МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Балтийский федеральный университет имени Иммануила Канта»

ОНК «Институт высоких технологий»

ОТЧЁТ

о прохождении учебной практики по получению первичных

профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности

на базе Высшей школы компьютерных наук и прикладной математики образовательно-научного кластера "Институт высоких технологий"

Выполнили:

Конрат Климентий

Белов Андрей

Студенты очной формы обучения 3 курса

специальности 10.05.01 Компьютерная безопасность

специализация «Математические методы защиты информации»

Руководитель практики от университета

доцент ОНК «ИВТ» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Киршанова Е.А.

г. Калининград 2022 г.

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc107438183)

[Глава 1. ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ 4](#_Toc107438184)

Глава 2. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В PYTHON. [4](#_Toc107438186)

ТЕОРИЯ. [4](#_Toc107438186)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 5](#_Toc107438187)

[Глава 3. ВЕРСТКА ПРЕЗЕНТАЦИИ В СИСТЕМЕ LATEX 9](#_Toc107438188)

[Заключение 10](#_Toc107438190)

[Список литературы 11](#_Toc107438191)

## Введение

Вид практики – учебная практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности, далее Учебная практика.

Цель учебной практики: практические навыки владения LaTeX и объектно-ориентированного программирования в Python, а также улучшить навыки публичного выступления и защиты работы.

Задачи учебной практики:

* Запрограммировать одну из игр на языке Python, используя объектно-ориентированное программирование, представить реализацию.
* Объяснить игру и стратегии, которые можно использовать.
* Составить презентацию, объясняющую суть игры, методы реализации, используя систему верстки LaTeX.
* Подготовить 10 минутное выступление, опирающееся на презентацию. Ответить на возможные вопросы касательно решения.

## Глава 1. ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА ПРАКТИКУ

**Суть и задачи**

Выбрать и запрограммировать одну из игр.

На реализацию была выбрана игра “NerdleGame”.

Сама игра чем-то напоминает «Быки и коровы»

В последней необходимо угадать заранее выбранную игрой (или другим игроком) комбинацию за ограниченное количество попыток. В качестве загаданного слова может быть как последовательность букв и цифр, так и последовательность цветов и пиктограмм. В случае угадывания правильной позиции и самого символа угадывающим ироком, данный символ «помечается» быком, если загагулина верно угадана, но её позиция не верна — помечается коровой.

Разумеется, роли при такой игре не равны. Игрок-загадывающий принимает пассивное участие, в то время как угадывающий должен анализировать как свои предыдущие попытки так и варианты, которые ему «подсказывают» интуиция и его логика; другими словами второй игрок — активный игрок.

Возвращаясь к сути NerdleGame, загаданное слово есть последовательность цифр и знаков операций, включая символ «=» (равно). Сама последовательность в целом является истинным равенством, которое должно поместиться в заранее выбранное количество игровых клеток. То есть если размер игрового поля в клетках равна восьми, то и длина неизвестного слова равна этой же величине.

Для реализации желательно придерживаться ООП, что даст не только удобство и целостность программы, но и гибкость настройки самой игры. Отдельные функции, опять же для структурности, будут находиться в отдельном модуле.

Разумеется, для демонстрации понадобиться что-то поинтересней и наглядней чем «сухой» код — необходимо сделать презентацию, для чего будем использовать LaTeX. В качестве бонуса, будем использовать данный документ.

## Глава 2. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В PYTHON.

**ТЕОРИЯ**

**Объектно-ориентированное программирование (ООП)** - это подход к программированию, основанный на **объектах** и **классах**.

Объектно-ориентированное программирование основано на следующих ключевых концепциях:

* Классы и объекты
* Наследование
* Инкапсуляция
* Полиморфизм
* Абстрактные классы

**Класс - это шаблон или схема, из которой будут создаваться экземпляры объектов.** Это похоже на форму, из которой могут быть созданы объекты. Класс состоит из набора состояний (также известных как **атрибуты** или **свойства**) и **поведений** (также известных как **методы**).

**Объект —** это **экземпляркласса.** Атрибуты объекта, сохраненные с использованием переменных для хранения данных, и поведение объекта реализованы в виде функций или процедур и могут использоваться для выполнения операций с данными.

