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: «Линейные списки»

Студентка гр. 2384 _____ Валеева А.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2023

Цель работы

Изучить принципы работы с линейными списками языке Си, научиться использовать их для решения практических задач.

Задание

Даны фигуры в двумерном пространстве.

Базовый класс - фигура *Figure*:

class Figure:

Поля объекта класса Figure:

- гериметр 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.

В отчете укажите:

1. Изображение иерархии описанных вами классов.
2. Методы, которые вы переопределили (в том числе методы класса object).
3. В каких случаях будут использованы методы __str__() и __add__().
4. Будут ли работать переопределенные методы класса list для PolygonList и CircleList? Объясните почему и приведите примеры.

Выполнение работы

Создаем класс *Figure,* прописываем у него конструктор, добавляя поля, если передаваемые аргументы правильные. Их правильность проверяем с помощью функций *check_positive_int() (*для проверки числа — целое, положительное*), check_color() (*проверка строки*) и ckeck_bool() (*проверка типа с помощью функции *isinstance).* Иначе с помощью *raise* передаем ошибки неправильного типа.

Следующий класс — наследуемый от класса *Figure — Polygon.* В конструкторе вызываем функцию super(), дополняем поля новыми аргументами, если они удовлетворяют условиям. Переопределяем методы *__str__(), __add__() и __eq__().*

Далее создаем ее один один дочерний класс класса *Figure — Circle.* Так же прописываем конструктор и переопределяем следующие методы: *__str__(), __add__(), __eq__().*

После создаем два класса-наследника класса *list - PolygonList и CircleList.* В первом из них мы прописываем конструктор, а также переопределяем метод *append()* и добавляем еще два метода: *print_colors(), print_count()*.

В следующем классе *CircleList* так же прописываем конструктор и переопределяем метод *extend* и добавляем еще три метода: *print_colors(), print_count()*, *total_area().*

Ответы на вопросы.

1. Иерархия описанных классов представлена на рис. 1.

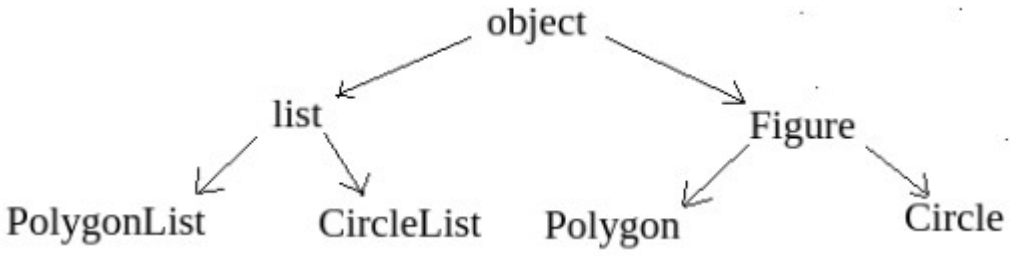

Рис. 1

2. Переопределенные методы.

1) В классе *Figure:*

object → __init__

2) В классе *Polygon:*

Figure → __init__

object→Figure→__str__

object→Figure→__add__

object→Figure→__eq__

3) В классе *Circle*:

Figure → __init__

object→Figure→__str__

object→Figure→__add__

object→Figure→__eq__

4) В классе *PolygonList:*

list →__init__

list → append

5) В классе *CircleList*:

list →__init__

list → extend

3. Использование методов __str__() и __add__().

Метод *__str__()* для отображения информации об объекте класса. Например для функций *print, str.*

Метод *__add__()* используется при применении бинарной операции

сложения двух объектов

4. Переопределенные методы будут работать, так как мы их переопределили, добавив дополнительные условия запуска этих методов. Пример: some_list = PolgonList(Pol) some_list.append(Polygon(10, 25, 'g', 4, True, 90)

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Вывод.

Были изучены основы ООП в языке программирования python. Разработана программа реализующая классы Figure, Polygon, Circle, PolygonList, CircleList, а также методы для взаимодействия с данными классами.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Valeyeva_Alina_lb1.py

```
def check_positive_int(num):
    return isinstance(num, int) and num > 0


def check_color(color):
    return color in ['r', 'g', 'b']


def check_bool(a):
     return isinstance(a, bool)  # isinstance проверка на класс, True False - тип bool


class Figure:
    def __init__(self, perimeter, area, color):
         if check_positive_int(perimeter) and check_positive_int(area) and check_color(color):
            self.perimeter = perimeter
            self.area = area
            self.color = color
        else:
            raise ValueError('Invalid value')



class Polygon(Figure):  # Наследуется от класса Figure
         def __init__(self, perimeter, area, color, angle_count, equilateral, biggest_angle):
        super().__init__(perimeter, area, color)
           if check_positive_int(angle_count) and angle_count > 2 and check_bool(equilateral) and check_positive_int(biggest_angle):
            self.angle_count = angle_count
            self.equilateral = equilateral
```

```
            self.biggest_angle = biggest_angle
        else:
            raise ValueError('Invalid value')

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

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

    def __eq__(self, other):
                return self.area == other.area and self.perimeter == other.perimeter and self.angle_count == other.angle_count


class Circle(Figure):  # Наследуется от класса Figure
    def __init__(self, perimeter, area, color, radius, diametr):
        super().__init__(perimeter, area, color)
         if check_positive_int(radius) and check_positive_int(diametr) and diametr == 2 * radius:
            self.radius = radius
            self.diametr = diametr
        else:
            raise ValueError('Invalid value')

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

    def __add__(self):
```

```
        return self.area + self.perimeter

    def __eq__(self, other):
        return 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):
        st = ""
        for i in range(
                len(self)):
            st += f"{i + 1} многоугольник: {self[i].color}\n"
        print(st[:len(st) - 1])

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


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

```
def extend(self, iterable):
    arr = [i for i in iterable if isinstance(i, Circle)]
    super().extend(arr)

def print_colors(self):
    st = ""
    for i in range(len(self)):
        st += f"{i + 1} окружность: {self[i].color}\n"
    print(st[:len(st) - 1])

def total_area(self):
    Area = 0
    for i in self:
        Area += i.area
    print(Area)
```

ПРИЛОЖЕНИЕ Б.
ТЕСТИРОВАНИЕ

Таблица Б.1 – результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	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.biggest_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, circle.color, circle.radius, circle.diametr)
	print(circle.__str__())
	print(circle.__add__())
		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
			Верный вывод

	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()
	circle_list.total_area()		

2.	try: #неправильные данные для фигуры
	fig = Figure(-10,25,'g')
	except (TypeError, ValueError):
	print('OK')
	
	try:
	fig = Figure(10,-25,'g')
	except (TypeError, ValueError):
	print('OK')
	
	try:
	fig = Figure(10,25,-1)
	except (TypeError, ValueError):
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
			Верный вывод

```
    print('OK')

try:
    fig = Figure(10,25,1)
except (TypeError, ValueError):
    print('OK')

try:
    fig = Figure(10,25,'a')
except (TypeError, ValueError):
    print('OK')

try:
    fig = Figure('a',25,'g')
except (TypeError, ValueError):
    print('OK')

try:
    fig = Figure(10,'a','g')
except (TypeError, ValueError):
    print('OK')

try:
    fig = Figure(0,25,'g')
except (TypeError, ValueError):
    print('OK')

try:
    fig = Figure(10,0,'g')
```

	except (TypeError, ValueError): print('OK')		