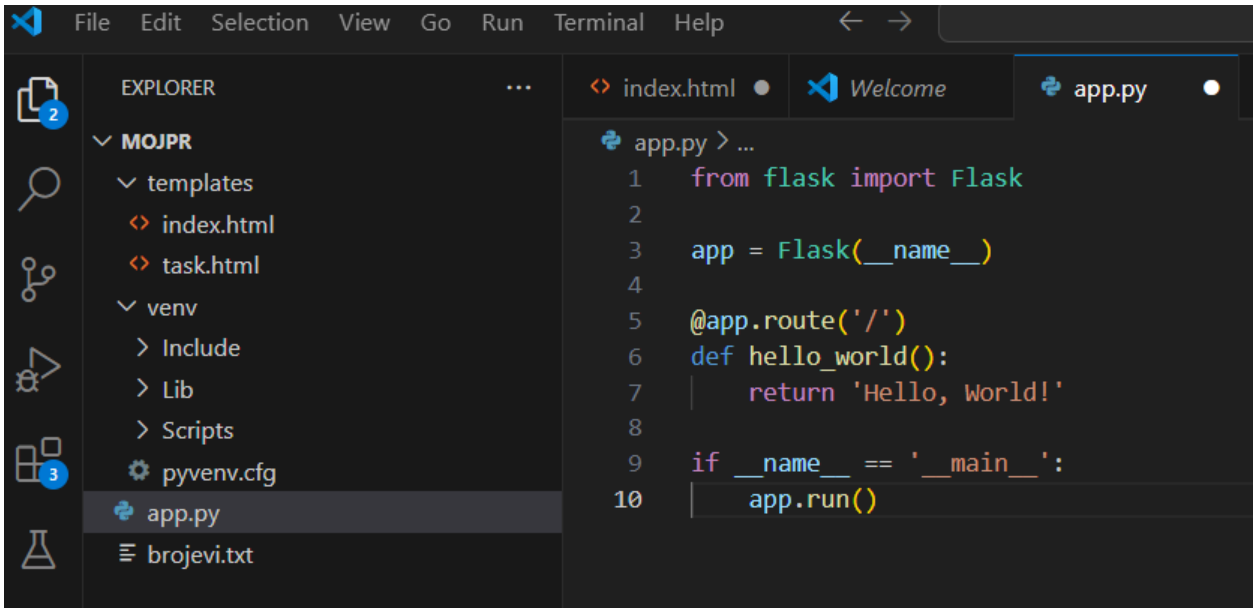
Nakon unosa naredbe `pip install Flask` u aktiviranom virtualnom okruženju, u Command Promptu bi se treb ali prikazati rezultati koji potvrđuju da je Flask uspješno instaliran. Tipičan izlaz mogao bi izgledati ovako:

```
PS C:\Users\emirh\OneDrive\Masaüstü\moj.diplomski> pip install Flask
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: Flask in c:\users\emirh\appdata\roaming\python\pyt
hon311\site-packages (3.0.3)
Requirement already satisfied: Werkzeug>=3.0.0 in c:\users\emirh\appdata\roaming\
python\python311\site-packages (from Flask) (3.0.3)
Requirement already satisfied: Jinja2>=3.1.2 in c:\users\emirh\appdata\roaming\py
thon\python311\site-packages (from Flask) (3.1.4)
Requirement already satisfied: itsdangerous>=2.1.2 in c:\users\emirh\appdata\roam
ing\python\python311\site-packages (from Flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in c:\users\emirh\appdata\roaming\pyt
hon\python311\site-packages (from Flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in c:\users\emirh\appdata\roaming\p
ython\python311\site-packages (from Flask) (1.8.2)
Requirement already satisfied: colorama in c:\users\emirh\appdata\roaming\python\
python311\site-packages (from click>=8.1.3->Flask) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\emirh\appdata\roaming\
python\python311\site-packages (from Jinja2>=3.1.2->Flask) (2.1.5)
```

*Slika 5 Izlaz naredbe "pip install Flask"*

Ovaj izlaz potvrđuje da su svi potrebni paketi i Flask instalirani u virtualnom okruženju.

Nakon instalacije Flask-a, preporučljivo je provjeriti je li proces instalacije uspješno dovršen. Ova provjera može se obaviti pokretanjem jednostavne Flask aplikacije. Prvo, potrebno je kreirati novu Python datoteku koja će sadržavati osnovni kod za Flask aplikaciju. Kao primjer, može se kreirati datoteka s nazivom app.py. U tu datoteku treba dodati sljedeći kod:




*Slika 6 Provjera instalacije Flask-a*

Ovaj kod predstavlja minimalnu Flask aplikaciju koja definira rutu ('/') i vraća jednostavan tekst "Hello, World!" kada se aplikacija pokrene. [22]

Nakon što je datoteka spremljena, aplikacija se pokreće putem terminala ili naredbenog retka pomoću naradbe: `python app.py`

