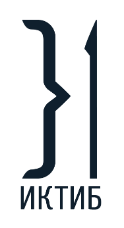
МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ



**ЛАБОРАТОРНАЯ РАБОТА № 4**

**ОТЧЁТ**

по дисциплине

**«Разработка прикладных решений на языке Python»**

Выполнил:

Студент группы

КТмо2-16

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Руденко К. Д. |
|  | *подпись* |  |

Проверил:

Доцент кафедры САиТ, к.т.н.

|  |  |  |
| --- | --- | --- |
|  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Лапшин В. С. |
|  | *подпись* |  |

Оценка

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

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**4 лабораторная работа.**

**Руденко Константин Дмитриевич – 3 вариант**

**Часть 1.**

**Вариант 3 :**

**Теоретический вопросы:**

1. Проблемы императивного (процедурного) подхода в программировании.

**Ответ: Код быстро разрастается из-за чего сложно поддерживать. Логика «размазана» по функциям поэтому трудно понять, где что меняется. Повторяющийся код поэтому высокая вероятность ошибок и тяжело понять что к чему относиться. Сложно адаптировать под изменения - каждое новое требование ломает старое.**

1. Понятие объекта и черного ящика (примеры). Парадигма объектно-ориентированного программирования. Цель объектно-ориентированного подхода. Как выглядит разработка приложения в объектно-ориентированном стиле?

**Ответ: Объект — это сущность, у которой есть данные (атрибуты) и поведение (методы). «Черный ящик» мы знаем, что объект делает, но не вникаем, как он это делает внутри. Например: объект автомобиль - мы жмём газ, а как работает двигатель – скрыто.**

**ООП (объектно-ориентированное программирование): программа строится как набор взаимодействующих объектов. То есть у нас есть классы – это объекты и у них какие-то поля они могут быть публичные могут приватными, и мы может что-то с ними делать с помощью функций. Цель - упростить разработку: код становится модульным, понятным и расширяемым. Разработка в ООП-стиле: думаем не «какие шаги выполнить», а «какие объекты будут и как они общаются».**

1. Атрибуты объекта и атрибуты класса: ключевые различия и варианты использования.

**Ответ: Атрибут объекта - уникален для каждого объекта.**

**Пример: у каждой машины свой цвет.**

**Атрибут класса - общий для всех объектов.**

**Пример: у всех машин есть число колёс = 4.**

**Использование:**

**Объектные атрибуты → для индивидуальных характеристик.**

**Классовые атрибуты → для общих свойств или «настроек по умолчанию».**

**Задача 1. Банковские вклады**

**Изображение выглядит как текст, алгебра, чек

Автоматически созданное описание**

from astroid import Global  
import random  
  
  
class Deposit:  
 def \_\_init\_\_(self, amount, rate, period):  
 self.amount = amount  
 self.rate = rate  
 self.period = period  
  
 def calculate\_profit(self):  
 raise NotImplementedError("Нужно реализовать в подклассе")  
  
class FixedDeposit(Deposit):  
 def calculate\_profit(self):  
 return self.amount \* self.rate \* self.period  
  
class BonusDeposit(Deposit):  
 def \_\_init\_\_(self, amount, rate, period, bonus\_threshold, bonus\_rate):  
 super().\_\_init\_\_(amount, rate, period)  
 self.bonus\_threshold = bonus\_threshold  
 self.bonus\_rate = bonus\_rate  
  
 def calculate\_profit(self):  
 profit = self.amount \* self.rate \* self.period  
 if self.amount >= self.bonus\_threshold:  
 profit += profit \* self.bonus\_rate  
 print("Бонус!")  
 return profit  
  
class CompoundDeposit(Deposit):  
 def calculate\_profit(self):  
 return self.amount \* ((1 + self.rate) \*\* self.period - 1)  
  
# Фабрика вкладов  
class DepositFactory:  
 @staticmethod  
 def create\_deposit(deposit\_type, \*\*kwargs):  
 if deposit\_type == "Срочный вклад":  
 return FixedDeposit(\*\*kwargs)  
 elif deposit\_type == "Бонусный вклад":  
 return BonusDeposit(\*\*kwargs)  
 elif deposit\_type == "Вклад со сложным процентом":  
 return CompoundDeposit(\*\*kwargs)  
 else:  
 raise ValueError("Неизвестный тип вклада")  
  
