

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Информатика»
Тема: Парадигмы программирования

Студент гр. 3342

Колесниченко М. А.

Преподаватель

Иванов Д. В.

Санкт-Петербург

2024

Цель работы

Изучение работы с классами и исключениями на языке Python.
Использование полученных знания для создания программы для работы с классами и вызова исключений в нужных ситуациях.

Задание

Даны фигуры в двумерном пространстве.

Базовый класс - фигура *Figure*:

class 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*:

class 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:

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>.

Метод `__add__()`:

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

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:

class 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:

Метод `__str__()`:

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

Метод `__add__()`:

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

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` – список многоугольников - наследуется от класса `list`.

Конструктор:

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

Необходимо реализовать следующие методы:

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>`

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

`<i>` многоугольник: `<color[i]>`

`<j>` многоугольник: `<color[j]>` ...

Метод `print_count()`: Вывести количество многоугольников в списке.

Окружности:

`class CircleList` – список окружностей - наследуется от класса `list`.

Конструктор:

1. Вызвать конструктор базового класса.

2. Передать в конструктор строку name и присвоить её полю name созданного объекта

Необходимо реализовать следующие методы:

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.

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

<i> окружность: <color[i]>

<j> окружность: <color[j]> ...

Метод `total_area()`: Посчитать и вывести общую площадь всех окружностей.

Выполнение работы

Покажем наследование классов. (см. на рис. 1)

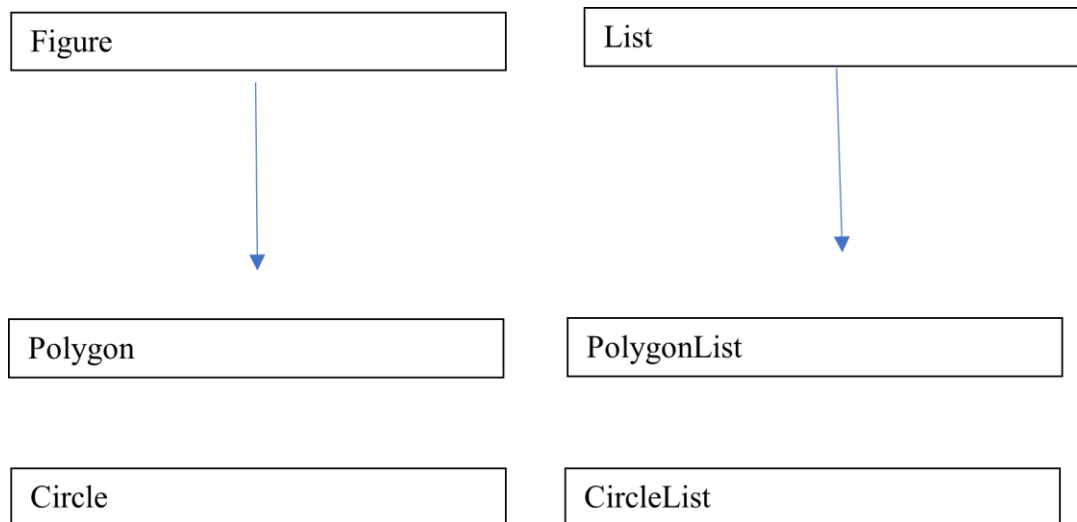


Рисунок 1 – Наследование классов

Опишем создание каждого класса.

Figure:

- 1) `__init__` - добавление в экземпляр класса соответствующих полей.

Вывод исключения, если данные неверные.

Polygon (наследование полей и функций от Figure):

- 1) `__init__` - переопределение для добавления новых полей
- 2) `__str__` - строковое представление экземпляра
- 3) `__eq__` - сравнение с другим экземпляром (other) по полям `perimeter`,

`area`, `angle_count`

Circle (наследование полей и функций от Figure):

- 1) `__init__`, `__str__` - по аналогии с `Polygon`
- 2) `__eq__` - сравнение экземпляров по полю `radius`

PolygonList (наследование полей и функций от list):

- 1) `__init__` - переопределение для добавления поля `name`
- 2) `append` – переопределение для добавления только элементов класса

`Figure`, иначе вызывается исключение

- 3) `print_count` – вывод количество многоугольников в списке.