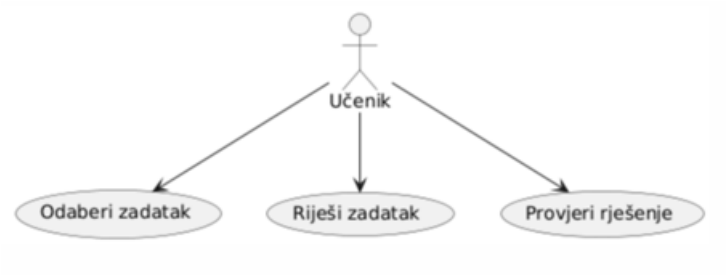
U slučaju da je Flask ispravno instaliran i virtualno okruženje aktivno, pojavit će se poruka koja obavještava da je aplikacija pokrenuta na lokalnom poslužitelju. Tipično će poslužitelj biti dostupan na adresi: http://127.0.0.1:5000

Otvaranjem ovog URL-a u web-pregledniku, prikazat će se poruka "Hello, World!", što je pokazatelj da aplikacija funkcionira i da je instalacija Flask-a bila uspješna. Ovaj proces ne samo da potvrđuje ispravnost instalacije, već omogućuje i provjeru je li sustav pravilno konfiguriran za razvoj aplikacija u Flask-u.

## 4.2 Modeliranje zahtjeva aplikacije

Prije početka izrade aplikacije, potrebno je definirati funkcionalnosti aplikacije. U ovom projektu aplikacija pruža osnovne funkcionalnosti za studente. Studenti imaju mogućnost odabira zadataka iz liste dostupnih zadataka koje sustav nudi. Svaki zadatak uključuje opis i definirani kod koji studenti trebaju dopuniti. Studenti unose vlastiti kod te ga zatim mogu poslati na provjeru kako bi testirali ispravnost svog rješenja putem aplikacije. Aplikacija automatski evaluira rješenje i vraća povratne informacije o eventualnim greškama ili uspješnom rješenju.

Ove funkcionalnosti omogućuju jednostavno rješavanje zadataka i dobivanje automatskih povratnih informacija, čime se potiče aktivno učenje programiranja. Funkcionalnosti aplikacije prikazat će se dijagramom slučajeva korištenja koji je kreiran prije samog početka izrade aplikacije. Na dijagramu će biti jasno prikazane sve funkcionalnosti aplikacije i način na koji korisnici, odnosno studenti, komuniciraju sa sustavom.




*Slika 70 Dijagram slučajeva korištenja*

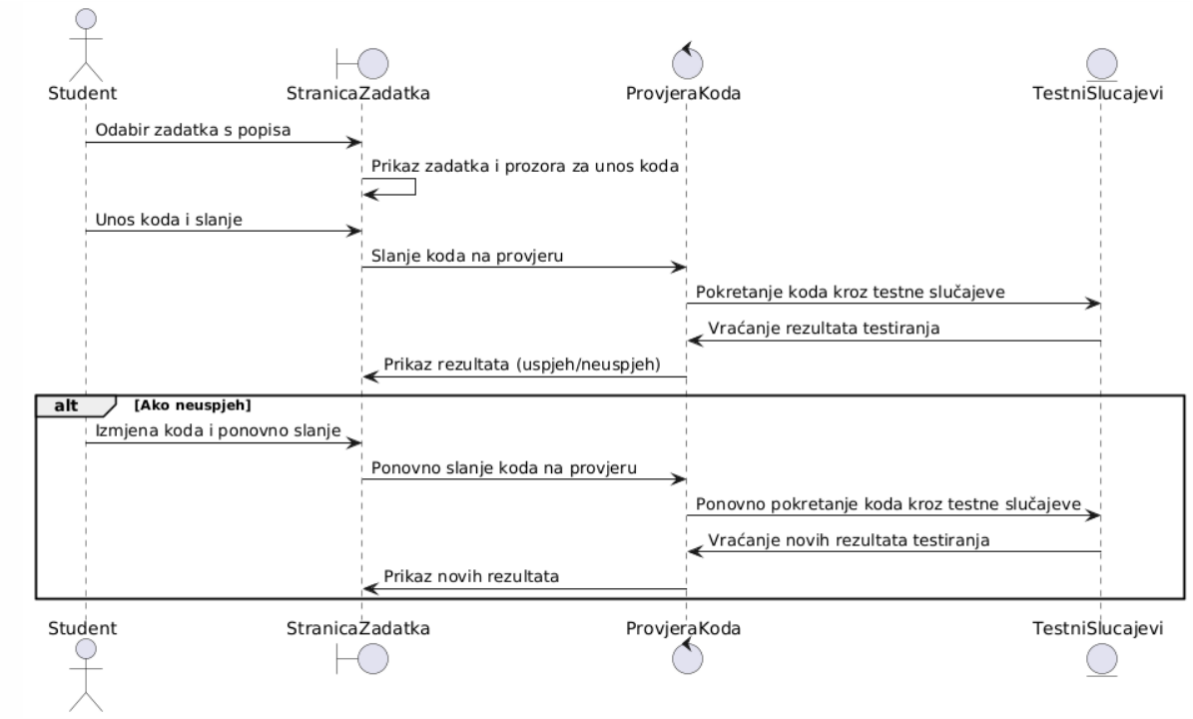
## 4.3 Modeliranje tijeka aplikacije

Dijagram slijeda opisuje ključni tijek interakcije studenta s aplikacijom prilikom rješavanja zadatka unutar autogradera. Proces započinje kada student odabere jedan od 35 dostupnih zadataka. Nakon odabira, stranica zadatka prikazuje opis zadatka te prozor za unos koda u kojem student može unijeti svoje rješenje.

