**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ИНЖЕНЕРНО-ПЕДАГОГИЧЕСКИ ФАКУЛТЕТ – СЛИВЕН**

#### Катедра ЕАИТ

#### ДИПЛОМНА РАБОТА

Тема: *Реализация на система за провеждане на състезания по алгоритмични проблеми при Python и PHP.*

Дипломант: ………………

Димитър Константинов Манолов

фак. № 19061007

Ръководител: ……………

ас. д-р Иван Торлаков

Сливен, 2023 г.

"Информационните технологии и компютърите са като въздух - те са нещо, без което не можем да живеем." - Bill Gates

"Информационните технологии са средство за преобразуване на живота на хората, за подобряване на бизнеса и за промяна на света." - Linda Stone

"Информационните технологии са ключът към бъдещето." - Steve Jobs

Съдържание

[Увод 4](#_Toc139276948)

[1. Python 5](#_Toc139276949)

[1.1 Програмен език Python 5](#_Toc139276950)

[1.1.1 Характеристики 5](#_Toc139276951)

[1.2 Предимства и недостатъци на Python 8](#_Toc139276952)

[1.2.1 Предимства 8](#_Toc139276953)

[1.2.2 Недостатъци 10](#_Toc139276954)

[1.2.3 Приложения 11](#_Toc139276955)

[1.2.4 Функции 14](#_Toc139276956)

[2 Judge System 15](#_Toc139276957)

[2.1 Създаване на Judge System 18](#_Toc139276958)

[2.1.1 Python Judge System 19](#_Toc139276959)

[2.1.2 Docker 20](#_Toc139276960)

[2.1.3 Sandbox в Judge System. 23](#_Toc139276961)

[2.1.4 Библиотеки 26](#_Toc139276962)

[2.1.5 Websocket. 28](#_Toc139276963)

[3 Указания за използване. 31](#_Toc139276964)

[3.1 3.1. Описание на кода 31](#_Toc139276965)

[3.1.1 Стартиране на кода 38](#_Toc139276966)

[4 Заключение 42](#_Toc139276967)

[5 Използвана литература 43](#_Toc139276968)

# Увод

Judge системата е софтуерно приложение или платформа, което се използва в състезания по програмиране и програмни предизвикателства. Тя служи за автоматично оценяване на решенията, представени от участниците, и дава точкова оценка въз основа на определени критерии.

Judge системите често се използват в онлайн платформи за състезателно програмиране като Softuni, Codeforces, Topcoder, AtCoder и други. Участниците представят своите програмни решения на дадена задача в предварително определен формат (например на езика за програмиране C++, Java или Python). След това judge системата компилира и изпълнява представените решения в сигурно изолирана среда.

След изпълнението на решението, judge системата проверява изходите от програмата на участника спрямо очакваните резултати, които са дефинирани от създателите на състезанието. Това може да включва проверка на коректността на изхода, времето за изпълнение и използваната памет. Judge системата дава точки на базата на успешността на програмното решение, като резултатите и ранговете се обновяват в реално време на лидерборда.

Целта на тази дипломна работа е да проучи и разработи judge система, която е специализиран инструмент за оценка на програмен код. Главната цел на този проект е да се създаде ефективна и надеждна система, която може да извършва автоматична оценка на предоставен код и да дава точна и обективна обратна връзка на програмистите. Ще се проучат и изградят необходимите компоненти и модули за изграждането на judge системата. Това включва разработване на сървърна инфраструктура, модули за зареждане и изпълнение на код, механизми за проверка на резултатите и съхранение на данните. Целта е да се изберат и приложат подходящи алгоритми, които ще осигурят точна и надеждна оценка на предоставения код. Тази дипломна работа има за цел да създаде инструмент, който ще подпомогне програмистите и преподавателите в процеса на оценка на програмен код. Чрез разработването на ефективна и надеждна judge система, тази работа ще допринесе за подобряване на качеството на обучението по програмиране и ще улесни процеса на проверка и оценка на програмни решения.

# Python

Judge системите са важен инструмент за автоматично оценяване и ранжиране на участниците в програмистки състезания. Те помагат за създаването на справедливи и обективни условия за състезателите и спомагат за оптимизиране на процеса на оценяване и обявяване на резултатите от състезанието.

Тези системи имат значително приложение не само в програмистките състезания, но и в учебни среди, където могат да бъдат използвани за автоматично оценяване на домашни и учебни задачи. Те също така могат да помогнат на работодателите и рекрутерите да оценят програмистите и техните умения по време на процесите на подбор и набиране на персонал.

Judge системите са съществен инструмент в програмистката общност, който помага за развитието на програмистите, измерването на техните умения и създаването на конкурентна среда. Те също така подпомагат обучението и усвояването на програмиране чрез предоставяне на обратна връзка и възможност за сравнение с други.

## Програмен език Python

Python е създаден от програмиста Гвидо ван Росум (Guido van Rossum). Разработката на езика започва през края на 1980-те години в Националния институт по математика и компютърни науки в Холандия. Гвидо ван Росум създава Python като отговор на необходимостта от по-прост, чист и ефективен език за програмиране. Python е първоначално пуснат в продукция през 1991 година и оттогава се развива и усъвършенства от общността на програмистите по цял свят. Гвидо ван Росум остава активен в развитието на езика и ръководи процеса на разработка на Python до 2018 година.

### Характеристики

**Синтаксис**: Python е известен със своя чист и четим синтаксис, който го прави лесен за научаване и разбиране. Програмите се пишат с помощта на интуитивни блокови структури и не изискват изрично деклариране на променливи.

**Променливи и типове данни**: В Python можете да декларирате променливи и да присвоите стойности на различни типове данни като числа (цели числа, числа с плаваща запетая), символни низове, булеви стойности (True/False) и други. Python е динамично типизиран език, което означава, че типовете на променливите се определят автоматично.

**Условни оператори**: Можете да използвате конструкции като if-else и switch-case за изпълнение на код в зависимост от условия.

**Цикли**: Python поддържа различни видове цикли, като for и while, които ви позволяват да повтаряте блокове от код множество пъти.

**Функции**: В Python можете да дефинирате собствени функции, които могат да приемат аргументи и да връщат резултати. Функциите са полезни за групиране на код в отделни блокове, които могат да бъдат използвани повторно.

**Структури от данни**: Python предлага вградени структури от данни като списъци (list), кортежи (tuple), множества (set) и речници (dictionary). Всички те имат различни функционалности и са полезни за различни видове задачи.

**Модули и пакети**: Модулите представляват файлове с Python код, които могат да се внасят и използват в други програми. Пакетите са директории, които съдържат множество модули.

**Обработка на изключения**: Python предоставя механизъм за обработка на изключения, който ви позволява да се справяте с грешки и неочаквани ситуации по време на изпълнение на програмата. Можете да използвате конструкции като try-except-finally за обработка на изключения и освобождаване на ресурси.

**Работа с файлове**: Python предлага богати възможности за работа с файлове. Можете да отворите, прочетете, запишете и затворите файлове с помощта на вградените функции и методи за работа с файлови обекти.

**Обектно-ориентирано програмиране (ООП)**: Python поддържа ООП и ви позволява да създавате класове, обекти, наследяване, полиморфизъм и инкапсулация. Това ви дава възможността да структурирате и организирате по-големи проекти по-лесно и да преизползвате код.

**Работа с бази данни**: Python има множество библиотеки и модули за работа с различни видове бази данни. Например, можете да използвате модула SQLite3 за работа с SQLite бази данни или модула psycopg2 за връзка с PostgreSQL бази данни.

**Уеб разработка**: Python се използва широко за уеб разработка. Има популярни фреймуърки като Django и Flask, които ви позволяват да създавате уеб приложения и API-та бързо и лесно.

**Научни изчисления и машинно самообучение**: Python е популярен сред научниците и изследователите, тъй като предлага богати библиотеки като NumPy, SciPy и Pandas за научни изчисления и обработка на данни, както и TensorFlow и PyTorch за машинно самообучение и дълбоко самообучение.

**Автоматизация на задачи**: Python е мощен инструмент за автоматизиране на различни задачи. Можете да използвате библиотеки като Selenium за автомат

**Регулярни изрази**: Python предоставя модулът re, който ви позволява да използвате регулярни изрази за търсене и манипулиране на текст. Регулярните изрази са мощно средство за търсене на шаблони в текст и извършване на операции като заместване и извличане на данни.

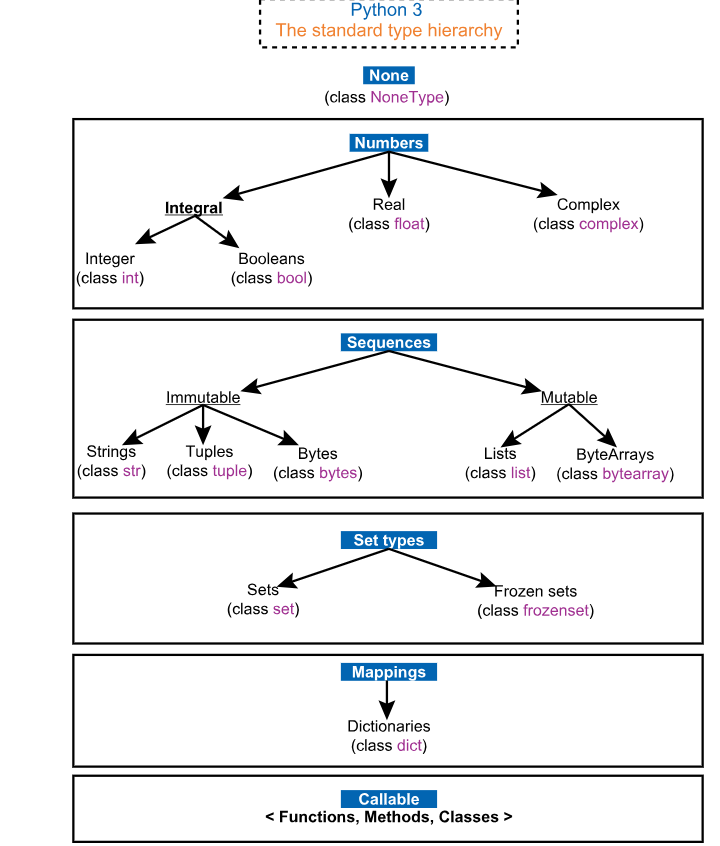
**Многонишково програмиране**: С Python можете да използвате модула threading или модула multiprocessing, за да създавате и управлявате нишки или процеси. Това ви позволява да използвате паралелно изпълнение и да ускорите изпълнението на програмата си, особено при изчислително интензивни задачи;

