

**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 и полиморфизм в языке Python»

по дисциплине «Основы программной инженерии»

Выполнил студент группы

ПИЖ-б-о-21-1

Зиберов Александр

« » мая 2023 г.

Подпись студента _____

Работа защищена

« » _____ 20__ г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь, 2023

Цель работы:

Приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: <https://github.com/afk552/lab4.3>

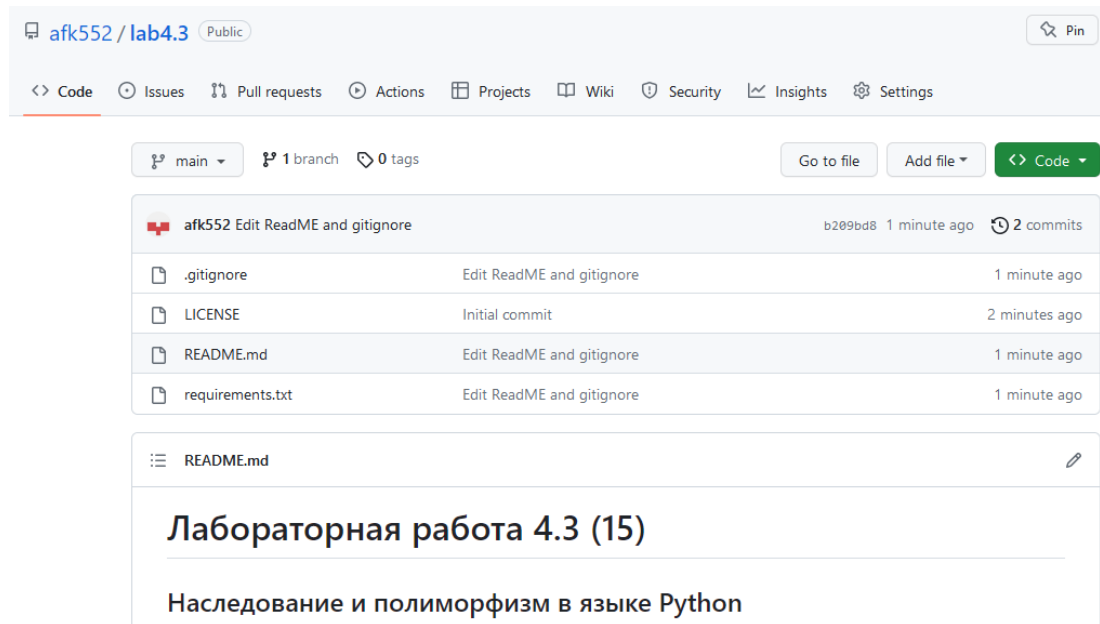


Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

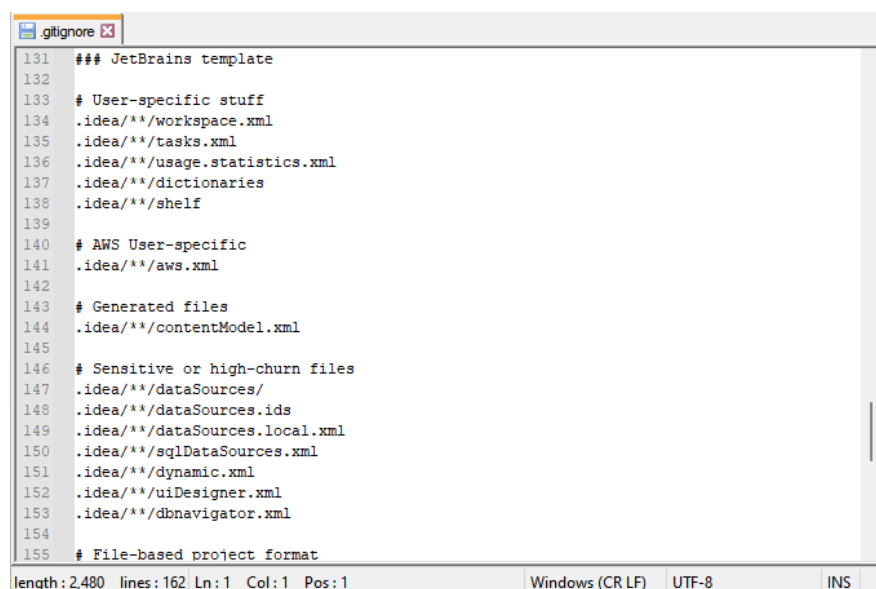


Рисунок 2 – Окно блокнота

Организируйте свой репозиторий в соответствие с моделью ветвления git-flow, рисунок 3.

```
C:\git\lab4.3>git checkout -b develop
Switched to a new branch 'develop'

C:\git\lab4.3>
```

Рисунок 3 – Окно командной строки

Примеры

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  class Figure:
6      def __init__(self, color):
7          self.__color = color
8
9      4 usages
10     @property
11     def color(self):
12         return self.__color
13
14     3 usages
15     @color.setter
16     def color(self, c):
17         self.__color = c
18
19  1 usage
20  class Rectangle(Figure):
21      def __init__(self, width, height, color):
22          super().__init__(color)
23          self.__width = width
24          self.__height = height
25
26  2 usages
27  @property
28  def width(self):
29      return self.__width
30
31  @width.setter
32  def width(self, w):
33      if w > 0:
34          self.__width = w
35      else:
36          raise ValueError
37
38  2 usages
39  @property
40  def height(self):
41      return self.__height
42
43  @height.setter
44  def height(self, h):
45      if h > 0:
46          self.__height = h
47      else:
48          raise ValueError
49
50  2 usages
51  @property
52  def height(self):
53      return self.__height
54
55  @height.setter
56  def height(self, h):
57      if h > 0:
58          self.__height = h
59      else:
60          raise ValueError
61
62  def area(self):
63      return self.__width * self.__height
