

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
дисциплины
«Объектно-ориентированное программирование»
Вариант № 3

Выполнил:
Левашев Тимур Рашидович
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Доцент департамента цифровых,
робототехнических систем и
электроники института перспективной
инженерии Воронкин Р.А

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г.

Тема работы: “Элементы объектно-ориентированного программирования в языке Python”.

Цель работы: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python

Ссылка на репозиторий: https://github.com/mazy99/oop_pract_1

Порядок выполнения работы:

1. Пример объявления класса Book и вывод атрибутов экземпляра класса.

```
1  #Класс Book
2  class Book:
3      #Атрибуты класса
4      material = "paper"
5      cover = "paperback"
6      all_books = []
7
8  def main():
9      print(f" Material: {Book.material}\n Cover: {Book.cover}\n Books: {Book.all_books}\n")
10     #Экземпляр класса Book
11     my_book = Book()
12     print(" Экземпляр класса Book: \n")
13     print(f" Material: {my_book.material}\n Cover: {my_book.cover}\n Books: {my_book.all_books}")
14
15
16 if __name__ == "__main__":
17     main()
```

Рисунок 1 – Объявление класса Book

```
Экземпляр класса Book:

Material: paper
Cover: paperback
Books: []
```

Рисунок 2 – Вывод экземпляра класса

2. Создание нового класса River с конструктором. Создание нового метода для данного класса.

```

1  class River:
2      all_rivers = []
3
4  def __init__(self, name, length):
5      self.name = name
6      self.length = length
7      River.all_rivers.append(self)
8
9  #Метод
10 def get_info(self):
11     return f"Длина {self.name} равна {self.length} км"
12
13
14 def main():
15
16     volga = River("Волга", 3530)
17     seine = River("Сена", 776)
18     nile = River("Нил", 6852)
19
20     for river in River.all_rivers:
21         print(f"Name: {river.name}, Length: {river.length}")
22     print(f"\n {volga.get_info()}\n {seine.get_info()}\n {nile.get_info()}")
23
24
25
26 if __name__ == "__main__":
27     main()

```

Рисунок 3 – Объявление класса River

```

Name: Волга, Length: 3530
Name: Сена, Length: 776
Name: Нил, Length: 6852

Длина Волга равна 3530 км
Длина Сена равна 776 км
Длина Нил равна 6852 км

```

Рисунок 4 – Вывод данных экземпляра класса с использованием метода
get_info

3. Создание нового класса Rectangle с разными уровнями доступа.

```

1  class Rectangle:
2      def __init__(self, width, height):
3          self._width = width
4          self._height = height
5
6      def get_width(self):
7          return self._width
8
9      def set_width(self, w):
10         self._width = w
11
12     def get_height(self):
13         return self._height
14
15     def set_height(self, h):
16         self._height = h
17
18     def area(self):
19         return self._width * self._height
20
21 def main():
22     rect = Rectangle(10, 5)
23     print(f"Width: {rect.get_width()}")
24     print(f"Height: {rect.get_height()}")
25     print(f"Area: {rect.area()}")
26
27 if __name__ == "__main__":
28     main()

```

Рисунок 5 – Объявление класса Rectangle

```
width: 10
Height: 5
Area: 50
```

Рисунок 6 – Вывод данных экземпляра класса

В данном случае и атрибуты, и методы класса являются публичными, а значит мы можем к ним обращаться напрямую.

```
1  class Rectangle:
2      def __init__(self, width, height):
3          self.__width = width
4          self.__height = height
5
6      def get_width(self):
7          return self.__width
8      def set_width(self, w):
9          self.__width = w
10     def get_height(self):
11         return self.__height
12     def set_height(self, h):
13         self.__height = h
14     def area(self):
15         return self.__width * self.__height
16
17 def main():
18     rect = Rectangle(10, 5)
19     print(f"Width: {rect.get_width()}")
20     print(f"Height: {rect.get_height()}")
21     print(f"Width: {rect._Rectangle__width}")
22     print(f"Height: {rect._Rectangle__height}")
23     print(f"Area: {rect.area()}")
24
25 if __name__ == "__main__":
26     main()
```

