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 № 18

Выполнил:
Текеева Мадина Азрет-Алиевна
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: 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.

Репозиторий: https://github.com/bickrosss/TM_oop_lab1

Порядок выполнения работы:

1. Создали класс Book с атрибутами класса. Определили атрибуты класса, общие для всех экземпляров. Показали доступ к ним. Создали экземпляр класса Book. Показали, что экземпляр имеет доступ к атрибутам класса.

```python
#Класс Book с атрибутами класса
class Book:
    material = "paper"
    cover = "paperback"
    all_books = []
# доступ к атрибутам класса через имя класса
print("Атрибуты класса через Book:")
print(Book.material)  # "paper"
print(Book.cover)  # "paperback"
print(Book.all_books)  # []

# экземпляр класса Book
my_book = Book()

# доступ к атрибутам класса через экземпляр
print("Атрибуты класса через экземпляр:")
print(my_book.material)  # "paper"
print(my_book.cover)  # "paperback"
print(my_book.all_books)  # []
```

Рисунок 1. Создание класса Book

```
Атрибуты класса через Book:
paper
paperback
[]
Атрибуты класса через экземпляр:
paper
paperback
[]
```

Рисунок 2. Вывод экземпляра класса

2. Создали класс River с конструктором init. Создали несколько рек и показали их в списке. Создали метод get_info с параметром self. Показали два способа вызова метода.

```
class River:
    all_rivers = []

    def __init__(self, name, length):
        self.name = name
        self.length = length
        River.all_rivers.append(self)

    def get_info(self):
        print("Длина {0} равна {1} км".format(self.name, self.length))

volga = River("Волга", 3530)
seine = River("Сена", 776)
nile = River("Нил", 6852)

# печатаем все названия рек из атрибута класса
print("Список всех рек:")
for river in River.all_rivers:
    print(river.name)

# вызов метода get_info() разными способами
print("Информация о реках:\n" + "Способ 1: self.method()")
volga.get_info()

print("Способ 2: class.method(self)")
River.get_info(volga)
```

Рисунок 3. Класс River

```
Список всех рек:
Волга
Сена
Нил
Информация о реках:
Способ 1: self.method()
Длина Волга равна 3530 км
Способ 2: class.method(self)
Длина Волга равна 3530 км
```

Рисунок 4. Вывод

3. Создали класс Pet с атрибутами класса и экземпляра. Показали разницу между изменением атрибутов класса и экземпляра.

```python
class Pet:
    kind = "mammal"
    n_pets = 0
    pet_names = []

    def __init__(self, spec, name):
        self.spec = spec
        self.name = name
        self.legs = 4
print("создание экземпляров:")
tom = Pet("cat", "Tom")
avocado = Pet("dog", "Avocado")
ben = Pet("goldfish", "Benjamin")

print(f"Число питомцев: {Pet.n_pets}")
print(f"Имена питомцев: {Pet.pet_names}")
print()

# доступ к атрибуту через класс
print("изменение атрибутов класса:")
Pet.n_pets = 3
print(Pet.n_pets)
print(tom.n_pets)
print(avocado.n_pets)
print(ben.n_pets)

```

Рисунок 5. Класс Pet

```
изменение через экземпляры:
3
4
4
4
mammal
mammal
mammal
fish
добавление имен:
['Tom', 'Avocado', 'Benjamin']
['Tom', 'Avocado', 'Benjamin']
['Tom', 'Avocado', 'Benjamin']
['Tom', 'Avocado', 'Benjamin']
создание отдельных списко:
['Tom', 'Avocado', 'Benjamin']
['Tom']
['Avocado']
['Benjamin']
изменение атрибутов
0
4
4
добавление новых атрибутов к классу:
['cat', 'dog', 'goldfish']
['cat', 'dog', 'goldfish']
['cat', 'dog', 'goldfish']
['cat', 'dog', 'goldfish']
corgi
```

Рисунок 6. Вывод

4. Создали класс Ship с методами. Показали работу методов.

```python
class Ship:
    def __init__(self, name, capacity):
        self.name = name
        self.capacity = capacity
        self.cargo = 0

    def sail(self):
        print("{} has sailed!".format(self.name))

    def convert_cargo(self):
        cargo_kg = self.cargo * 1000
        return cargo_kg

    def name_captain(self, cap):
        self.captain = cap
        print("{} is the captain of the {}".format(self.captain, self.name))

    def load_cargo(self, weight):
        if self.cargo + weight <= self.capacity:
            self.cargo += weight
            print("Loaded {} tons".format(weight))
        else:
            print("Cannot load that much")

    def unload_cargo(self, weight):
        if self.cargo - weight >= 0:
            self.cargo -= weight
            print("Unloaded {} tons".format(weight))
        else:
            print("Cannot unload that much")

    def free_space(self):
        return self.capacity - self.cargo
```

