

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ  
ФЕДЕРАЦИИ**  
**Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций**

**Институт цифрового развития**

**ОТЧЁТ**

**по лабораторной работе**

Дисциплина: «Объектно – ориентированное программирование»

Выполнил: студент 3 курса

группы ИВТ-б-о-21-1

Уланбекова Айканыш Уланбековна

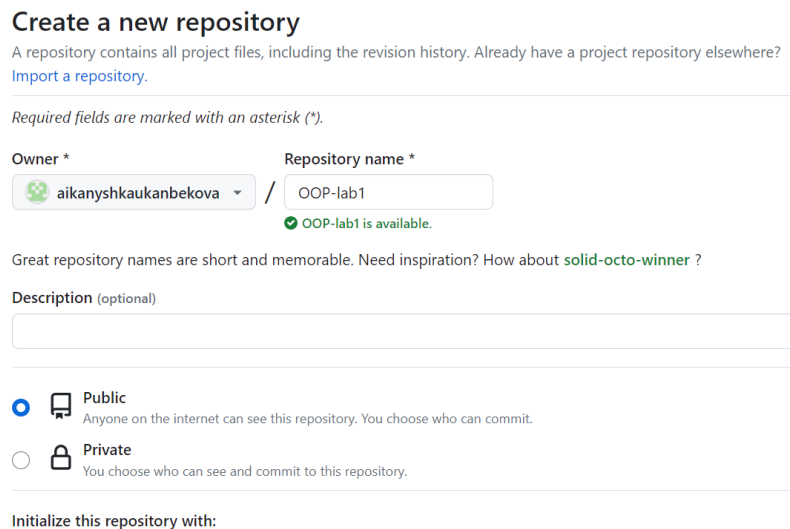
Ставрополь 2022

# Элементы объектно-ориентированного программирования в языке Python

**Цель работы:** приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

## Порядок выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

---

*Required fields are marked with an asterisk (\*).*

**Owner \*** aikanyshkaukanbekova / **Repository name \*** OOP-lab1  
✔ OOP-lab1 is available.

Great repository names are short and memorable. Need inspiration? How about [solid-octo-winner](#) ?

**Description (optional)**

---

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

---

Initialize this repository with:

Рисунок 1. Создание репозитория

2. Выполните клонирование созданного репозитория.

```
C:\Users\User\Documents\3,1 курс\Объектно-ориентированное\лаб 1>git clone https://github.com/aikanyshkaukanbekova/OOP-lab1
Cloning into 'OOP-lab1'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
C:\Users\User\Documents\3,1 курс\Объектно-ориентированное\лаб 1>_
```

Рисунок 2. Клонирование репозитория

3. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

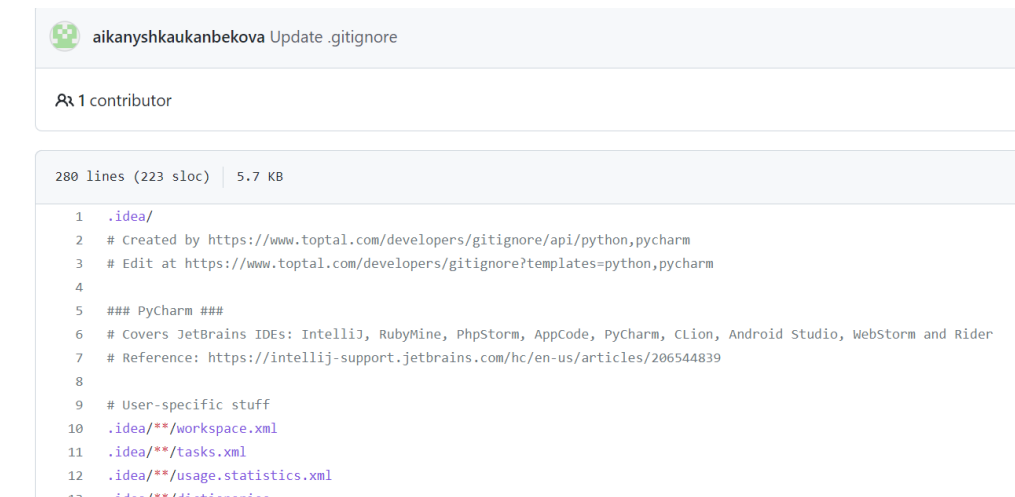


Рисунок 3. Дополнение файла .gitignore

## Практическая часть:

### Вариант 19

#### Задание 1

19. Число сочетаний по  $k$  объектов из  $n$  объектов ( $k < n$ ) вычисляется по формуле

$$C(n, k) = n! / ((n - k)! \times k!) \quad (2)$$

Поле first — целое положительное число,  $k$ ; поле second — положительное целое число,  $n$ .

Реализовать метод `combination()` — вычисление  $C(n, k)$ .