**Наследование** — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью.

**Инкапсуляция**— свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе.

**Полиморфизм подтипов** (либо просто полиморфизм) — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

**Абстрагирование** означает выделение значимой информации и исключение из рассмотрения незначимой.

В представленной программе будут использоваться некоторые из концепций, рассказанных выше. А именно – классы и объекты, поэтому о них далее будет идти речь.

В Python класс определяется с помощью ключевого слова – *class*. Это похоже на то, как определяются функции с помощью *def*. Как только класс определен, его можно вызывать. Вызов класса немного отличается от вызова функции. Когда вызывается функция, на выходе получается возвращаемое значение функции. Когда вызывается класс, на выходе получается **объект,** тип**которого соответствует этому классу**.

Действия над классом выполняются с помощью методов. **Метод**- это функция, которая живет в классе и, в частности, **работает с экземплярами этого класса**. Методы, как правило, либо получают доступ к данным из экземпляра класса, либо изменяют данные в экземпляре класса.

Прежде чем создавать какие-либо новые методы в классе, необходимо создать метод «\_\_init\_\_». Он позволяет принимать аргументы нового класса. Что еще более важно, данный метод позволяет присваивать начальные значения различным атрибутам в экземплярах класса.

**ПРАКТИЧЕСКАЯ ЧАСТЬ**

В модуле Calculus имеем две функции

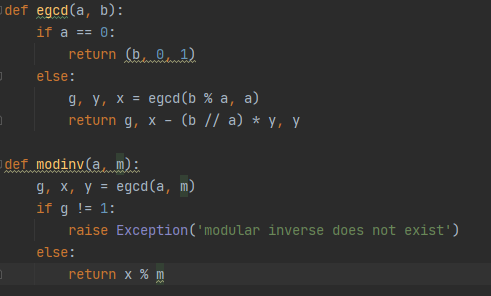


Рисунок 1. Модуль Calculus

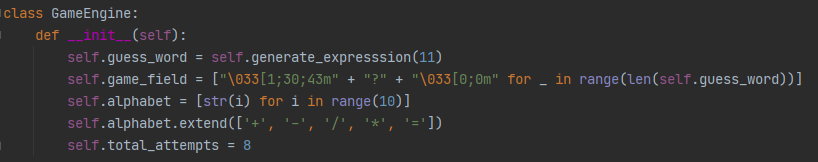
Первая представляет из себя расширенный алгоритм Евклида. В то время, как вторая функция вычисляет обратный элемент по модулю, на основе резульата работы вышеописанной функции. Эти функции напрямую к проекту не относятся так как нужны для переопределения оператора деления. Удобно было вынести их в отдельный модуль.

Следующим классом для объяснения будет «OperationDefiner» или «Определитель операции», который определяет то, как будут себя вести операции, точнее говоря то, что будет происходить с двумя элементами, как и какой результат мы в итоге получим.

Первоначально, в конструкторе, создаём словарь, где каждому ключу строке-операции ставим в соответствие выражение, которое будет соответствовать этой операции. Как мы видим, в констурктор так же передаётся аргумент — операция. В действительности, если мы не генерируем выражение, а работаем с уже имеющимся, то в констуркторе мы, опустившись в условие «if» прекратим выполнение. Иначе мы, конечно, сделаем то же самое, только предварительно вытянем жребий на ту или иную операцию. Магические методы «call» и «str» позволят нам использовать экземпляр класса как функцию и представлять его как строку соответственно.

Рисунок 2. Класс GameEngine, конструктор

Далее идёт массивный класс игрового движка, начнём с конструктора:

Рисунок 3. Класс GameEngine, конструктор

При создании экземпляра мы сразу определяем несколько полей: алфавит, игровое поле (по-умолчанию полностью состоящее из знаков «?») и генерируем слово изпользуя ниже описанный метод.