Рисунок 7. Класс Ship

```
создание корабля:
корабль создан: Black Pearl, вместимость: 800 тонн
вызов метода sail:
Black Pearl has sailed!
конвертация груза в килограммы:
груз в килограммах: 0 кг
создание атрибута captain через метод name_captain:
Jack Sparrow is the captain of the Black Pearl
капитан корабля: Jack Sparrow
работа с грузом:
текущий груз: 0 тонн
свободное место: 800 тонн
загрузка груза:
Loaded 600 tons
Cannot load that much
текущий груз: 600 тонн
свободное место: 200 тонн
разгрузка груза:
Unloaded 400 tons
Cannot unload that much
текущий груз: 200 тонн
свободное место: 600 тонн
```

Рисунок 8. Вывод

5. Создали класс Rectangle с инкапсуляцией. Показали работу с приватными атрибутами

```python
class Rectangle1:
    def __init__(self, width, height):
        self._width = width
        self._height = height

    def get_width(self):
        return self._width

    def set_width(self, w):
        self._width = w

    def get_height(self):
        return self._height

    def set_height(self, h):
        self._height = h

    def area(self):
        return self._width * self._height


print("работа с защищенными атрибутами:")
rect1 = Rectangle1(10, 20)
print(f"через getter: {rect1.get_width()}")
print(f"прямой доступ: {rect1._width}")
print(f"площадь: {rect1.area()}")

print("изменение защищенных атрибутов:")
rect1.set_width(15)
print(f"через setter: {rect1.get_width()}")
rect1._width = 25
print(f"прямое изменение: {rect1._width}")
print()
```

Рисунок 9. Объявление класса Rectangle с инкапсуляцией

```
работа с защищенными атрибутами:
через getter: 10
прямой доступ: 10
площадь: 200
изменение защищенных атрибутов:
через setter: 15
прямое изменение: 25
```

Рисунок 10. Вывод данных экземпляра класса

```python
class Rectangle2:
    def __init__(self, width, height):
        self.__width = width
        self.__height = height

    def get_width(self):
        return self.__width

    def set_width(self, w):
        self.__width = w

    def get_height(self):
        return self.__height

    def set_height(self, h):
        self.__height = h

    def area(self):
        return self.__width * self.__height


print("работа с приватными атрибутами:")
rect2 = Rectangle2(30, 40)
print(f"через getter: {rect2.get_width()}")
print(f"площадь: {rect2.area()}")

print(f"{rect2._Rectangle2__width}")
rect2._Rectangle2__width = 35
print(f"после изменения: {rect2.get_width()}")
print()
```

Рисунок 11. Класса Rectangle с приватными атрибутами

```
работа с приватными атрибутами:
через getter: 30
площадь: 1200
30
после изменения: 35
```

Рисунок 12. Вывод

6. Создали класс Rectangle со свойствами @property.

```python
class Rectangle3:
    def __init__(self, width, height):
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height


print("работа со свойствами")
```

Рисунок 13. Класс Rectangle со свойствами @property

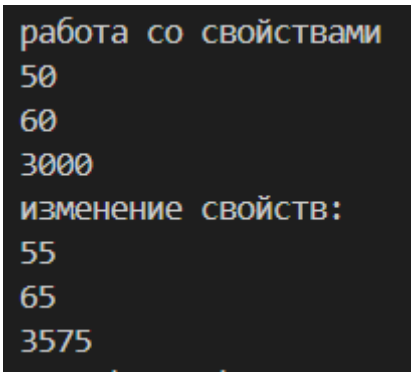


Рисунок 14. Вывод

7. Создали класс Rational для работы с дробями. Показали арифметические операции с дробями.