**Работа с API**: Python предлага много библиотеки за работа с различни API-та (Application Programming Interfaces). Това включва RESTful API, SOAP API, Twitter API, Google API и други. Можете да използвате библиотеки като requests или urllib за изпращане на HTTP заявки и получаване на отговори от различни уеб услуги;

**Тестово управление**: Python разполага с различни библиотеки и инструменти за тестване на софтуер. Някои от тях са unittest, pytest и nose. Те ви позволяват да пишете и изпълнявате автоматизирани тестове за вашия код, което помага за осигуряване на правилното функциониране и качеството на софтуера ви;

**Визуализация на данни**: Python предлага библиотеки като Matplotlib и Seaborn, които ви позволяват да създавате графики и визуализации на данни. Тези библиотеки ви предоставят мощни инструменти за визуализиране и представяне на данни по ясен и привлекателен начин.

**Документиране на код**: В Python има конвенции и инструменти за документиране на кода. Най-популярният инструмент е Sphinx, който ви позволява да генерирате качествена документация.



Фиг. 1 Стандартен тип йерархия

## Предимства и недостатъци на Python

Python е изключително полезен и мощен език за програмиране, който има множество предимства. Ето някои от основните предимства и слабости на Python.

### Предимства

**Лесен за научаване и разбиране**: Python е известен със своя чист и четим синтаксис, който го прави лесен за научаване дори за начинаещи програмисти. Това е добър избор за хора, които искат да започнат да програмират.

**Голяма общност и богата екосистема**: Python има огромна и активна общност от програмисти, което означава, че има множество ресурси, документация и поддръжка. Също така, Python разполага с обширна екосистема от библиотеки и модули, които обхващат различни области като уеб разработка, научни изследвания, машинно самообучение и други.

**Портативност**: Python е портативен език, който работи на различни операционни системи, включително Windows, macOS и Linux. Това означава, че можете да пишете код на една платформа и да го изпълните на друга без да променяте кода.

**Широк спектър на приложения**: Python е много универсален език, който може да бъде използван в различни области и сектори. От уеб разработка до научни изследвания и машинно самообучение, Python предоставя инструменти и библиотеки за различни видове проекти.

**Бързо прототипиране**: Python е отличен за бързото прототипиране на идеи и концепции. Синтаксисът му и богатата библиотека от инструменти ви позволяват да изградите прототипи бързо и да тествате идеите си преди да инвестирате повече време и усилия.

**Машинно самообучение и научни изчисления**: Python има силна подкрепа за машинно самообучение и научни изчисления. Библиотеки като NumPy, SciPy, Pandas, TensorFlow и PyTorch предоставят мощни инструменти и функци. Голяма подкрепа за библиотеки и модули: Python разполага с огромен брой библиотеки и модули, които могат да се използват за различни задачи. Независимо дали става въпрос за уеб разработка, научни изследвания, обработка на данни, графики или други области, вероятно ще намерите подходяща библиотека в Python, която ви помага да ускорите разработката на проекти.

**Голяма захватност и лесна интеграция**: Python е известен със своята възможност да се интегрира с други езици и системи. Можете да използвате Python за създаване на скриптове, които взаимодействат с програми написани на C/C++, Java, .NET и много други езици. Python има и интерфейси към множество системи и протоколи, което го прави идеален за разработка на разнообразни приложения.

**Платформено независим**: Python работи на различни операционни системи, включително Windows, macOS и Linux. Това ви позволява да пишете еднокодови приложения, които могат да се изпълняват на различни платформи без промени в кода.

**Добра четимост на кода**: Python е известен със своя чист и лесно разбираем синтаксис. Структурата на езика и насочването към четимост правят кода написан на Python лесен за разбиране и поддръжка. Това е от особена полза, когато работите в екип или когато преглеждате и поддържате код на по-късен етап.

**Голяма заедница и поддръжка**: Python има голяма и активна общност от програмисти, което означава, че има множество ресурси и поддръжка. Можете да намерите общности, форуми, блогове, уроци и примери, които ви помагат да научите и подобрите уменията си в Python.оналности, които улесняват изчисленията и разработката

### Недостатъци

**Изпълнение на кода**: Python е интерпретиран език, което означава, че изпълнението на кода може да бъде по-бавно в сравнение с компилираните езици като C++ или Java. Въпреки че Python има оптимизации и JIT (Just-in-Time) компилатори като PyPy, все още може да бъде по-бавен от други езици в изчислително интензивни задачи.

**Ограничена поддръжка на мобилни платформи**: Въпреки че Python работи на множество операционни системи, поддръжката му за мобилни платформи, като Android и iOS, е ограничена. Въпреки това съществуват някои разширения и фреймуърки, които могат да ви помогнат да разработите мобилни приложения с Python, но те не са толкова популярни и разпространени като другите езици, специално проектирани за мобилни платформи.

**Изисква по-голямо количество памет**: Поради динамичната типизация и вградената поддръжка на обектно-ориентирано програмиране, Python може да изисква повече памет в сравнение с по-ниско нивни езици или езици, които са по-близки до машинния код. Това може да бъде проблематично, особено при работа с по-големи масиви от данни или в ограничени околности, където наличието на ограничени ресурси е от значение.

**Global Interpreter Lock (GIL)**: GIL е механизъм в Python, който ограничава едновременното изпълнение на нишки в рамките на един процес. Това означава, че Python не може да използва многонишково програмиране за изпълнение на паралелни задачи с използване на множество процесорни ядра. Въпреки това, Python предлага алтернативни решения, като използването на множество процеси.

**Използване на памет**: Python може да използва повече памет в сравнение с някои други езици. Това се дължи на факта, че Python използва динамично управление на паметта и има вградена събирачка на отпадъци (garbage collector), която автоматично освобождава неизползваната памет. Това може да създаде проблеми при използването на Python за приложения с огромни обеми от данни или в ситуации, където паметта е ограничена.

**Скорост на изпълнение**: Въпреки че Python е лесен за използване и разработка, той не е най-бързият език за изпълнение. Интерпретирането и динамичната типизация на Python могат да направят кода малко по-бавен в сравнение с компилирани езици като C++ или Java, особено при изчислително интензивни задачи. Въпреки това, за много приложения скоростта на изпълнение на Python е достатъчна и не е пречка за успешната разработка на проекти.

**Политика на обратно съвместимост**: Въпреки усилията на общността на програмисти да поддържа обратна съвместимост, всяка голяма актуализация на Python може да доведе до непълна съвместимост с предишни версии. Това може да създаде проблеми при преносимостта на съществуващ код или при взаимодействие с външни библиотеки и модули. Въпреки това, съществуват инструменти и подходи, които могат да помогнат при управлението на проблемите със съвместимост

### Приложения

**Уеб разработка**: Python се използва широко за създаване на уеб приложения и уеб сайтове. Има много популярни фреймуърки като Django и Flask, които предоставят удобни инструменти за разработка на уеб приложения.

**Научни изследвания и обработка на данни**: Python има богата екосистема от библиотеки като NumPy, pandas и SciPy, които предоставят мощни инструменти за научни изчисления и обработка на данни. Python се използва активно в области като машинно самообучение, изкуствен интелект, геномика и други.

**Анализ на данни и визуализация**: Python предоставя библиотеки като matplotlib и Seaborn, които позволяват анализ и визуализация на данни. Тези инструменти са изключително полезни при изследване на данни и представяне на резултатите от анализа.

**Автоматизация на задачи**: Python е отличен избор за автоматизиране на различни задачи. Можете да създавате скриптове, които автоматизират рутинни задачи, обработват файлове и данни, взаимодействат със системни ресурси и други.

**Игрова разработка**: Python може да се използва за създаване на компютърни игри. Има библиотеки като Pygame, които предоставят функционалности за създаване на игри с графика, анимации и управление на потребителски вход.

**Мрежово програмиране**: Python предлага мощни библиотеки като Requests и Twisted, които улесняват разработката на мрежови приложения, включително работа с HTTP, TCP/IP протоколи, сокети и други.

**Роботика**: Python се използва в роботиката за програмиране на различни видове роботи. Фреймуърки като ROS (Robot Operating System) имат поддръжка за Python и предоставят инструменти за разработка на роботични приложения.

**Графичен дизайн и обработка на изображения**: Python предлага библиотеки като Pillow и OpenCV, които позволяват обработка на изображения, създаване на графики и визуални ефекти.

**Системно администриран**е: Python се използва за автоматизиране на задачи в системното администриране, като управление на конфигурации, мониторинг на системи и администриране на сървъри.

**Блокчейн технологии**: Python предлага библиотеки като Web3 и Pyethereum, които позволяват разработка на блокчейн приложения и смарт контракти.

Еднo от основните предимства на Python е неговата голяма библиотечна екосистема. Има хиляди библиотеки и модули, които предлагат функционалности за различни нужди, като работа с бази данни, обработка на език, генериране на документация, работа с XML и JSON, разработка на игри и други.

Python е език с широко приложение и голямо общество от програмисти, което го прави изключително полезен в много области на софтуерната разработка и също така удобен за учене и бързо разработване на проекти.