Kada student završi s unosom koda, šalje ga na provjeru pritiskom na gumb za slanje. Stranica zadatka tada prenosi uneseni kod komponenti zaduženoj za provjeru rješenja. Ta komponenta pokreće studentski kod kroz definirane testne slučajeve. Testni slučajevi izvršavaju kod te vraćaju rezultate o tome je li rješenje ispravno.

Nakon što se provjeri ispravnost rješenja, komponenta za provjeru vraća rezultate stranici zadatka. Rezultati se prikazuju studentu, pokazujući je li rješenje prošlo ili nije. Ako rješenje nije ispravno, student ima mogućnost izmijeniti kod i ponovno ga poslati na provjeru. Taj se postupak može ponavljati neograničeno, sve dok student ne pronađe ispravno rješenje.

Cijeli proces jasno ilustrira kako sustav postupa s unosima koda, provjerava rješenja te vraća povratne informacije studentima, omogućujući im da kontinuirano rade na svom rješenju dok ne postignu točan rezultat.




*Slika 8 Dijagram slijeda*

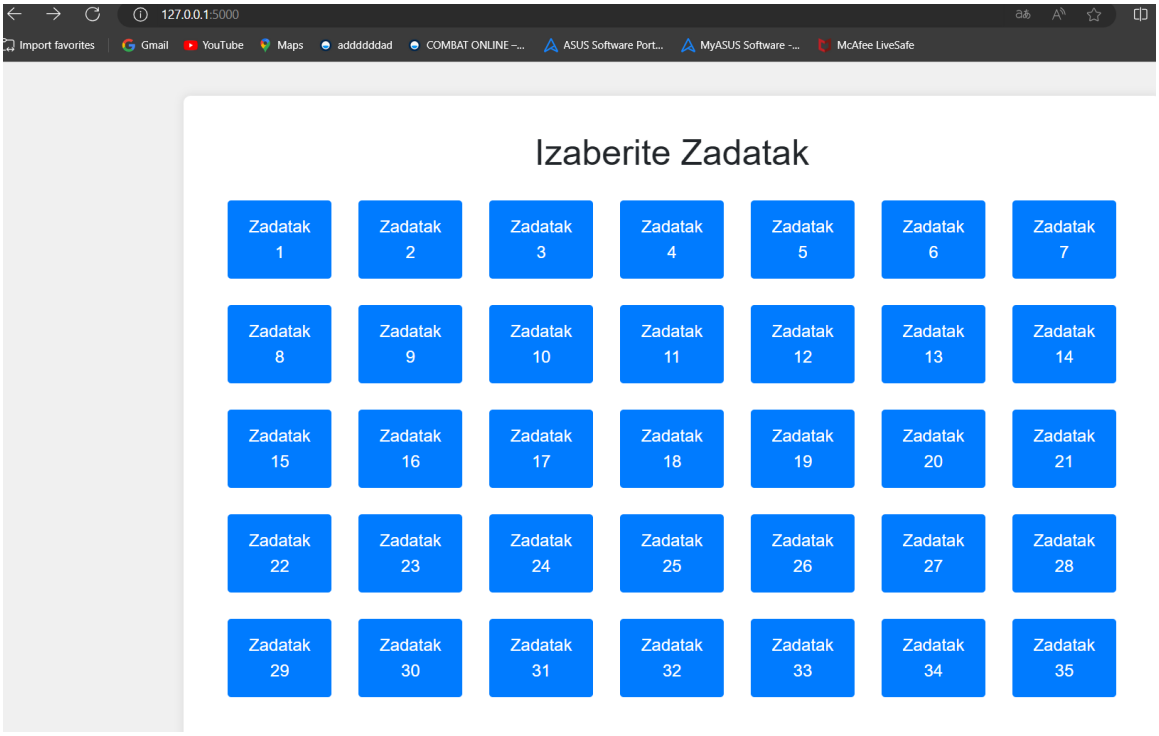
# 5 Implementacija autogradera za Python

Arhitektura automatskog sustava za ocjenjivanje, izrađenog korištenjem Flask frameworka, uključuje nekoliko ključnih komponenti koje omogućuju automatsku evaluaciju studentskih zadataka u Python programskom jeziku. Projekt koristi model klijent-poslužitelj, gdje studenti predaju svoje rješenje putem web sučelja, a sustav na serveru obrađuje ta ista rješenja te generira povratne informacije. Cilj ovog sustava je smanjiti ručno ocjenjivanje te omogućiti brzu, točnu i nepristranu evaluaciju studentskih radova.