64
65  if __name__ == '__main__':
66      rect = Rectangle(10, 20, "green")
67      print(rect.width,
68            rect.height,
69            rect.color
70            )
71      rect.color = "red"
72      print(rect.color)
```

C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex1.py
10 20 green
red
Process finished with exit code 0

Рисунок 4 – Пример 1

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  class Figure:
6      def __init__(self, color):
7          self.__color = color
8
9      @property
10     def color(self):
11         return self.__color
12
13     @color.setter
14     def color(self, c):
15         self.__color = c
16
17     def info(self):
18         print("Figure")
19         print("Color: " + self.__color)
20
21
22 class Rectangle(Figure):
23     def __init__(self, width, height, color):
24         super().__init__(color)
25         self.__width = width
26         self.__height = height
27
28     @property
29     def width(self):
30         return self.__width
31
32     @width.setter
33     def width(self, w):
34         if w > 0:
35             self.__width = w
36         else:
37             raise ValueError
38
39     @property
40     def height(self):
41         return self.__height
42
43     @height.setter
44     def height(self, h):
45         if h > 0:
46             self.__height = h
47         else:
48             raise ValueError
49
50     def area(self):
51         return self.__width * self.__height
52
53     def info(self):
54         print("Rectangle")
55         print("Color: " + self.color)
56         print("Width: " + str(self.width))
57         print("Height: " + str(self.height))
58         print("Area: " + str(self.area()))
59
60
61 if __name__ == '__main__':
62     fig = Figure("orange")
63     fig.info()
64     rect = Rectangle(10, 20, "green")
65     rect.info()
66
67 C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex2.py
68 Figure
69 Color: orange
70 Rectangle
71 Color: green
72 Width: 10
73 Height: 20
74 Area: 200
75
76 Process finished with exit code 0
```