Рисунок 6 – Объявление нового класса Rectangle

```
width: 10
Height: 5
width: 10
Height: 5
Area: 50
```

Рисунок 7 – Вывод данных экземпляра класса

В данном случае атрибуты класса являются приватными, а значит обратиться напрямую к ним невозможно, если не использовать внешний атрибут.

```

1 class Rectangle:
2     def __init__(self, width, height):
3         self.__width = width
4         self.__height = height
5     @property
6     def width(self):
7         return self.__width
8     @width.setter
9     def width(self, w):
10        if w>0:
11            self.__width = w
12        else:
13            raise ValueError
14    @property
15    def height(self):
16        return self.__height
17    @height.setter
18    def height(self, h):
19        if h>0:
20            self.__height = h
21        else:
22            raise ValueError
23    def area(self):
24        return self.__width * self.__height
25
26 def main():
27     rect = Rectangle(10, 5)
28     print(f"Width: {rect.width}")
29     print(f"Height: {rect.height}")
30     print(f"Area: {rect.area()}")
31
32 if __name__ == "__main__":
33     main()

```

Рисунок 7 – Объявление нового класса Rectangle

```

Width: 10
Height: 5
Area: 50

```

Рисунок 8 – Вывод данных экземпляра класса

В данном случае атрибуты и методы класса являются приватными, а значит обратиться напрямую к ним невозможно, если не использовать внешний атрибут. Для доступа к атрибутам класса были добавлены свойства, и сеттеры для изменяя атрибутов экземпляра.

4. Выполнение индивидуального задания 1.

Задание 1

Парой называется класс с двумя полями, которые обычно имеют имена *first* и *second*. Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать:

- метод инициализации `__init__`; метод должен контролировать значения аргументов на корректность;
- ввод с клавиатуры `read`;
- вывод на экран `display`.

Реализовать внешнюю функцию с именем `make_тип()`, где `тип` — тип реализуемой структуры. Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанного класса.

Рисунок 9 – Индивидуально задание 1

3. Поле *first* — целое положительное число, числитель; поле *second* — целое положительное число, знаменатель. Реализовать метод `ipart()` — выделение целой части дроби *first/second*. Метод должен проверять неравенство знаменателя нулю.

Рисунок 10 – Вариант индивидуального задания

Объявление класса `FractionPart` с приватными атрибутами *first* и *second*.

```
1  from math import gcd as gsd
2
3  class FractionPart:
4      def __init__(self, first=0, second=1):
5          if second == 0:
6              raise ValueError("Знаменатель не может быть 0")
7
8          self.__first = int(abs(first))
9          self.__second = int(abs(second))
10
11         self.normalize()
```

Рисунок 11 – Объявление класса FractionPart

Реализация методов `read` и `display` для ввода и вывода данных.

```
13  def read(self):
14      self.__first = int(input("Введите числитель: "))
15      self.__second = int(input("Введите знаменатель: "))
16
17      if self.__second == 0:
18          raise ValueError("Знаменатель не может быть 0")
19
20
21  def display(self):
22
23      print(f"{self.__first}/{self.__second}")
24      print(f"Целая часть дроби: {self.ipart()}")
```

Рисунок 12 – Методы для ввода и вывода данных

Создание нового метода для нормализации вывода, чтобы автоматически сокращать дробь.

```
26     def normalize(self):
27         g = gsd(self.__first, self.__second)
28         self.__first //= g
29         self.__second //= g
30
```

Рисунок 13 – Метод для нормализации данных

```
32     @property
33     def first(self):
34         return self.__first
35
36
37     @property
38     def second(self):
39         return self.__second
```

Рисунок 14 – Методы для вывода атрибутов экземпляра класса
Реализация основного метода для выделения целой части дроби.

```
41
42     def ipart(self):
43         return self.__first // self.__second
44
```

