ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

Специальность 09.02.07

«Информационные системы и программирование»

Практическое занятие №6

Руководитель:

\_\_\_\_\_\_\_\_ А.А. Отыргашева

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка, дата)

Выполнил:

Студент группы ИС(ПРО)-31

\_\_\_\_\_\_\_\_ Д. А. Коробкова

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Абакан 2025 г.

**Фото конспекта**

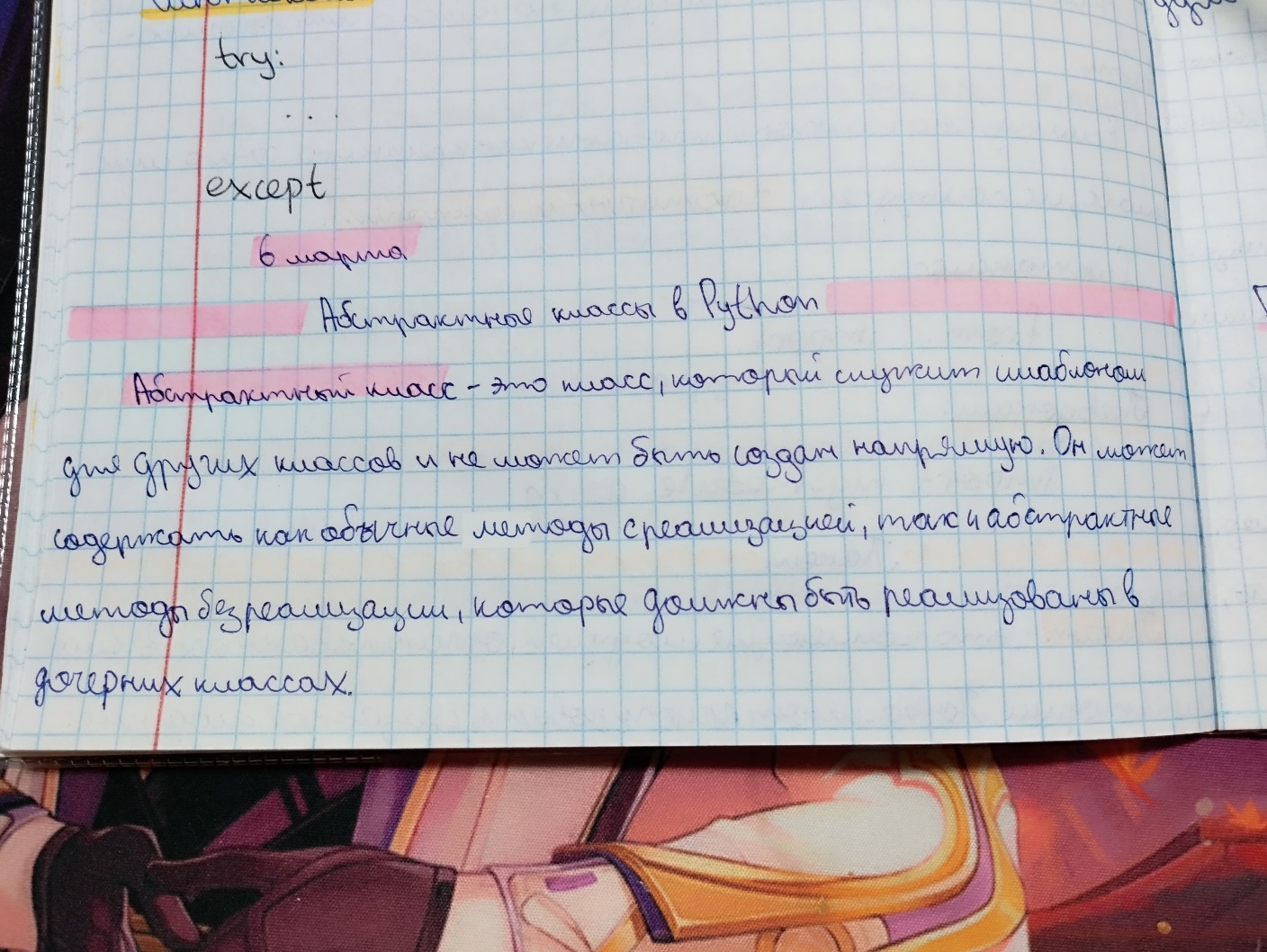