Рисунок 5 – Пример 2

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  class Table:
6      def __init__(self, l, w, h):
7          self.length = l
8          self.width = w
9          self.height = h
10
11
12 class DeskTable(Table):
13     def square(self):
14         return self.width * self.length
15
16
17 if __name__ == '__main__':
18     t1 = Table(1.5, 1.8, 0.75)
19     t2 = DeskTable(0.8, 0.6, 0.7)
20     print(t2.square())
21
22 DeskTable > square()
23
24 Run ex3
25
26 C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex3.py
27 0.48
28
29 Process finished with exit code 0
```

Рисунок 6 – Пример 3

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  1 usage
5  class Table:
6      def __init__(self, l, w, h):
7          self.length = l
8          self.width = w
9          self.height = h
10
11  ===
12  class KitchenTable(Table):
13      def __init__(self, l, w, h, p):
14          Table.__init__(self, l, w, h)
15          self.places = p
16  ===
17
18  1 usage
19  class KitchenTable(Table):
20      def __init__(self, l, w, h, p):
21          super().__init__(l, w, h)
22          self.places = p
23
24
25  if __name__ == '__main__':
26      t4 = KitchenTable(1.5, 2, 0.75, 6)
27      print(t4.height)
28

```

Run ex4

C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex4.py
0.75

Process finished with exit code 0

Рисунок 7 – Пример 4

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  # Python program showing
6  # implementation of abstract
7  # class through subclassing
8  import abc
9
10
11  3 usages
12  class parent:
13      def geeks(self):
14          pass
15
16  2 usages
17  class child(parent):
18      def geeks(self):
19          print("child class")
20
21  if __name__ == '__main__':
22      print(issubclass(child, parent))
23      print(isinstance(child(), parent))
24

```

Run ex5

C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex5.py
True
True

Process finished with exit code 0

Рисунок 8 – Пример 5

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  # Python program invoking a
6  # method using super()
7  from abc import ABC
8
9
10 1 usage
11 class R(ABC):
12     1 usage
13     def rk(self):
14         print("Abstract Base Class")
15
16
17 1 usage
18 class K(R):
19     1 usage
20     def rk(self):
21         super().rk()
22         print("subclass")
23
24
25 if __name__ == '__main__':
26     r = K()
27     r.rk()
```