## 5.1 Struktura i funkcionalnost početnog zaslona aplikacije

`index.html` predstavlja početnu stranicu aplikacije i služi kao središnje sučelje za odabir zadataka. Na ovoj stranici nalaze se gumbi koji predstavljaju svaki od trideset i pet zadataka, pružajući korisnicima mogućnost izbora zadatka koji žele riješiti. Od tih trideset i pet zadataka, njih petnaest su zadatci iz algoritama, deset iz lista i nizova te zadnjih deset zadataka su zadatci vezani za datoteke.

- Izgled aplikacije: Aplikacija je organizirana tako da sadrži pregled svih dostupnih zadataka. Svaki zadatak je predstavljen posebnim gumbom koji je jasno označen brojem ili nazivom zadatka, omogućujući korisnicima da na jednostavan način pronađu zadatak koji žele riješiti.




*Slika 1 Izgled početnog zaslona aplikacije*

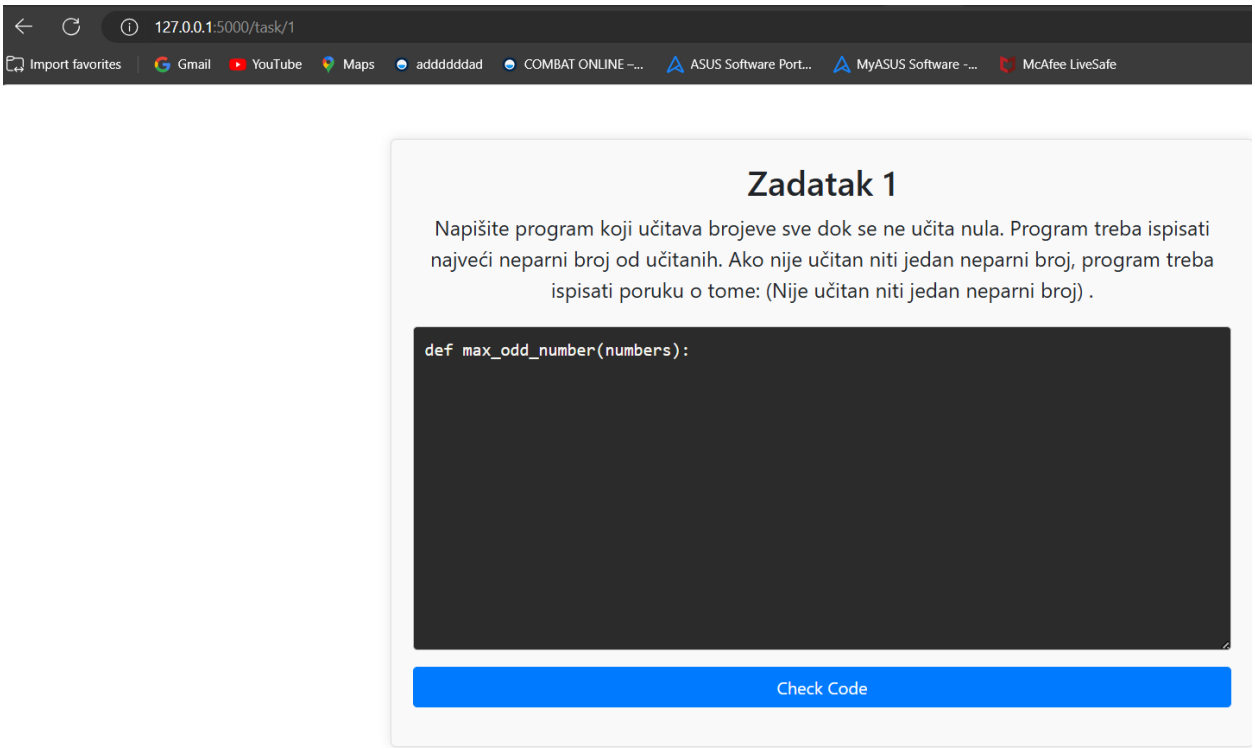
- Navigacija: Kada korisnik klikne na određeni gumb, aplikacija preusmjerava korisnika na specifičnu stranicu koja prikazuje detalje o odabranom zadatku. Ova navigacija omogućava korisnicima brz i efikasan pristup svakom zadatku, a backend aplikacije prepoznaje koji je zadatak odabran i vraća odgovarajuću stranicu putem Flaskove funkcionalnosti za renderiranje predložaka (engl. render_template).

- Primjena gumba: Gumbi su ključni za interakciju s korisnikom. Svaki gumb poziva backend, šaljući korisnički odabir kroz GET zahtjev Flask aplikaciji. Backend zatim obrađuje taj zahtjev i vraća HTML stranicu za odgovarajući zadatak, što pruža dojam fluidne i intuitivne navigacije kroz aplikaciju.

## 5.2 Struktura i funkcionalnost stranice zadataka

Stranica `task.html` koristi se za prikaz detalja specifičnih zadataka. Kada korisnik odabere zadatak na početnoj stranici, preusmjerava se na ovu stranicu, gdje može vidjeti puni opis zadatka i unijeti rješenje u obliku programskog koda.

