

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «Информационные технологии»**  
**Тема: Парадигмы программирования**

Студент гр. 3343

Наумкин А.Д.

Преподаватель

Иванов Д.В.

Санкт-Петербург

2024

## **Цель работы**

Изучение основ работы с классами в языке Python. Реализация наследования классов, определение\переопределение «магических» и пользовательских методов.

## Задание

Даны фигуры в двумерном пространстве.

Базовый класс - фигура ***Figure***:

Поля объекта класса *Figure*:

- периметр фигуры(в сантиметрах, целое положительное число)
- площадь фигуры(в квадратных сантиметрах, целое положительное число)
- цвет фигуры(значение может быть одной из строк: 'r', 'b', 'g').

При создании экземпляра класса *Figure* необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение *ValueError* с текстом 'Invalid value'.

Многоугольник — ***Polygon***: (Наследуется от класса *Figure*)

Поля объекта класса *Polygon*:

- периметр фигуры(в сантиметрах, целое положительное число)
- площадь фигуры(в квадратных сантиметрах, целое положительное число)
- цвет фигуры(значение может быть одной из строк: 'r', 'b', 'g')
- количество углов(неотрицательное значение, больше 2)
- равносторонний(значениями могут быть или True, или False)
- самый большой угол(или любого угла, если многоугольник равносторонний) (целое положительное число)

При создании экземпляра класса *Polygon* необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение *ValueError* с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:

1) Метод `__str__()`:

Преобразование к строке вида: *Polygon*: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, количество углов <кол-во углов>.

равносторонний <равносторонний>, самый большой угол <самый большой угол>.

2) Метод `__add__()`:

Сложение площади и периметра многоугольника. Возвращает число, полученное при сложении площади и периметра многоугольника.

3) Метод `__eq__()`:

Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа *Polygon* равны, если равны их периметры, площади и количество углов.

Окружность — ***Circle***: (Наследуется от класса *Figure*)

Поля объекта класса *Circle*:

- периметр фигуры(в сантиметрах, целое положительное число)
- площадь фигуры(в квадратных сантиметрах, целое положительное число)
- цвет фигуры(значение может быть одной из строк: 'r', 'b', 'g').
- радиус(целое положительное число)
- диаметр(целое положительное число, равен двум радиусам)

При создании экземпляра класса *Circle* необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение *ValueError* с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:

1) Метод `__str__()`:

Преобразование к строке вида: *Circle*: Периметр <периметр>, площадь <площадь>, цвет фигуры <цвет фигуры>, радиус <радиус>, диаметр <диаметр>.

2) Метод `__add__()`:

Сложение площади и периметра окружности. Возвращает число, полученное при сложении площади и периметра окружности.

### 3) Метод `__eq__()`:

Метод возвращает `True`, если два объекта класса равны и `False` иначе. Два объекта типа *Circle* равны, если равны их радиусы.

Необходимо определить список *list* для работы с фигурами:

#### **Многоугольники (*class PolygonList*):**

Конструктор:

- Вызвать конструктор базового класса.
- Передать в конструктор строку `name` и присвоить её полю `name` созданного объекта

Необходимо реализовать следующие методы:

#### 1) Метод `append(p_object)`:

Переопределение метода `append()` списка. В случае, если *p\_object* - многоугольник (объект класса *Polygon*), элемент добавляется в список, иначе выбрасывается исключение *TypeError* с текстом: `Invalid type <тип_объекта p_object>`

#### 2) Метод `print_colors()`:

Вывести цвета всех многоугольников в виде строки (нумерация начинается с 1):

<i> многоугольник: <color[i]>

<j> многоугольник: <color[j]> ...

#### 3) Метод `print_count()`:

Вывести количество многоугольников в списке.

#### **Окружности (*class CircleList*):**

Конструктор:

- Вызвать конструктор базового класса.