Run ex6

C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex6.py

Abstract Base Class

subclass

Рисунок 9 – Пример 6

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  13 usages
6  class Rational:
7      def __init__(self, a=0, b=1):
8          a = int(a)
9          b = int(b)
10         if b == 0:
11             raise ValueError()
12         self.__numerator = abs(a)
13         self.__denominator = abs(b)
14         self.__reduce()
15         # Сокращение дроби
16
17  2 usages
18     def __reduce(self):
19         # Функция для нахождения наибольшего общего делителя
20         def gcd(a, b):
21             if a == 0:
22                 return b
23             elif b == 0:
24                 return a
25             elif a >= b:
26                 return gcd(a % b, b)
27             else:
28                 return gcd(a, b % a)
29
30         c = gcd(self.__numerator, self.__denominator)
31         self.__numerator //= c
32         self.__denominator //= c
33
34  14 usages
35     @property
36     def numerator(self):
37         return self.__numerator
38
39  18 usages
40     @property
41     def denominator(self):
42         return self.__denominator
43
44     # Прочитать значение дроби с клавиатуры. Дробь вводится
45     # как a/b.
46     1 usage
47     def read(self, prompt=None):
48         line = input() if prompt is None else input(prompt)
49         parts = list(map(int, line.split('/', maxsplit=1)))
50         if parts[1] == 0:
51             raise ValueError()
52         self.__numerator = abs(parts[0])
53         self.__denominator = abs(parts[1])
54         self.__reduce()
55
56  51
57     # Вывести дробь на экран
58     6 usages
59     def display(self):
60         print(f"{self.__numerator}/{self.__denominator}")
61
62     # Сложение обыкновенных дробей.
63     1 usage
64     def add(self, rhs):
65         if isinstance(rhs, Rational):
66             a = self.__numerator * rhs.__denominator + \
67                 self.__denominator * rhs.__numerator
68             b = self.__denominator * rhs.__denominator
69             return Rational(a, b)
70         else:
71             raise ValueError()
72
73     # Вычитание обыкновенных дробей.
74     1 usage
75     def sub(self, rhs):
76         if isinstance(rhs, Rational):
77             a = self.__numerator * rhs.__denominator - \
78                 self.__denominator * rhs.__numerator
79             b = self.__denominator * rhs.__denominator
80             return Rational(a, b)
81         else:
82             raise ValueError()
83
84     # Умножение обыкновенных дробей.
85     1 usage
86     def mul(self, rhs):
87         if isinstance(rhs, Rational):
88             a = self.__numerator * rhs.__numerator
89             b = self.__denominator * rhs.__denominator
90             return Rational(a, b)
91         else:
92             raise ValueError()
93
94     # Деление обыкновенных дробей.
95     1 usage
96     def div(self, rhs):
97         if isinstance(rhs, Rational):
98             a = self.__numerator * rhs.__denominator
99             b = self.__denominator * rhs.__numerator
100             return Rational(a, b)
101         else:
102             raise ValueError()
103
104     # Отношение обыкновенных дробей.
105     def equals(self, rhs):
106         if isinstance(rhs, Rational):
107             return (self.__numerator == rhs.__numerator) and \
108                 (self.__denominator == rhs.__denominator)
109         else:
110             return False
111
112     def greater(self, rhs):
113         if isinstance(rhs, Rational):
114             v1 = self.__numerator / self.__denominator
115             v2 = rhs.__numerator / rhs.__denominator
116             return v1 > v2
117         else:
118             return False
119
120     def less(self, rhs):
121         if isinstance(rhs, Rational):
122             v1 = self.__numerator / self.__denominator
123             v2 = rhs.__numerator / rhs.__denominator
124             return v1 < v2
125         else:
126             return False
127
128 if __name__ == '__main__':
129     r1 = Rational(3, 4)
130     r1.display()
131     r2 = Rational()
132     r2.read("Введите обыкновенную дробь: ")
133     r2.display()
134     r3 = r2.add(r1)
135     r3.display()
136     r4 = r2.sub(r1)
137     r4.display()
138     r5 = r2.mul(r1)
139     r5.display()
140     r6 = r2.div(r1)
141     r6.display()
142
143 C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex_task1.py
144 3/4
145 Введите обыкновенную дробь: 3/2
146 3/2
147 9/4
148 3/4
149 9/8
150 2/1
151
152 Process finished with exit code 0
```

Рисунок 10 – Пример 7

```

1  #!/usr/bin/env python3
2  #-*- coding: utf-8 -*-
3
4  # Python program showing
5  # abstract base class work
6  from abc import ABC, abstractmethod
7
8
9  class Polygon(ABC):
10     @abstractmethod
11     def noofsides(self):
12         pass
13
14
15     1 usage
16     class Triangle(Polygon):
17         # overriding abstract method
18         1 usage
19         def noofsides(self):
20             print("I have 3 sides")
21
22     1 usage
23     class Pentagon(Polygon):
24         # overriding abstract method
25         1 usage
26     def noofsides(self):
27         print("I have 5 sides")
28
29
30     1 usage
31     class Hexagon(Polygon):
32         # overriding abstract method
33         1 usage
34     def noofsides(self):
35         print("I have 6 sides")
36
37
38     1 usage
39     class Quadrilateral(Polygon):
40         # overriding abstract method
41         1 usage
42     def noofsides(self):
43         print("I have 4 sides")
44
45
46     # Driver code
47     if __name__ == '__main__':
48         R = Triangle()
49         R.noofsides()
50         K = Quadrilateral()
51         K.noofsides()
52         R = Pentagon()
53         R.noofsides()
54         K = Hexagon()
55         K.noofsides()

```

C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex_task2.py

I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

Process finished with exit code 0

Рисунок 11 – Пример 8

