Московский государственный технический университет им. Н.Э. Баумана

Отчет
Лабораторная работа 1

ИСПОЛНИТЕЛЬ:

Группа ИБМ3-34Б
Михайлова В.И.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

2025 г.

Цель работы

Изучение основных конструкций языка Python:

1. Работу с различными типами данных (действительные числа).
2. Использование условных операторов (if/elif/else) для принятия решений.
3. Реализацию циклов (while) для валидации ввода данных.
4. Работа с модулями (sys, math) и параметрами командной строки.
5. Разработка программы с использованием процедурной парадигмы.

Задание

Разработать консольное приложение на языке Python,
решающее **биквадратное уравнение** вида $Ax^4 + Bx^2 + C = 0.$

Алгоритм

1. **Ввод данных:** коэффициенты должны считываться из параметров командной строки. Если параметры не заданы или некорректны, осуществляется ввод с клавиатуры.
2. **Валидация:** при вводе с клавиатуры программа должна **повторно** запрашивать коэффициент, пока не будет введено
   корректное действительное число.
3. **Решение:** уравнение решается заменой, что приводит к квадратному уравнению
4. **Обработка случаев:**
   - Вычисляется дискриминант
   - Находятся действительные корни с учетом условия .
   - Предусматривается обработка вырожденных случаев.
5. **Вывод:** на экран выводятся все действительные корни уравнения.

3. Текст программы (Процедурная парадигма)

```
import sys
import math

def get_coefficient(name, default_value=None):
    """
    Запрашивает коэффициент у пользователя, пока не будет введено
    корректное действительное число, с учетом приоритета параметров
    командной строки.
    """
    # [Здесь вставьте полный код функции get_coefficient]
    # (Функция для валидации ввода)
```

```
    if default_value is not None:
        try:
            return float(default_value)
        except ValueError:
            print(f"Некорректное значение '{default_value}' для {name} из
командной строки. Запрашиваю ввод с клавиатуры...")
            pass

    while True:
        try:
            value = input(f"Введите коэффициент {name}: ")
            coefficient = float(value)
            return coefficient
        except ValueError:
            print(f"Ошибка: '{value}' не является действительным числом.
Повторите ввод.")

def solve_biquadratic(A, B, C):
    """
    Решает биквадратное уравнение Ax^4 + Bx^2 + C = 0.
    """
    print(f"\nРешаем уравнение: {A}x^4 + {B}x^2 + {C} = 0")

    # [Здесь вставьте полный код функции solve_biquadratic]
    # (Основная логика решения уравнения)
    if A == 0:
        if B == 0:
            if C == 0:
                return "Бесконечное множество корней (0 = 0)"
            else:
                return "Нет корней (константа C != 0)"
        else:
            # Уравнение Bx^2 + C = 0 (квадратное)
            print("Уравнение вырождается в квадратное: x^2 = -C/B")
            x2 = -C / B
            if x2 >= 0:
                sqrt_x2 = math.sqrt(x2)
                return [f"x1 = {sqrt_x2}", f"x2 = {-sqrt_x2}"]
            else:
                return "Нет действительных корней"

    # Решает квадратное уравнение относительно y = x^2: Ay^2 + By + C = 0
    D = B*B - 4*A*C
    print(f"Дискриминант D = B^2 - 4AC = {B}*{B} - 4*{A}*{C} = {D}")
```

```
    if D < 0:
        return "Дискриминант < 0. Нет действительных корней."

    sqrt_D = math.sqrt(D)
    y1 = (-B + sqrt_D) / (2 * A)
    y2 = (-B - sqrt_D) / (2 * A)

    print(f"Корни вспомогательного уравнения (y = x^2): y1 = {y1:.4f}, y2 = {y2:.4f}")

    real_roots = []

    # [Проверка корней y1 и y2 на положительность и вычисление x]
    if y1 >= 0:
        root_p = math.sqrt(y1)
        real_roots.append(f"x = {root_p:.4f}")
        if root_p != 0:
            real_roots.append(f"x = {-root_p:.4f}")

    if y2 >= 0 and abs(y2 - y1) > 1e-9: # Учет дублирования корней при D=0
        root_p = math.sqrt(y2)
        real_roots.append(f"x = {root_p:.4f}")
        if root_p != 0:
            real_roots.append(f"x = {-root_p:.4f}")

    unique_roots = sorted(list(set(real_roots)), key=lambda x: (float(x.split(' ')[2]), x)) # Для удобного вывода

    if not unique_roots:
        return "Нет положительных корней y. Нет действительных корней x."
    else:
        return "\nДействительные корни уравнения:\n" + "\n".join(unique_roots)

def main():
    """
    Главная функция, управляющая вводом и решением.
    """
    args = sys.argv[1:]

    A_val = args[0] if len(args) > 0 else None
    B_val = args[1] if len(args) > 1 else None
    C_val = args[2] if len(args) > 2 else None

    A = get_coefficient("A", A_val)
    B = get_coefficient("B", B_val)
```

```
    C = get_coefficient("C", C_val)

    result = solve_biquadratic(A, B, C)
    print("\n--- Результат ---")

    if isinstance(result, list):
        print("Нет положительных корней y. Нет действительных корней x.") # Если список пустой, то нет корней
    else:
        print(result)

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

Вывод кода:

```
/usr/local/bin/python3.13 /Users/mikhailovavi/PycharmProjects/PythonProject/1.1.py
Введите коэффициент A: 2
Введите коэффициент B: 3
Введите коэффициент C: 2

Решаем уравнение: 2.0x^4 + 3.0x^2 + 2.0 = 0
Дискриминант D = B^2 - 4AC = 3.0*3.0 - 4*2.0*2.0 = -7.0

--- Результат ---
Дискриминант < 0. Нет действительных корней.

Process finished with exit code 0
```

```
/usr/local/bin/python3.13 /Users/mikhailovavi/PycharmProjects/PythonProject/1.1.py
Введите коэффициент A: 1
Введите коэффициент B: 4
Введите коэффициент C: 2

Решаем уравнение: 1.0x^4 + 4.0x^2 + 2.0 = 0
Дискриминант D = B^2 - 4AC = 4.0*4.0 - 4*1.0*2.0 = 8.0
Корни вспомогательного уравнения (y = x^2): y1 = -0.5858, y2 = -3.4142

--- Результат ---
Нет положительных корней y. Нет действительных корней x.

Process finished with exit code 0
```

```
/usr/local/bin/python3.13 /Users/mikhailovavi/PycharmProjects/PythonProject/1.1.py
Введите коэффициент A: 0.1
Введите коэффициент B: -1
Введите коэффициент C: 0

Решаем уравнение: 0.1x^4 + -1.0x^2 + 0.0 = 0
Дискриминант D = B^2 - 4AC = -1.0*-1.0 - 4*0.1*0.0 = 1.0
Корни вспомогательного уравнения (y = x^2): y1 = 10.0000, y2 = 0.0000

--- Результат ---

Действительные корни уравнения:
x = -3.1623
x = 0.0000
x = 3.1623

Process finished with exit code 0
```

```
/usr/local/bin/python3.13 /Users/mikhailovavi/PycharmProjects/PythonProject/1.1.py
Введите коэффициент A: one
Ошибка: 'one ' не является действительным числом. Повторите ввод.
Введите коэффициент A:
```

Дополнительное задание 1: Объектно-Ориентированная Парадигма

3. Текст программы (ООП-парадигма)

В этой версии логика решения уравнения инкапсулирована
в класс BiquadraticEquation, что соответствует принципам ООП (инкапсуляция, методы). Функции ввода (get_coefficient_oop) и управления (main_oop) используются для взаимодействия с классом.

Код:

```
import sys

import math

class BiquadraticEquation:

    """Класс для решения биквадратного уравнения Ax^4 + Bx^2 + C = 0."""

    def __init__(self, A, B, C):

        """Конструктор класса, инициализирует коэффициенты."""

        self.A = A
```

```
        self.B = B
        self.C = C
    def solve(self):
        """Основной метод для решения уравнения."""
        if self.A == 0:
            return self._handle_degenerate_case()
        # Решаем квадратное уравнение относительно y = x^2
        D = self.B * self.B - 4 * self.A * self.C
        print(f"Дискриминант D = {D:.4f}")
        if D < 0:
            return "Дискриминант < 0. Нет действительных корней."
        sqrt_D = math.sqrt(D)
        y1 = (-self.B + sqrt_D) / (2 * self.A)
        y2 = (-self.B - sqrt_D) / (2 * self.A)
        print(f"Корни вспомогательного уравнения (y = x^2): y1 = {y1:.4f}, y2 =
{y2:.4f}")
        return self._find_real_roots(y1, y2)
    def _handle_degenerate_case(self):
        """Приватный метод: Обрабатывает случаи, когда A=0."""
        if self.B == 0:
            return "Бесконечное множество корней" if self.C == 0 else "Нет корней"
        #Уравнение Bx^2 + C = 0
        x2 = -self.C / self.B
        if x2 >= 0:
            sqrt_x2 = math.sqrt(x2)
```

```
            return f"Уравнение вырождается в квадратное. Корни: x1 =
{sqrt_x2:.4f}, x2 = {-sqrt_x2:.4f}"

        else:

            return "Уравнение вырождается в квадратное. Нет действительных
корней."

    def _find_real_roots(self, y1, y2):

        """Приватный метод: Находит действительные корни x, зная y1 и y2."""