Рисунок 1 – Первая страница конспекта

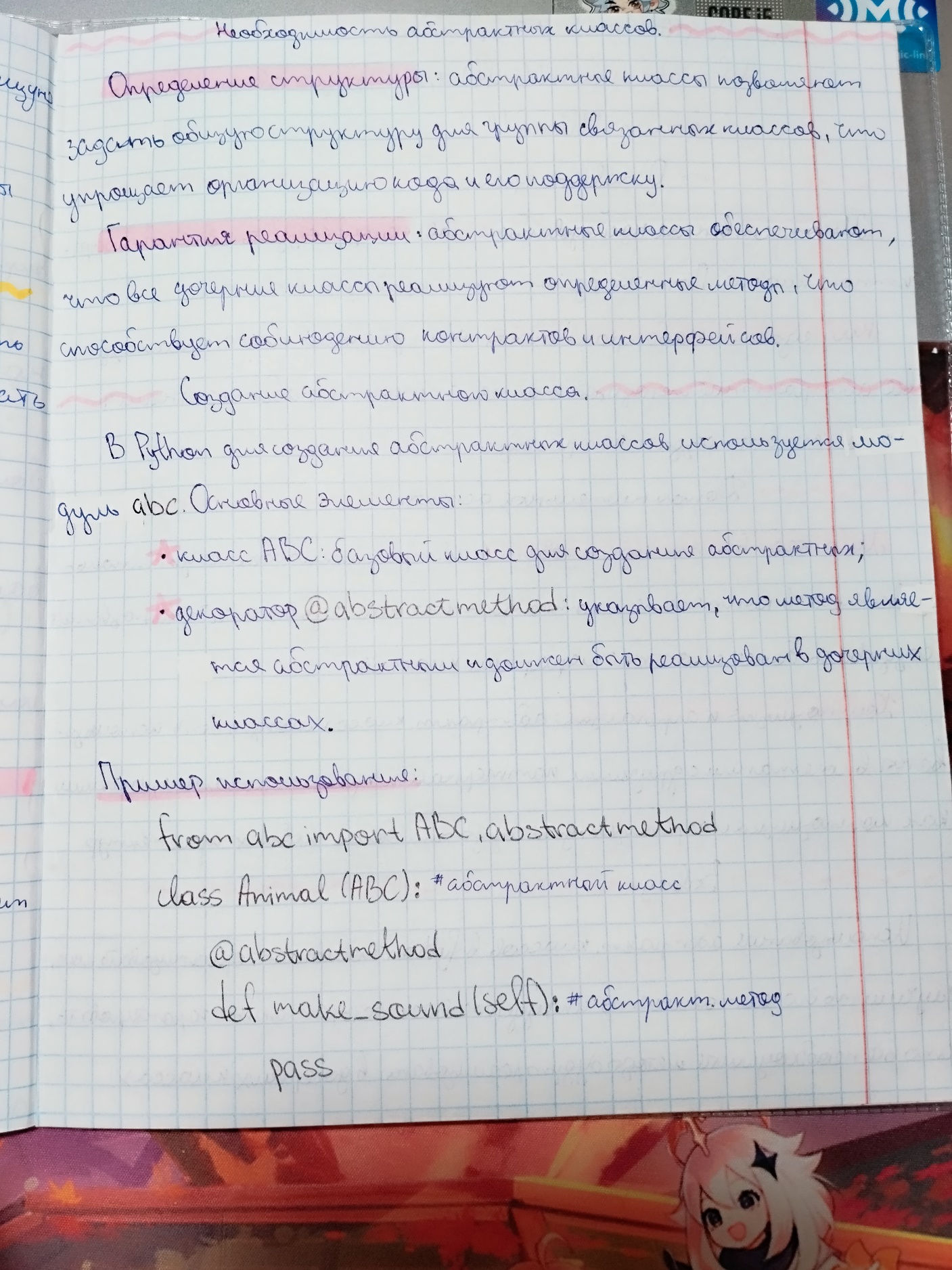


Рисунок 2 – Вторая страница конспекта

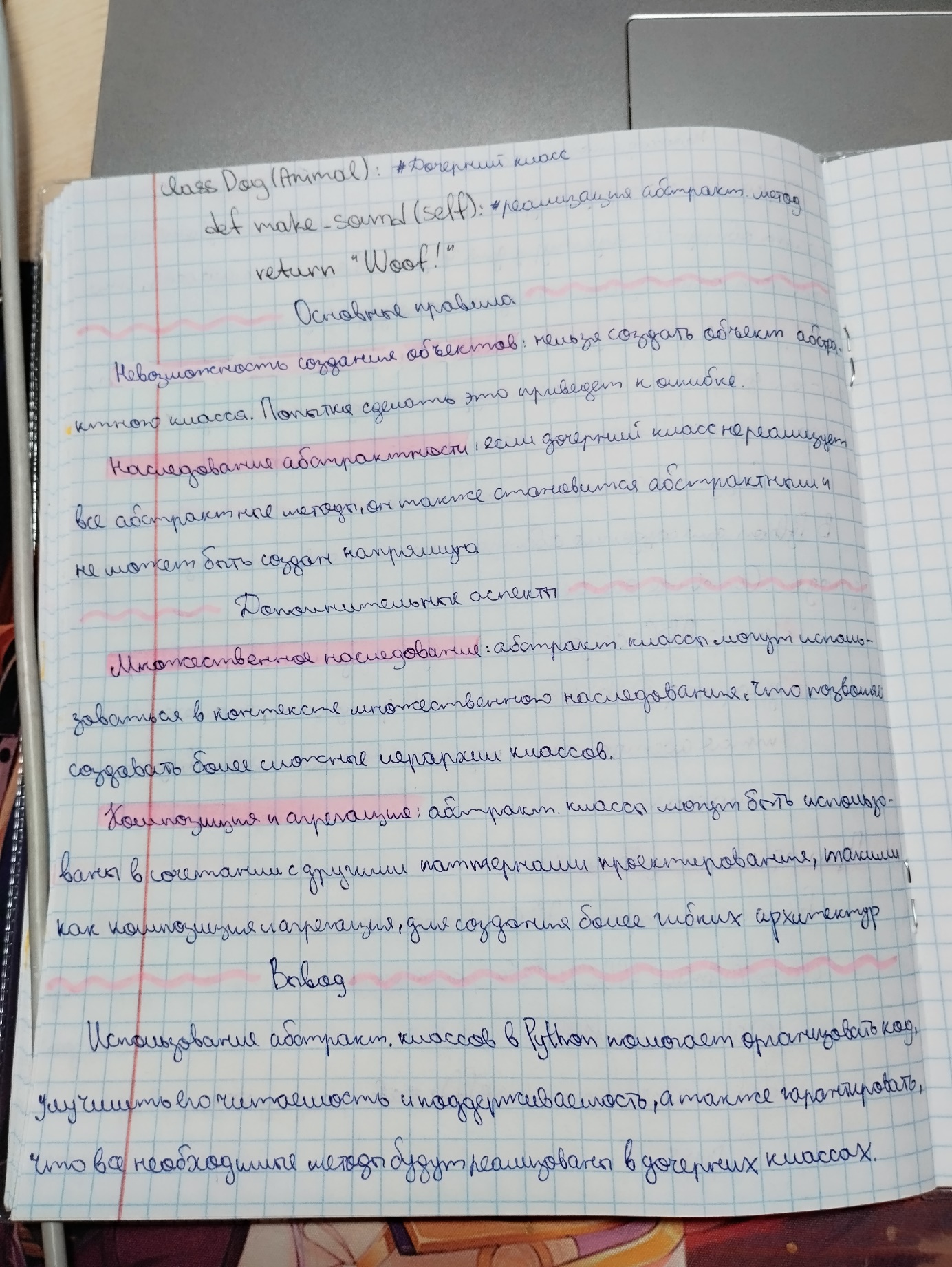


Рисунок 3 – Третья страница конспекта

**Задание №1**

Вариант 2: Иерархия "Персонажи игры"