```

1  #!/usr/bin/env python3
2  #-*- coding: utf-8 -*-
3
4  # Python program showing
5  # abstract base class work
6  from abc import ABC
7
8
9
10     4 usages
11     class Animal(ABC):
12         def move(self):
13             pass
14
15     1 usage
16     class Human(Animal):
17         1 usage
18         def move(self):
19             print("I can walk and run")
20
21     1 usage
22     class Snake(Animal):
23         1 usage
24         def move(self):
25             print("I can crawl")
26
27
28     1 usage
29     class Dog(Animal):
30         1 usage
31         def move(self):
32             print("I can bark")
33
34
35     1 usage
36     class Lion(Animal):
37         1 usage
38         def move(self):
39             print("I can roar")
40
41
42     # Driver code
43     if __name__ == '__main__':
44         R = Human()
45         R.move()
46         K = Snake()
47         K.move()
48         R = Dog()
49         R.move()
50         K = Lion()
51         K.move()

```

C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\examples\ex_task3.py

I can walk and run
I can crawl
I can bark
I can roar

Рисунок 12 – Пример 9

8, 9. Решите задачу, разработайте программу по следующему описанию.

В некоей игре-стратегии есть солдаты и герои. У всех есть свойство, содержащее уникальный номер объекта, и свойство, в котором хранится принадлежность команде. У солдат есть метод "иду за героем", который в качестве аргумента принимает объект типа "герой". У героев есть метод увеличения собственного уровня.

В основной ветке программы создается по одному герою для каждой команды. В цикле генерируются объекты-солдаты. Их принадлежность команде определяется случайно.

Солдаты разных команд добавляются в разные списки.

Измеряется длина списков солдат противоборствующих команд и выводится на экран. У героя, принадлежащего команде с более длинным списком, увеличивается уровень.

Отправьте одного из солдат первого героя следовать за ним. Выведите на экран идентификационные номера этих двух юнитов.

```
25 # 2 usages
26 class Unit:
27     __number = 0
28
29     def __init__(self, team):
30         self.__identifier = self.__number
31         self.team = team
32         self.__class__.__number += 1
33
34     @property
35     def identifier(self):
36         return self.__identifier
37
38     @property
39     def team(self):
40         return self.__team
41
42     @team.setter
43     def team(self, value):
44         self.__team = value
45
46 # 3 usages
47 class Hero(Unit):
48     def __init__(self, team):
49         super().__init__(team)
50         self.level = 0
51
52     @property
53     def level(self):
54         return self.__level
55
56     @level.setter
57     def level(self, value):
58         self.__level = value
59
60 # 2 usages
61 def increment_level(self):
62     self.level += 1
63
64 # 2 usages
65 def display(self):
66     print(f"Команда: {self.team}")
67     print(f"ID: {self.identifier}")
68     print(f"После: {self.level}")
69
70 class Soldier(Unit):
71     def __init__(self, team):
72         super().__init__(team)
73         self.follows = False
74         self.hero_id = -1
75
76 # 1 usage
77 def start_following(self, hero):
78     if isinstance(hero, Hero):
79         self.follows = True
80         self.hero_id = hero.identifier
81     else:
82         raise ValueError
83
84 # 3 usages
85 @property
86 def follows(self):
87     return self.__follows
88
89 # 2 usages
90 @hero_id.setter
91 def hero_id(self, value):
92     self.__hero_id = value
93
94 # 2 usages
95 @hero_id.setter
96 def hero_id(self, value):
97     self.__hero_id = value
98
99 # 2 usages
100 def display(self):
101     print(f"Команда: {self.team}")
102     print(f"ID: {self.identifier}")
103     print(f"Следует за героем: {self.follows}")
104     if self.follows:
105         print(f"ID героя: {self.hero_id}")
106
107 if __name__ == '__main__':
108     # Создаем героев
109     h1 = Hero(1)
110     h2 = Hero(2)
111
112     # Создаем команду
113     team1 = []
114     team2 = []
115
116     # Рандомно и добавляем в команды
117     amount = 10
118     for i in range(amount):
119         team = randint(1, 2)
120         if team == 1:
121             team1.append(Soldier(1))
122         else:
123             team2.append(Soldier(2))
124
125     # Увеличение уровня героя в зависимости от длины списка
126     if len(team1) > len(team2):
127         h1.increment_level()
128     else:
129         h2.increment_level()
130
131     # Следование за героем
132     rand_unit = randint(0, len(team1) - 1)
133     team1[rand_unit].start_following(h1)
134     print(f"Герой W1:")
135     h1.display()
136     print(f"Герой W2:")
137     h2.display()
138     print(f"Солдат команды W2:")
139     team2[0].display()
140     print(f"Солдат, следующий за героем W1:")
141     team1[rand_unit].display()
142
143 C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\indiv_task\task8_9.py
144 Герой W1:
145 Команда: 1
146 ID: 0
147 Уровень: 0
148
149 Герой W2:
150 Команда: 2
151 ID: 1
152 Уровень: 1
153
154 Солдат команды W2:
155 Команда: 2
156 ID: 1
157 Следует за героем: False
158
159 Солдат, следующий за героем W1:
160 Команда: 1
161 ID: 0
162 Следует за героем: True
163 ID героя: 0
164
165 Process finished with exit code 0
```

Рисунок 13 – Код программы заданий 8, 9

Индивидуальное задание.

Задание 1. Создать класс Man (человек), с полями: имя, возраст, пол и вес. Определить методы переназначения имени, изменения возраста и изменения веса. Создать производный класс Student, имеющий поле года обучения. Определить методы переназначения и увеличения года обучения.