- Передать в конструктор строку `name` и присвоить её полю `name` созданного объекта

Необходимо реализовать следующие методы:

1) Метод *extend(iterable)*:

Переопределение метода `extend()` списка. В качестве аргумента передается итерируемый объект `iterable`, в случае, если элемент `iterable` - объект класса `Circle`, этот элемент добавляется в список, иначе не добавляется.

2) Метод *print\_colors()*:

Вывести цвета всех окружностей в виде строки (нумерация начинается с 1):

<i> окружность: <color[i]>

<j> окружность: <color[j]> ...

3) Метод *total\_area()*:

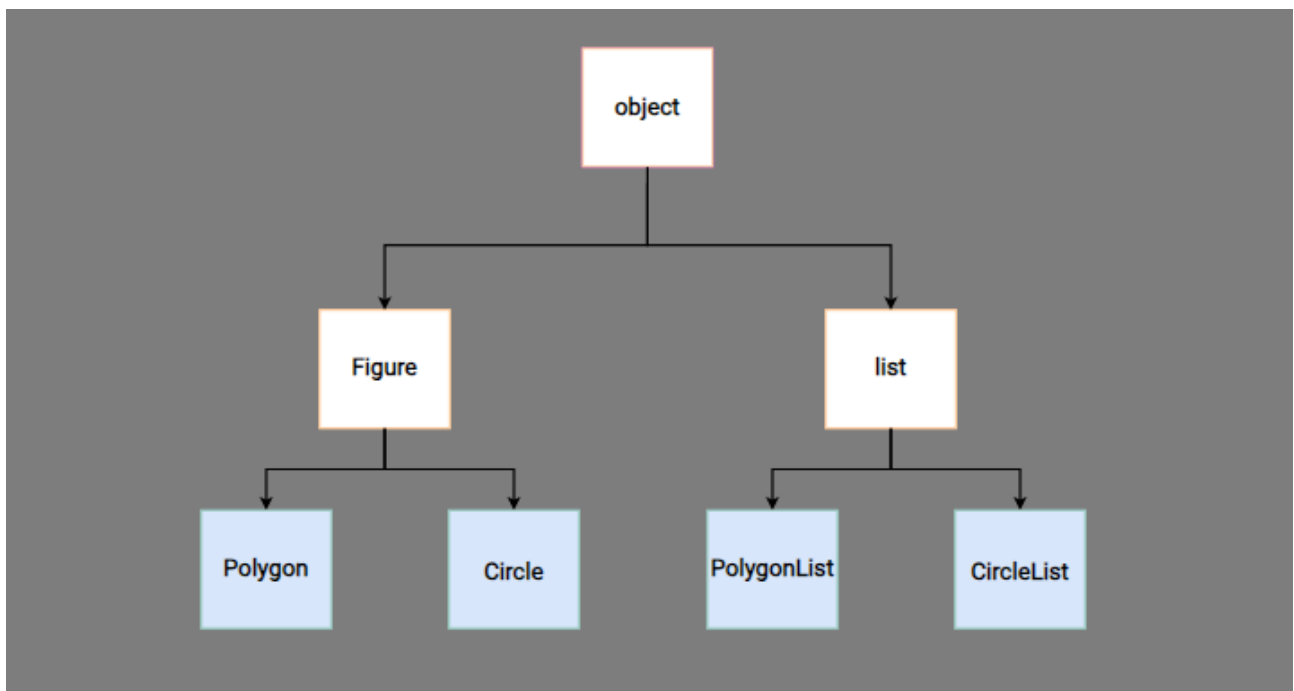
Посчитать и вывести общую площадь всех окружностей.

## Выполнение работы

Требовалось реализовать 2 класса, которые наследовались от пользовательского класса, и 2 класса, которые наследовались от встроенного класса `list`.

Каждый из классов при инициализации экземпляра сохраняет входящие данные в защищенные атрибуты. Для доступа к этим атрибутам были реализованы геттеры и сеттеры через декоратор свойства (`@property`).