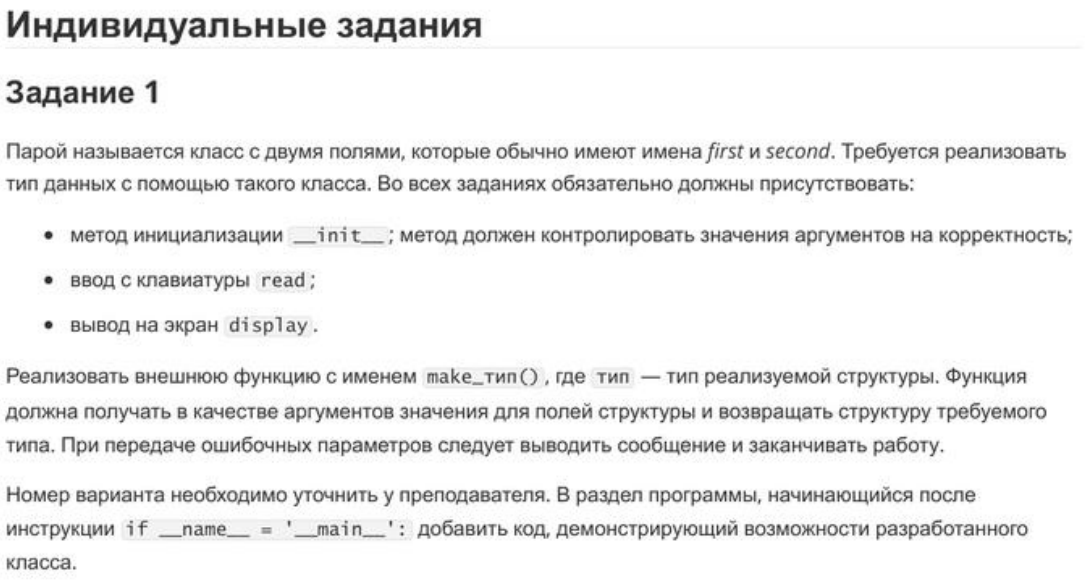
```python
class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError()
        self.__numerator = abs(a)
        self.__denominator = abs(b)
        self.__reduce()

    def __reduce(self):
        def gcd(a, b):
            if a == 0:
                return b
            elif b == 0:
                return a
            elif a >= b:
                return gcd(a % b, b)
            else:
                return gcd(a, b % a)
        c = gcd(self.__numerator, self.__denominator)
        self.__numerator //= c
        self.__denominator //= c

    @property
    def numerator(self):
        return self.__numerator

    @property
    def denominator(self):
        return self.__denominator

    def read(self, prompt=None):
        line = input() if prompt is None else input(prompt)
```

Рисунок 15. Класс Rational

```
демонстрация работы с рациональными дробями:
r1 = 3/4
введите обыкновенную дробь (формат a/b): 4/9
r2 = 4/9

арифметические операции:
r2 + r1 = 43/36
r2 - r1 = 11/36
r2 * r1 = 1/3
r2 / r1 = 16/27

операции сравнения:
r1 == r2: False
r1 > r2: True
r1 < r2: False

дополнительные примеры:
r7 = 6/8 = 3/4
r1 == r7: True
```

Рисунок 16. Вывод

8. Выполнили индивидуальное задание 1.



Индивидуальные задания

Задание 1

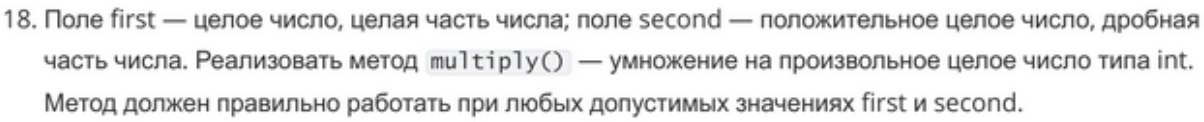
Парой называется класс с двумя полями, которые обычно имеют имена *first* и *second*. Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать:

- 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 17. Индивидуально задание 1



18. Поле first — целое число, целая часть числа; поле second — положительное целое число, дробная часть числа. Реализовать метод `multiply()` — умножение на произвольное целое число типа int. Метод должен правильно работать при любых допустимых значениях first и second.

Рисунок 18. Вариант 18 индивидуального задания

Создание класса.

```
class Pair:
    def __init__(self, first=0, second=0):
        self.first = int(first)
        self.second = int(second)
    def read(self):
        print("Введите целую часть числа: ")
        first_value = input()
        self.first = int(first_value)
        print("Введите дробную часть числа: ")
```

```
        second_value = input()
        self.second = int(second_value)
    def display(self):
        print(f"Число: {self.first}.{self.second}")
    def multiply(self, number):
        full_number = float(f"{self.first}.{self.second}")
        result_number = full_number * number
        result_str = str(result_number)
        parts = result_str.split('.')
        new_first = int(parts[0])
        new_second = int(parts[1]) if len(parts) > 1 else 0
        result_pair = Pair(new_first, new_second)
        return result_pair
def make_pair(first, second):
    new_pair = Pair(first, second)
    return new_pair
```

Тестирование правильности работы класса и его методов.