```
C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\indiv_task\indiv_1.py
```

```
Имя: Andrey  
Пол: Male  
Возраст: 18  
Вес: 50
```

```
Имя: Andrey  
Пол: Male  
Возраст: 20  
Вес: 43
```

```
Имя: Test  
Пол: Female  
Возраст: 23  
Вес: 100  
Год обучения: 2
```

```
Имя: Jean  
Пол: Female  
Возраст: 22  
Вес: 100  
Год обучения: 3
```

```
Process finished with exit code 0
```

Рисунок 14 – Результат выполнения программы индивидуального задания 1

```

12 class Man:
13     # Инициализация класса "Человек"
14     def __init__(self, name, age, sex, weight):
15         self.name = name
16         self.age = age
17         self.sex = sex
18         self.weight = weight
19
20     2 usages
21     @property
22     def name(self):
23         return self.__name
24
25     2 usages
26     @property
27     def age(self):
28         return self.__age
29
30     2 usages
31     @property
32     def sex(self):
33         return self.__sex
34
35     2 usages
36     @property
37     def weight(self):
38         return self.__weight
39
40     2 usages
41     @name.setter
42     def name(self, value):
43         self.__name = value
44
45     2 usages
46     @age.setter
47     def age(self, value):
48         self.__age = value
49
50     1 usage
51     @sex.setter
52     def sex(self, value):
53         self.__sex = value
54
55     2 usages
56     @weight.setter
57     def weight(self, value):
58         self.__weight = value
59
60     1 usage
61     def set_name(self, name):
62         self.name = name
63
64     1 usage
65     def set_weight(self, weight):
66         self.weight = weight
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109

```

```

2 usages
def set_age(self, age):
    self.age = age

# Функция вывода
3 usages
def display(self):
    print()
    print(f"Имя: {self.name}")
    print(f"Пол: {self.sex}")
    print(f"Возраст: {self.age}")
    print(f"Вес: {self.weight}")

# Производный класс "Человек" -> "Студент"
2 usages
class Student(Man):
    def __init__(self, name, age, sex, weight, study_year):
        # Доступ к наследуемым элементам
        super().__init__(name, age, sex, weight)
        # Поле "Год обучения"
        self.study_year = study_year