  
# --- Пример использования ---  
if \_\_name\_\_ == "\_\_main\_\_":  
 for i in range(10):  
 a = random.randint(5000, 100000) # сумма вклада  
 r = random.randint(100, 2000) / 10000 # ставка от 0.01 до 0.2  
 p = random.randint(1, 10) # срок в годах  
 b = random.randint(20000, 25000) # порог для бонуса  
  
 deposit1 = DepositFactory.create\_deposit(  
 "Срочный вклад", amount=a, rate=r, period=p  
 )  
 deposit2 = DepositFactory.create\_deposit(  
 "Бонусный вклад", amount=a, rate=r, period=p,  
 bonus\_threshold=b, bonus\_rate=0.09  
 )  
 deposit3 = DepositFactory.create\_deposit(  
 "Вклад со сложным процентом", amount=a, rate=r, period=p  
 )  
 l = 0  
 global obj  
 for dep in [deposit1, deposit2, deposit3]:  
 temp = dep.calculate\_profit()  
 if temp > l:  
 obj, l = dep, temp  
 print(f"Денюшки были под вкладом {dep.\_\_class\_\_.\_\_name\_\_}: и мы получили прибыль = {temp:.2f} руб. за {dep.period} лет.")  
  
 print(f"Самый выгодный вклад в данном случае - {obj.\_\_class\_\_.\_\_name\_\_} \n \n")  
  
  
# C:\Users\korudenko\PycharmProjects\botconfectioner\.venv2\Scripts\python.exe C:\Users\korudenko\PycharmProjects\Py3semestr2025RudenkoKTmo2-16\laba4\Task4-1.py   
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 7692.19 руб. за 3 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 8384.49 руб. за 3 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 8106.28 руб. за 3 лет.  
# Самый выгодный вклад в данном случае - BonusDeposit   
  
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 5626.17 руб. за 2 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 6132.53 руб. за 2 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 5799.46 руб. за 2 лет.  
# Самый выгодный вклад в данном случае - BonusDeposit   
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 39176.57 руб. за 3 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 42702.46 руб. за 3 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 47003.31 руб. за 3 лет.  
# Самый выгодный вклад в данном случае - CompoundDeposit   
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 11579.32 руб. за 3 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 12621.46 руб. за 3 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 12135.66 руб. за 3 лет.  
# Самый выгодный вклад в данном случае - BonusDeposit   
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 1618.68 руб. за 4 лет.  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 1618.68 руб. за 4 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 1756.40 руб. за 4 лет.  
# Самый выгодный вклад в данном случае - CompoundDeposit   
  
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 76069.35 руб. за 6 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 82915.59 руб. за 6 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 108900.41 руб. за 6 лет.  
# Самый выгодный вклад в данном случае - CompoundDeposit   
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 10027.08 руб. за 4 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 10929.52 руб. за 4 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 10724.44 руб. за 4 лет.  
# Самый выгодный вклад в данном случае - BonusDeposit  
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 36998.19 руб. за 3 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 40328.03 руб. за 3 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 43319.66 руб. за 3 лет.  
# Самый выгодный вклад в данном случае - CompoundDeposit   
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 17418.47 руб. за 5 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 18986.13 руб. за 5 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 18880.12 руб. за 5 лет.  
# Самый выгодный вклад в данном случае - BonusDeposit   
  
# Денюшки были под вкладом FixedDeposit: и мы получили прибыль = 65925.64 руб. за 10 лет.  
# Бонус!  
# Денюшки были под вкладом BonusDeposit: и мы получили прибыль = 71858.95 руб. за 10 лет.  
# Денюшки были под вкладом CompoundDeposit: и мы получили прибыль = 113806.04 руб. за 10 лет.  
# Самый выгодный вклад в данном случае - CompoundDeposit   
  
# Process finished with exit code 0

Часть 2.