- Prikaz zadatka: Na ovoj stranici prikazuje se opis odabranog zadatka, uključujući jasne upute o tome što korisnik treba postići svojim rješenjem. Ovaj opis pomaže korisnicima da razumiju zadatak prije nego što započnu s pisanjem koda.




*Slika 2*

- Unos rješenja: Korisnicima se pruža tekstualno polje (textbox) u kojem mogu unijeti svoje rješenje u obliku programskog koda. Ovo polje je centralni dio stranice jer omogućava interaktivnost aplikacije.
- Gumb za provjeru koda: Nakon unosa koda, korisnik može pritisnuti gumb "Provjeri kod", čime se kod šalje na evaluaciju putem POST zahtjeva prema backendu. Funkcionalnost ovog gumba omogućava dinamičku interakciju s aplikacijom, pri čemu se korisnički kod evaluira i rezultati se vraćaju korisniku.

## 5.3 Sustav za testiranje aplikacije

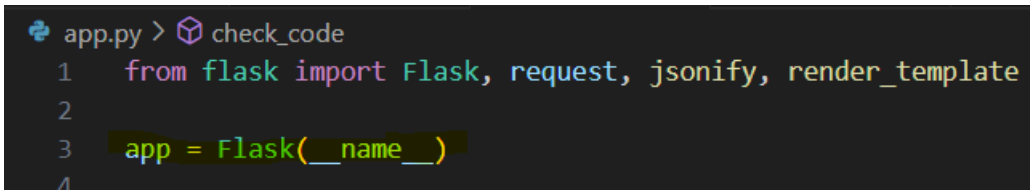
Datoteka `app.py` sadrži glavnu logiku aplikacije i služi kao most između korisničkog sučelja i evaluacijskog sustava. Ova datoteka koristi Flask framework za rukovanje zahtjevima s frontenda i vraćanje odgovarajućih odgovora.

Na početku koda uvoze se potrebne biblioteke:

- `Flask`: Služi za izradu web aplikacije.
- `request`: Omogućuje dohvaćanje podataka iz HTTP zahtjeva, npr. korisničkog koda.
- `jsonify`: Omogućuje vraćanje podataka u JSON formatu kao odgovor na HTTP zahtjev.
- `render_template`: Omogućuje prikaz HTML datoteka kao odgovora na zahtjev.

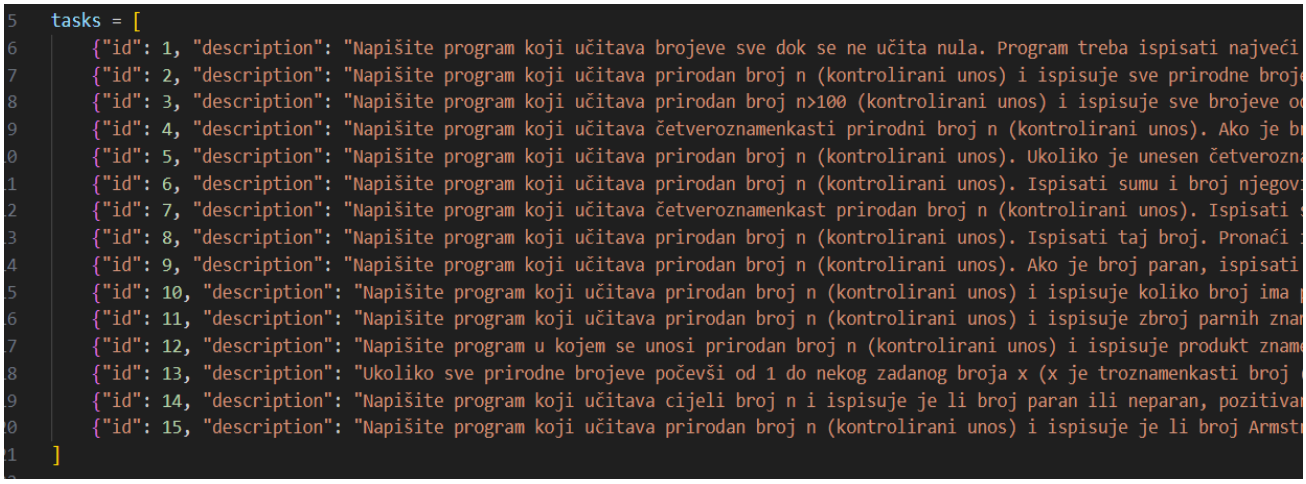

*Slika 3 Biblioteke potrebne za backend*

Flask aplikacija se inicijalizira putem `app = Flask(__name__)`. Ovo stvara instancu Flask aplikacije koja će obrađivati zahtjeve korisnika.



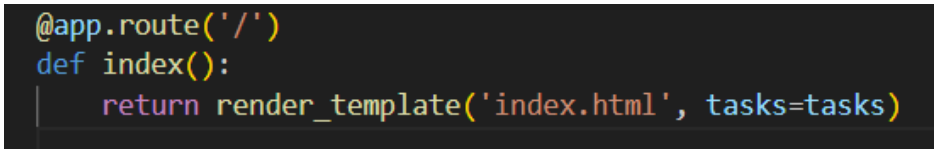

*Slika 15 Inicijalizacija aplikacije*

Zadaci su definirani kao popis Python rječnika, gdje svaki rječnik predstavlja jedan zadatak. Svaki zadatak ima svoj jedinstveni broj (id) i opis (description). Ovo služi za pohranjivanje 35 različitih zadataka koje korisnici mogu rješavati.




