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 №4.1
дисциплины «Объектно-ориентированное программирование»

Выполнила:
Мурашко Анастасия Юрьевна
3 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Проверил:
Богданов С.С., кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____Дата защиты_____

Ставрополь, 2024 г.

Тема: Элементы объектно-ориентированного программирования в языке Python.

Цель работы: приобретение навыков написания многопоточных приложений на языке программирования Python версии 3.x.

Ход работы:

Изучила теоретический материал работы, создала общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавил файл .gitignore с необходимыми правилами.

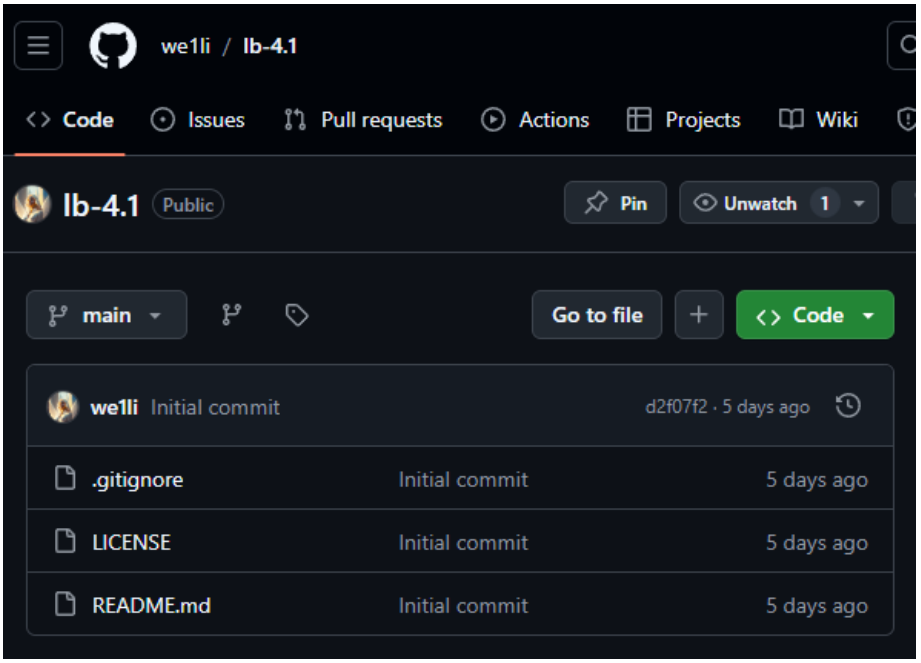

Рисунок 1. Новый репозиторий

Проклонировала свой репозиторий на свой компьютер.

Организовала свой репозиторий в соответствие с моделью ветвления git-flow, появилась новая ветка develop.