В классе *Figure* были переопределены следующие методы класса *object*: `__init__`, `__str__`, `__add__`, `__eq__`. Последние два метода дочерние классы использовали без переопределения. Методы `__str__` и `__init__` класса *Figure* вызываются в и дочерних классах (работа родительского метода дополняется в каждом из дочерних классов по-разному), это сделано во избежание дублирования кода.



Если метод `__init__` используется единожды для инициализации значениями нового экземпляра класса, то метод `__str__` используется для «неформального» строкового представления экземпляра класса. Данный метод неявно вызывается, например, при передаче экземпляра в функцию `print()`. Метод `__add__` неявно вызывается при использовании оператора «+», когда

объект класса стоит слева от знака сложения. Сам метод описывает логику сложения.

При наследовании от класса *list*, переопределяя метод *append*, использовалась функция *super()* с вызовом родительского метода *append* для того, чтобы возможно было добавить элемент в пользовательский список, при этом избежав рекурсивных вызовов (это произошло бы, если, например, при переопределении метода *append* в его теле использовалась не строка *super().append()*, а *self.append()*). Но для *CircleList* мы имели право использовать последний вариант написания, т.к *self.append()* вызывался при переопределении метода *extend()*. В этой ситуации вызывается родительский метод *append()* для экземпляра класса *CircleList*.



## Тестирование

Результаты тестирования содержатся в таблице 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<pre> polygon = Polygon(10,25,'g',4, True, 90)  polygon2 = Polygon(10,25,'g',4, True, 90)  print(polygon.perimeter, polygon.area, polygon.color, polygon.angle_count, polygon.equilateral, polygon.biggest_angle)  print(polygon.__str__())  print(polygon.__add__())  print(polygon.__eq__(polygon2))  polygon_list = PolygonList(Polygon)  polygon_list.append(polygon)  polygon_list.append(polygon2)  polygon_list.print_colors()  polygon_list.print_count()</pre>	<pre> 10 25 g 10 25 g 4 True 90 Polygon: Периметр 10, площадь 25, цвет фигуры g, количество углов 4, равносторонний True, самый большой угол 90. 35 True 1 многоугольник: g 2 многоугольник: g 2</pre>	Тесты с экземплярами класса Polygon и работа со списком PolygonList

## **Выводы**

В результате работы были изучены классы в Python и способы работы с ними: наследование, переопределение методов, особенности работы «магических» методов.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Figure:
    def __init__(self, perimeter, area, color):
        if not isinstance(perimeter, int) or not isinstance(area,
int) or not isinstance(color, str) or perimeter <= 0 or area <= 0
or color not in 'rgb':
            raise ValueError('Invalid value')
        self.perimeter = perimeter
        self.area = area
        self.color = color

class Polygon(Figure):
    def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
        super().__init__(perimeter, area, color)
        if not isinstance(angle_count, int) or not
isinstance(equilateral, bool) or not isinstance(biggest_angle,
int) or angle_count <= 2 or biggest_angle <= 0:
            raise ValueError('Invalid value')
        self.angle_count = angle_count
        self.equilateral = equilateral
        self.biggest_angle = biggest_angle

    def __str__(self):
        return f'Polygon: Периметр {self.perimeter}, площадь
{self.area}, цвет фигуры {self.color}, ' \
            f'количество углов {self.angle_count},
равносторонний {self.equilateral}, самый большой угол
{self.biggest_angle}.'
```

```
    def __add__(self):
        return self.perimeter + self.area

    def __eq__(self, other):
```

```

        return isinstance(other, Polygon) and self.perimeter ==
other.perimeter \
        and self.area == other.area and self.angle_count ==
other.angle_count

```

```

class Circle(Figure):
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
        if not isinstance(radius, int) or not isinstance(diametr,
int) or diametr != 2 * radius:
            raise ValueError('Invalid value')
        self.radius = radius
        self.diametr = diametr

    def __str__(self):
        return f'Circle: Периметр {self.perimeter}, площадь
{self.area}, цвет фигуры {self.color}, ' \
        f'радиус {self.radius}, диаметр {self.diametr}.'