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math

class Pair:
    """
    Класс, хранящий введенные числа k и n в полях first и second
    """

    def __init__(self, first, second):
        """
        Конструктор класса, принимает два параметра, валидирует их и
        сохраняет в поля
        """
        # Удостоверимся, что first является целым числом
        if not isinstance(first, int):
            raise TypeError("Значение first должно быть целым положительным
числом")

        # Удостоверимся, что second является целым числом
        if not isinstance(second, int):
            raise TypeError("Значение second должно быть целым положительным
числом")

        # Удостоверимся, что first является положительным числом
        if first < 0:
            raise ValueError("Значение first должно быть положительным")
```

```

        # Удостоверимся, что second является целым числом
        if second < 0:
            raise ValueError("Значение second должно быть положительным")

        # Удостоверимся, что число first больше second
        if first > second:
            raise ValueError("Значение first должно быть меньше либо равно
чем second")

        # Записываем значения в поля
        self.first = first
        self.second = second

    def combination(self):
        """
        Метод высчитывает значения по формуле, с использованием переданных в
конструкторе значений
        """
        return math.factorial(self.second) / (math.factorial(self.second -
self.first) * math.factorial(self.first))

    def display(self):
        """
        Метод выводит экземпляра класса в консоль с указанием переданных в
конструктор значений полей first и second
        """
        print(f"({self.first}, {self.second})")

    @classmethod
    def read(cls):
        """
        Статический метод для создания экземпляра класса с запрашиванием
значений в консоли
        """
        k = int(input("Введите число k: "))
        n = int(input("Введите число n: "))

        return cls(k, n)

def make_pair(first, second):
    """
    Функция создания экземпляра класса Pair, принимая значения полей как
аргументы
    """
    return Pair(first, second)

if __name__ == '__main__':
    # Создаем экземпляр класса, запрашивая значения через консоль
    pair = Pair.read()
    # Отображаем созданный объект в консоли
    pair.display()
    # Выводим посчитанное по формуле значение с использованием метода
combination
    print(pair.combination())

```

```
Введите число k: 5
Введите число n: 10
(5, 10)
252.0
```

Рисунок 1. Результат 1 задания

## Задание 2

4. Создать класс Triangle для представления треугольника. Поля данных должны включать углы и стороны. Требуется реализовать операции: получения и изменения полей данных, вычисления площади, вычисления периметра, вычисления высот, а также определения вида треугольника (равносторонний, равнобедренный или прямоугольный).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math

class Triangle:
    """
    Класс "Треугольник" хранящий в себе информацию о длине трех сторон и
    углов
    """

    def __init__(self, a, b, c):
        """
        Конструктор класса, принимающий значения сторон a, b и c
        """
        self.a = a
        self.b = b
        self.c = c
        # Вызываем приватный метод для подсчета углов треугольника
        self._calculate_degrees()

    def get_a(self):
        """
        Возвращает значение стороны a
        """
        return self.a

    def get_b(self):
        """
        Возвращает значение стороны b
        """
        return self.b

    def get_c(self):
        """
        Возвращает значение стороны c
        """
        return self.c

    def set_a(self, value):
        """
        Устанавливает значение стороны a
        """
```

```

        """
        self.a = value
        # Вызываем приватный метод для подсчета углов треугольника
        self._calculate_degrees()

def set_b(self, value):
    """
    Возвращает значение стороны b
    """
    self.b = value
    # Вызываем приватный метод для подсчета углов треугольника
    self._calculate_degrees()

def set_c(self, value):
    """
    Возвращает значение стороны c
    """
    self.c = value
    # Вызываем приватный метод для подсчета углов треугольника
    self._calculate_degrees()

def area(self):
    """
    Возвращает площадь треугольника
    """
    s = (self.a + self.b + self.c) / 2
    # Вычисляем площадь треугольника по формуле
    # math.sqrt - функция квадратного корня
    return math.sqrt(s * (s - self.a) * (s - self.b) * (s - self.c))

def perimeter(self):
    """
    Возвращает периметр треугольника
    """
    return self.a + self.b + self.c

def height_a(self):
    """
    Возвращает высоту стороны a
    """
    return 2 * self.area() / self.a

def height_b(self):
    """
    Возвращает высоту стороны b
    """
    return 2 * self.area() / self.b

def height_c(self):
    """
    Возвращает высоту стороны c
    """
    return 2 * self.area() / self.c

def print_type(self):
    """
    Метод выводит в консоль тип треугольника
    Если тип не удалось вывести - не выводит ничего
    """
    # У равностороннего треугольника все три стороны равны между собой
    if self.a == self.b == self.c:
        print("Треугольник равносторонний")
    # У равнобедренного треугольника два катета равны
    elif self.a == self.b or self.a == self.c or self.b == self.c:

