

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«Северо-Кавказский федеральный университет»**

Кафедра инфокоммуникаций

**Отчет по лабораторной работе № 4.3
«Наследование полиморфизм в языке Python»**

по дисциплине «Объектно-ориентированное программирование»

Выполнил студент группы ИВТ-б-о-20-1

Бобров Н.В. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2022

Цель работы: приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

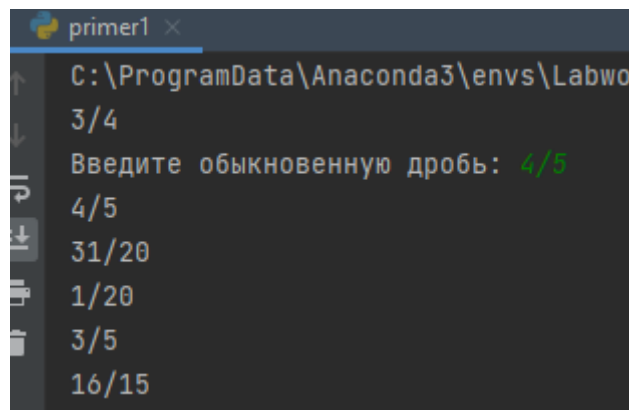
Ход работы:

1. Создал общедоступный репозиторий и клонировал его на локальный сервер.
2. Изучив теоретический материал, приступил к выполнению примеров.



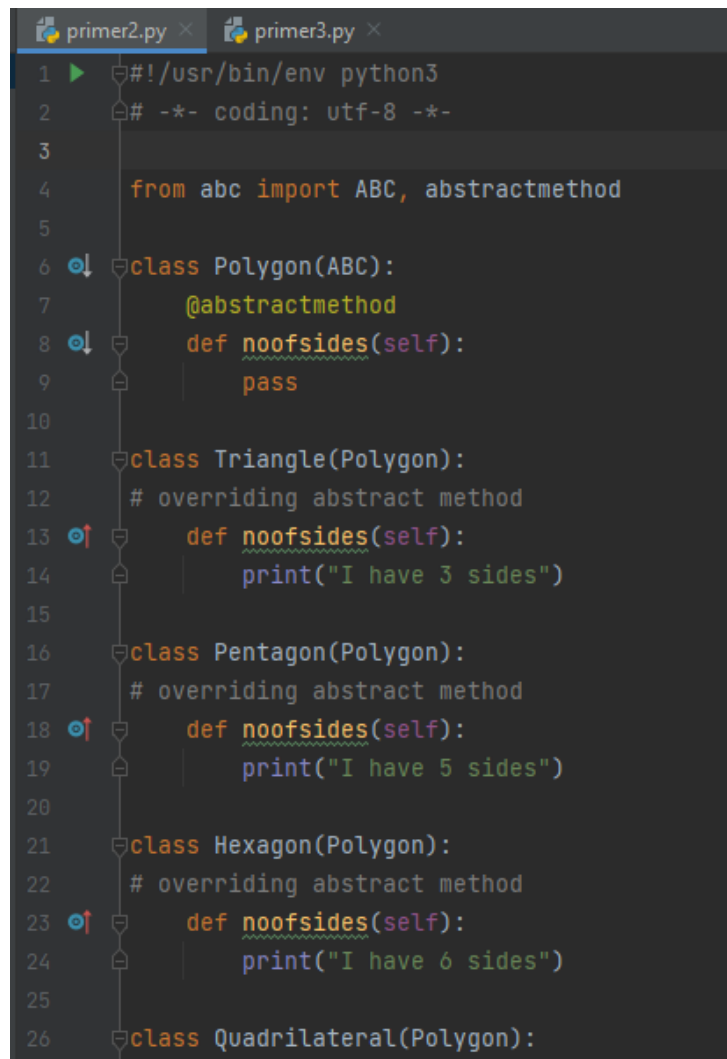
```
primer1.py x primer2.py x primer3.py x
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  class Rational:
5      def __init__(self, a=0, b=1):
6          a = int(a)
7          b = int(b)
8          if b == 0:
9              raise ValueError()
10         self.__numerator = abs(a)
11         self.__denominator = abs(b)
12         self.__reduce()
13
14     def __reduce(self):
15
16         def gcd(a, b):
17             if a == 0:
18                 return b
19             elif b == 0:
20                 return a
21             elif a >= b:
22                 return gcd(a % b, b)
23             else:
24                 return gcd(a, b % a)
25         c = gcd(self.__numerator, self.__denominator)
26         self.__numerator //= c
```

