# Автоматизация тестирования кода для предметов в области программирования с помощью контейнеров

Павлов В. О.

ФГАОУ ВО «СВФУ имени М.К. Аммосова», Педагогический институт, кафедра информатики и вычислительной техники, БА-ИВТ-17

В современной России количество строительства новых школ не соответствует количеству увеличения рождения детей, что порождает переполнения учебных классов. Преподавателям становится всё сложнее проверять и делать индивидуальный анализ заданий, выполняемых учениками. Поэтому мною предложена система, которая позволит преподавателям дисциплин, связанных с программированием, частично автоматизировать проверку решений различных задач обучающихся.

Данная система должна решать следующие проблемы:

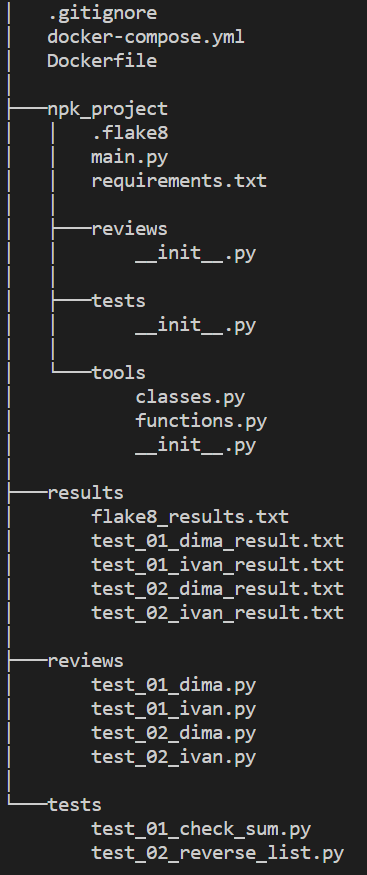
1. Проверку скриптов на решение своих задач.
2. Логирование ошибок выполнения.
3. Подсчет времени исполнения скриптов.
4. Проверку кода на соответствие Style Guide’ам языков.
5. Вывод анализа для последующей аналитики.
6. Легкий запуск системы на компьютерах или серверах образовательных учреждений.
7. Удобный интерфейс для взаимодействия преподавателей с системой.
8. Возможность устанавливать свои зависимости, если они нужны в задаче.
9. Изоляция тестовой среды от внешних параметров системы и возможность работать без интернета – в режиме оффлайн.

Предлагаемое решение на примере задач по алгоритмам с использованием языка Python сводится к следующему. Для решения задачи проверки скриптов на правильность их исполнения требуется решить две проблемы: 1) создание кейсов для тестирования решений и 2) создания данных, которые будут проверяться на соответствие. Для первой задачи, в контексте языка Python, будем использовать собственные тест классы, так как задача имеет специфичные требования. Это фреймворк, входящий в стандартную библиотеку языка Python, который позволяет описывать различные кейсы для тестирования кода обучающихся. Для генерации данных рекомендуется рассмотреть Faker, он позволит сделать независимые входные данные для различных скриптов.

Преимущества и логические этапы предлагаемой системы:

1. Возможность написания кейсов тестирования скриптов на своем языке программирования и использование библиотек для генерации данных.
2. Для выполнения логирования приложения подойдет loguru.
3. В качестве инструмента для измерения времени выполнения программы приемлемым вариантом является timeit, он входит в стандартную библиотеку python и позволяет проводить более сложные замеры времени.
4. Как инструмент для проверки кода необходимым требованиям удовлетворяет flake8, который имеет очень высокую расширяемость и возможность глубокой настройки правил.
5. В качестве вывода для последующей аналитики соответствует формат json с основными параметрами тестов: результат, время выполнения, ошибки в оформлении и ошибки в исполнении программы
6. В качестве среды для запуска рекомендуется использовать контейнеры для виртуализации, поскольку они имеют хорошую возможность изоляции и быстрое развертывание как на большинстве современных систем для персональных компьютеров, так и на различные серверные дистрибутивы unix-систем. В качестве программного обеспечения для управления контейнерами следует обратить внимание на Docker, поскольку он имеет удобный интерфейс для управления контейнерами и переключением между windows и unix контейнерами.
7. В качестве интерфейса возможны два сценария использования – как самостоятельную программу и как сервис для различного рода продуктов, например CMS-систем.
8. Также должна быть возможность устанавливать собственные зависимости внутри контейнеров, поэтому предлагается рассмотреть два варианта запуска: с помощью своего файла зависимостей или стандартного, который уже будет помещен в контейнер. В качестве пакетного менеджера для Python это будут Pip и файл зависимостей requirements.txt.
9. Для работы в режиме оффлайн, например в местах, где есть проблемы с интернетом, необходимо предусмотреть заранее возможность запуска контейнера без различных внешних зависимостей и с возможностью беспроблемного переноса.

Пример расположения файлов:



В данном примере есть npk\_project для сборки контейнера, Dockerfile для описания сборки, docker-compose как удобный менеджер контейнеров и подключения томов.

В качестве томов создаются 3 папки: tests для тестов преподавателей, reviews для ответов обучающихся и results, где в удобном формате выводятся результаты проверки тестов.

Пример самого простого теста, для проверки функций сложения:

import faker

from tools import TestClass

faker = faker.Faker()

class SomeTest(TestClass):

    def right\_version(self, data):

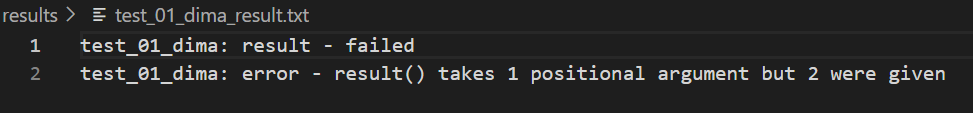
        return data[0] + data[1]

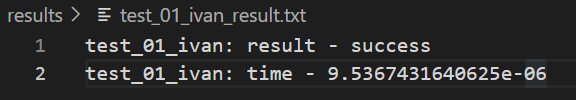
    def generate\_data(self):

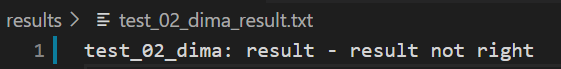
        return [faker.random\_int(), faker.random\_int()]

Создав класс SomeTest преподаватель должен определить входные данные для проверки и результат, который хочет получить по итогу обработки данных. Пример сделан с использованием библиотеки faker для генерации случайных значений.

В папке результатов создается файл проверки StyleGuide – flake8\_results.txt, и файлы проверки скриптов учеников. Пока сделано три формата вывода – успешно, не успешно и ошибка.







В итоге мы имеем удобный подход для проведения тестов обучающихся по языку программирования Python, который можно перевести на физические формулы популярных форматов типа – LaTeX, или другие языки программирования.