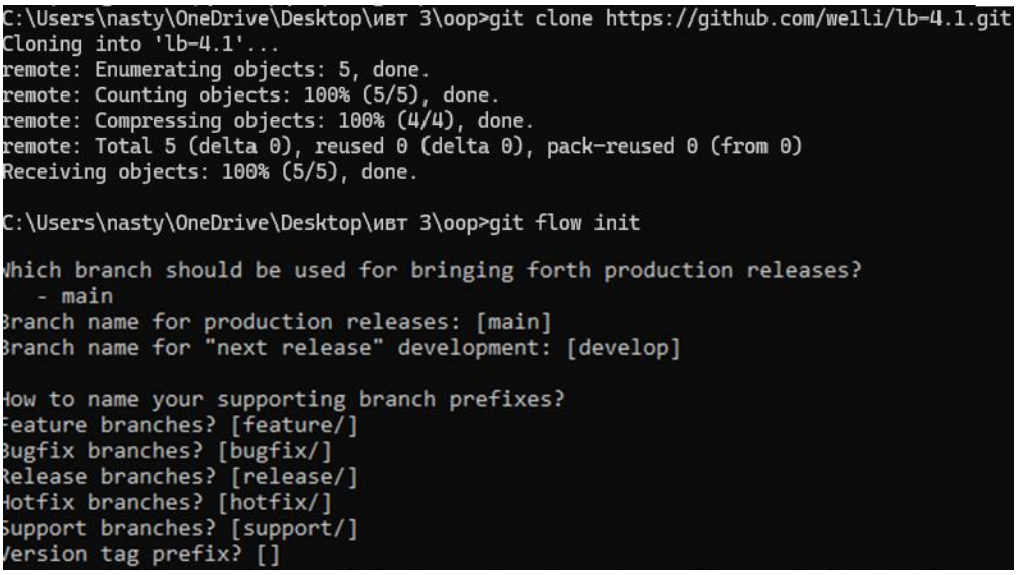

Рисунок 2. Клонирование и модель ветвления git-flow

Реализовывала примеры и индивидуальные задания на основе ветки develop, без создания дополнительной ветки feature/(название ветки) по указанию преподавателя.

Проработала пример лабораторной работы.

Пример 1: Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a — числитель, b — знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции:

- сложения;
- вычитания;
- умножения;
- деления;
- сравнения.

Должна быть реализована приватная функция сокращения дроби reduce, которая обязательно вызывается при выполнении арифметических операций .

Напишем программу для решения поставленной задачи.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
```

```
        if b == 0:
            raise ValueError("Знаменатель не может быть равен нулю.")
        self.__numerator = abs(a)
        self.__denominator = abs(b)
        self.__reduce()

    # Сокращение дроби
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        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

    # Прочитать значение дроби с клавиатуры. Дробь вводится как a/b.
    def read(self, prompt=None):
        line = input() if prompt is None else input(prompt)
        parts = list(map(int, line.split('/', maxsplit=1)))
        if parts[1] == 0:
            raise ValueError("Знаменатель не может быть равен нулю.")
        self.__numerator = abs(parts[0])
        self.__denominator = abs(parts[1])
        self.__reduce()

    # Вывести дробь на экран
    def display(self):
        print(f"{self.__numerator}/{self.__denominator}")

    # Сложение обыкновенных дробей
    def add(self, rhs):
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator + self.denominator *
rhs.numerator
            b = self.denominator * rhs.denominator
```

```
            return Rational(a, b)
        else:
            raise ValueError("Аргумент должен быть объектом класса Rational.")

    # Вычитание обыкновенных дробей
    def sub(self, rhs):
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator - self.denominator *
rhs.numerator
            b = self.denominator * rhs.denominator
            return Rational(a, b)
        else:
            raise ValueError("Аргумент должен быть объектом класса Rational.")

    # Умножение обыкновенных дробей
    def mul(self, rhs):
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.numerator
            b = self.denominator * rhs.denominator
            return Rational(a, b)
        else:
            raise ValueError("Аргумент должен быть объектом класса Rational.")

    # Деление обыкновенных дробей
    def div(self, rhs):
        if isinstance(rhs, Rational):
            a = self.numerator * rhs.denominator
            b = self.denominator * rhs.numerator
            if b == 0:
                raise ValueError("Деление на ноль невозможно.")
            return Rational(a, b)
        else:
            raise ValueError("Аргумент должен быть объектом класса Rational.")

    # Отношение обыкновенных дробей (равенство)
    def equals(self, rhs):
        if isinstance(rhs, Rational):
            return (self.numerator == rhs.numerator) and (self.denominator ==
rhs.denominator)
        else:
            return False

    # Сравнение: больше
    def greater(self, rhs):
        if isinstance(rhs, Rational):
            v1 = self.numerator / self.denominator
            v2 = rhs.numerator / rhs.denominator
            return v1 > v2
        else:
            return False
```