Рисунок 15 – Основной метод для выделения целой части дроби

```
2/1
Целая часть дроби: 2
Введите числитель: 10
Введите знаменатель: 5
10/5
Целая часть дроби: 2
(venv) PS C:\учеба\ООП\1 лаба\oop_pract_1>
```

Рисунок 16 – Пример работы класса

Тестирование правильности работы класса и его методов.

```

1  import pytest
2  from unittest.mock import patch
3  from io import StringIO
4  from tasks.fraction_part import FractionPart
5
6  def test_fraction_io_read():
7      frac = FractionPart(5, 2)
8
9      with patch('builtins.input', side_effect=['3', '4']):
10         with patch('sys.stdout', new_callable=StringIO) as mock_stdout:
11             frac.read()
12             frac.display()
13
14             output = mock_stdout.getvalue()
15             assert frac.first == 3
16             assert frac.second == 4
17             assert "3/4" in output
18             assert "Целая часть дроби: 0" in output
19
20  def test_fraction_part_initialization():
21      with pytest.raises(ValueError):
22          FractionPart(1, 0)
23
24  def test_fraction_ipart():
25      fraction = FractionPart(7, 3)
26      assert fraction.ipart() == 2

```

Рисунок 17 – Тесты класса FractionPart

```

20  def test_fraction_part_initialization():
21      with pytest.raises(ValueError):
22          FractionPart(1, 0)
23
24  def test_fraction_ipart():
25      fraction = FractionPart(7, 3)
26      assert fraction.ipart() == 2
27
28
29  def test_fraction_denominator_zero():
30      with pytest.raises(ValueError, match="Знаменатель не может быть 0"):
31          FractionPart(1, 0)
32
33  def test_fraction_normalization():
34      fraction = FractionPart(8, 4)
35      assert fraction.first == 2
36      assert fraction.second == 1
37      assert fraction.ipart() == 2
38
39  def test_fraction_negative_values():
40      fraction = FractionPart(-3, -4)
41      assert fraction.first == 3
42      assert fraction.second == 4
43      assert fraction.ipart() == 0

```

Рисунок 18 – Тесты класса FractionPart

```

tests/test_fraction_part.py::test_fraction_io_read PASSED [ 16%]
tests/test_fraction_part.py::test_fraction_part_initialization PASSED [ 33%]
tests/test_fraction_part.py::test_fraction_ipart PASSED [ 50%]
tests/test_fraction_part.py::test_fraction_denominator_zero PASSED [ 66%]
tests/test_fraction_part.py::test_fraction_normalization PASSED [ 83%]
tests/test_fraction_part.py::test_fraction_negative_values PASSED [100%]

```

Рисунок 19 – Результаты тестов

5. Выполнение второго индивидуального задания.

Задание 2

Составить программу с использованием классов и объектов для решения задачи. Во всех заданиях, помимо указанных в задании операций, обязательно должны быть реализованы следующие методы:

- метод инициализации `__init__`;
- ввод с клавиатуры `read`;
- вывод на экран `display`.

Номер варианта необходимо уточнить у преподавателя. В раздел программы, начинающийся после инструкции `if __name__ == '__main__':` добавить код, демонстрирующий возможности разработанного класса.

Рисунок 20 – Индивидуальное задание 2

3. Создать класс `Money` для работы с денежными суммами. Число должно быть представлено двумя полями: типа `int` для рублей и копеек. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой. Реализовать сложение, вычитание, деление сумм, деление суммы на дробное число, умножение на дробное число и операции сравнения.