    3 usages
    @property
    def study_year(self):
        return self.__study_year

    2 usages
    @study_year.setter
    def study_year(self, value):
        self.__study_year = value

    def set_study_year(self, value):
        self.study_year = value

    1 usage
    def increase_study_year(self):
        self.study_year += 1

    2 usages
    def display(self):
        super(Student, self).display()
        print(f"Год обучения: {self.study_year}")

```

```

if __name__ == "__main__":
    m1 = Man(name="Andrey", age=18, sex="Male", weight=50)
    m1.display()
    m1.set_age(20)
    m1.set_weight(43)
    m1.display()

    m2 = Student(name="Test", age=23, sex="Female", weight=100, study_year=2)
    m2.display()
    m2.increase_study_year()
    m2.set_name("Jean")
    m2.set_age(22)
    m2.display()

```

Рисунок 15 – Код программы индивидуального задания 1

Задание 2. Создать абстрактный базовый класс Triangle для представления треугольника с виртуальными функциями вычисления площади и периметра. Поля данных должны включать две стороны и угол между ними. Определить классы-наследники: прямоугольный треугольник, равнобедренный треугольник, равносторонний треугольник со своими функциями вычисления площади и периметра.

```

19 from math import pi, sqrt, cos, sin
20 from abc import ABC, abstractmethod
21
22 # Базовый абстрактный класс "Треугольник"
23 3 usages
24 class Triangle(ABC):
25     @abstractmethod
26     def __init__(self):
27         pass
28
29 1 usage
30 @abstractmethod
31 def square(self):
32     pass
33
34 1 usage
35 @abstractmethod
36 def perimeter(self):
37     pass
38
39 # Функция вывода
40 3 usages
41 def print_info(self):
42     print(f"Площадь треугольника = {self.square()}")
43     print(f"Периметр треугольника = {self.perimeter()}")
44
45 # Класс-наследник "Прямоугольный треугольник"
46 1 usage
47 class RightTriangle(Triangle):
48     def __init__(self, side1, side2, angle):
49         self.__side1 = side1
50         self.__side2 = side2
51         self.__angle = angle * pi
52
53 1 usage
54 def square(self):
55     return 0.5 * self.__side1 * self.__side2
56
57 1 usage
58 def perimeter(self):
59     return (
60         self.__side1
61         + self.__side2
62         + sqrt(
63             self.__side1**2
64             + self.__side2**2
65             - 2 * self.__side1 * self.__side2 * cos(self.__angle)
66         )
67     )
68
69 if __name__ == "__main__":
70     # Проверка классов, задание значений, расчет
71     tr1 = RightTriangle(3, 4, 30)
72     print(f"Площадь прямоугольного треугольника: {tr1.square()}")
73     print(f"Периметр прямоугольного треугольника: {tr1.perimeter()}")
74     tr2 = IsoscelesTriangle(2, 2, 45)
75     print(f"Площадь равнобедренного треугольника: {tr2.square()}")
76     print(f"Периметр равнобедренного треугольника: {tr2.perimeter()}")
77     tr3 = EquilateralTriangle(5, 5, 60)
78     print(f"Площадь равностороннего треугольника: {tr3.square()}")
79     print(f"Периметр равностороннего треугольника: {tr3.perimeter()}")
80     print("-" * 20)
81     # Проверка функции вывода посчитанных значений
82     tr1.print_info()
83     tr2.print_info()
84     tr3.print_info()
85
86 class IsoscelesTriangle(Triangle):
87     def __init__(self, side1, side2, angle):
88         self.__side1 = side1
89         self.__side2 = side2
90         self.__angle = angle * pi
91
92 1 usage
93 def square(self):
94     return 0.5 * self.__side1 * self.__side2 * sin(self.__angle)
95
96 1 usage
97 def perimeter(self):
98     return (
99         self.__side1
100         + self.__side2
101         + sqrt(
102             self.__side1**2
103             + self.__side2**2
104             - 2 * self.__side1 * self.__side2 * cos(self.__angle)
105         )
106     )
107
108 # Класс-наследник "Равнобедренный треугольник"
109 1 usage
110 class EquilateralTriangle(Triangle):
111     def __init__(self, side1, side2, angle):
112         # Если неверно заданы стороны или угол -> ошибка
113         if side1 != side2 or angle != 60:
114             raise ValueError("Не равнобедренный треугольник!")
115         else:
116             self.__side1 = side1
117             self.__side2 = side2
118             self.__angle = angle * pi
119
120 1 usage
121 def square(self):
122     return (sqrt(3.0) / 4) * self.__side1**2
123
124 1 usage
125 def perimeter(self):
126     return 3 * self.__side1

```

C:\git\lab4.3\venv\Scripts\python.exe C:\git\lab4.3\indiv_task\indiv_2.py