```
    # Сравнение: меньше
    def less(self, rhs):
        if isinstance(rhs, Rational):
            v1 = self.numerator / self.denominator
            v2 = rhs.numerator / rhs.denominator
            return v1 < v2
        else:
            return False


if __name__ == '__main__':
    r1 = Rational(3, 4)
    r1.display()

    r2 = Rational()
    r2.read("Введите обыкновенную дробь: ")
    r2.display()

    r3 = r2.add(r1)
    r3.display()

    r4 = r2.sub(r1)
    r4.display()

    r5 = r2.mul(r1)
    r5.display()

    r6 = r2.div(r1)
    r6.display()
```

```
3/4
Введите обыкновенную дробь: 5/6
5/6
19/12
1/12
5/8
10/9
PS C:\Users\Student\Downloads>
```

Рисунок 3. Результат работы примера

Индивидуальное задание 1.

Для своего варианта лабораторной работы 4.1 необходимо реализовать тип данных с помощью класса с двумя полями, которые обычно имеют имена first и second.

Вариант – 8.

Поле first — целое число, левая граница диапазона, включается в

диапазон; поле second — целое число, правая граница диапазона, не включается в диапазон. Пара чисел представляет полуоткрытый интервал [first, second). Реализовать метод rangecheck() — проверку заданного целого числа на принадлежность диапазону.

Эта программа реализует класс Pair, который представляет пару целых чисел, где первое число (first) должно быть меньше второго (second). Программа включает функции для работы с такими парами чисел, проверки интервалов, а также обработки ошибок, связанных с некорректными данными.

Разбор программы:

1. **Импорт библиотеки**
2. **Класс Pair**
    - **Конструктор (__init__)**:
        - Инициализирует объект с двумя числами first и second.
        - Проверяет, являются ли оба аргумента целыми числами.
        - Проверяет, что значение first меньше значения second. Если эти условия не выполняются, возбуждается исключение ValueError.
    - **Метод read**:
        - Позволяет вводить значения first и second с клавиатуры.
        - Выполняется проверка, что введённое значение first меньше, чем second.
        - Если возникают ошибки (например, если введённое значение не целое число), выводится сообщение об ошибке.
    - **Метод display**:
        - Выводит текущие значения пары чисел в формате [first, second), где правая граница интервала не включается.
    - **Метод rangecheck**:
        - Принимает число number и проверяет, попадает ли оно в интервал [first, second), то есть находится ли число в диапазоне между first (включительно) и second.

3. **Функция make_pair**:
    - Принимает два аргумента и создает объект Pair.
    - Если при создании объекта возникает ошибка (например, переданы некорректные данные), она обрабатывается, и выводится сообщение об ошибке.

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Pair:
    def __init__(self, first, second):
        # Инициализация объекта Pair с двумя целыми числами: first и second.
        # Проверка, что оба числа являются целыми.
        if not isinstance(first, int) or not isinstance(second, int):
            raise ValueError("Поля first и second должны быть целыми числами.")
        # Проверка, что first меньше second.
        if first >= second:
            raise ValueError("Поле first должно быть меньше поля second.")
        self.first = first
        self.second = second

    def read(self):
        # Чтение значений first и second с клавиатуры.
        try:
            self.first = int(input("Введите первое число (левая граница
диапазона): "))
            self.second = int(input("Введите второе число (правая граница
диапазона, не включается): "))
            # Проверка, что first меньше second.
            if self.first >= self.second:
                raise ValueError("Левая граница должна быть меньше правой
границы.")
        except ValueError as e:
            print(f"Ошибка ввода: {e}")

    def display(self):
        # Отображение текущей пары чисел.
        print(f"Пара чисел: [{self.first}, {self.second})")

    def rangecheck(self, number):
        """Проверка, принадлежит ли число number интервалу [first, second)."""
        return self.first <= number < self.second

def make_pair(first, second):
    """Функция для создания объекта типа Pair."""
    try:
```

```
        return Pair(first, second)
    except ValueError as e:
        print(f"Ошибка создания объекта: {e}")
        return None

if __name__ == '__main__':
    # Демонстрация работы программы

    # Создание объекта Pair
    pair = make_pair(10, 20)
    if pair:

        # Ввод новой пары чисел с клавиатуры.
        print("\nВвод новой пары с клавиатуры:")
        pair.read()
        pair.display()

        # Проверка числа, введенного пользователем, на принадлежность интервалу.
        number_to_check = int(input("\nВведите число для проверки принадлежности
интервалу: "))
        print(f"Число {number_to_check} в интервале [{pair.first},
{pair.second}): {pair.rangecheck(number_to_check)}")
```

```
Ввод новой пары с клавиатуры:
Введите первое число (левая граница диапазона): 3
Введите второе число (правая граница диапазона, не включается): 6
Пара чисел: [3, 6)

Введите число для проверки принадлежности интервалу: 11
Число 11 в интервале [3, 6): False
```

```
Ввод новой пары с клавиатуры:
Введите первое число (левая граница диапазона): 3
Введите второе число (правая граница диапазона, не включается): 6
Пара чисел: [3, 6)

Введите число для проверки принадлежности интервалу: 3
Число 3 в интервале [3, 6): True
```