Рисунок 21 – Вариант индивидуального задания

Создание нового класса `Money` с приватными полями `rub` и `kop`.

```
class Money:
    def __init__(self, rub: int, kop: int):
        self.__rub = int(rub)
        self.__kop = int(kop)
    @property
    def rub(self):
        return self.__rub
    @property
    def kop(self):
        return self.__kop
```

Рисунок 22 – Объявление класса `Money`

```
15     def read(self):
16         self.__rub = int(input("Введите количество рублей: "))
17         self.__kop = int(input("Введите количество копеек: "))
18
19
20     def display(self):
21         print(f"Количество денег: {self.__rub},{self.__kop} руб.")
22
```

Рисунок 23 – Методы для ввода и вывода данных

```

23     def _to_kop(self):
24         return self.rub * 100 + self.kop
25
26     def normalize(self):
27         if self.__kop >= 100:
28             self.__rub = self.__rub + self.__kop // 100
29             self.__kop = self.__kop % 100
30         elif self.__kop < 0:
31             self.__rub = self.__rub - (-self.__kop // 100)
32             self.__kop = -self.__kop % 100
33

```

Рисунок 24 – Методы для нормализации данных

```

34     def add(self, other):
35         total = self._to_kop() + other._to_kop()
36         return Money(0, total)
37
38     def sub(self, other):
39         total = self._to_kop() - other._to_kop()
40         return Money(0, total)
41
42     def div(self, other):
43         if other._to_kop() == 0:
44             raise ZeroDivisionError
45         total = round(self._to_kop() / other._to_kop())
46         return total
47
48     def div_num(self, num: float):
49         if num == 0:
50             raise ZeroDivisionError
51         total = self._to_kop() / num
52         return Money(0, total)
53
54     def mul_num(self, num: float):
55         total = self._to_kop() * num
56         return Money(0, total)

```

Рисунок 25 – Методы сложения, вычитания, умножения, деления

```

58  def eq(self, other):
59      return self._to_kop() == other._to_kop()
60
61  def lt(self, other):
62      return self._to_kop() < other._to_kop()
63
64  def gt(self, other):
65      return self._to_kop() > other._to_kop()
66
67  def le(self, other):
68      return self._to_kop() <= other._to_kop()
69
70  def ge(self, other):
71      return self._to_kop() >= other._to_kop()
72
73  def ne(self, other):
74      return self._to_kop() != other._to_kop()

```

Рисунок 26 – Методы, используемые для сравнения

```

76  if __name__ == "__main__":
77      money_1 = Money(1, 2)
78      money_2 = Money(3, 4)
79
80      result_1 = money_1.add(money_2)
81      result_1.normalize()
82      result_1.display()
83
84      money_3 = Money(5, 6)
85      money_4 = Money(7, 8)
86
87      result_2 = money_3.sub(money_4)
88      result_2.normalize()
89      result_2.display()
90
91      money_5 = Money(10, 40)
92      money_6 = Money(5, 20)
93
94      result_3 = money_5.div(money_6)
95      print(result_3)
96
97      money_7 = Money(13, 14)
98      result_4 = money_7.div_num(2)
99      result_4.normalize()
100     result_4.display()
101
102     money_8 = Money(15, 16)
103     result_5 = money_8.mul_num(2)
104     result_5.normalize()
105     result_5.display()
106
107
108     print(money_1.eq(money_2))
109     print(money_1.lt(money_2))
110     print(money_1.gt(money_2))
111     print(money_1.le(money_2))

```

Рисунок 27 – Код для демонстрации работы

```
Количество денег: 4,6 руб.  
Количество денег: -2,2 руб.  
2  
Количество денег: 6,57 руб.  
Количество денег: 30,32 руб.  
False  
True  
False  
True  
False  
True
```

Рисунок 27 – Пример вывода данных

```
1  import pytest  
2  from unittest.mock import patch  
3  from io import StringIO  
4  from tasks.money import Money  
5  
6  def test_io_read():  
7      money = Money(5,20)  
8  
9      with patch('builtins.input', side_effect=['5', '20']):  
10         with patch('sys.stdout', new_callable=StringIO) as mock_stdout:  
11             money.read()  
12             money.display()  
13  
14             output = mock_stdout.getvalue()  
15             assert money.rub == 5  
16             assert money.kop == 20  
17             assert "Количество денег: 5,20 руб." in output  
18  
19  def test_add():  
20      m1 = Money(5,20)  
21      m2 = Money(10,30)  
22  
23      result = m1.add(m2)  
24      result.normalize()  
25  
26      assert result.rub == 15  
27      assert result.kop == 50
```