Има много известни компании, които използват Python в своите проекти и системи. Ето някои от тях:

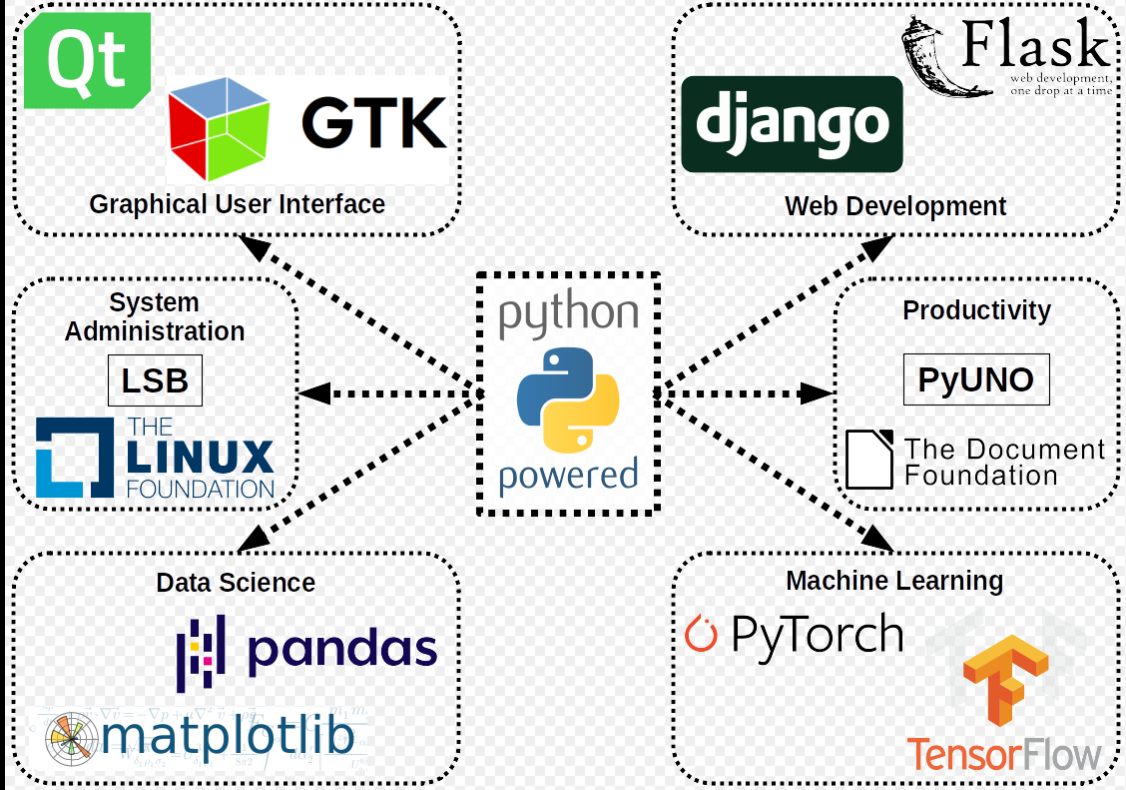
**Google**: Google използва Python за разработка на множество приложения и инструменти, включително популярния фреймуърк за уеб разработка Django.

**Facebook**: Python е използван в различни части на инфраструктурата на Facebook, включително в разработката на технологии за изкуствен интелект и обработка на данни.

**Instagram**: Instagram, който е собственост на Facebook, също използва Python за разработка на своите услуги и системи.

**Netflix**: Python се използва от Netflix в различни аспекти на тяхната платформа, включително за управление на системите за доставка на видео съдържание и персонализация на потребителските препоръки.

**Dropbox**: Python е избран език за разработка на бекенд системата на Dropbox и се използва за обработка на данни, синхронизация на файлове и други функции.



Фиг. 2 Приложения на Python

### Функции

**Философията на Python**: Python притежава "The Zen of Python" (Зенът на Python) - сет от принципи и философия, които насърчават четимостта и едно значение пред множество. Можете да видите Зенът на Python, като изпълните командата "import this" в интерпретатора на Python.

**Интуитивен и лесен за учене**: Python е известен с лесното си научаване и разбиране. Синтаксисът на Python е чист и езикът има малък брой ключови думи. Това го прави достъпен и за начинаещи програмисти.

**Обширна библиотечна екосистема**: Python разполага с огромен брой библиотеки, които предлагат различни функционалности. Библиотеки като NumPy, pandas, TensorFlow, Django и много други предоставят инструменти за разработка в различни области.

**Използва се в големи компании**: Python е популярен език за програмиране и се използва в големи технологични компании като Google, Facebook, Instagram, Netflix, Dropbox и др. Той е особено подходящ за разработка на уеб приложения, обработка на данни и машинно самообучение.

**Множество фреймуърки**: Python предлага множество фреймуърки за различни цели. Django е популярен за уеб разработка, Flask е известен със своята лекота и гъвкавост, а Scikit-learn и TensorFlow са известни в областта на машинното самообучение и изкуствен интелект.

**Python в космическите програми**: Python се използва в различни космически програми и агенции като NASA. Той се използва за разработка на софтуерни приложения за управление на космически мисии, обработка на данни от сензори и други.

**Python** е един от най-бързо развиващите се езици за програмиране в света и е известен със своята активна и подкрепена общност от програмисти.

**Python** има вариант на име "MicroPython", който е оптимизиран за работа на малки микроконтролери и вградени системи.

**Guido van Rossum**, създателят на Python, нарича езика "любимото си хоби". Той започна разработката на Python през 1989 година и го управляваше до 2018 година.

**Python** е един от езиците, които подкрепят функционалното програмиране, обектно-ориентираното програмиране и процедурното програмиране. Това го прави гъвкав и подходящ за различни стилове на програмиране.

**Python** поддържа множество операционни системи, включително Windows, macOS, Linux и други. Това го прави персоналният избор на програмистите, независимо от тяхната операционна система.

Въпреки че Python е интерпретиран език, съществуват и компилатори, които могат да преведат Python код в изпълним файл за по-бързо изпълнение.

**Python** е известен със своята четимост на кода, като се насърчава използването на експлицитен и ясен код за по-добро разбиране.

**Python** има широка гама от инструменти за тестване на код, включително модули като unittest, pytest и doctest, които помагат на програмистите да разработват и поддържат стабилен и надежден код.

**Популярност и широко приложение**: Python вече е един от най-популярните и широко използвани езици за програмиране в света. Той се използва в различни области, като уеб разработка, научни изследвания, машинно обучение, изкуствен интелект, анализ на данни и др. С този широк спектър от приложения, Python ще продължи да има значителна роля във високотехнологичната индустрия.

**Постоянно развитие и активност на общността**: Python има активна общност от програмисти, които продължават да развиват и подобряват езика. Python Software Foundation (PSF), некомерсиална организация, подкрепя и координира развитието на Python. Това допринася за бъдещето на Python, като се осигурява наличие на нови функционалности, подобрения в езика и поддръжка на съвместимост.

# Judge System

Judge System е софтуерна система, която се използва за автоматично оценяване на програмен код. Тя е предназначена да подпомага образователни институции, училища и състезания по програмиране, като предоставя

Judge системите имат различни функционалности и възможности, които ги правят полезни за програмистите и състезателите. Ето някои от тях:

Judge системата компилира и изпълнява представените програмни кодове, което позволява автоматично тестване на решенията. Това включва проверка на правилността на изходите, използването на очаквани резултати и проверка на ограниченията за време и памет.

Judge системите обикновено поддържат няколко популярни езика за програмиране, като C++, Java, Python, C# и други. Това позволява на участниците да представят решения в предпочитания от тях език.

Някои judge системи предлагат вградена онлайн среда за разработка, която позволява на участниците да пишат и тестват своите програми директно в платформата без да се налага инсталирането на допълнителен софтуер.

Judge системите актуализират лидерборда в реално време, като отразяват точковите оценки на участниците и им помагат да следят своята позиция в ранглистата.

След приключване на състезанието, judge системата обикновено позволява на участниците да видят правилните решения на задачите. Това е полезно за учене и сравняване на собствените решения с оптималните решения.

Judge системите могат да предоставят информация за времето за изпълнение и използваната памет на представените решения. Това помага на участниците да оптимизират своите програми и да постигнат по-добра ефективност.

Някои judge системи имат вградена функционалност за автоматично генериране на тестови примери за задачите. Това помага на състезателите да проверят решенията си върху различни входни данни и да се уверят, че те работят правилно във всички случаи.

Judge системите позволяват на организаторите на състезанието да създават и качват задачи, които участниците трябва да решат. Това включва описание на задачата, примерни входни данни и очаквани изходи.

Judge системите обработват различни видове грешки, които могат да възникнат по време на изпълнението на решенията, като неправилни изходи, прекалено дълго време за изпълнение или грешки при компилацията. Това помага на участниците да идентифицират и отстранят проблемите в своите програми.

Judge системите са проектирани да поддържат множество задачи в рамките на едно състезание или платформа. Това позволява на участниците да решават и представят решения за различни задачи, като получават точкови оценки за всяка от тях.

Въпреки ползите, които предоставят, judge системите имат и някои потенциални слаби страни:

Judge системите оценяват решенията въз основа на предварително зададени тестови примери и очаквани резултати. Това означава, че решенията, които са правилни, но се различават по отношение на структура или подход, може да получат по-ниска оценка. Някои състезания включват и ръчна проверка от съдии, за да се компенсира това ограничение.

Качеството на задачите, които се предлагат в judge системата, има голямо значение за резултатите и опита на участниците. Ако задачите не са добре дефинирани или има грешки в тестовите примери, това може да доведе до неравнопоставени условия за състезателите.

Judge системите оценяват решенията само по техническите аспекти, като коректност, ефективност и паметови ограничения. Те не вземат предвид контекстуални фактори, като иновативност, стил на програмиране или коментари в кода. Това ограничение може да доведе до липса на пълна оценка на уменията на програмистите.