Task

```
from pair import Pair
from pair import make_pair
if __name__ == '__main__':
    print("Создание объектов через конструктор:")
    pair1 = Pair(3, 5)
```

```
    pair1.display()
    pair2 = Pair(2, 44)
    pair2.display()
    print("Создание объектов через функцию make_pair:")
    pair3 = make_pair(7, 8)
    pair3.display()
    pair4 = make_pair(4, 20)
    pair4.display()
    print("Ввод данных с клавиатуры:")
    pair5 = Pair()
    pair5.read()
    print("Результат ввода:")
    pair5.display()
    print("Операция умножения числа на целое число:")
    test_pair = Pair(2, 5)
    print("Исходное число:")
    test_pair.display()
    print("Введите целое число для умножения: ")
    multiplier = int(input())
    result = test_pair.multiply(multiplier)
    print(f"Результат умножения {test_pair.first}.{test_pair.second} на
{multiplier}:")
    result.display()
```

```
print("Работа с различными множителями:")
original_pair = Pair(3, 14)
multipliers = [2, -1, 0, 5]
for multiplier in multipliers:
    result = original_pair.multiply(multiplier)
    print(f"{original_pair.first}.{original_pair.second} * {multiplier} = ", end="")
    result.display()
print()
print("Демонстрация цепочки операций:")
start_pair = Pair(1, 5)
print("Начальное число:")
start_pair.display()
print("Введите первый множитель: ")
mult1 = int(input())
intermediate = start_pair.multiply(mult1)
print(f"После умножения на {mult1}:")
intermediate.display()
print("Введите второй множитель: ")
mult2 = int(input())
final_result = intermediate.multiply(mult2)
print(f"После умножения на {mult2}:")
final_result.display()
```

Test

```
from tasks.pair import Pair
from tasks.pair import make_pair
def test_pair_creation():
    print("Тест 1: Создание объекта Pair через конструктор")
    pair = Pair(3, 14)
    print("Ожидаем: 3.14")
    pair.display()
def test_make_pair_function():
    print("Тест 2: Создание объекта Pair через функцию make_pair")
    pair = make_pair(2, 5)
    print("Ожидаем: 2.5")
    pair.display()
def test_pair_multiply_positive():
    print("Тест 3: Умножение числа на положительное число")
    pair = Pair(2, 5)
    print("Исходное число: 2.5")
    result = pair.multiply(2)
    print("После умножения на 2:")
    result.display()
    print("Ожидаем: 5.0")
def test_pair_multiply_negative():
    print("Тест 4: Умножение числа на отрицательное число")
```

```
    pair = Pair(1, 5)
    print("Исходное число: 1.5")
    result = pair.multiply(-2)
    print("После умножения на -2:")
    result.display()
    print("Ожидаем: -3.0")
def test_pair_multiply_zero():
    print("Тест 5: Умножение числа на ноль")
    pair = Pair(5, 5)
    print("Исходное число: 5.5")
    result = pair.multiply(0)
    print("После умножения на 0:")
    result.display()
    print("Ожидаем: 0.0")
def test_pair_multiply_large():
    print("Тест 6: Умножение числа на большое число")
    pair = Pair(10, 25)
    print("Исходное число: 10.25")
    result = pair.multiply(100)
    print("После умножения на 100:")
    result.display()
    print("Ожидаем: 1025.0")
if __name__ == "__main__":
```

```
test_pair_creation()

test_make_pair_function()

test_pair_multiply_positive()

test_pair_multiply_negative()

test_pair_multiply_zero()

test_pair_multiply_large()
```

9. Выполнили индивидуальное задание 2.

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 19. 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 20. Вариант 18