Рисунок 28 – Тесты для класса Money

```
19  def test_add():  
20      m1 = Money(5,20)  
21      m2 = Money(10,30)  
22  
23      result = m1.add(m2)  
24      result.normalize()  
25  
26      assert result.rub == 15  
27      assert result.kop == 50  
28  
29  def test_sub():  
30      m1 = Money(10,30)  
31      m2 = Money(5,20)  
32  
33      result = m1.sub(m2)  
34      result.normalize()  
35  
36      assert result.rub == 5  
37      assert result.kop == 10  
38  
39      m3 = Money(10,30)  
40      m4 = Money(10,30)  
41  
42      result = m3.sub(m4)  
43      result.normalize()  
44  
45      assert result.rub == 0  
46      assert result.kop == 0  
47  
48      m5 = Money(5,20)  
49      m6 = Money(10,30)  
50  
51      result = m5.sub(m6)  
52      result.normalize()
```

Рисунок 29 – Тесты для класса Money

```

43     result.normalize()
44
45     assert result.rub == 0
46     assert result.kop == 0
47
48     m5 = Money(5,20)
49     m6 = Money(10,30)
50
51     result = m5.sub(m6)
52     result.normalize()
53
54     assert result.rub == -5
55     assert result.kop == 10
56
57 def test_div():
58     m1 = Money(20,60)
59     m2 = Money(10,30)
60
61     result = m1.div(m2)
62
63     assert result == 2
64
65     m3 = Money(10,30)
66     m4 = Money(10,30)
67
68     result = m3.div(m4)
69
70     assert result == 1

```

Рисунок 30 – Тесты для класса Money

```

68     result = m3.div(m4)
69
70     assert result == 1
71
72     m5 = Money(10,20)
73     m6 = Money(5,20)
74
75     result = m5.div(m6)
76
77     assert result == 2
78
79     m7 = Money(5,20)
80     m8 = Money(10,20)
81
82     result = m7.div(m8)
83
84     assert result == 1
85
86     m9 = Money(5,20)
87     m10 = Money(0,0)
88
89
90     with pytest.raises(ZeroDivisionError):
91         result = m9.div(m10)
92
93 def test_div_num():
94     m1 = Money(20,60)
95     m2 = 2
96
97     result = m1.div_num(m2)

```

Рисунок 31 – Тесты для класса Money

```

92
93 def test_div_num():
94     m1 = Money(20,60)
95     m2 = 2
96
97     result = m1.div_num(m2)
98     result.normalize()
99
100     assert result.rub == 10
101     assert result.kop == 30
102
103
104     m3 = Money(0,30)
105     m4 = 2
106
107     result = m3.div_num(m4)
108     result.normalize()
109
110     assert result.rub == 0
111     assert result.kop == 15
112
113     m5 = Money(10,20)
114     m6 = 0
115
116
117     with pytest.raises(ZeroDivisionError):
118         result = m5.div_num(m6)
119
120 def test_mul_num():
121     m1 = Money(20,60)
122     m2 = 2
123
124     result = m1.mul_num(m2)
125     result.normalize()
126

```

Рисунок 32 – Тесты для класса Money

```

123
124     result = m1.mul_num(m2)
125     result.normalize()
126
127     assert result.rub == 41
128     assert result.kop == 20
129
130     m1 = Money(20,60)
131     m2 = 0
132
133     result = m1.mul_num(m2)
134     result.normalize()
135
136     assert result.rub == 0
137     assert result.kop == 0
138
139 def test_eq():
140     m1 = Money(20,60)
141     m2 = Money(20,60)
142
143     assert m1.eq(m2)
144
145 def test_lt():
146     m1 = Money(20,60)
147     m2 = Money(10,30)
148
149     assert m2.lt(m1)
150
151 def test_gt():
152     m1 = Money(20,60)
153     m2 = Money(10,30)
154
155     assert m1.gt(m2)