Въпреки усилията за обработка на грешки, judge системите не могат да разрешат всички възможни проблеми. Някои сложни грешки, свързани с алгоритмични проблеми или логически грешки, могат да бъдат пропуснати от judge системата и да доведат до неточни оценки.

Judge системите изискват представянето на решенията в определен формат и на определени езици за програмиране, което може да ограничи творческата свобода на участниците. Например, ако judge системата не поддържа даден език, програмистите ще трябва да се адаптират към езика, който е разрешен от judge системата.

Идеята зад изпълнението на тестови случаи е да се провери правилността и коректността на предоставеното решение чрез сравнение на изходите му с очакваните резултати.

- При подготовка се дефинират и подготвят различни тестови случаи. Тези случаи включват входни данни или сценарии, които трябва да бъдат подадени на решението за изпълнение. Тестовите случаи могат да бъдат разнообразни и да включват различни ситуации, гранични стойности, невалидни входни данни и други.

- При изпълнение на решението, то е предоставено от потребителя или участника и се изпълнява с помощта на Judge System. Решението може да бъде компилирано или интерпретирано в зависимост от езика на програмиране. За всякакви входни данни или сценарии от тестовите случаи се подават на решението.

- Изходите от изпълнението на решението се сравняват с очакваните резултати за всякакъв тестов случай. Ако изходът съвпада с очаквания резултат, се приема, че решението е правилно за този конкретен тест. Ако изходът не съвпада, се счита, че решението е грешно.

Идеята зад изпълнението на тестови случаи е да се осигури достоверно оценяване на решенията, като се проверят тяхната функционалност, коректност и съответствие с очакваните изходи. Тестовите случаи позволяват на потребителите да видят как решението се справя с различни входни ситуации и да го подобрят в случай на грешки или неправилности.

## Създаване на Judge System

За създаване на judge система са необходими няколко основни компонента и функционалности:

**Оценяване на решенията**: Judge системата трябва да може да приема представени решения от програмистите и да ги оценява автоматично. Това включва компилация (ако е необходимо) и изпълнение на представените програми, както и проверка на правилността на резултата спрямо зададените тестови примери.

**Тестови примери**: За оценяване на решенията е необходима колекция от тестови примери. Тези примери представляват входни данни, към които се прилагат решенията, и очаквани изходни резултати. Judge системата трябва да изпълнява представените решения срещу тези тестови примери и да проверява правилността на получените изходни резултати.

**Езикова поддръжка**: Judge системата трябва да поддържа различни езици за програмиране, които програмистите могат да използват за представяне на решенията. Това включва компилатори или интерпретатори за различните езици, както и поддръжка на синтаксиса и особеностите на всякакъв език.

**Система за оценяване и резултати**: Judge системата трябва да може да оценява и рангира представените решения, базирайки се на правилността и ефективността им. Тя трябва да предоставя ясни резултати и обратна връзка за програмистите, които да им помогнат да разберат какво може да бъде подобрено в техните решения.

**Сигурност**: За сигурността на judge системата е важно да се предприемат мерки за предотвратяване на злонамерени или опасни действия от представените решения. Това може да включва изолиране на решенията в контролирани среди, ограничаване на достъпа до системните ресурси и други сигурностни механизми.

### Python Judge System

Ако judge системата е написана на Python, то това означава, че ядрото или централната част на системата е изградена с използването на Python. Някои от предимствата на използването на Python за създаване на judge система включват:

Python е език с чист и четим синтаксис, който прави разработката на системата по-проста и удобна. Той предлага богата библиотека от инструменти и модули, които могат да бъдат използвани за разработка на различни функционалности на judge системата.

Python е известен със своята гъвкавост и разширяемост. Това позволява бързо и лесно добавяне на нови функционалности към judge системата или промяна на съществуващите. Python предлага и механизми за управление на зависимости и пакетиране, които правят процеса на интеграция и разширение по-удобен.

Python има голяма общност от програмисти, които споделят знания и опит. Това означава, че има наличие на много ресурси, форуми, библиотеки и инструменти, които могат да помогнат при разработката и поддръжката на judge системата.

Python е мултиплатформен език, което означава, че judge системата може да бъде използвана на различни операционни системи като Windows, Linux и macOS. Това прави системата по-достъпна за потребителите на различни платформи.

При създаване на Python Judge System се предоставя среда за оценяване на програми, написани на езика за програмиране Python. Тази система е специализирана в изпълнението на Python код и проверката на резултатите от него. Основните цели на Python Judge System включват:

-Изпълнение на Python код: Системата позволява на потребителите да предават свои Python програми за изпълнение. Тя осигурява изолирана среда, в която кодът се компилира или интерпретира, в зависимост от спецификите на системата.

-Оценяване на решения: Python Judge System изпълнява предоставения Python код с помощта на тестови случаи или входни данни. Той сравнява изходите от програмата с очакваните резултати и дава оценка в зависимост от това дали резултатите съвпадат или не.

### Docker

Docker е отворен софтуер, който предоставя платформа за изолиране и управление на приложения в контейнери. Контейнерите са леки и преносими пакети, които включват всичко необходимо за изпълнението на софтуерно приложение, включително код, системни инструменти, библиотеки и зависимости. Docker осигурява стандартизирана и предсказуема среда за изпълнение на приложенията, независимо от операционната система или инфраструктурата, на която се изпълняват.

В основата на Docker стои концепцията на контейнеризация, която позволява пакетиране на приложенията и тяхното изпълнение в изолирана среда. Вместо да инсталирате и конфигурирате всички необходими компоненти на дадена система, Docker ви позволява да създавате контейнери, които включват всички нужни ресурси, и да ги използвате на различни системи без проблеми със съвместимостта.

Ето някои от ключовите концепции и компоненти в Docker:

**Docker контейнери**: Контейнерите са изолирани инстанции на приложения, които се изпълняват върху Docker платформата. Всяко приложение може да бъде пакетирано и стартирано в контейнер, който включва всички нужни ресурси.

**Docker образи**: Образите са шаблони, от които се създават контейнерите. Те съдържат цялата необходима информация за конфигуриране и стартиране на приложение в контейнер. Образите са леки, преносими и могат да бъдат споделяни и повторно използвани.

**Dockerfile**: Dockerfile е текстови файл, който съдържа инструкции за създаването на Docker образ. Чрез дефиниране на инструкции за инсталиране на софтуер, копиране на файлове и настройка на околната среда, можете да определите начина, по който образът ще бъде създаден.

**Docker регистър**: Docker регистърът е централизирано място за съхранение и споделяне на Docker образи. Регистърите позволяват на потребителите да изтеглят и качват образи, които могат да бъдат използвани от всички, които имат достъп до регистъра.

Docker се използва в много различни сфери, включително разработка на софтуер, DevOps, развитие на микросервизи, тестови среди и много други. Ето някои от основните причини, поради които Docker е толкова популярен:

Docker контейнерите осигуряват изолация на приложенията и тяхната зависима среда. Това позволява разработчиците да работят върху различни проекти с различни зависимости, без да се притесняват от взаимни конфликти. Освен това образите могат да бъдат споделяни и повторно използвани, което намалява времето за конфигурация и настройка на средите за разработка и изпълнение.

Docker контейнерите са преносими и могат да бъдат изпълнявани върху всяка съвместима платформа. Това позволява разработчиците да използват еднакви среди както на своите локални машини, така и на продукционните сървъри.

Docker използва контейнерната виртуализация, която е по-лека от традиционната виртуализация на виртуални машини. Това означава, че контейнерите стартират по-бързо и изискват по-малко оперативна памет и дисково пространство.

Docker предлага инструменти за управление на контейнерите и автоматизиране на развойните процеси. С помощта на Docker Swarm или Kubernetes може да се управлява множество контейнери и да се разпределят натоварванията в зависимост от нуждите на приложението.

Docker е широко използвана технология в индустрията на софтуерното разработване и DevOps. Ето някои от известните технологии и проекти, които използват Docker:

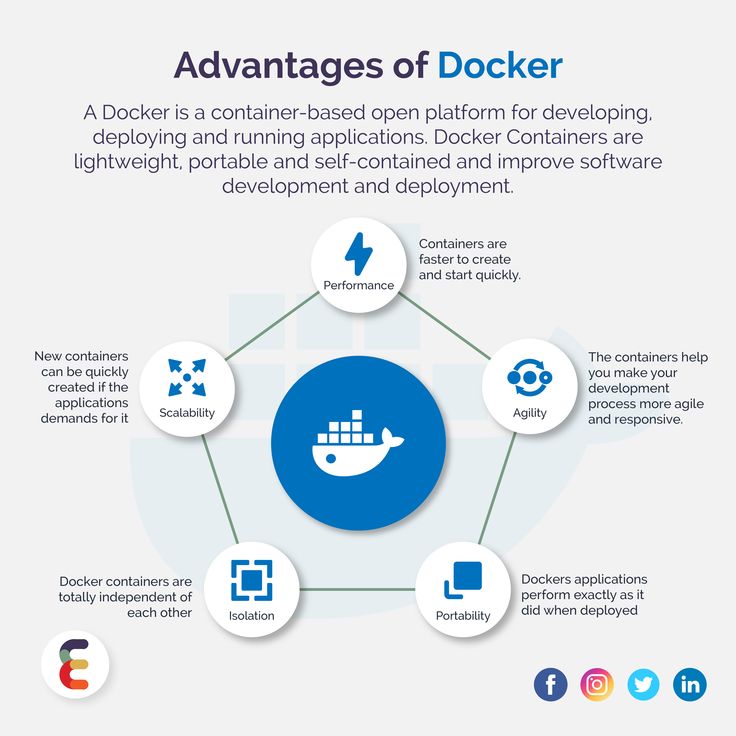
Kubernetes е популярна оркестрационна система за контейнери, която управлява и скалира автоматично Docker контейнери. Kubernetes позволява на разработчиците да управляват и развиват големи разпределени приложения в контейнери.