Рисунок 1 – Код первого примера



```
primer1 x
C:\ProgramData\Anaconda3\envs\Labwor
3/4
Введите обыкновенную дробь: 4/5
4/5
31/20
1/20
3/5
16/15
```

Рисунок 2 – Результат работы кода

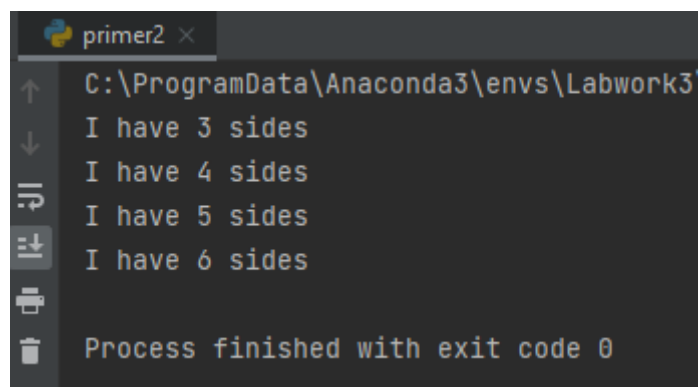


```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  from abc import ABC, abstractmethod
5
6  class Polygon(ABC):
7      @abstractmethod
8      def noofsides(self):
9          pass
10
11  class Triangle(Polygon):
12      # overriding abstract method
13      def noofsides(self):
14          print("I have 3 sides")
15
16  class Pentagon(Polygon):
17      # overriding abstract method
18      def noofsides(self):
19          print("I have 5 sides")
20
21  class Hexagon(Polygon):
22      # overriding abstract method
23      def noofsides(self):
24          print("I have 6 sides")
25
26  class Quadrilateral(Polygon):

```

Рисунок 3 – Код из второго примера



```

C:\ProgramData\Anaconda3\envs\Labwork3
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
Process finished with exit code 0

```

Рисунок 4 – Работа кода

```

1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      from abc import ABC, abstractmethod
5
6  ⚡  class Animal(ABC):
7      ⚡  @abstractmethod
8      ⚡  def move(self):
9          pass
10
11     class Human(Animal):
12     ⚡  def move(self):
13         print("I can walk and run")
14
15     class Snake(Animal):
16     ⚡  def move(self):
17         print("I can crawl")
18
19     class Dog(Animal):
20     ⚡  def move(self):
21         print("I can bark")
22
23     class Lion(Animal):
24     ⚡  def move(self):
25         print("I can roar")

```

Рисунок 5 – Код из третьего примера

```

primer3 x
C:\ProgramData\Anaconda3\envs\Labwork
I can walk and run
I can crawl
I can bark
I can roar
Process finished with exit code 0

```

Рисунок 6 – Результат работы кода

3. Затем приступил к выполнению общего для всей группы задания.

```

task.py x
45
46 def main():
47     hero1 = Hero(1, 1, 'Blue') # Создаем героев с номерами 1 и 2
48     hero2 = Hero(2, 2, 'Red')
49     team1, team2 = [], []
50     for i in range(2, 12): # Генерим нечетное количество солдат
51         n = random.randint(0, 1)
52         if n:
53             team1.append(Soldier(i, 1))
54             print('Солдат с номером', i, 'добавлен в армию', hero1.name)
55         else:
56             team2.append(Soldier(i, 2))
57             print('Солдат с номером', i, 'добавлен в армию', hero2.name)
58
59     print('Армия героя', hero1.name, ': ', len(team1))
60     print('Армия героя', hero2.name, ': ', len(team2))
61
62     if len(team1) > len(team2):
63         print('В армии', hero1.name, 'больше солдат, увеличиваем его уровень')
64         hero2.inclevel()
65     else:
66         print('В армии', hero2.name, 'больше солдат, увеличиваем его уровень')
67         hero2.inclevel()
68
69     team1[1].gotohero(hero2)
70