*Slika 4 Popis zadataka*

Aplikacija pruža jednostavno sučelje gdje se prikazuju svi zadaci. Stranica koristi render_template funkciju koja učitava HTML stranicu `index.html`. Na toj stranici svaki zadatak ima gumb koji preusmjerava korisnika na odgovarajuću stranicu zadatka putem Flask rute `/task/<int:task_id>`.




*Slika 18 Preusmjeravanje na početnu stranicu*

Za svaki zadatak kreirana je specifična ruta unutar Flask aplikacije. Kada korisnik zatraži određeni zadatak putem GET zahtjeva, aplikacija koristi `render_template` funkciju kako bi vratila HTML stranicu specifičnu za taj zadatak.

- Prvo se traži odgovarajući zadatak iz popisa tasks koristeći `task_id`.
- Ako zadatak ne postoji, vraća se HTTP status `404` s porukom "`Invalid task ID`".
- Ako zadatak postoji, prikazuje se odgovarajuća HTML stranica.

```python
@app.route('/task/<int:task_id>')
def task(task_id):
    task = next((task for task in tasks if task['id'] == task_id), None)
    if task is None:
        return "Invalid task ID", 404
    return render_template('task.html', task=task)
```

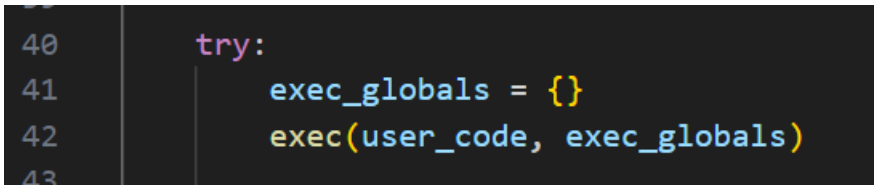
*Slika 5 Prikaz pojedinačnog zadatka*

Jedna od ključnih funkcionalnosti aplikacije je evaluacija koda unesenog na `task.html` stranici. Kada korisnik pošalje kod, backend ga prima putem POST zahtjeva na ruti `/check_code`. Koristi se `request.get_json()` kako bi se dohvatili korisnički kod i `task_id` iz zahtjeva.

```python
@app.route('/check_code', methods=['POST'])
def check_code():
    data = request.get_json()
    user_code = data['code']
    task_id = int(data['task_id'])
```

*Slika 6 POST zahtjev*

Funkcija `exec` se koristi za izvršavanje korisničkog koda u Pythonu. Kreira se novi prazan rječnik `exec_globals`, unutar kojeg se izvršava korisnički kod. Funkcija `exec` omogućuje dinamičko izvršavanje korisničkog koda, ali zbog sigurnosnih razloga ograničeno je okruženje.




*Slika 7 Korištenje funkcije exec*

Za svaki zadatak definiran je popis testnih slučajeva. Na primjer, za prvi zadatak (task_id == 1), korisnički kod se provjerava na nekoliko ulaza (testnih slučajeva). Svaki testni slučaj sadrži ulazne podatke (input) i očekivani izlaz (expected). Aplikacija izvršava funkciju definiranu u korisničkom kodu s tim ulaznim podacima i uspoređuje izlaz s očekivanim vrijednostima. Ako izlaz odgovara očekivanom, test se smatra uspješnim, a rezultat se pohranjuje u popis rezultata.

```
results = []

if task_id == 1:
    test_cases = [
        {"input": [2, 3, 5, 0], "expected": 5},
        {"input": [2, 4, 6, 0], "expected": "Nije učitan niti jedan neparni broj"},
    ]
    for test in test_cases:
        result = exec_globals['max_odd_number'](test['input'])
        results.append({"input": test['input'], "output": result, "expected": test['expected'], "passed": result == test['expected']})

elif task_id == 2:
    test_cases = [
        {"input": 10, "expected": [2, 6, 10]},
        {"input": 20, "expected": [2, 6, 10, 14, 18]},
    ]
    for test in test_cases:
        result = exec_globals['even_not_four'](test['input'])
        results.append({"input": test['input'], "output": result, "expected": test['expected'], "passed": result == test['expected']})

elif task_id == 3:
    test_cases = [
        {"input": 150, "expected": [19, 28, 37, 46, 55, 64, 73, 82, 91, 109, 118, 127, 136, 145]},
        {"input": 110, "expected": [19, 28, 37, 46, 55, 64, 73, 82, 91]},
    ]
    for test in test_cases:
        result = exec_globals['sum_digits_equals_ten'](test['input'])
        results.append({"input": test['input'], "output": result, "expected": test['expected'], "passed": result == test['expected']})
```