Jenkins е известен инструмент за непрекъсната интеграция и доставка (CI/CD). Той може да бъде конфигуриран да изгражда, тества и доставя приложения в Docker контейнери, което улеснява автоматизирания процес на разработка и доставка на софтуер.

GitLab е платформа за разработка на софтуер, която предоставя система за управление на изходния код, CI/CD инструменти и други функционалности. GitLab използва Docker за изолиране на приложенията и предоставя възможности за автоматично изграждане и доставка на контейнеризирани приложения.

Elastic Beanstalk е услуга за управление на приложенията в облака на Amazon Web Services (AWS). Тя поддържа Docker контейнери като един от форматите за пакетиране и разпространение на приложения в Elastic Beanstalk.

Azure е облачна платформа на Microsoft, която предлага различни услуги за хостинг и управление на приложения. Azure поддържа Docker контейнери и предоставя различни инструменти за развитие и доставка на контейнеризирани приложения.

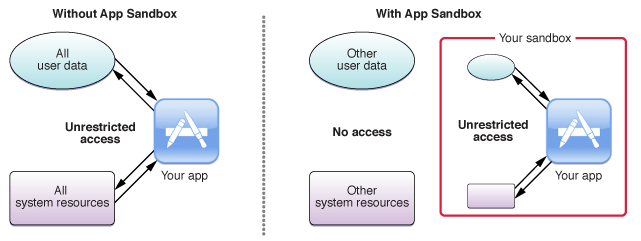
Google Cloud Platform (GCP) е облачна платформа, която предлага услуги за разработчици и оператори. GCP поддържа Docker контейнери и предоставя инструменти като Google Kubernetes Engine (GKE) за управление на контейнери и разпределени приложения.

Фиг. 3 Предимства при работа с Docker

Идеята на Docker в Judge System е да осигури изолирана и консистентна среда за изпълнение на програмите и да улесни процеса на оценяване на решенията.

В една Judge система често се изпълняват множество програми или решения, предоставени от потребителите или участниците. В този контекст - Docker позволява инсталирането и поддръжката на различни езици за програмиране и техните версии. Това е полезно в Judge системата, където се използват различни езици като Python, C++, Java и други. Всяка програма може да бъде изпълнена в подходящ контейнер, който съдържа съответния език и версия.

### Sandbox в Judge System.



Фиг. 4 Схема при ползване на Sandbox

Sandbox в Judge System е среда, в която се изпълняват и оценяват решенията на задачи по програмиране или други изпълними файлове. Това е контролирана и изолирана среда, предназначена да предостави сигурно и надеждно изпълнение на кода, като същевременно предотвратява възможността за злоупотреби или нежелани последици.

Sandbox-ът осигурява следните възможности и ограничения:

**-Изолация на процеси**: Решенията на задачите се изпълняват в отделни процеси, които са изолирани един от друг. Това предотвратява един код да се намеси или повлияе върху друг код, който се изпълнява в същата среда.

-**Ограничение на ресурси**: Sandbox-ът може да наложи ограничения върху използването на ресурси като процесорно време, памет, дисково пространство и мрежови връзки. Това предотвратява злоупотреби и гарантира, че всеки код използва разумно количество ресурси.

**-Ограничение на достъп до системни ресурси**: Sandbox-ът може да ограничи достъпа на кода до системни ресурси, като файлова система, операционна система или мрежа. Това предотвратява вредни операции върху системата или нежелани промени.

**-Безопасност и сигурност**: Sandbox-ът може да има механизми за откриване и предотвратяване на опасни или забранени операции, като например изпълнение на вредоносен код или достъп до забранени системни функции.

Judge System използва sandbox-и, за да гарантира обективност и сигурност при оценяването на решенията на задачи по програмиране. Те позволяват на потребителите да изпращат своите решения, които се изпълняват в контролирана и изолирана среда, където се проверява коректността и ефективността на кода според дефинираните критерии.

Ако не се използва sandbox в Judge System или ако не се осигури подходяща изолация и сигурност при изпълнението на решенията на задачи по програмиране, могат да възникнат някои рискове и проблеми:

**-Сигурност на системата:** Без изолация и sandbox, кодът на потребителите може да има достъп до системни ресурси и операционната система на сървъра. Това отваря вратата за потенциални атаки и злоупотреби, като например изпълнение на вредоносен код, промени в системните файлове или създаване на вредни мрежови връзки.

**-Конфликти и взаимодействие между решения**: Ако решенията на различни потребители се изпълняват без изолация, те могат да взаимодействат неправилно или да взаимодействат един с друг. Това може да доведе до нежелани последици и неточни резултати при оценяването на задачите.

**-Непредсказуемо поведение и несигурност на резултатите**: Без контролирана среда за изпълнение, резултатите от решенията на задачи могат да бъдат несигурни и непредсказуеми. Това може да доведе до неправилни оценки, неправилно функциониращи приложения или нежелани проблеми в продукционни среди.

**-Използване на нежелани системни ресурси:** Ако не се наложат ограничения върху използването на ресурси, решенията на задачи могат да консумират прекомерно много процесорно време, памет или други системни ресурси. Това може да доведе до затормозяване или дори срив на системата, което засяга други потребители и приложения.

Sandbox в контекста на Docker се използва за създаване на изолирана и контролирана среда за изпълнение на код. В следващите стъпки ще обясня как работи sandbox-ът през Docker:

**-Създаване на Docker образ**: Първоначално се създава Docker образ, който съдържа всички необходими зависимости и компоненти за изпълнение на кода. Образът може да включва операционна система, библиотеки, програмни езици, конфигурационни файлове и други необходими компоненти.

**-Изпълнение на контейнер:** След създаването на образа, той се използва за стартиране на Docker контейнер. Контейнерът представлява инстанция на образа, където кодът ще бъде изпълнен.

**-Изолация и ограничаване на ресурсите**: Docker предоставя механизми за изолация на контейнерите от хост системата и други контейнери. Кодът, изпълняван в контейнера, е изолиран и не може да взаимодейства директно с хост системата или други контейнери, освен ако не се настроят специални мрежови връзки.

Освен изолацията, Docker предоставя и възможности за ограничаване на ресурсите, които контейнерът може да използва. Могат да бъдат наложени ограничения върху процесорното време, паметта, дисковото пространство и мрежовите ресурси, които контейнерът може да използва по време на изпълнение.

**-Управление на входни и изходни данни:** Docker предоставя механизми за управление на входните и изходните данни на контейнера. Кодът може да получи входни данни чрез параметри или файлове, а резултатът от изпълнението му може да бъде получен чрез връщане на стойности, запис на файлове и други подобни операции.

**-Използване на Docker регистри**: Docker регистрите са централизирани репозитории, които позволяват споделяне и достъп до Docker образите. Те могат да бъдат използвани за съхранение на образите, които се използват в sandbox-а на Judge System, и за лесно обновяване или възстановяване на образите.

Чрез използване на Docker, се създава изолирана и контролирана среда, където кодът на потребителите може да се изпълнява безопасно и надеждно. Това осигурява сигурност, предотвратява конфликти и позволява лесно управление на ресурсите и данните, свързани с изпълнението на кода.

### Библиотеки

Библиотека в програмирането е съвкупност от предварително написани функции, класове и ресурси, които се използват за улесняване на разработката на софтуерни приложения. Тези функции и ресурси се предоставят от различни разработчици и организации и могат да бъдат включени в програмата чрез импортиране на съответната библиотека.

Те предоставят готови решения за често срещани задачи и функционалности, като обработка на данни, мрежово взаимодействие, графични интерфейси, математически операции, работа с бази данни и много други. Те позволяват на програмистите да използват вече написан код и да се възползват от функционалностите му, без да е необходимо да го пишат отново.

Библиотеките играят важна роля в разработката на софтуер, като помагат за ускоряване на процеса, повишаване на ефективността и предоставяне на разнообразни възможности за програмистите. Популярни примери за библиотеки в Python включват NumPy, Pandas, TensorFlow, Matplotlib, requests и много други.

Ето някои от библиотеките, които използвах за създаването на judge системата:

-**asyncio**. asyncio е библиотека в Python, която предоставя инфраструктура за създаване на асинхронен код. asyncio е вградена библиотека, налична в стандартната библиотека на Python от версия 3.4 и нататък.

asyncio предоставя средства за създаване на асинхронни приложения, като използва концепцията на корутини (coroutines), събития (events) и селектори (selectors). Това позволява на програмистите да пишат ефективен асинхронен код, който може да използва не-блокиращи операции и да осигури конкурентност и паралелизъм в програмата.

asyncio е особено полезна за създаване на сървърни приложения, мрежови услуги, уеб сървъри и други системи, които изискват множество паралелни операции. Тя предоставя инструменти за управление на конкурентността, обработка на събития и взаимодействие с различни протоколи като HTTP, TCP и UDP.

asyncio е станала изключително популярна в Python общността и се използва широко за разработка на асинхронни приложения и системи.

**-subprocess**. subprocess е библиотека в Python, която се използва за изпълнение на външни команди и манипулиране на процеси от Python скриптове. Тя е също вградена библиотека в стандартната библиотека на Python.

subprocess предоставя различни функции и класове, които позволяват на програмата ви да създава нови процеси, да комуникира с тях, да изпълнява команди в операционната система и да управлява входа и изхода на процесите.

С помощта на subprocess можете да изпълнявате команди от командния ред, да четете и пишете данни от и към процесите, да получавате изходни данни и грешки от процесите и да контролирате различни аспекти на процесите, като например тяхното приоритизиране или околната среда.

subprocess е използвана широко при автоматизация на системни задачи, обработка на данни, изпълнение на външни програми и други сценарии, които изискват взаимодействие с процесите в операционната система.

**-json**. json е библиотека в Python, която предоставя функционалности за работа с JSON (JavaScript Object Notation) данни. Тя е вградена библиотека в стандартната библиотека на Python.

json библиотеката предоставя набор от функции, които позволяват на програмата да кодира (serialize) Python обекти в JSON формат или да декодира (deserialize) JSON данни в Python обекти.