```

Площадь прямоугольного треугольника: 6.0
Периметр прямоугольного треугольника: 8.0
Площадь равнобедренного треугольника: 3.9163938347251765e-15
Периметр равнобедренного треугольника: 8.0
Площадь равностороннего треугольника: 10.825317547305483
Периметр равностороннего треугольника: 15
-----
Площадь треугольника = 6.0
Периметр треугольника = 8.0
Площадь треугольника = 3.9163938347251765e-15
Периметр треугольника = 8.0
Площадь треугольника = 10.825317547305483
Периметр треугольника = 15

```

Process finished with exit code 0

Рисунок 16 – Индивидуальное задание 2 (код, результат)

Вывод: Были приобретены и применены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

Контрольные вопросы:

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

В организации наследования участвуют как минимум два класса: класс родитель и класс потомок. При этом возможно множественное наследование, в этом случае у класса потомка может быть несколько родителей. Не все языки программирования поддерживают множественное наследование, но в Python можно его использовать. По умолчанию все классы в Python являются наследниками от object, явно этот факт указывать не нужно. 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 возможность объектов с одинаковой сигнатурой методов вызывать разные реализации этого метода в зависимости от текущего типа объекта. В Python полиморфизм реализуется через вызов методов класса объекта без необходимости указывать явно тип объекта.

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

"Утиная" типизация - это стиль программирования, при котором проверка на соответствие типу объекта происходит во время выполнения, а не на этапе компиляции. В Python все объекты имеют общий тип object, и проверка соответствия типу может быть выполнена с помощью ключевого слова isinstance.

4. Каково назначение модуля abc языка программирования Python?

По умолчанию Python не предоставляет абстрактных классов. Python поставляется с модулем, который обеспечивает основу для определения абстрактных базовых классов (ABC), и имя этого модуля - ABC. ABC работает, декорируя методы базового класса как абстрактные, а затем регистрируя конкретные классы как реализации абстрактной базы. Метод становится абстрактным, если он украшен ключевым словом `@abstractmethod`.

5. Как сделать некоторый метод класса абстрактным?

Для того чтобы сделать метод класса абстрактным, нужно создать абстрактный метод в базовом классе с помощью декоратора `@abstractmethod`. Этот метод не должен иметь реализации в базовом классе, и должен быть переопределен в каждом наследнике.

6. Как сделать некоторое свойство класса абстрактным?

Для того чтобы сделать свойство класса абстрактным, нужно создать абстрактное свойство в базовом классе с помощью декоратора `@abstractmethod`. Это свойство не должно иметь реализации в базовом классе, и должно быть переопределено в каждом наследнике.

7. Каково назначение функции isinstance?

Функция `isinstance` используется для проверки соответствия типа объекта указанному классу или его наследнику. Она принимает два аргумента: объект, тип которого нужно проверить, и класс или кортеж классов, с которым нужно сравнить тип объекта. Если объект является экземпляром указанного класса или его наследника, то функция возвращает `True`, в противном случае - `False`.