*Slika 8 Testiranje zadataka*

Ovaj dio koda koristi se za slanje HTTP odgovora u JSON formatu kada se dogodi greška ili kada se zadatak uspješno izvrši. Evo detaljnog objašnjenja:

```
else:
    return jsonify({"error": "Invalid task ID"}), 400
```

*Slika 29 Dio koda koji se koristi za slanje HTTP odgovora u JSON formatu*

- Ovaj blok koda dolazi nakon provjere ID-a zadatka. Ako zadani ID zadatka nije prepoznat (npr. nije definiran u sustavu ili je nevažeći), ovaj blok se izvršava.
- Vraća JSON odgovor s porukom o grešci. Ključ "error" u JSON objektu sadrži poruku "Invalid task ID" koja obavještava klijenta da je ID zadatka nevažeći.
- Vraća status kod **400** (Bad Request), što znači da je došlo do greške zbog neispravnog unosa ili zahtjeva od strane klijenta.

Ovaj dio koda se izvršava kada je zadatak uspješno provjeren i evaluiran. Vraća JSON objekt s ključem "results", a vrijednost ovog ključa je rezultat evaluacije zadatka. Ovaj rezultat može biti popis ili bilo koja druga struktura podataka koja sadrži povratne informacije o zadatku (npr. je li zadatak točno riješen, ispravni izlazi, itd.).

```
return jsonify({"results": results})
```

*Slika 103 Dio koda koji se izvršava kada je zadatak uspješno provjeren i evaluiran*

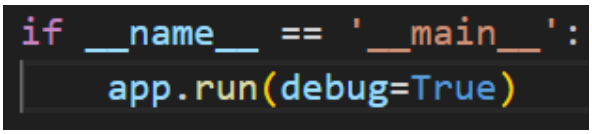
Ovaj blok se koristi za hvatanje bilo kakvih iznimki (grešaka) koje se dogode prilikom izvršavanja koda unutar `try` bloka. Ako dođe do bilo kakve iznimke, ona se hvata i pretvara u string pomoću funkcije `str(e)`. Tada se ta greška vraća u JSON formatu s ključem "`error`", kako bi korisnik dobio obavijest o čemu se radi. Također vraća status kod **400** (Bad Request), budući da je došlo do greške u obradi zahtjeva.

```
except Exception as e:
    return jsonify({"error": str(e)}), 400
```

*Slika 211 Kod koji se koristi za hvatanje grešaka*

Ovo je standardni Python konstruktor koji provjerava je li skripta pokrenuta direktno ili je uvezena kao modul u neki drugi program. Ako je skripta pokrenuta direktno, Flask aplikacija se pokreće pomoću `app.run()` i omogućava pristup putem lokalnog servera. `debug=True` omogućava "debugging mode", koji pomaže u otkrivanju grešaka tako što prikazuje detaljne poruke o pogreškama i omogućava automatsko ponovno pokretanje aplikacije kad se promijeni kôd.

Dakle, kada se plikacija pokrene, ona će slušati zahtjeve na zadanoj adresi (obično http://127.0.0.1:5000/) i omogućiti interakciju s njom putem preglednika ili API klijenta. Ovaj blok osigurava da aplikacija radi u razvojnom okruženju s dodatnim opcijama za otkrivanje grešaka.




*Slika 25 Standardni Python konstruktor koji provjerava je li skripta pokrenuta direktno ili je uvezena kao modul u neki drugi program*

# Zaključak

Automatski sustav za ocjenjivanje, tzv. autograder predstavljen u ovom radu nudi značajne prednosti za proces evaluacije programskih zadataka. Automatizirani sustav ocjenjivanja omogućava bržu i precizniju analizu studentskih rješenja, pružajući povratne informacije u realnom vremenu. Time se smanjuje potreba za ručnim ocjenjivanjem, što znatno olakšava rad nastavnicima, a studentima daje priliku da odmah isprave eventualne pogreške i uče na vlastitim greškama. Ovaj pristup također doprinosi standardizaciji ocjenjivanja jer se svi studenti ocjenjuju prema istim kriterijima.

Međutim, autograder nije bez svojih nedostataka. Jedna od glavnih slabosti je ograničena fleksibilnost u prepoznavanju različitih stilova kodiranja. Iako sustav može ispravno ocijeniti funkcionalnost koda, ne uzima u obzir kreativne ili alternativne načine rješavanja zadataka koji također mogu biti ispravni. Također, postoji rizik da studenti razviju preveliku ovisnost o automatiziranom sustavu, umjesto da sami pokušavaju riješiti složenije probleme i kritički analiziraju vlastiti kod.

Kako bi se autograder poboljšao, nekoliko aspekata može biti unaprijeđeno. Proširenje testnih slučajeva omogućilo bi bolju pokrivenost različitih rješenja i rubnih slučajeva. Integracija naprednijih alata za prepoznavanje logičkih grešaka mogla bi doprinijeti preciznijem ocjenjivanju i dubljoj analizi koda, čime bi se studenti potaknuli da unaprijede svoje vještine. Također, razvijanje sustava za evaluaciju stila kodiranja te uvođenje povratnih informacija o optimizaciji koda i njegovoj čitljivosti predstavljaju područja u kojima bi se autograder mogao dodatno razviti.