Рисунок 4. Результат работы программы

Индивидуальное задание 2.

Составить программу с использованием классов и объектов для решения задачи.

Создать класс Time для работы со временем в формате «час:минута:секунда». Класс должен включать в себя не менее четырех функций инициализации: числами, строкой (например, «23:59:59»), секундами

и временем. Обязательными операциями являются: вычисление разницы между двумя моментами времени в секундах, сложение времени и заданного количества секунд, вычитание из времени заданного количества секунд, сравнение моментов времени, перевод в секунды, перевод в минуты (с округлением до целой минуты)

1. Класс Time

Класс представляет собой модель времени с параметрами hours, minutes и seconds.

- **Инициализация (__init__)**:
  - Конструктор принимает три аргумента: hours, minutes и seconds, которые по умолчанию равны 0.
  - После инициализации времени вызывается приватный метод _normalize(), который нормализует время (преобразует значения минут и секунд, если они выходят за пределы допустимых значений).
- **Метод _normalize()**:

Приводит значения времени к корректному формату:

Если секунд больше или равно 60, они конвертируются в минуты;

Если минут больше или равно 60, они конвертируются в часы;

Если часов больше или равно 24, они циклически обновляются.

2. Методы класса (@classmethod):

- **from_string(time_string)**:
  - Создает объект Time из строки формата HH:MM:SS.
  - Разделяет строку по символу: и преобразует отдельные элементы в целые числа.
  - В случае неправильного формата выводит сообщение об ошибке и возвращает None.
- **from_seconds(total_seconds)**:
  - Создает объект Time, переводя количество секунд в часы, минуты и секунды.

- Количество часов вычисляется как целое деление на 3600, минуты – через остаток от деления на 3600, а секунды — через остаток от деления на 60.

3. Операции с временем:

- **to_seconds()**:
  - Преобразует текущее время в общее количество секунд.
- **to_minutes()**:
  - Преобразует текущее время в минуты с округлением до ближайшего целого значения.
- **time_difference(other_time)**:
  - Вычисляет разницу между двумя объектами Time в секундах. Возвращает абсолютное значение разницы, чтобы результат всегда был положительным.
- **add_seconds(seconds)**:
  - Добавляет к текущему времени указанное количество секунд. Возвращает новый объект Time с обновленным временем.
- **subtract_seconds(seconds)**:
  - Вычитает из текущего времени указанное количество секунд и возвращает новый объект Time.

4. Методы сравнения (__eq__, __lt__, __gt__):

- **__eq__(other)**:
  - Проверяет, равны ли два объекта времени, сравнивая их в секундах.
- **__lt__(other)**:
  - Проверяет, меньше ли текущее время, чем другое.
- **__gt__(other)**:
  - Проверяет, больше ли текущее время, чем другое.

5. Чтение и вывод времени:

- **read()**:
  - Позволяет вводить время с клавиатуры в формате HH:MM:SS.

Вызывает метод from_string для преобразования строки в объект Time.

- **display()**:
  - Выводит время в формате HH:MM:SS, где часы, минуты и секунды выводятся с ведущими нулями, если требуется.