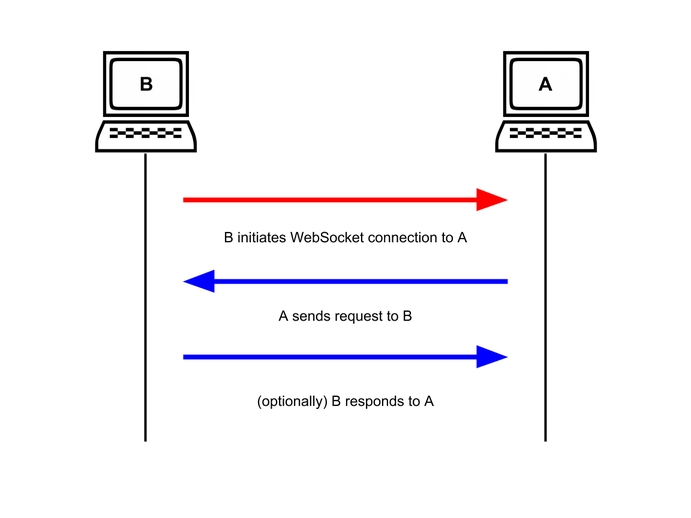
С помощта на json може да превръщате Python обекти като списъци, речници, низове и числа в JSON формат, който е текстов формат за представяне на структурирани данни. Също така, можете да превръщате JSON данни в Python обекти, които можете да използвате във вашата програма.

Библиотеките в Judge системата предоставят допълнителни функционалности и ресурси, които улесняват процеса на оценяване на решенията и подобряват функционалността на самата система. Библиотеките могат да бъдат използвани за различни цели, включително:

-Управление на изпълнението на програмите: Библиотеките могат да предоставят функции за контрол на изпълнението на програмите в средата на Judge системата. Това може да включва управление на времето за изпълнение, изолация на програмите, измерване на потреблението на памет и други аспекти, свързани с изпълнението.

Интеграция с други инструменти и услуги: Библиотеките могат да предоставят средства за интеграция с други инструменти и услуги, които се използват в Judge системата. Това може да включва интеграция с системи за управление на бази данни, уеб услуги, системи за съхранение на файлове и други външни ресурси.

### Websocket.



Фиг. 5 Механизъм работа на Websocket

WebSocket е протокол за двупосочна комуникация между клиент и сървър чрез уеб сокет (WebSocket) връзка. Това е разширение на HTTP протокола, което позволява постоянна връзка между клиента и сървъра, като позволява изпращането на съобщения в реално време.

Стандартният HTTP протокол е заявка-отговор базиран протокол, където клиентът прави заявка към сървъра и той отговаря с резултат. След като се получи отговорът, връзката се затваря и трябва да се установи нова връзка, ако клиентът иска да изпрати друга заявка.

WebSocket протоколът променя това като позволява постоянна двупосочна комуникация между клиента и сървъра. Той осигурява персистентността на връзката и позволява клиента и сървъра да изпращат съобщения един на друг в реално време, без да се налага постоянно заявяване на нови връзки.

WebSocket протоколът е често използван за изграждане на интерактивни уеб приложения и реално време услуги. Той поддържа библиотеки и фреймуърки на различни програмни езици, включително Python, които позволяват лесна имплементация на WebSocket комуникация и обработка на съобщенията в реално време.

С него клиентът и сървърът могат да изпращат съобщения по инициатива на всяка от страните, без постоянно заявяване на нови връзки. Това го прави изключително полезен за сценарии, където се изисква бърза, ефективна и двупосочна комуникация между уеб приложенията и сървърите.

WebSocket протоколът е проектиран да бъде лек, ефективен и ниско натоварващ за комуникация в реално време. Някои от основните характеристики на WebSocket включват:

WebSocket позволява клиентът и сървърът да изпращат съобщения един на друг в реално време. Това означава, че и двата участника могат да инициират комуникацията и да изпращат данни без постоянно заявяване на нови връзки.

WebSocket установява постоянна връзка между клиента и сървъра, което позволява по-бързо и ефективно изпращане на данни. Това отпада нуждата от повторно установяване на връзка при всяка заявка, като намалява натоварването и увеличава ефективността.

WebSocket е изграден върху TCP протокола и използва по-ниско ниво на протокола от HTTP. Това го прави по-ефективен за изпращане на данни в реално време и по-подходящ за сценарии, които изискват ниска латентност и висока производителност.

WebSocket поддържа различни версии на протокола, като например WebSocket Protocol Version 13, което гарантира съвместимост между различни клиенти и сървъри.

WebSocket се използва в широк спектър от сценарии, включително чат системи, реално време обновяване на данни, игри в реално време, финансови приложения и други, където двупосочна комуникация и изпращане на данни в реално време са от съществено значение.

В Python има няколко библиотеки, които предоставят поддръжка на WebSocket протокола, като websocket-client, websockets, tornado.websocket и други. Тези библиотеки позволяват разработчиците да създават WebSocket клиенти и сървъри, които могат да комуникират чрез WebSocket връзка.

WebSocket също може да се използва в judge системите за целите на комуникацията между клиента и сървъра. Използването на WebSocket в judge системите може да предложи следните предимства:

WebSocket позволява бърза и ефективна двупосочна комуникация между клиента и сървъра в реално време. Това е полезно в judge системите, където е важно да се осигури бързо и незабавно обмяна на данни, като например изпращане на информация за напредъка на изпълнението на кода или получаване на резултати в реално време.

WebSocket поддържа постоянни връзки между клиента и сървъра, което означава, че не е необходимо да се установяват нови връзки при всяка заявка. Това намалява натоварването на сървъра и подобрява ефективността на комуникацията.

С WebSocket, сървърът може да изпрати известие до клиента, когато задачата за изпълнение на кода приключи. Това е полезно в judge системите, където се изисква незабавно уведомяване на клиента за завършено изпълнение или за наличие на грешки.

WebSocket позволява на сървъра да изпраща резултатите от изпълнението на кода към клиента в реално време. Това е полезно в judge системите, където е важно да се предоставят резултати на потребителите в реално време, без да е необходимо да се актуализира страницата или да се изпълняват повторни заявки.

WebSocket може да се използва в judge системите, за да се подобри комуникацията, изпълнението на кода и предоставянето на резултати в реално време между клиента и сървъра. Това подобрява потребителското изживяване и ефективността на системата.

WebSocket е широко използван протокол за комуникация в реално време, който се прилага в много различни области и от множество компании. Ето някои от известните компании, които използват WebSocket:

Facebook използва WebSocket в своите чат и месинджър приложения, за да осигури бърза и двупосочна комуникация между потребителите.

Google използва WebSocket в своите приложения като Google Docs и Google Drive, за да позволи реално време сътрудничество и споделяне на файлове.

Идеята на WebSocket в Judge системата е да осигури двупосочна комуникация между клиента (например уеб браузър) и сървъра на Judge системата.

В контекста на Judge системата, WebSocket се използва за следните цели:

- Предаване на информация за състоянието на изпълнението: WebSocket позволява на сървъра да предава непрекъснат поток от информация за състоянието на изпълнението на програмите към клиента. Това може да включва информация за статуса на изпълнението (например "изпълнява се", "изчакване", "завършено"), съобщения за грешки, времеви отчети и други съобщения от Judge системата.

- Актуализация на резултатите на оценяването: WebSocket позволява на сървъра да предава незабавни актуализации за резултатите на оценяването на решението. Това може да включва информация за точките, преминатите тестови случаи, общия брой точки и други релевантни данни. Клиентът (например уеб браузър) може да получава и визуално да отразява актуализациите на резултатите в реално време.

- Предаване на команди от клиента към сървъра: WebSocket позволява на клиента да изпраща команди към сървъра на Judge системата. Това може да включва команди за стартиране или спиране на изпълнение на програми, промяна на настройки или конфигурации, управление на тестовите случаи и други действия, свързани с процеса на оценяване на решенията.

Използването на WebSocket в Judge системата подобрява взаимодействието между клиента и сървъра, предоставя по-бърза и актуална информация за състоянието на оценяването и осигурява по-добро потребителско изживяване при проследяване на резултатите на оценката.

# Указания за използване

## Описание на кода

import asyncio

import websockets

import subprocess

import json

class OfflineJudge:

def \_\_init\_\_(self):

self.program\_source = None

def run\_test\_case(self, expected\_result):

try:

with open("temp\_program.py", "w") as file:

file.write(self.program\_source)

process = subprocess.Popen(

["python", "temp\_program.py"],

stdout=subprocess.PIPE, stderr=subprocess.PIPE,

universal\_newlines=True

)

output, error = process.communicate(timeout=5)

if process.returncode != 0:

return False, f"Runtime Error:\n{error}"

output = output.strip()

if output == str(expected\_result):

return True, "Correct Answer"

else:

return False, f"Wrong Answer:\nExpected Result: {expected\_result}\nActual Result: {output}"

except subprocess.TimeoutExpired:

return False, "Time Limit Exceeded"

except Exception as e:

return False, f"Error occurred: {str(e)}"

async def start\_websocket\_server(websocket, path):

offline\_judge = OfflineJudge()

async for message in websocket:

data = json.loads(message)

program\_source = data.get('source')

expected\_result = data.get('expected')

offline\_judge.program\_source = program\_source

result, message = offline\_judge.run\_test\_case(expected\_result)

response = {

"result": result,

"message": message

}

await websocket.send(json.dumps(response))

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.get\_event\_loop().run\_until\_complete(

websockets.serve(start\_websocket\_server, "localhost", 8765)

)

asyncio.get\_event\_loop().run\_forever()

Програмен код 1 main.py

`import asyncio` - Импортира библиотеката `asyncio`, която предоставя инфраструктура за създаване на асинхронен код.