    def __add__(self):
        return self.perimeter + self.area

    def __eq__(self, other):
        return isinstance(other, Circle) and self.radius ==
other.radius

```

```

class PolygonList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def append(self, p_object):
        if isinstance(p_object, Polygon):
            super().append(p_object)
        else:
            raise TypeError(f'Invalid type {type(p_object)}')

```

```

def print_colors(self):
    count = 0
    for p in self:
        count += 1
        print(f'{count} многоугольник: {p.color}')

def print_count(self):
    print(len(self))

class CircleList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def extend(self, iterable):
        for item in iterable:
            if isinstance(item, Circle):
                self.append(item)

    def print_colors(self):
        count = 0
        for c in self:
            count += 1
            print(f'{count} окружность: {c.color}')

    def total_area(self):
        total = sum(c.area for c in self)
        print(total)

    @color.setter
    def color(self, value):
        if value in ('r', 'g', 'b'):
            self._color = value
        else:
            raise ValueError('Invalid value')

    def __str__(self):

```

```

        return f'{self.__class__.__name__}: Периметр
{self.perimeter}, площадь {self.area}, цвет фигуры {self.color}, '

def __add__(self):
    return self.area + self.perimeter

def __eq__(self, other):
    if isinstance(other, Polygon):
        return (self.perimeter == other.perimeter) and
(self.area == other.area) and (self.angle_count ==
other.angle_count)
    if isinstance(other, Circle):
        return self.radius == other.radius

class Polygon(Figure):
    def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
        super().__init__(perimeter, area, color)
        self.angle_count = angle_count
        self.equilateral = equilateral
        self.biggest_angle = biggest_angle

    @property
    def angle_count(self):
        return self._angle_count

    @angle_count.setter
    def angle_count(self, value):
        if isinstance(value, int) and value > 2:
            self._angle_count = value
        else:
            raise ValueError('Invalid value')

    @property
    def equilateral(self):
        return self._equilateral

    @equilateral.setter

```

```

def equilateral(self, value):
    if isinstance(value, bool):
        self._equilateral = value
    else:
        raise ValueError('Invalid value')

@property
def biggest_angle(self):
    return self._biggest_angle

@biggest_angle.setter
def biggest_angle(self, value):
    if isinstance(value, int) and value > 0:
        self._biggest_angle = value
    else:
        raise ValueError('Invalid value')

def __str__(self):
    return super().__str__() + f'количество углов
{self.angle_count}, равносторонний {self.equilateral}, самый
большой угол {self.biggest_angle}.'

class Circle(Figure):
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
        self.radius = radius
        self.diametr = diametr

@property
def radius(self):
    return self._radius

@radius.setter
def radius(self, value):
    if isinstance(value, int) and value > 0:
        self._radius = value
    else:
        raise ValueError('Invalid value')

```

```

@property
def diametr(self):
    return self._diametr

@diametr.setter
def diametr(self, value):
    if isinstance(value, int) and value > 0 and value ==
self.radius * 2:
        self._diametr = value
    else:
        raise ValueError('Invalid value')

def __str__(self):
    return super().__str__() + f'радиус {self.radius}, диаметр
{self.diametr}.'

class PolygonList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name

@property
def name(self):
    return self._name

@name.setter
def name(self, value):
    self._name = value

def append(self, __object):
    if isinstance(__object, Polygon):
        super().append(__object)
    else:
        raise TypeError(f'Invalid type {__object.__class__}')

def print_colors(self):
    for obj_num in range(len(self)):

```



```

        print(f'{obj_num + 1} многоугольник:
{self[obj_num].color}')

    def print_count(self):
        print(len(self))

class CircleList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        self._name = value

    def extend(self, __iterable):
        for obj in __iterable:
            if isinstance(obj, Circle):
                self.append(obj)

    def print_colors(self):
        for obj_num in range(len(self)):
            print(f'{obj_num + 1} окружность:
{self[obj_num].color}')

    def total_area(self):
        print(reduce(lambda x, y: x + y.area, self, 0))

```