6. Основная часть программы (if __name__ == '__main__':):

- **Ввод времени**:
  - Программа предлагает пользователю ввести два момента времени (первый и второй) с клавиатуры.
- **Сравнение времени**:
  - Сравнивает два времени и выводит результат сравнения:
    - Если они равны, выводится сообщение "Времена равны".
    - Если первое время меньше второго, выводится сообщение "Первое время меньше второго".
    - Если первое больше второго, выводится сообщение "Первое время больше второго".
- **Добавление и вычитание секунд**:
  - Пользователь вводит количество секунд для добавления или вычитания из первого времени, и программа выводит результат.
- **Перевод в секунды и минуты**:
  - Программа выводит первое время в виде общего количества секунд и общего количества минут (с округлением).

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Time:
    def __init__(self, hours=0, minutes=0, seconds=0):
        self.hours = hours
        self.minutes = minutes
        self.seconds = seconds
        self._normalize()

    @classmethod
    def from_string(cls, time_string):
```

```
        try:
            h, m, s = map(int, time_string.split(':'))
            return cls(h, m, s)
        except ValueError:
            print("Неправильный формат времени. Ожидается формат 'HH:MM:SS'.")
            return None

    @classmethod
    def from_seconds(cls, total_seconds):
        hours = total_seconds // 3600
        minutes = (total_seconds % 3600) // 60
        seconds = total_seconds % 60
        return cls(hours, minutes, seconds)

    def _normalize(self):
        """Нормализация времени для корректного представления."""
        if self.seconds >= 60:
            self.minutes += self.seconds // 60
            self.seconds %= 60
        if self.minutes >= 60:
            self.hours += self.minutes // 60
            self.minutes %= 60
        if self.hours >= 24:
            self.hours %= 24

    def to_seconds(self):
        """Перевод времени в общее количество секунд."""
        return self.hours * 3600 + self.minutes * 60 + self.seconds

    def to_minutes(self):
        """Перевод времени в минуты с округлением до целого числа."""
        total_seconds = self.to_seconds()
        return round(total_seconds / 60)

    def time_difference(self, other_time):
        """Вычисление разницы между двумя моментами времени в секундах."""
        return abs(self.to_seconds() - other_time.to_seconds())

    def add_seconds(self, seconds):
        """Добавление секунд к текущему времени."""
        total_seconds = self.to_seconds() + seconds
        return Time.from_seconds(total_seconds)

    def subtract_seconds(self, seconds):
        """Вычитание секунд из текущего времени."""
        total_seconds = self.to_seconds() - seconds
        return Time.from_seconds(total_seconds)

    def __eq__(self, other):
        return self.to_seconds() == other.to_seconds()
```

```
    def __lt__(self, other):
        return self.to_seconds() < other.to_seconds()

    def __gt__(self, other):
        return self.to_seconds() > other.to_seconds()

    def read(self):
        """Ввод времени с клавиатуры в формате HH:MM:SS."""
        time_string = input("Введите время в формате HH:MM:SS: ")
        new_time = Time.from_string(time_string)
        if new_time:
            self.hours, self.minutes, self.seconds = new_time.hours,
new_time.minutes, new_time.seconds

    def display(self):
        """Вывод времени в формате HH:MM:SS."""
        print(f"{self.hours:02}:{self.minutes:02}:{self.seconds:02}")

if __name__ == '__main__':
    # Ввод первого времени с клавиатуры
    print("Введите первое время:")
    time1 = Time()
    time1.read()

    # Ввод второго времени с клавиатуры для сравнения
    print("\nВведите второе время для сравнения:")
    time2 = Time()
    time2.read()

    # Вывод времени
    print("\nПервое время:")
    time1.display()
    print("Второе время:")
    time2.display()

    # Сравнение времени
    print("\nСравнение:")
    if time1 == time2:
        print("Времена равны.")
    elif time1 < time2:
        print("Первое время меньше второго.")
    else:
        print("Первое время больше второго.")