`import websockets` - Импортира библиотеката `websockets`, която предоставя възможност за създаване на WebSocket сървър и клиент.

`import subprocess` - Импортира модула `subprocess`, който позволява изпълнението на външни команди и процеси.

`import json` - Импортира модула `json` за работа с JSON данни.

`class OfflineJudge:` - Дефинира класа `OfflineJudge`, който съдържа методи за изпълнение на програмен код и оценка на резултата.

`def \_\_init\_\_(self):` - Дефинира конструктора на класа `OfflineJudge`. Тук се инициализира променливата `self.program\_source` със стойност `None`.

`def run\_test\_case(self, expected\_result):` - Дефинира метода `run\_test\_case`, който изпълнява подадения му програмен код и сравнява резултата с очаквания резултат.

`with open("temp\_program.py", "w") as file:` - Отваря файл с име "temp\_program.py" за запис и го свързва с променливата `file`.

`file.write(self.program\_source)` - Записва съдържанието на `self.program\_source` в отворения файл.

`process = subprocess.Popen([...])` - Стартира нов процес, изпълняващ командата `python temp\_program.py` чрез `subprocess.Popen`. Входа и изхода от процеса се пренасочват чрез `stdout=subprocess.PIPE` и `stderr=subprocess.PIPE`.

`output, error = process.communicate(timeout=5)` - Комуникира със стартирания процес и чете изхода и грешките му. Изчаква до 5 секунди за завършване на процеса.

`if process.returncode != 0:` - Проверява връщания от процеса код за грешка.

`return False, f"Runtime Error:\n{error}"` - Ако кодът на връщане не е 0, връща се грешка и съобщението за грешка.

output = output.strip() - Премахва празните пространства около изхода от процеса.

if output == str(expected\_result): - Проверява дали изходът от процеса е равен на очаквания резултат.

return True, "Correct Answer" - Ако изходът е равен на очаквания резултат, връща се успех и съобщение за правилния отговор.

return False, f"Wrong Answer:\nExpected Result: {expected\_result}\nActual Result: {output}" - В противен случай връща се неуспех и съобщение за грешния отговор.

except subprocess.TimeoutExpired: - Хваща изключението, което се генерира при изтичане на времето за изпълнение на процеса.

return False, "Time Limit Exceeded" - Връща неуспех и съобщение за надвишаване на лимита за време.

except Exception as e: - Хваща общото изключение Exception и го свързва с променливата e.

return False, f"Error occurred: {str(e)}" - Връща неуспех и съобщение за възникнала грешка.

async def start\_websocket\_server(websocket, path): - Дефинира асинхронната функция start\_websocket\_server, която се изпълнява при всяко ново WebSocket съобщение от клиента.

offline\_judge = OfflineJudge() - Създава нов обект от класа OfflineJudge.

async for message in websocket: - Чете съобщението, получено от клиента, чрез WebSocket връзката.

data = json.loads(message) - Преобразува JSON съобщението в Python обект.

program\_source = data.get('source') - Извлича програмния код от полето "source" на JSON обекта.

expected\_result = data.get('expected') - Извлича очаквания резултат от полето "expected" на JSON обекта.

offline\_judge.program\_source = program\_source - Записва програмния код в offline\_judge.program\_source.

result, message = offline\_judge.run\_test\_case(expected\_result) - Изпълнява програмния код чрез offline\_judge.run\_test\_case() и получава резултата и съобщението.

response = {"result": result, "message": message} - Създава обект от тип JSON с полета "result" и "message", съдържащи резултата и съобщението.

await websocket.send(json.dumps(response)) - Изпраща JSON съобщението като отговор през WebSocket връзката.

if \_\_name\_\_ == "\_\_main\_\_": - Проверява дали сценарият се изпълнява като основен сценарий.

asyncio.get\_event\_loop().run\_until\_complete([...]) - Създава нов събитиен цикъл от asyncio и стартира асинхронното изпълнение на WebSocket сървъра чрез websockets.serve().

asyncio.get\_event\_loop().run\_forever() - Започва безкраен цикъл на асинхронния събитиен цикъл, който позволява на сървъра да работи постоянно и да очаква нови връзки.

import subprocess  
  
class OfflineJudge:  
 def \_\_init\_\_(self):  
 self.program\_path = None  
 def run\_test\_case(self, expected\_result):  
 try:  
 process = subprocess.Popen(  
 ["python", self.program\_path],  
 stdout=subprocess.PIPE, stderr=subprocess.PIPE,  
 universal\_newlines=True  
 )  
 output, error = process.communicate(timeout=5)  
 if process.returncode != 0:  
 return False, f"Runtime Error:\n{error}"  
 output = output.strip()  
 if output == str(expected\_result):  
 return True, "Correct Answer"  
 else:  
 return False, f"Wrong Answer:\nExpected Result: {expected\_result}\nActual Result: {output}"  
 except subprocess.TimeoutExpired:  
 return False, "Time Limit Exceeded"  
 except Exception as e:  
 return False, f"Error occurred: {str(e)}"

Програмен код 2 offline\_judge.py

`import subprocess` - Импортира модула `subprocess`, който позволява изпълнението на външни команди и процеси.

`class OfflineJudge:` - Дефинира класа `OfflineJudge`, който съдържа методи за изпълнение на външни програми и оценка на резултатите.

`def \_\_init\_\_(self):` - Дефинира конструктора на класа `OfflineJudge`. Тук се инициализира променливата `self.program\_path` със стойност `None`.

`def run\_test\_case(self, expected\_result):` - Дефинира метода `run\_test\_case`, който изпълнява външната програма, зададена от `self.program\_path`, и сравнява резултата с очаквания резултат.

`process = subprocess.Popen([...])` - Стартира нов процес, изпълняващ командата `python self.program\_path` чрез `subprocess.Popen`. Входа и изхода от процеса се пренасочват чрез `stdout=subprocess.PIPE` и `stderr=subprocess.PIPE`.

`output, error = process.communicate(timeout=5)` - Комуникира със стартирания процес и чете изхода и грешките му. Изчаква до 5 секунди за завършване на процеса.

`if process.returncode != 0:` - Проверява връщания от процеса код за грешка.

`return False, f"Runtime Error:\n{error}"` - Ако кодът на връщане не е 0, връща се грешка и съобщението за грешка.

`output = output.strip()` - Премахва празните пространства около изхода от процеса.

`if output == str(expected\_result):` - Проверява дали изходът от процеса е равен на очаквания резултат.

`return True, "Correct Answer"` - Ако изходът е равен на очаквания резултат, връща се успех и съобщение за правилния отговор.

`return False, f"Wrong Answer:\nExpected Result: {expected\_result}\nActual Result: {output}"` - В противен случай връща се неуспех и съобщение за грешния отговор.

`except subprocess.TimeoutExpired:` - Хваща изключението, което се генерира при изтичане на времето за изпълнение на процеса.

return False, "Time Limit Exceeded" - Връща неуспех и съобщение за надвишаване на лимита за време.

except Exception as e: - Хваща общото изключение Exception и го свързва с променливата ‘e’.

return False, f"Error occurred: {str(e)}" - Връща неуспех и съобщение за възникнала грешка.

FROM python:3.9  
  
WORKDIR judge-system-main  
  
COPY /src/main.py ./  
COPY /src/offline\_judge.py ./  
COPY /src/testing1.py ./  
  
CMD ["python", "main.py", "src.main:judge"]  
  
EXPOSE 80

Програмен код 3 Dockerfile

Този код представлява Dockerfile, който се използва за създаване на Docker образ за система за оценяване (judge system).

`FROM python:3.9` - Избира базов образ на Docker, в този случай Python версия 3.9. Това означава, че създаденият образ ще съдържа Python версия 3.9.

`WORKDIR judge-system-main` - Задава работната директория в образа, където ще се извършва останалата част от командите. В този случай, работната директория е `judge-system-main`.

`COPY /src/main.py ./` - Копира файлът `main.py` от локалната директория `/src` в текущата работна директория на образа.

`COPY /src/offline\_judge.py ./` - Копира файлът `offline\_judge.py` от локалната директория `/src` в текущата работна директория на образа.

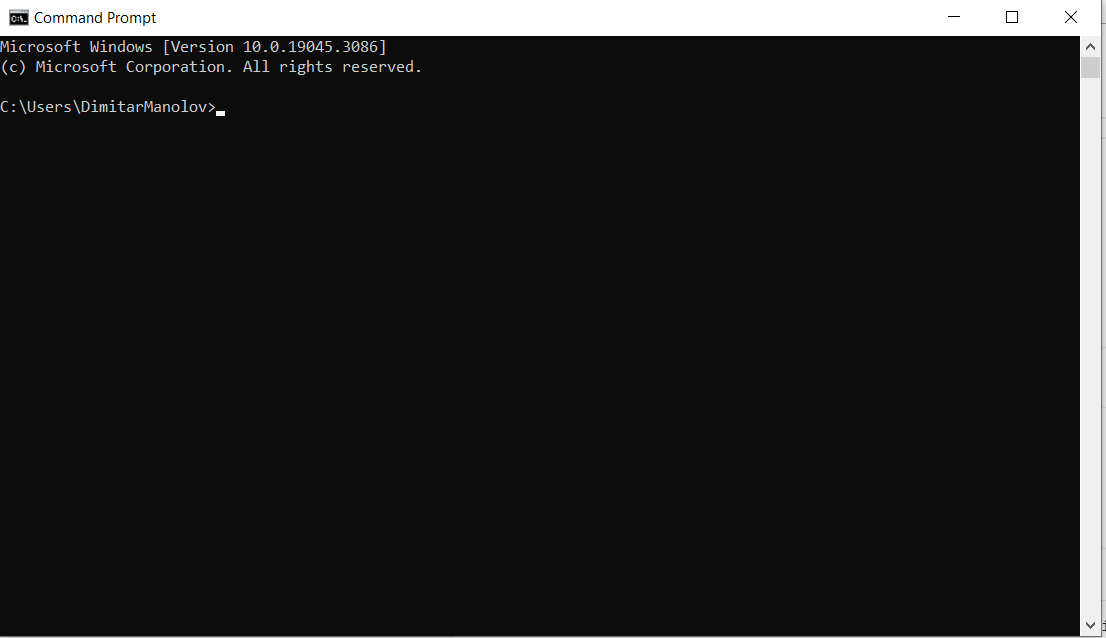
COPY /src/testing1.py ./` - Копира файлът `testing1.py` от локалната директория `/src` в текущата работна директория на образа.