        real_roots = []

        roots_y = set() # Используем множество для уникальности корней y

        if y1 >= 0: roots_y.add(y1)

        if y2 >= 0: roots_y.add(y2)

        for y in roots_y:

            if y == 0:

                real_roots.append("x = 0.0000")

            elif y > 1e-9: # Игнорируем малые отрицательные числа из-за
погрешностей

                root_p = math.sqrt(y)

                real_roots.append(f"x = {root_p:.4f}")

                real_roots.append(f"x = {-root_p:.4f}")

        unique_roots = sorted(list(set(real_roots)), key=lambda x: (float(x.split('
')[2]), x))

        if not unique_roots:

            return "Нет положительных корней y. Нет действительных корней x."

        else:

            return "\nДействительные корни уравнения (ООП):\n" +
"\n".join(unique_roots)
```

```
# Функции для ввода и управления (вне класса)

def get_coefficient_oop(name, default_value=None):

    """

    Валидирует ввод коэффициентов.

    """

    if default_value is not None:

        try:

            return float(default_value)

        except ValueError:

            print(f"Некорректное значение '{default_value}' для {name}.
Запрашиваю ввод с клавиатуры...")

    while True:

        try:

            value = input(f"Введите коэффициент {name} (ООП): ")

            coefficient = float(value)

            return coefficient

        except ValueError:

            print(f"Ошибка: '{value}' не является действительным числом.
Повторите ввод.")

def main_oop():

    """Точка входа для ООП-реализации."""

    args = sys.argv[1:]

    #Считывание коэффициентов с учетом приоритета и валидации

    A_val = args[0] if len(args) > 0 else None

    B_val = args[1] if len(args) > 1 else None
```

```
    C_val = args[2] if len(args) > 2 else None
    A = get_coefficient_oop("A", A_val)
    B = get_coefficient_oop("B", B_val)
    C = get_coefficient_oop("C", C_val)
    #Создание объекта и вызов его метода solve()
    equation = BiquadraticEquation(A, B, C)
    print(f"\nРешаем уравнение (ООП): {A}x^4 + {B}x^2 + {C} = 0")
    result = equation.solve()
    print("\n--- Результат (ООП) ---")
    print(result)
if __name__ == "__main__":
    main_oop()
```

Результат вывода кода:

```
/usr/local/bin/python3.13 /Users/mikhailovavi/PycharmProjects/PythonProject/1.1.py
Введите коэффициент A (ООП): 1
Введите коэффициент B (ООП): -10
Введите коэффициент C (ООП): 9

Решаем уравнение (ООП): 1.0x^4 + -10.0x^2 + 9.0 = 0
Дискриминант D = 64.0000
Корни вспомогательного уравнения (y = x^2): y1 = 9.0000, y2 = 1.0000

--- Результат (ООП) ---

Действительные корни уравнения (ООП):
x = -3.0000
x = -1.0000
x = 1.0000
x = 3.0000

Process finished with exit code 0
```

Вывод:

В ходе выполнения ЛР1 была достигнута цель — изучение и практическое применение ключевых конструкций Python на примере решения комплексной математической задачи: нахождения действительных корней биквадратного уравнения .

1. Освоение базового функционала:

- Была реализована основная **математическая логика** решения уравнения, включающая вычисление **дискриминанта** и последующее нахождение действительных корней с учетом условия .
- На практике отработаны **условные операторы** (if/elif/else) для обработки различных сценариев (четыре корня, два корня, один корень, отсутствие корней, вырожденные случаи).

2. Реализация системных требований:

- Обеспечена **гибкость ввода данных** путём использования модуля sys для считывания коэффициентов из **параметров командной строки**.
- Разработан надежный механизм **валидации ввода** с клавиатуры (через цикл while и обработку исключений try-except), что гарантирует корректность обрабатываемых данных (действительных чисел) и повышает устойчивость программы к ошибкам пользователя.

3. Сравнение парадигм программирования (Дополнительное задание):

Критерий	Процедурная парадигма	Объектно-Ориентированная парадигма (ООП)
Структура	Логика разделена на **независимые функции** (get_coefficient, solve_biquadratic).	Логика инкапсулирована в **класс BiquadraticEquation** с его **методами** (solve).
Модульность	Ниже, данные (коэффициенты) передаются между функциями.	**Выше**, данные и методы, работающие с ними, объединены в один объект.
Результат	Программа наглядно продемонстрировала, что для решения даже такой относительно простой задачи, как решение уравнений, **ООП-подход**обеспечивает лучшую **организацию кода** и его потенциал для расширения.	

В итоге, обе разработанные программы (процедурная и ООП) полностью соответствуют требованиям задания. В процессе работы были успешно закреплены навыки работы с **типами данных**, **управляющими конструкциями**, **модулями** и принципами **надежной обработки пользовательского ввода** в Python.