```

Рисунок 33 – Тесты для класса Money

```
tests/test_money.py::test_io_read PASSED [ 8%]
tests/test_money.py::test_add PASSED [ 16%]
tests/test_money.py::test_sub PASSED [ 25%]
tests/test_money.py::test_div PASSED [ 33%]
tests/test_money.py::test_div_num PASSED [ 41%]
tests/test_money.py::test_mul_num PASSED [ 50%]
tests/test_money.py::test_eq PASSED [ 58%]
tests/test_money.py::test_lt PASSED [ 66%]
tests/test_money.py::test_gt PASSED [ 75%]
tests/test_money.py::test_le PASSED [ 83%]
tests/test_money.py::test_ge PASSED [ 91%]
tests/test_money.py::test_ne PASSED [100%]
```

Рисунок 34 – Результаты тестирования кода

Ответы на контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

Объявление класса осуществляется с помощью ключевого слова `class`, за которым следует имя класса (обычно с заглавной буквы) и двоеточие. Тело класса пишется с отступом.

```
class MyClass:
    # атрибуты и методы класса
    pass
```

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

1) **Атрибуты класса** принадлежат самому классу. Все экземпляры этого класса разделяют один и тот же атрибут класса. Изменение атрибута класса затронет все экземпляры.

2) **Атрибуты экземпляра** принадлежат конкретному объекту (экземпляру) класса. Они уникальны для каждого экземпляра. Изменение атрибута одного экземпляра не затронет другие.

3. Каково назначение методов класса?

Методы класса определяют поведение объектов этого класса. Они являются функциями, которые описывают, что объекты класса могут делать. Методы могут работать как с атрибутами экземпляра, так и с атрибутами класса.

4. Для чего предназначен метод `__init__()` класса?

Метод `__init__()` — это конструктор экземпляра класса. Он автоматически вызывается при создании нового объекта (экземпляра) класса.

Его основное назначение — инициализировать атрибуты нового объекта, то есть задать ему начальное состояние.

5. Каково назначение self?

self — это ссылка на текущий экземпляр класса. Через self методы получают доступ к атрибутам и другим методам этого конкретного объекта. Это первый параметр любого метода экземпляра, хотя сам язык не запрещает назвать его иначе, но это сильно противоречит общепринятым соглашениям.

6. Как добавить атрибуты в класс?

1) **Атрибуты класса:** объявляются прямо внутри тела класса, но вне любых методов.

```
class MyClass:  
    class_attr = "Я атрибут класса" # Добавление атрибута класса
```

2) **Атрибуты экземпляра:** обычно добавляются и инициализируются в методе `__init__` через self.

```
class MyClass:  
    def __init__(self, value):  
        self.instance_attr = value # Добавление атрибута экземпляра
```

Также атрибут экземпляру можно добавить динамически в любом другом методе или даже вне класса, обратившись к объекту:

```
obj = MyClass()  
  
obj.new_attr = 10
```

7. Как осуществляется управление доступом к методам и атрибутам в языке Python?

Python использует модель «согласованного доступа» (we are all consenting adults here), а не строгое управление доступом. Для обозначения уровня доступа используются соглашения об именовании:

1) **Публичные (public)** атрибуты: attr_name

2) **Защищенные (protected)** атрибуты (сигнал для программиста, что с ним нужно быть осторожнее, а не для интерпретатора): `_attr_name`

3) **Приватные (private)** атрибуты (интерпретатор искажает их имя, чтобы усложнить случайный доступ извне, но прямой доступ все равно возможен): `__attr_name`

8. Каково назначение функции `isinstance`?

Функция `isinstance(object, classinfo)` проверяет, является ли объект `object` экземпляром класса `classinfo` или экземпляром любого из его подклассов (дочерних классов). Возвращает `True` или `False`. Она используется для проверки типа объекта.

Вывод: в ходе работы были приобретены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python