`CMD ["python", "main.py", "src.main:judge"]` - Задава командата, която ще се изпълни по подразбиране при стартиране на контейнера, базиран на създадения образ. В този случай, командата е `python main.py src.main:judge`. Това означава, че при стартиране на контейнера ще се изпълни Python скриптът `main.py` с аргумент `src.main:judge`.

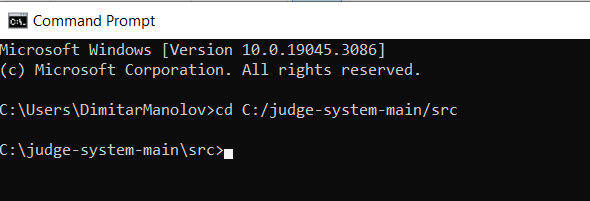
`EXPOSE 80` - Специфицира, че контейнерът ще слуша на порт 80 и може да приема входящи връзки на този порт.

### Стартиране на кода

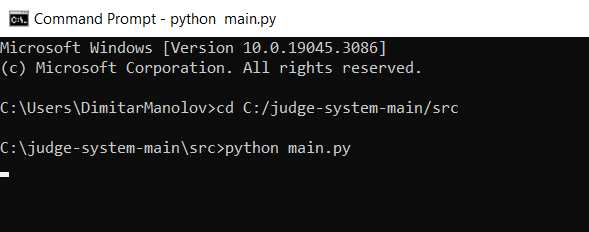
Отваря се Command Prompt



Посочва се пътя до main.py



Стартира се main.py с команда python main.py

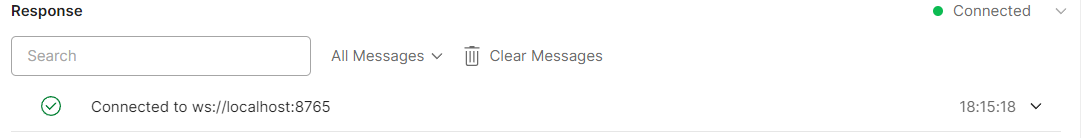


След това се отваря Postman (Когато се използ Postman с WebSocket, може да се установи WebSocket връзка със сървъра и да се изпращат и получават съобщения по време на активната връзка. Това дава възможност да се изпращат WebSocket заявки към сървъра и да се наблюдават отговорите му в реално време.) Въвеждаме ws://localhost:8765 в полето ENTER URL

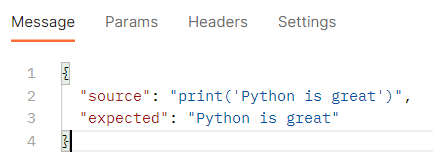
ws://localhost:8765 е адресът на WebSocket сървъра, който служи за приемане на кодови предложения, изпълнение на предоставения код и връщане на резултатите от изпълнението.



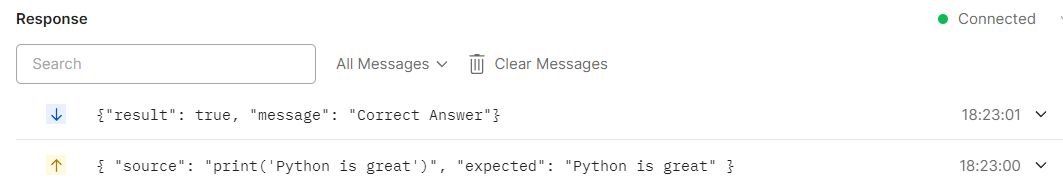
След това ще се установи връзка



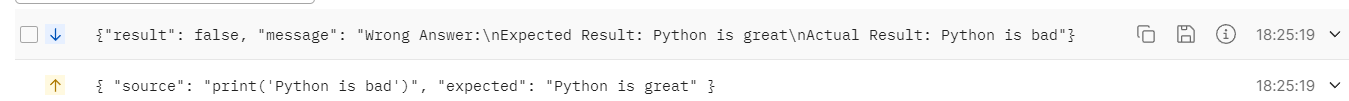
Изпраща се съобщение до Websocket сървъра



Сървърът приема съобщението, изпълнява предоставения код и сравнява получения резултат с очаквания. След това връща отговор в JSON формат с полета "result" и "message", които указват дали изпълнението е било успешно и съобщение, свързано с резултата.



Ако отговорът не е правилен:



- В `offline\_judge.py` има клас `OfflineJudge`, който представлява офлайн оценяващ съдия. Той има метод `run\_test\_case`, който приема очаквания резултат и изпълнява предоставения код.

- В метода `run\_test\_case` се използва модула `subprocess` за изпълнение на предоставения код в Python интерпретатор. Помощникът отваря нов процес с `subprocess.Popen` и предава предоставения код за изпълнение.

- След като кодът приключи изпълнението си, проверява се резултата. Ако кодът се изпълни успешно (връща код за успешно завършване), сравнява се изхода от изпълнението с очаквания резултат.

- Ако изходът е същият като очаквания резултат, връща резултат "Вярно" и съобщение "Правилен отговор". В противен случай, връща резултат "Грешка" и подробно съобщение, показващо очаквания и получения резултат.

- В `main.py` има функцията `start\_websocket\_server`, която се изпълнява, когато клиент се свърже със сървъра чрез WebSocket протокола.

- В функцията `start\_websocket\_server` се четат съобщенията, получени от клиента, като се използва цикъл `async for`. Съобщенията се предават в JSON формат и съдържат полета `"source"` (сорс кода) и `"expected"` (очаквания резултат).

-Създава се инстанция на класа `OfflineJudge` и се задава предоставения сорс код като поле на инстанцията. След това се извиква метода `run\_test\_case` с очаквания резултат.

- В зависимост от резултата от метода `run\_test\_case`, се формира отговор, който съдържа полета `"result"` и `"message"`, които указват дали резултатът е верен и съобщение за резултата.

- Отговорът се изпраща обратно към клиента чрез WebSocket връзката.

- В основния блок на `main.py` стартираме WebSocket сървъра, който слуша на адрес "localhost" и порт 8765. След това изпълнява безкраен цикъл `asyncio.get\_event\_loop().run\_forever()`, който позволява на сървъра да продължи да работи и да приема нови клиенти.

Компонентите са свързани по следния начин:

- Postman: Postman позволява да се изпращат HTTP заявки към Judge сървъра. Използвах Postman, за да изпращатя заявки с код към Judge сървъра и да получа отговори с резултатите от изпълнението на кода.

-WebSocket сървър: Judge сървърът използва библиотеката `websockets` за създаване на WebSocket сървър. Той слуша за входящи връзки от клиентите (например Postman) и установява WebSocket връзка с тях. След като връзката е установена, сървърът приема заявки с програмен код и изпраща отговори с резултатите от изпълнението.

- OfflineJudge клас: Това е класът, който извършва изпълнението на кода и проверката на резултатите. Когато Judge сървърът получи заявка с програмен код от клиента, той изпраща кода към инстанцията на OfflineJudge класа. OfflineJudge класът изпълнява кода в изолирана среда, изчаква резултата и след това го връща обратно на сървъра.

- Docker контейнер: За да изолира изпълнението на кода и осигури сигурност, Judge системата използва Docker контейнери. Когато OfflineJudge класът получи кода за изпълнение, той създава временен файл с кода и го изпраща към Docker контейнера. Контейнерът изпълнява кода в изолирана среда и връща резултатите обратно на OfflineJudge класът.

Тези компоненти работят заедно, за да осигурят функционалността на Judge системата. Postman се използва за изпращане на заявки към Judge сървъра, който чрез WebSocket връзка комуникира с OfflineJudge класа, който изпълнява кода в Docker контейнер и връща резултатите обратно на сървъра, който ги препраща към клиента.

# Заключение

В заключение, тази дипломна работа разглежда разработването на judge система като специализиран инструмент за оценка на програмен код. Целта на проекта беше създаването на ефективна и надеждна система, която може автоматично да оценява предоставен код и да предоставя обективна обратна връзка на програмистите.

Процесът на разработка включваше проектиране на компонентите на judge системата, изследване и избор на подходящи алгоритми за оценка на програмен код, както и разработване на потребителски интерфейс за лесно взаимодействие със системата.

Чрез обширното тестване и оценка на judge системата, беше потвърдено, че тя е успешен инструмент за оценка на програмен код. Системата демонстрира функционалност, надеждност и ефективност при изпълнение на различни типове тестове. Надявам се, че тази дипломна работа ще послужи като основа за бъдещи подобни проекти и ще допринесе за развитието на инструменти за оценка на програмен код, които играят важна роля в сферата на софтуерното обучение и разработката.

# Използвана литература

1. <https://dl.acm.org/doi/abs/10.1145/3143560>
2. <https://devopscube.com/what-is-docker/>
3. <https://docs.docker.com/get-started/overview/>
4. <https://en.wikipedia.org/wiki/Docker_(software)>
5. <https://en.wikipedia.org/wiki/Library_(computing)>
6. <https://en.wikipedia.org/wiki/Python_(programming_language)>
7. <https://gatefy.com/blog/sandbox-what-it-and-how-it-works/>
8. <https://www.javatpoint.com/postman>
9. <https://www.knowledgehut.com/blog/web-development/what-is-websocket>
10. <https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is>
11. <https://www.worldscientific.com/doi/pdf/10.1142/S0218194022500346>