**Вариант 3 :**

**Задача 1. Установка и использование модулей**

**Изображение выглядит как текст, Шрифт, алгебра, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Содержимое, созданное искусственным интеллектом, может быть неверным.**

import roman  
  
roman\_num = input("Введите римское число: ").upper() # X  
arabic\_num = int(input("Введите арабское число: ")) # 5  
  
a = roman.fromRoman(roman\_num)  
b = arabic\_num  
  
# Операции  
operations = {  
 "+": a + b,  
 "-": a - b,  
 "\*": a \* b,  
 "//": a // b if b != 0 else "деление на ноль"  
}  
  
# Выводим результаты  
print(f"Римское число {roman\_num} = {a}")  
print(f"Арабское число = {b}\n")  
  
for op, result in operations.items():  
 if isinstance(result, int) and result > 0: # только положительные можно в римские  
 try:  
 roman\_result = roman.toRoman(result)  
 except:  
 roman\_result = "нельзя перевести"  
 else:  
 roman\_result = "нельзя перевести"  
 print(f"{a} {op} {b} = {result} (римскими: {roman\_result})")

C:\Users\korudenko\PycharmProjects\botconfectioner\.venv2\Scripts\python.exe C:\Users\korudenko\PycharmProjects\Py3semestr2025RudenkoKTmo2-16\laba4\Task4-2-1.py

Введите римское число: x

Введите арабское число: 3

Римское число X = 10

Арабское число = 3

10 + 3 = 13 (римскими: XIII)

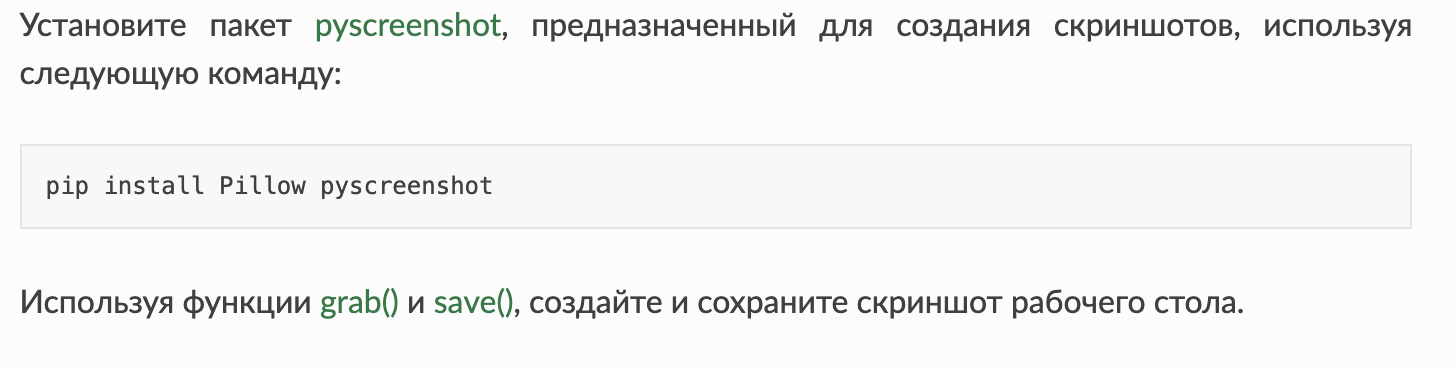
10 - 3 = 7 (римскими: VII)

10 \* 3 = 30 (римскими: XXX)