```

```

        print("Треугольник равнобедренный")
        # У прямоугольного треугольника один угол равен 90 градусов
        elif self.first_angle_degree == 90 or self.second_angle_degree == 90
or self.third_angle_degree == 90:
            print("Треугольник прямоугольный")

    def _calculate_degrees(self):
        """
        Приватный метод для подсчета градусов всех трех углов.
        Учитывая длины сторон a, b и c взятых из полей класса
        """
        self.first_angle_degree = math.degrees(
            math.acos((self.b ** 2 + self.c ** 2 - self.a ** 2) / (2 * self.b
* self.c)))
        self.second_angle_degree = math.degrees(
            math.acos((self.a ** 2 + self.c ** 2 - self.b ** 2) / (2 * self.a
* self.c)))
        self.third_angle_degree = math.degrees(
            math.acos((self.a ** 2 + self.b ** 2 - self.c ** 2) / (2 * self.a
* self.b)))

    def display(self):
        """
        Метод выводит в консоль значения сторон треугольника
        """
        print(f"(a: {self.a}, b: {self.b}, c: {self.c})")

if __name__ == '__main__':
    # Создаем произвольный треугольник
    triangle = Triangle(6, 8, 10)

    # Выводим его в консоль
    triangle.display()
    # Выводим тип треугольника в консоль
    triangle.print_type()

    # Выводим значение стороны a в консоль
    print(triangle.get_a())
    # Устанавливаем новое значение для стороны a
    triangle.set_a(10)
    # Выводим треугольник в консоль
    triangle.display()

    # Выводим в консоль площадь, периметр и высоту всех трех сторон
    треугольника
    print(f"Площадь: {triangle.area()}")
    print(f"Периметр: {triangle.perimeter()}")
    print(f"Высота a: {triangle.height_a()}")
    print(f"Высота b: {triangle.height_b()}")
    print(f"Высота c: {triangle.height_c()}")

```

```
(a: 6, b: 8, c: 10)
Треугольник прямоугольный
6
(a: 10, b: 8, c: 10)
Площадь: 36.66060555964672
Периметр: 28
Высота a: 7.332121111929344
Высота b: 9.16515138991168
Высота c: 7.332121111929344
```

Рисунок . Результат работы 2 задания

### Контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

Для создания класса в Python используется инструкция class. Она сильно похожа на объявление функций def и так же, как и def, class создаёт объект.

Инструкция class имеет следующий синтаксис:

```
**class <Name> ([<Superclass1>], [<Superclass2>]):  
<name declarations>**
```

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибуты экземпляра и класса отличаются способом получения доступа к ним. Другими словами, речь идет об использовании названия класса и использовании названия экземпляра. С другой стороны, глобальные и локальные переменные отличаются своими областями видимости, другими словами, местами, где к ним может быть получен доступ.

3. Каково назначение методов класса?

Методы определяют набор действий, которые доступны классу (часто говорят, что они определяют поведение класса). Метод описывается один раз, а может вызываться для различных объектов класса столько раз, сколько необходимо. Общий формат записи методов класса имеет следующий вид: [атрибуты] [спецификаторы] тип метода имя метода ([параметры]) {тело метода}.

4. Для чего предназначен метод \_\_init\_\_() класса?



Метод `__init__` в определении класса позволяет нам инициализировать атрибуты или переменные экземпляра для всех экземпляров класса. Метод `__init__` вызывается каждый раз, когда создается новый экземпляр класса. Цель наличия нескольких методов `__init__` в классе Python – предоставить несколько конструкторов для создания объектов.

#### 5. Каково назначение `self` ?

Ключевое слово `self` в Python используется для ссылки на текущий экземпляр объекта класса. Оно обычно используется в методах класса, чтобы обращаться к атрибутам и методам этого объекта. Когда мы вызываем метод объекта класса, Python автоматически передает ссылку на этот объект в качестве первого аргумента метода, который мы обычно называем `self`. Таким образом, мы можем обращаться к атрибутам и методам объекта через `self`, как в примере выше, где мы сохраняем имя объекта в атрибуте `name` и выводим его через метод `say_hello`.

#### 6. Как добавить атрибуты в класс?

Атрибуты могут быть добавлены в класс путем определения их внутри класса. Например:

```
class MyClass:
    def __init__(self, attribute1, attribute2):
        self.attribute1 = attribute1
        self.attribute2 = attribute2
```

#### 7. Как осуществляется управление доступом к методам и атрибутам в языке Python?

Управление доступом к методам и атрибутам в языке Python осуществляется с помощью модификаторов доступа. В Python есть три уровня доступа: `public`, `protected` и `private`.

#### 8. Каково назначение функции `isinstance`?

Функция `isinstance` в языке Python используется для проверки принадлежности объекта определенному классу. Она принимает два аргумента: объект и класс, и возвращает `True`, если объект принадлежит к указанному классу или его наследникам, и `False` в противном случае. Функция `isinstance` может быть полезна, например, при проверке типов аргументов функции или при обработке объектов разных классов в цикле.