```

Рисунок 7 – Код выполненного задания

```

task x
C:\ProgramData\Anaconda3\envs\Labwork3\python.exe C:/00P/Labwork3/task.py
Солдат с номером 2 добавлен в армию Blue
Солдат с номером 3 добавлен в армию Red
Солдат с номером 4 добавлен в армию Blue
Солдат с номером 5 добавлен в армию Blue
Солдат с номером 6 добавлен в армию Blue
Солдат с номером 7 добавлен в армию Blue
Солдат с номером 8 добавлен в армию Blue
Солдат с номером 9 добавлен в армию Red
Солдат с номером 10 добавлен в армию Red
Солдат с номером 11 добавлен в армию Blue
Армия героя Blue : 7
Армия героя Red : 3
В армии Blue больше солдат, увеличиваем его уровень
Уровень героя Red увеличен на 1 и равен 2
Солдат номер 4 следует за героем Red с номером 2

Process finished with exit code 0

```

Рисунок 8 – Результат работы кода

4. Затем приступил к выполнению индивидуальных заданий.

```

individual1.py x
12 class Car:
13     def __init__(self, mark, cylinders, power):
14         self.__mark = mark
15         self.__cylinders = cylinders
16         self.__power = power
17
18     @property
19     def mark(self):
20         return self.__mark
21
22     @mark.setter
23     def mark(self, inp):
24         self.__mark = inp
25
26     @property
27     def cylinders(self):
28         return self.__cylinders
29
30     @cylinders.setter
31     def cylinders(self, inp):
32         self.__cylinders = inp
33
34     @property
35     def power(self):
36         return self.__power
37

```

```

individual1.py x
41
42
43 class Lorry(Car):
44     def __init__(self, mark, cylinders, power, capacity):
45         super().__init__(mark, cylinders, power)
46         self.__capacity = capacity
47
48     @property
49     def capacity(self):
50         return self.__capacity
51
52     @capacity.setter
53     def capacity(self, inp):
54         self.__capacity = inp
55
56
57 def main():
58     truck = Lorry("Газель", 4, 119, 2500)
59     passenger = Car("Лада", 8, 105)
60     print(f"Марка: {truck.mark}, количество цилиндров: {truck.cylinders}, "
61           f"мощность: {truck.power} л.с, грузоподъемность: {truck.capacity} т")
62     print(f"Марка: {passenger.mark}, количество цилиндров: {passenger.cylinders}")
63     truck.power = 60
64     truck.mark = "MAN"
65     print(f"Марка: {truck.mark}, кол-во цилиндров: {truck.cylinders}, "
66           f"мощность: {truck.power} л.с, грузоподъемность: {truck.capacity} т")

```

Рисунок 9 – Код для решения задачи

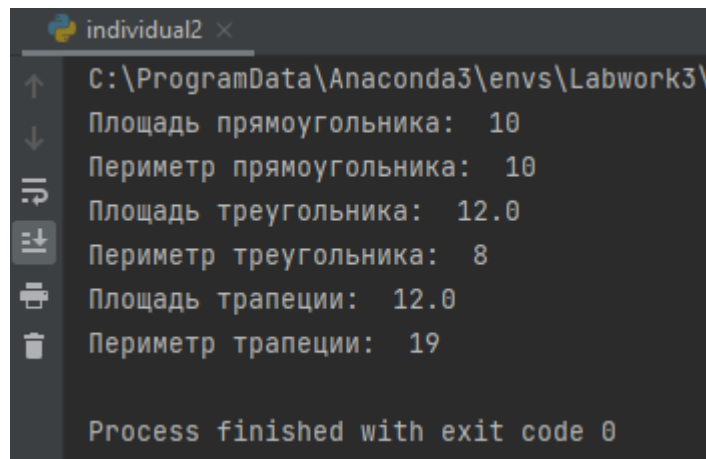
```
individual1 x
C:\ProgramData\Anaconda3\envs\Labwork3\python.exe C:/00P/Labwork3/individual1.py
Марка: Газель, количество цилиндров: 4, мощность: 119 л.с, грузоподъемность: 2500 т
Марка: Лада, количество цилиндров: 8
Марка: MAN, кол-во цилиндров: 4, мощность: 60 л.с, грузоподъемность: 2500 т
Process finished with exit code 0
```