CircleList (наследование полей и функций от list):

- 1) `__init__` - переопределение для добавления поля `name`
- 2) `extend` - добавления из переданного аргумента только элементов класса `Circle`
- 3) `print_colors` - вывод цвета всех окружностей в списке

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<pre>fig = Figure(10,25,'g') #фигура print(fig.perimeter, fig.area, fig.color) polygon = Polygon(10,25,'g',4, True, 90) #многоугольник polygon2 = Polygon(10,25,'g',4, True, 90) print(polygon.perimeter, polygon.area, polygon.color, polygon.angle_count, polygon.equilateral, polygon.big- gest_angle) print(polygon.__str__()) print(polygon.__add__()) print(polygon.__eq__(polygon2)) circle = Circle(13, 13,'r', 2, 4) #окружность circle2 = Circle(13, 13,'g', 2, 4) print(circle.perimeter, circle.area, cir- cle.color, circle.radius, circle.diametr) print(circle.__str__()) print(circle.__add__()) print(circle.__eq__(circle2)) polygon_list = PolygonList(Polygon) #список многоугольников polygon_list.append(polygon) polygon_list.append(polygon2) polygon_list.print_colors() polygon_list.print_count() circle_list = CircleList(Circle) #список окружностей circle_list.extend([circle, circle2]) circle_list.print_colors()</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 13 13 r 2 4 Circle: Периметр 13, площадь 13, цвет фигуры r, радиус 2, диаметр 4. 26 True 1 многоугольник: g 2 многоугольник: g 2 1 окружность: r 2 окружность: g 26</pre>	Верный вывод

	<code>circle_list.total_area()</code>		
--	---------------------------------------	--	--

Выводы

Была разработана программа, описаны классы и их методы. Обработаны ключевые ситуации, в которых может возникнуть ошибка и использованы исключения для предотвращения этого.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Figure:
    '''Поля объекта класс Figure:
        perimeter - периметр фигуры (в сантиметрах,
        целое положительное число)
        area - площадь фигуры (в квадратных санти
        метрах, целое положительное число)
        color - цвет фигуры (значение может быть о
        дной из строк: 'r', 'b', 'g')

        При создании экземпляра класса Figure нео
        бходимо убедиться, что переданные в конст рукт
        ор параметры удовлетворяют требованиям, ина ч
        е выбросить исключение ValueError с текстом 'Invalid
        value'.
    '''
    def __init__(self, perimeter, area, color):
        if not isinstance(perimeter, int) or perimeter <= 0:
            raise ValueError('Invalid value')

        if not isinstance(area, int) or area <= 0:
            raise ValueError('Invalid value')

        if color not in ['r', 'b', 'g']:
            raise ValueError('Invalid value')

        self.perimeter = perimeter
        self.area = area
        self.color = color

class Polygon(Figure): #Наследуется от класса Figure
    '''Поля объекта класс Polygon:
        perimeter - периметр фигуры (в сантиметрах,
        целое положительное число)
        area - площадь фигуры (в квадратных санти
        метрах, целое положительное число)
        color - цвет фигуры (значение может быть о
        дной из строк: 'r', 'b', 'g')
        angle_count - количество углов (целое положи
        тельное значение, больше 2)
        equilateral - равносторонний (значениями мо
        гут быть или True, или False)
```

biggest_angle - самый большой угол (или любой угол, если многоугольник равносторонний) (в градусах, 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'.

```
'''
def __init__(self, perimeter, area, color, angle_count,
equilateral, biggest_angle):
    if not isinstance(perimeter, int) or perimeter <= 0:
        raise ValueError('Invalid value')

    if not isinstance(area, int) or area <= 0:
        raise ValueError('Invalid value')

    if color not in ['r', 'b', 'g']:
        raise ValueError('Invalid value')

    if not isinstance(angle_count, int) or angle_count <= 2:
        raise ValueError('Invalid value')

    if not isinstance(equilateral, bool):
        raise ValueError('Invalid value')

    if not isinstance(biggest_angle, int) or biggest_angle <=
0:
        raise ValueError('Invalid value')

    self.perimeter = perimeter
    self.area = area
    self.color = color
    self.angle_count = angle_count
    self.equilateral = equilateral
    self.biggest_angle = biggest_angle

def __str__(self):
    '''Преобразование к строке вида: Polygon:
Периметр <периметр>, площадь <площадь>, цвет фи
гуры <цвет фигуры>, равносторонний <равностор
онний>, прямоугольный <прямоугольный>.'''
    return f"Polygon: Периметр {self.perimeter}, площ
адь {self.area}, цвет фигуры {self.color}, количество у
глов {self.angle_count}, равносторонний {self.equilateral},
самый большой угол {self.biggest_angle}."