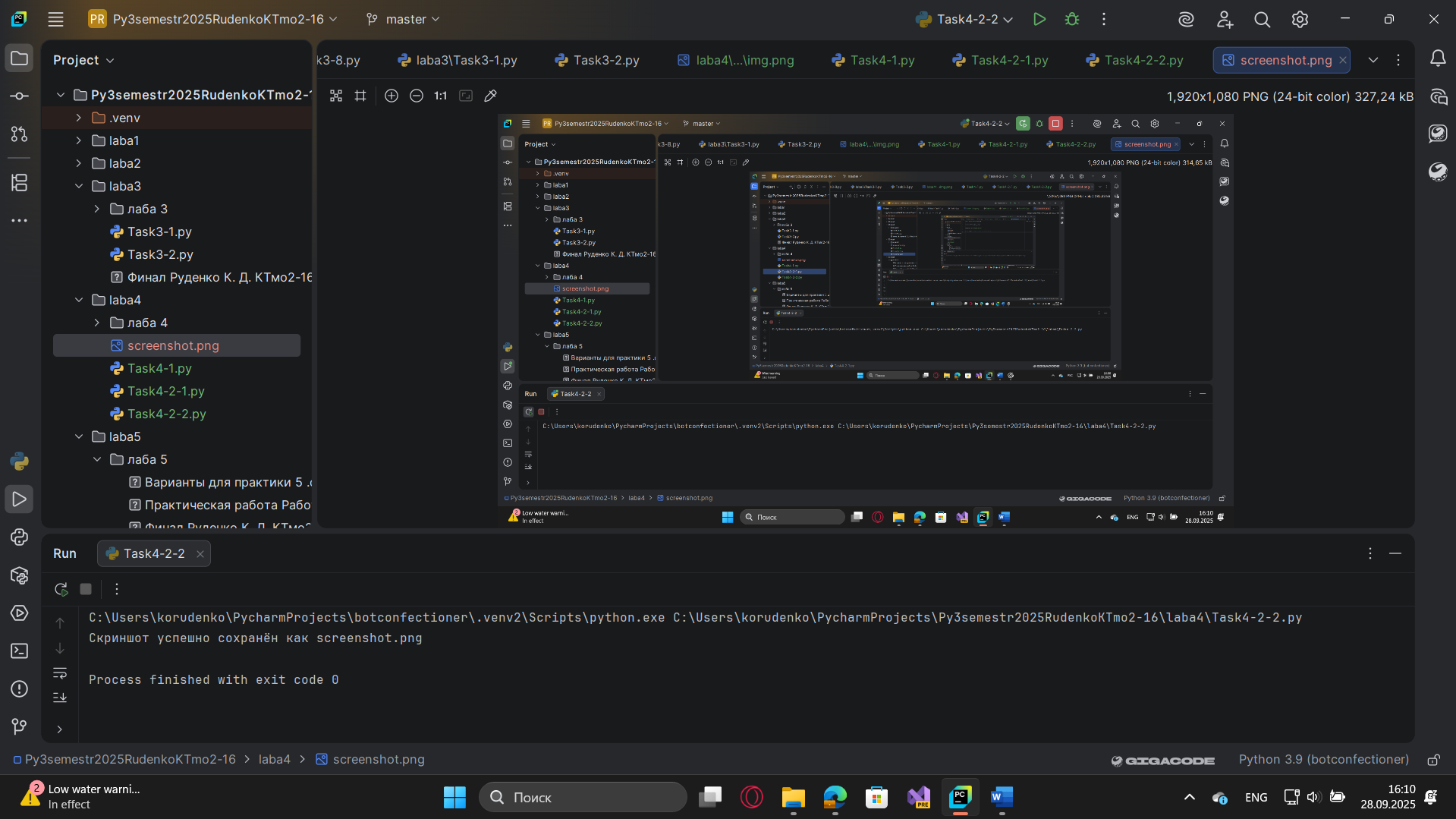
10 // 3 = 3 (римскими: III)

Process finished with exit code 0

**Задача 2. Установка и использование модулей**

****

import pyscreenshot as ImageGrab  
  
# Делаем скриншот всего экрана  
screenshot = ImageGrab.grab()  
  
screenshot.save("screenshot.png")  
  
print("Скриншот успешно сохранён как screenshot.png")

****

**Задача 3.Создание модулей**

**Изображение выглядит как текст, снимок экрана, Шрифт, алгебра

Автоматически созданное описание**

import math  
  
def gcd(a, b):  
 while b:  
 a, b = b, a % b  
 return abs(a)  
  
def lcm(a, b):  
 return abs(a \* b) // gcd(a, b) if a != 0 and b != 0 else 0  
  
def is\_prime(n):  
 if n <= 1:  
 return False  
 for i in range(2, int(math.sqrt(n)) + 1):  
 if n % i == 0:  
 return False  
 return True  
  
def reciprocal(n):  
 if n == 0:  
 raise ZeroDivisionError("Невозможно найти обратное число к нулю")  
 return 1 / n  
  
def square\_root(n):  
 if n < 0:  
 raise ValueError("Невозможно извлечь корень из отрицательного числа")  
 return math.sqrt(n)  
  
  
def gcd\_multiple(\*numbers):  
 if not numbers:  
 return 0  
 result = numbers[0]  
 for num in numbers[1:]:  
 result = gcd(result, num)  
 return result  
  
  
def lcm\_multiple(\*numbers):  
 if not numbers:  
 return 0  
 result = numbers[0]  
 for num in numbers[1:]:  
 result = lcm(result, num)  
 return result  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Тестирование модуля numbers.py:")  
 print(f"НОД(12, 18) = {gcd(12, 18)}")  
 print(f"НОК(12, 18) = {lcm(12, 18)}")  
 print(f"Простое число 17: {is\_prime(17)}")  
 print(f"Простое число 15: {is\_prime(15)}")  
 print(f"Обратное число к 4: {reciprocal(4)}")  
 print(f"Корень из 16: {square\_root(16)}")  
  
 print(f"\nНОД(24, 36, 48) = {gcd\_multiple(24, 36, 48, 60)}")  
 print(f"НОК(3, 4, 6) = {lcm\_multiple(3, 4, 6)}")