    # Ввод количества секунд для добавления
    seconds_to_add = int(input("\nВведите количество секунд для добавления к
первому времени: "))
    new_time = time1.add_seconds(seconds_to_add)
    print(f"\nНовое время после добавления {seconds_to_add} секунд:")
    new_time.display()
```

```
    # Ввод количества секунд для вычитания
    seconds_to_subtract = int(input("\nВведите количество секунд для вычитания из
первого времени: "))
    new_time = time1.subtract_seconds(seconds_to_subtract)
    print(f"\nНовое время после вычитания {seconds_to_subtract} секунд:")
    new_time.display()

    # Вывод времени в секундах и минутах
    print(f"\nПервое время в секундах: {time1.to_seconds()} секунд")
    print(f"Первое время в минутах (с округлением): {time1.to_minutes()} минут")
```

```
Введите первое время:
Введите время в формате HH:MM:SS: 12:54:12

Введите второе время для сравнения:
Введите время в формате HH:MM:SS: 13:12:26

Первое время:
12:54:12
Второе время:
13:12:26

Сравнение:
Первое время меньше второго.

Введите количество секунд для добавления к первому времени: 3600

Новое время после добавления 3600 секунд:
13:54:12

Введите количество секунд для вычитания из первого времени: 3600

Новое время после вычитания 3600 секунд:
11:54:12

Первое время в секундах: 46452 секунд
Первое время в минутах (с округлением): 774 минут
```

Рисунок 5. Результат работы программы

Контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

Объявление класса в Python выполняется с помощью ключевого слова class, за которым следует имя класса и двоеточие. Внутри класса определяются атрибуты и методы. Например:

class MyClass:

```
    def __init__(self):
        self.attribute = "value"
```

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибуты класса — это переменные, объявленные на уровне класса, которые являются общими для всех экземпляров класса. Они определяются непосредственно в теле класса и их значение разделяется всеми объектами этого класса.

Атрибуты экземпляра — это переменные, связанные с конкретным экземпляром (объектом) класса. Они обычно определяются в методе __init__ с использованием ключевого слова self.

Пример:

```
class MyClass:
    class_attribute = "Атрибут класса"
    def __init__(self):
        self.instance_attribute = "Атрибут экземпляра"
```

3. Каково назначение методов класса?

Методы класса — это функции, определенные внутри класса, которые позволяют манипулировать данными, связанными с объектами этого класса, и выполнять действия, связанные с поведением объектов. Методы класса могут работать с атрибутами экземпляра, изменять их или выполнять другие операции.

4. Для чего предназначен метод __init__() класса?

Метод __init__() — это инициализатор, который автоматически вызывается при создании нового экземпляра класса. Он предназначен для инициализации атрибутов экземпляра и выполнения начальных настроек. Это своего рода "конструктор" в Python.

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

5. Каково назначение self ?

self — это ссылка на текущий экземпляр класса. Она используется для доступа к атрибутам и методам класса изнутри самого класса. В Python все методы класса должны иметь self в качестве первого параметра, чтобы иметь возможность работать с данными конкретного объекта.

```
class MyClass:
    def __init__(self, value):
        self.value = value
    def display_value(self):
        print(self.value)
```

6. Как добавить атрибуты в класс?

Атрибуты можно добавить в класс:

На уровне класса – просто объявив их в теле класса.

На уровне экземпляра – в методе __init__() или в любом другом методе с использованием self.

```
class MyClass:
    class_attribute = "Атрибут класса"
    def __init__(self):
        self.instance_attribute = "Атрибут экземпляра"
```

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

В Python управление доступом к методам и атрибутам осуществляется с помощью соглашений об именах:

Атрибуты и методы, начинающиеся с одного подчеркивания (_), считаются защищенными (protected) и предназначены для использования внутри класса и его подклассов.

Атрибуты и методы, начинающиеся с двух подчеркиваний (__), считаются частными (private) и подвергаются механизму "маскировки имен" (name mangling), что затрудняет доступ к ним за пределами класса.

Однако, все атрибуты и методы в Python остаются доступны и могут быть вызваны.

8. Каково назначение функции isinstance?

Функция isinstance() проверяет, является ли объект экземпляром указанного класса или его подклассов. Она возвращает True, если объект принадлежит классу, и False в противном случае.

```
obj = MyClass()
print(isinstance(obj, MyClass))  # True
```

Эта функция полезна для проверки типов данных и обеспечивает безопасность типов в программах.

Вывод: в ходе лабораторной работы приобретены навыки по работе с классами и объектами при написании программ помощью языка программирования Python версии 3.x.