Рисунок 10 – Результат работы кода

5. Приступил к выполнению второго индивидуального задания.

```
individual2.py x
16 class Figure(ABC):
17
18     @abstractmethod
19     def square(self):
20         pass
21
22     @abstractmethod
23     def perimetr(self):
24         pass
25
26
27 class Rectangle(Figure):
28     def square(self, a, b):
29         print("Площадь прямоугольника: ", a * b)
30
31     def perimetr(self, a, b):
32         print("Периметр прямоугольника: ", 2 * (a + b))
33
34
35 class Circle(Figure):
36     def square(self, b, h):
37         print("Площадь треугольника: ", 0.5 * b * h)
38
39     def perimetr(self, a, b, c):
40         print("Периметр треугольника: ", a + b + c)
41
42
43
44
45
46
47
48
49
50 def main():
51     r1 = Rectangle()
52     r1.square(2, 5)
53     r1.perimetr(3, 2)
54
55     c1 = Circle()
56     c1.square(4, 6)
57     c1.perimetr(1, 4, 3)
58
59     t1 = Trapezium()
60     t1.square(4, 8, 2)
61     t1.perimetr(2, 4, 6, 7)
62
63
64 if __name__ == "__main__":
65     main()
```

Рисунок 11 – Код программы



```
individual2 x
C:\ProgramData\Anaconda3\envs\Labwork3\
Площадь прямоугольника: 10
Периметр прямоугольника: 10
Площадь треугольника: 12.0
Периметр треугольника: 8
Площадь трапеции: 12.0
Периметр трапеции: 19
Process finished with exit code 0
```

Рисунок 12 – Результат работы кода

Контрольные вопросы:

1. Что такое наследование как оно реализовано в языке Python?

Наследование — это возможность расширения (наследования) ранее написанного программного кода класса с целью дополнения, усовершенствования или привязки под новые требования.

Синтаксически создание класса с указанием его родителя выглядит так:
`class имя_класса(имя_родителя1, [имя_родителя2,..., имя_родителя_n])`

2. Что такое полиморфизм и как он реализован в языке Python?

Полиморфизм - это способность выполнять действие над объектом независимо от его типа. Это обычно реализуется путем создания базового класса и наличия двух или более подклассов, которые все реализуют методы с одинаковой сигнатурой.

3. Что такое "утиная" типизация в языке программирования Python?

Эта концепция адаптирована из следующего абдуктивного умозаключения:

Если что-то выглядит как утка, плавает как утка и крикает как утка, это наверняка и есть утка.

Концепция утиной типизация в основном принята в языках программирования, поддерживающих динамическую типизацию, таких как Python и JavaScript. Общей особенностью этих языков является возможность объявления переменных без указания их типа.

При использовании пользовательских типов для определённых целей,

реализация связанных функций важнее, чем точные типы данных.

Утиная типизация подчёркивает реализацию связанных выполняемых функций, а конкретные типы данных менее важны

4. Каково назначение модуля `abc` языка программирования Python?

Начиная с версии языка 2.6 в стандартную библиотеку включается модуль `abc`, добавляющий в язык абстрактные базовые классы (далее АБК).

АБК позволяют определить класс, указав при этом, какие методы или свойства обязательно переопределить в классах-наследниках.

5. Как сделать некоторый метод класса абстрактным?

Перед методом класса необходимо добавить декоратор модуля `abc`: `@abstractmethod`.

6. Как сделать некоторое свойство класса абстрактным?

Абстрактные классы включают в себя атрибуты в дополнение к методам, вы можете потребовать атрибуты в конкретных классах, определив их с помощью `@abstractproperty`.

7. Каково назначение функции `isinstance` ?

Функция `isinstance ()` в Python используется для проверки, является ли объект экземпляром указанного класса или нет.

Вывод: в ходе выполнения лабораторной работы были приобретены простейшие навыки по работе с наследованием и абстрактными методами классов в языке программирования Python.