Создали класс для решения поставленной задачи.

```
class Money:

    def __init__(self, rubles=0, kopecks=0):

        self.rubles = rubles

        self.kopecks = kopecks
```

```
    self._normalize()


def _normalize(self):
    if self.kopecks >= 100:
        self.rubles += self.kopecks // 100
        self.kopecks = self.kopecks % 100
    elif self.kopecks < 0:
        self.rubles -= (-self.kopecks) // 100 + 1
        self.kopecks = 100 - ((-self.kopecks) % 100)


def read(self):
    self.rubles = int(input("Введите рубли: "))
    self.kopecks = int(input("Введите копейки: "))
    self._normalize()


def display(self):
    print(f"{self.rubles} руб. {self.kopecks:02d} коп.")


def add(self, other):
    new_rubles = self.rubles + other.rubles
    new_kopecks = self.kopecks + other.kopecks
    return Money(new_rubles, new_kopecks)
```

```
def subtract(self, other):
    new_rubles = self.rubles - other.rubles
    new_kopecks = self.kopecks - other.kopecks
    return Money(new_rubles, new_kopecks)

def divide_amount(self, n):
    total_kopecks = self.rubles * 100 + self.kopecks
    result_kopecks = total_kopecks // n
    return Money(0, result_kopecks)

def divide_by_money(self, other):
    total_kopecks1 = self.rubles * 100 + self.kopecks
    total_kopecks2 = other.rubles * 100 + other.kopecks
    return total_kopecks1 / total_kopecks2

def multiply(self, factor):
    total_kopecks = self.rubles * 100 + self.kopecks
    result_kopecks = int(total_kopecks * factor)
    return Money(0, result_kopecks)

def equals(self, other):
    return (self.rubles == other.rubles) and (self.kopecks == other.kopecks)
```

```
def greater(self, other):
    total1 = self.rubles * 100 + self.kopecks
    total2 = other.rubles * 100 + other.kopecks
    return total1 > total2


def less(self, other):
    total1 = self.rubles * 100 + self.kopecks
    total2 = other.rubles * 100 + other.kopecks
    return total1 < total2
```

Тестирование правильности работы класса и его методов.

Task

```
from money import Money


if __name__ == "__main__":
    print("демонстрация работы класса Money:")
    print()

    # Создание объектов
    money1 = Money(150, 75)
    money2 = Money(80, 50)

    print("Цена1 = ", end="")
    money1.display()
```

```
print("Цена2 = ", end="")
money2.display()
print()

# Арифметические операции
sum_money = money1.add(money2)
print("Цена1 + Цена2 = ", end="")
sum_money.display()

diff_money = money1.subtract(money2)
print("Цена1 - Цена2 = ", end="")
diff_money.display()
print()

# Деление и умножение
divided = money1.divide_amount(3)
print("Цена1 / 3 = ", end="")
divided.display()

ratio = money1.divide_by_money(money2)
print(f"Цена1 / Цена2 = {ratio:.2f}")

multiplied = money2.multiply(1.5)
```

```
    print("Цена2 * 1.5 = ", end="")
    multiplied.display()
    print()

    # Сравнение
    print(f"money1 == money2: {money1.equals(money2)}")
    print(f"money1 > money2: {money1.greater(money2)}")
    print(f"money1 < money2: {money1.less(money2)}")
```

Test

```
import sys
import os

sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from tasks.money import Money

def test_money_creation():
    print("Тест 1: Создание объекта Money")
    money = Money(100, 50)
    print("Ожидаем: 100 руб. 50 коп.")
    money.display()
```

```
def test_money_addition():
    print("Тест 2: Сложение денежных сумм")
    money1 = Money(100, 50)
    money2 = Money(50, 75)
    result = money1.add(money2)
    print("100.50 + 50.75 = ", end="")
    result.display()
    print("Ожидаем: 151 руб. 25 коп.")

def test_money_subtraction():
    print("Тест 3: Вычитание денежных сумм")
    money1 = Money(100, 50)
    money2 = Money(50, 75)
    result = money1.subtract(money2)
    print("100.50 - 50.75 = ", end="")
    result.display()
    print("Ожидаем: 49 руб. 75 коп.")

def test_money_division():
    print("Тест 4: Деление денежной суммы на число")
    money = Money(100, 50)
    result = money.divide_amount(2)
    print("100.50 / 2 = ", end="")
```

```
    result.display()
    print("Ожидаем: 50 руб. 25 коп.")

def test_money_multiplication():
    print("Тест 5: Умножение денежной суммы на число")
    money = Money(100, 50)
    result = money.multiply(1.5)
    print("100.50 * 1.5 = ", end="")
    result.display()
    print("Ожидаем: 150 руб. 75 коп.")

def test_money_comparison():
    print("Тест 6: Сравнение денежных сумм")
    money1 = Money(100, 50)
    money2 = Money(50, 75)

    print(f"100.50 == 50.75: {money1.equals(money2)}")
    print(f"100.50 > 50.75: {money1.greater(money2)}")
    print(f"100.50 < 50.75: {money1.less(money2)}")
    print("Ожидаем: False, True, False")

if __name__ == "__main__":
    test_money_creation()
```

```
test_money_addition()

test_money_subtraction()

test_money_division()

test_money_multiplication()

test_money_comparison()

print("Все тесты Money завершены!")
```

Вывод: в ходе выполнения лабораторной работы были приобретены навыки работы с классами и объектами при написании программ с помощью языка программирования Python.