C:\Users\korudenko\PycharmProjects\botconfectioner\.venv2\Scripts\python.exe C:\Users\korudenko\PycharmProjects\Py3semestr2025RudenkoKTmo2-16\laba4\Task4-2-3.py

Тестирование модуля numbers.py:

НОД(12, 18) = 6

НОК(12, 18) = 36

Простое число 17: True

Простое число 15: False

Обратное число к 4: 0.25

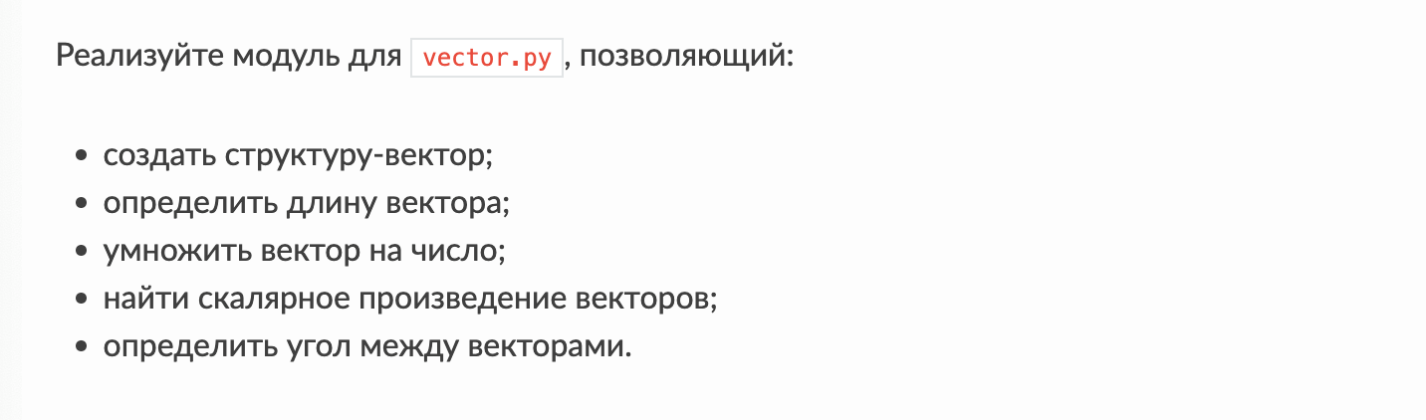
Корень из 16: 4.0

НОД(24, 36, 48) = 12

НОК(3, 4, 6) = 12

Process finished with exit code 0

**Задание 4. Создание модулей**

****

import math  
  
  
class Vector:  
 def \_\_init\_\_(self, \*components):  
 if not components:  
 raise ValueError("Вектор должен иметь хотя бы одну компоненту")  
 self.components = tuple(float(x) for x in components)  
  
 def \_\_repr\_\_(self):  
 return f"Vector{self.components}"  
  
 def \_\_str\_\_(self):  
 return f"Vector{self.components}"  
  
 def \_\_len\_\_(self):  
 return len(self.components)  
  
 def \_\_getitem\_\_(self, index):  
 return self.components[index]  
  
 def \_\_eq\_\_(self, other):  
 if not isinstance(other, Vector):  
 return False  
 return self.components == other.components  
  
 def magnitude(self):  
 return math.sqrt(sum(x \* x for x in self.components))  
  
 def \_\_mul\_\_(self, other):  
 if isinstance(other, (int, float)):  
 return Vector(\*(x \* other for x in self.components))  
 elif isinstance(other, Vector):  
 if len(self) != len(other):  
 raise ValueError("Векторы должны иметь одинаковую размерность")  
 return sum(a \* b for a, b in zip(self.components, other.components))  
 else:  
 raise TypeError("Можно умножать только на число или вектор")  
  