1. Создайте абстрактный класс Character с абстрактными методами:
   * attack() — для атаки.
   * defend() — для защиты.
2. Реализуйте дочерние классы:
   * Warrior (воин) — принимает уровень силы.
   * Mage (маг) — принимает уровень маны.
   * Archer (лучник) — принимает точность стрельбы.
3. В основном скрипте создайте объекты каждого персонажа и продемонстрируйте их методы.

**Код с комментариями**

# Импорт класса ABC и декоратора abstractmethod из модуля abc

from abc import ABC, abstractmethod

# Создание абстрактного класса character

class character(ABC):

# Использование декоратора для определения абстракт. методов в абстракт. классах

@abstractmethod

# Создание абстрактного метода для обозначения атаки

def attack(self):

# Использование оператора pass для реализации метода в дочерних классах

pass

# Использование декоратора для определения абстракт. методов в абстракт. классах

@abstractmethod

# Создание абстрактного метода для обозначения защиты

def defend(self):

# Использование оператора pass для реализации метода в дочерних классах

pass

# Создание дочернего класса для персонажа «Воин»

class warrior(character):

# Создание функции для определения силы

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

# Присваивание значения параметра сила атрибуту self.strength

self.strength = strength

# Создание функции для вывода показателя атаки

def attack(self):

# Возвращение строки

return f"Воин атакует с показателем {self.strength \* 3}"

# Создание функции для вывода показателя защиты  
def defend(self):

# Возвращение строки

return f"Воин защищается с показателем: {self.strength}"

# Создание дочернего класса для персонажа «Маг»

class mage(character):

# Создание функции для определения маны

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

# Присваивание значения параметра мана атрибуту self.mana

self.mana = mana

# Создание функции для вывода показателя атаки

def attack(self):

# Возвращение строки

return f"Маг атакует с показателем: {self.mana \* 4}"

# Создание функции для вывода показателя защиты

def defend(self):

# Возвращение строки

return f"Маг защищается с показателем: {self.mana / 2}"

# Создание дочернего класса для персонажа «Лучник»

class archer(character):

# Создание функции для определения точности

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

# Присваивание значения параметра точность атрибуту self.accuracy

self.accuracy = accuracy

# Создание функции для вывода показателя атаки

def attack(self):

# Возвращение строки

return f"Лучник стреляет с показателем: {self.accuracy}"

# Создание функции для вывода показателя защиты

def defend(self):

# Возвращение строки

return f"Лучник защищается с показателем: {self.accuracy/2}"

# Создание объекта Warrior класса warrior()

Warrior = warrior(strength=10)

# Создание объекта Mage класса mage()

Mage = mage(mana=20)

# Создание объекта Archer класса archer()

Archer = archer(accuracy=15)

# Вывод атаки воина  
print(Warrior.attack())

# Вывод защиты воина  
print(Warrior.defend())

# Вывод пустой строки

print()

# Вывод атаки мага

print(Mage.attack())

# Вывод защиты мага

print(Mage.defend())

# Вывод пустой строки

print()

# Вывод атаки лучника  
print(Archer.attack())

# Вывод защиты лучника

print(Archer.defend())

**Демонстрация выполнения методов**

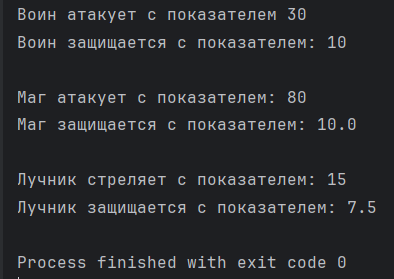


Рисунок 4 – Выполнение кода