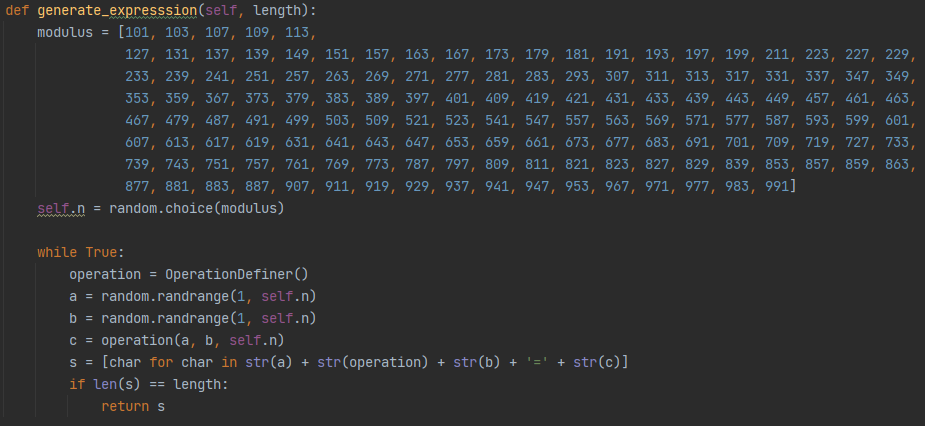
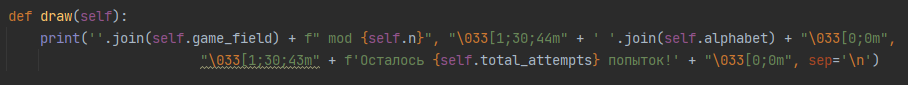


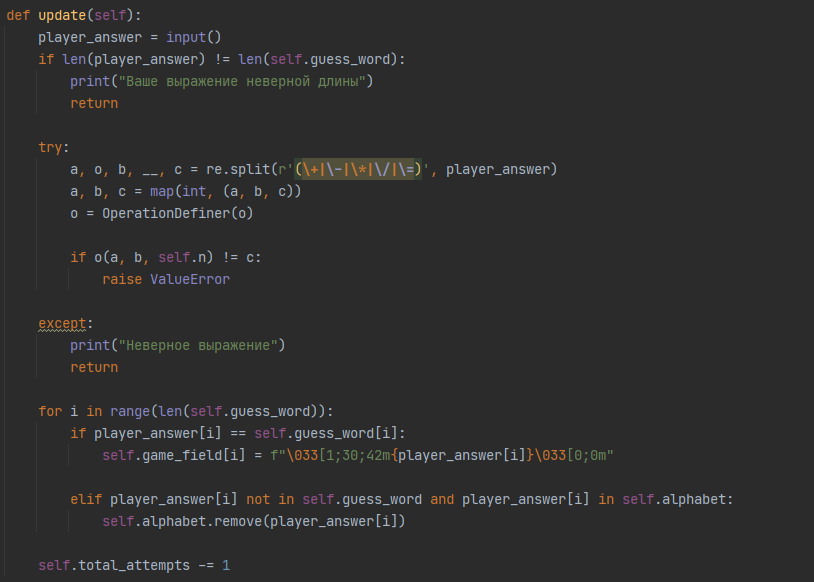
Рисунок 4. Класс GameEngine, генератор выражений

Метод принимает в себя единственный аргумент — длину угадываемого выражения, которое задаётся не пользователем, а программно ещё в конструкторе. Так как наже выражение является истинным выражением в поле, то желательно не ждать вечность, пока выражение будет генерироваться со знаком «/». Поэтому берём на выбор, для генерации поля, одно из простых чисел с тремя цифрами (можно и большие/меньшие). Далее обращаемся к определителью операций чтобы знать, что вообще будет происходить с выражением. Выбираем два счастливых числа из выбранного поля, и считаем, что будет с ними в результате выполнения определившийся операции; генерируем выражение в строковом виде и проверяем равна ли его длина изначальной длине загаданного слова (определённой в конструкторе GameEngine). Если длина не совпала, повторяем.

На данном этапе игровые данные готовы для игры, но пользователь-игрок ничего не увидит. Правильно, ведь смотреть не на что — вывода нет. Смотрим реализацию: (Рисунок 5)

Рисунок 5. Класс GameEngine, отображение

По сути, что хотим увидеть, то и пишем. Выводим в одном замечательном print’е игровое поле (наши угаданные и неугаданные символы), модуль, оставшийся доступный алфавит и количество оставшихся попыток, чтобы пользователь понимал, когда дело пахнет жареным. Разумеется, осталось завернуть это в яркую упаковку разукрашивающих служебных символов для подсветки нашей игры.