 def \_\_rmul\_\_(self, other):  
 return self \* other  
  
 def dot(self, other):  
 return self \* other  
  
 def angle(self, other):  
 if len(self) != len(other):  
 raise ValueError("Векторы должны иметь одинаковую размерность")  
  
 dot\_product = self.dot(other)  
 mag\_self = self.magnitude()  
 mag\_other = other.magnitude()  
  
 if mag\_self == 0 or mag\_other == 0:  
 raise ValueError("Нулевой вектор не имеет определенного направления")  
  
 cos\_angle = dot\_product / (mag\_self \* mag\_other)  
 cos\_angle = max(-1.0, min(1.0, cos\_angle))  
  
 return math.acos(cos\_angle)  
  
 def angle\_degrees(self, other):  
 return math.degrees(self.angle(other))  
  
 def \_\_add\_\_(self, other):  
 if not isinstance(other, Vector):  
 raise TypeError("Можно складывать только векторы")  
 if len(self) != len(other):  
 raise ValueError("Векторы должны иметь одинаковую размерность")  
 return Vector(\*(a + b for a, b in zip(self.components, other.components)))  
  
 def \_\_sub\_\_(self, other):  
 if not isinstance(other, Vector):  
 raise TypeError("Можно вычитать только векторы")  
 if len(self) != len(other):  
 raise ValueError("Векторы должны иметь одинаковую размерность")  
 return Vector(\*(a - b for a, b in zip(self.components, other.components)))  
  
 def normalize(self):  
 mag = self.magnitude()  
 if mag == 0:  
 raise ValueError("Невозможно нормализовать нулевой вектор")  
 return Vector(\*(x / mag for x in self.components))  
  
  
def distance(v1, v2):  
 if len(v1) != len(v2):  
 raise ValueError("Векторы должны иметь одинаковую размерность")  
 return (v1 - v2).magnitude()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Тестирование модуля vector.py:")  
  
 # Создание векторов  
 v1 = Vector(1, 2, 3)  
 v2 = Vector(4, 5, 6)  
 print(f"v1 = {v1}")  
 print(f"v2 = {v2}")  
  
 # Длина вектора  
 print(f"Длина v1: {v1.magnitude():.2f}")  
  
 # Умножение на число  
 v3 = v1 \* 2  
 print(f"v1 \* 2 = {v3}")  
  
 # Скалярное произведение  
 dot\_result = v1.dot(v2)  
 print(f"Скалярное произведение v1 и v2: {dot\_result}")  
  
 # Угол между векторами  
 angle\_rad = v1.angle(v2)  
 angle\_deg = v1.angle\_degrees(v2)  
 print(f"Угол между v1 и v2: {angle\_rad:.2f} радиан ({angle\_deg:.2f} градусов)")  
  
 # Сложение и вычитание  
 print(f"v1 + v2 = {v1 + v2}")  
 print(f"v1 - v2 = {v1 - v2}")  
  
 # Нормализация  
 v\_norm = v1.normalize()  
 print(f"Нормализованный v1: {v\_norm}, длина: {v\_norm.magnitude():.2f}")

C:\Users\korudenko\PycharmProjects\botconfectioner\.venv2\Scripts\python.exe C:\Users\korudenko\PycharmProjects\Py3semestr2025RudenkoKTmo2-16\laba4\Task4-2-4.py

Тестирование модуля vector.py:

v1 = Vector(1.0, 2.0, 3.0)

v2 = Vector(4.0, 5.0, 6.0)

Длина v1: 3.74

v1 \* 2 = Vector(2.0, 4.0, 6.0)

Скалярное произведение v1 и v2: 32.0

Угол между v1 и v2: 0.23 радиан (12.93 градусов)

v1 + v2 = Vector(5.0, 7.0, 9.0)

v1 - v2 = Vector(-3.0, -3.0, -3.0)

Нормализованный v1: Vector(0.2672612419124244, 0.5345224838248488, 0.8017837257372732), длина: 1.00

Process finished with exit code 0

Ссылка на гитхаб: [all/Py3semestr2025RudenkoKTmo2-16/laba3 at master · Kosten-73/all](https://github.com/Kosten-73/all/tree/master/Py3semestr2025RudenkoKTmo2-16/laba3)