Gledajući prema budućnosti, moguće je očekivati da će autograderi postati još sofisticiraniji. Razvojem umjetne inteligencije mogli bi se implementirati sustavi koji ne samo da evaluiraju točnost koda, već i razumiju njegovu svrhu i kontekst. Takvi napredniji sustavi mogli bi ponuditi personaliziranu povratnu informaciju svakom studentu, prilagođenu njihovom znanju i napretku, što bi doprinijelo još efikasnijem učenju programiranja. Unatoč izazovima i ograničenjima, jasno je da autograderi imaju ključnu ulogu u modernizaciji obrazovnog procesa, osobito u području računalnih znanosti i programiranja.

# Bibliografija


[1] C. L. D. &. O. J. Douce, » Automatic Test-Based Assessment of Programming,« *Educational Resources in Computing*, 2005.

[2] D. Documentation.. [Mrežno]. Available: https://docs.docker.com/get-started/.

[3] Fluent Python: Clear, Concise Programming, O'Reilly Media., 2015.

[4] M. &. L. H. Unger, » "Automated Grading for Programming Exercises in MOOCs: A Review of the Current State.",« 2016.

[5] G. Wilson, »Wilson, G. (2016). Teaching Tech Together: How to Create and Deliver Lessons that Work and Build a Teaching Community Around Them,« [Mrežno]. Available: https://teachtogether.tech/.

[6] Y. &. H. X. Zhou, »The Impact of Automated Grading Systems on Students' Learning: A Case Study from a Programming Course,« *Computer Applications in Engineering Education*, 2020.

[7] A. &. P. D. Fox, »From Online Education to MOOC University.,« pp. 44-49, 2012.

[8] S. H. Edwards, Web-CAT: Automatic Grading of Programming Assignments., 2009.

[9] C. S. Horstmann, » CodeCheck: Automated Assessment of Programming Assignments,« 8 Rujan 2020.. [Mrežno]. Available: https://horstmann.com/codecheck/index.html.

[10] C. A. &. P. J. Gonzalez, »Automated Assessment Systems in Programming: An Overview of Current Tools.,« *Journal of Computer Science Education*, 2022.

[11] C. &. H. S. Murray, »CS50 and the Future of Programming Education,« *Communications of the ACM*, 2018.

[12] D. &. K. M. Kuklinski, »The Use of Automated Assessment Tools in Education: Challenges and Opportunities,« *Journal of Educational Technology & Society*, 2019.

[13] K. &. W. J. Huang, »Evaluation of Automatic Grading Systems in MOOC.,« 2017.

[14] M. Lutz, Learning Python (5th ed.)., O'Reilly Media., 2013.

[15] A. B. Downey, Think Python: How to Think Like a Computer Scientist., Green Tea Press., 2015.

[16] D. E. Knuth, The Art of Computer Programming, Volume 1: Fundamental Algorithms, 1997..

[17] J. Manning, Automate the Boring Stuff with Python, 2016.

[18] A. Meyers, The Complete Guide to Flask: Python Web Development for Beginners, 2019.

[19] »Python Software Foundation. pip Documentation.,« [Mrežno]. Available: https://pip.pypa.io/en/stable/.

[20] D. Powers, Flask by Example, 2017.

[21] »Python Software Foundation,« [Mrežno]. Available: https://www.python.org/downloads/.

[22] M. Grinberg, Flask Web Development: Developing Web Applications with Python., O'Reilly Media, 2018.

# Popis slika

Slika 1  Autograderi u MOOC platformama....10
Slika 2 Sučelje autogradera CodeCheck....11
Slika 3 Sučelje autogradera CS50....12
Slika 4 Provjera instalacije Pythona....22
Slika 5 Navigiranje do direktorija projekta....23
Slika.6 Kreiranje virtualnog okruženja....23
Slika 7 Naziv okruženja „naziv_okruženja“ unutar mape „moj.diplomski“ ....24
Slika 8 Izlaz naredbe "pip install Flask"....24
Slika 9 Provjera instalacije Flask-a....25
Slika 10  Dijagram slučajeva korištenja ....26
Slika 11 Dijagram slijeda....27
Slika 12 Izgled početnog zaslona aplikacije ....28
Slika 13 Izgled aplikacije nakon odabira željenog zadatka ....29
Slika 14 Biblioteke potrebne za backend....30
Slika 16 Popis zadataka ....31
Slika 18 Prikaz pojedinačnog zadatka....32
Slika 19 POST zahtjev....32
Slika 20 Korištenje funkcije exec....33
Slika 21 Testiranje zadataka ....33
Slika 22 Dio koda koji se koristi za slanje HTTP odgovora u JSON formatu....34
Slika 23 Dio koda koji se izvršava kada je zadatak uspješno provjeren i evaluiran....34
Slika 24 Kod koji se koristi za hvatanje grešaka....34
Slika 25 Standardni Python konstruktor koji provjerava je li skripta pokrenuta direktno ili je uvezena kao modul u neki drugi program ....35

# Tablice

Tablica 1 Usporedba različitih autogradera s glavnim karakteristikama .....15