Рисунок 6. Класс GameEngine, обновление

Чтобы игрок мог выиграть (или проиграть) данные прогресса последнего надо обновлять — реализуем следующим образом (Рисунок 6). Считываем пользовательский ввод, и даём игроку вводить до тех пор, пока длина его выражения не совпадёт с загаданной, попытку мы у него не заберём, а сообщением предупредим.

Так же мы его предупредим его в том случае, если он будет считать, что выражение «0 \* 0 = 2» или любое ему подобное верное, как видно из кода ниже. Здесь мы используем библиотеку регулярных выражений, чтобы разбить пользовательский ввод по сосовляющим — операторы отдельно, операнды отдельно; проверить верно ли выражение по своей сути. Если всё в порядке, то для всех угаданных символов на своей позиции заменяем надоевший знак «?» на тот, что за ним «стоит». Если пользовательский символ вообще не присутствует в загаданном выражении — убираем его из доступного алфавита.

Не забудем лишить игрока одной попытки.

К слову, было бы обидно, если бы мы не дали возможность выиграть в игре. И вот как это будет работать:



Рисунок 7. Класс GameEngine, проверка на победу

Принцип прост — если знаков «?» на поле нет — значит всё угадано верно и пользователь-игрок победил. Иначе… Не победил, я полагаю.

В конце концов надо запустить нашу игру, что мы и сделаем с помощью игрового цикла:

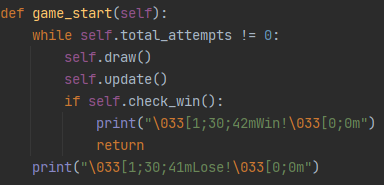
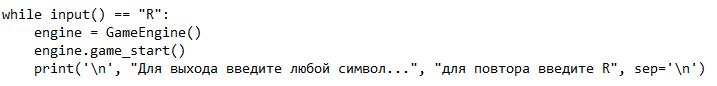


Рисунок 8. Класс GameEngine, запуск игры

Игра будет идти до того момента, пока игрок имеет попытки, в силу того, что это условие проверяется позже проверки на победу, игрок сможет победить даже при последней попытке! Всё очень просто — каждую игровую итерацию мы отрисовывем поле, обновляем его и проверяем победу игрока. При любом раскладе выложим в терминале игроку красивейшую красную табличку со словом «Lose!» или, если нас переиграли, внушающую гордость зелёную надпись «Win!» - коротко и ясно. Есть модификация, позволяющая перезапускать игру без повторного запуска главного файла (Рисунок 9).

 В конечном итоге игра выглядит так как на следующих рисунках:

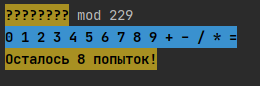


Рисунок 9. Игровое поле



Рисунок 10. Ошибка длины выполнения (по умолчанию — 8)

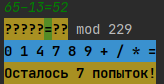


Рисунок 11. Пример игры после одного игрового такта.

## 

Рисунок 12. Поражение

## 

Рисунок 13. Победа

## Глава 3. ВЕРСТКА ПРЕЗЕНТАЦИИ В СИСТЕМЕ LATEX

С помощью редактора для LaTeX мы создали презентацию. Был использован пакет Beamer.