def __add__(self):
```

```

        '''Сложение площади и периметра много
        угольника. Возвращает число, полученное при с
        ложении площади и периметра многоугольника.'''
        return self.perimeter + self.area

    def __eq__(self, other):
        '''Метод возвращает True, если два объе
        кта класса равны и False иначе. Два объекта типа
        Polygon равны, если равны их периметр, площадь и
        количество углов.'''
        return self.perimeter == other.perimeter and self.area ==
        other.area and self.angle_count == other.angle_count

class Circle(Figure): #Наследуется от класса Figure
    '''Поля объекта класс Circle:
        perimeter - периметр фигуры (в сантиметрах,
        целое положительное число)
        area - площадь фигуры (в квадратных санти
        метрах, целое положительное число)
        color - цвет фигуры (значение может быть о
        дной из строк: 'r', 'b', 'g')
        radius - радиус (целое положительное число)
        diametr - диаметр (целое положительное чис
        ло, равен двум радиусам)

        При создании экземпляра класса Circle нео
        бходимо убедиться, что переданные в конструкт
        ор параметры удовлетворяют требованиям, инач
        е выбросить исключение ValueError с текстом 'Invalid
        value'.
    '''
    def __init__(self, perimeter, area, color, radius, diametr):
        if not isinstance(perimeter, int) or perimeter <= 0:
            raise ValueError('Invalid value')

        if not isinstance(area, int) or area <= 0:
            raise ValueError('Invalid value')

        if color not in ['r', 'b', 'g']:
            raise ValueError('Invalid value')

        if not isinstance(radius, int) or radius <= 0:
            raise ValueError('Invalid value')

        if not isinstance(diametr, int) or diametr != 2*radius:
            raise ValueError('Invalid value')

        self.perimeter = perimeter

```

```

        self.area = area
        self.color = color
        self.radius = radius
        self.diametr = diametr

    def __str__(self):
        '''Преобразование к строке вида: Circle:
Периметр <периметр>, площадь <площадь>, цвет фи
гуры <цвет фигуры>, радиус <радиус>, диаметр <д
иаметр>.'''
        return f"Circle: Периметр {self.perimeter}, площа
дь {self.area}, цвет фигуры {self.color}, радиус
{self.radius}, диаметр {self.diametr}."

    def __add__(self):
        '''Сложение площади и периметра окруж
ности. Возвращает число, полученное при сложе
нии площади и периметра окружности.'''
        return self.perimeter + self.area

    def __eq__(self, other):
        '''Метод возвращает True, если два объе
кта класса равны и False иначе. Два объекта типа
Circle равны, если равны их радиусы.'''
        return self.radius == other.radius

class PolygonList(list):
    def __init__(self, name):
        '''1. Вызвать конструктор базового кла
сса.

        2. Передать в конструктор строку name
и присвоить её полю name созданного объекта'''
        super().__init__()
        self.name = name

    def append(self, p_object):
        '''Переопределение метода append() списк
а. В случае, если p_object - многоугольник (объект
класса Polygon), элемент добавляется в список, ин
аче выбрасывается исключение TypeError с текстом:
Invalid type <тип_объекта p_object>'''
        if isinstance(p_object, Polygon):
            super().append(p_object)
        else:
            raise TypeError(f"Invalid type {type(p_object)}")
    def print_colors(self):

```

```

        '''Вывести цвета всех многоугольников
в.'''
        for i in range(len(self)):
            print(f"{i+1} многоугольник: {self[i].color}")
    def print_count(self):
        '''Вывести количество многоугольников
в. в списке'''
        print(len(self))

class CircleList(list):
    def __init__(self, name):
        '''1. Вызвать конструктор базового класса.

        2. Передать в конструктор строку name
и присвоить её полю name созданного объекта'''
        super().__init__()
        self.name = name

    def extend(self, 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.'''
        for item in iterable:
            if isinstance(item, Circle):
                super().extend([item])

    def print_colors(self):
        '''Вывести цвета всех изогнутых фигур.'''
        for i in range(len(self)):
            print(f"{i+1} окружность: {self[i].color}")

    def total_area(self):
        '''Посчитать и вывести общую площадь
всех окружностей.'''
        a = 0
        for x in self:
            a += x.area
        print(a)

```