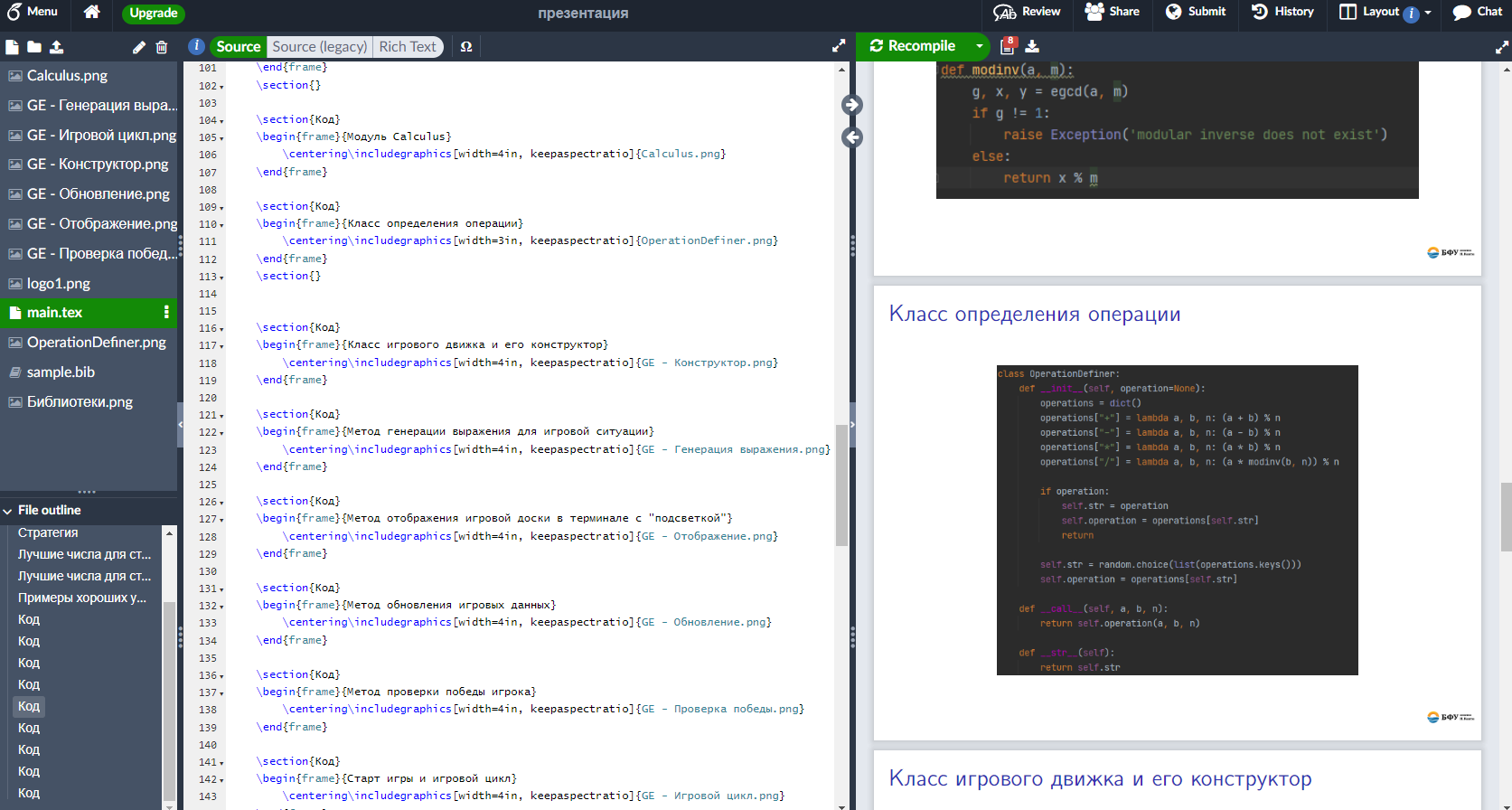


Рисунок 14. Пример интерфейса

## **Дополнение. Стратегии.**

Решение числового уравнения сразу может быть более сложным.  
Существует две основные стратегии в играх такого рода:  
 - первая стратегия заключается в том чтобы как можно использовать уникальных цифр в первых двух попытках.  
 - вторая стратегия заключается в том чтобы использовать как можно больше операторов в первых двух попытках.

Нетрудно догадаться, что первая стратегия самая лучшая для нас ведь в игре всего один оператор, не считая «равно». Для краткости, приведу лишь один пример крайне успешного выражения, фильтрующие циры и знак: 48 - 36 = 12

За расширенной информацией можно обратиться к презентации.

## Заключение

В течение учебной практики все поставленные задачи были выполнены, а цели достигнуты. Мы повторили некоторые моменты при работе с Python и улучшили наши умения при создании презентации на LaTeX. Отдельно отметить хотелось работу с регулярными выражениями (модуль re) – теперь эту вещь понимать легче, когда непосредственно на практике её применил. Должно быть мы научились делать некоторые вещи в команде и, не без этого, вникли в азы такого раздела математики, как теория игр.

**Интернет – ресурсы:**

1. Классы в Python - https://docs.python.org/3/tutorial/classes.html
2. Модуль re - https://habr.com/ru/company/skillbox/blog/552360/
3. ООП - [https://ru.wikipedia.org](https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование)