Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №1
«Основные конструкции языка Python»

Выполнил:
студент группы ИУ5-32Б
Мажитов В.

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Описание задания

Разработать программу для решения биквадратного уравнения.

1. Программа должна быть разработана в виде консольного приложения на языке Python.
2. Программа осуществляет ввод с клавиатуры коэффициентов A, B, C, вычисляет дискриминант и ДЕЙСТВИТЕЛЬНЫЕ корни уравнения (в зависимости от дискриминанта).
3. Коэффициенты A, B, C могут быть заданы в виде параметров командной строки. Если они не заданы, то вводятся с клавиатуры в соответствии с пунктом 2.
4. Если коэффициент A, B, C введен или задан в командной строке некорректно, то необходимо проигнорировать некорректное значение и вводить коэффициент повторно пока коэффициент не будет введен корректно. Корректно заданный коэффициент - это коэффициент, значение которого может быть без ошибок преобразовано в действительное число.
5. Дополнительное задание 1. Разработайте две программы на языке Python - одну с применением процедурной парадигмы, а другую с применением объектно-ориентированной парадигмы.
6. Дополнительное задание 2. Разработайте две программы - одну на языке Python, а другую на любом другом языке программирования (кроме C++).

Текст программы

Python, объектно-ориентированная парадигма

Файл main.py

```python
from entity.exceptions import WrongInput, NoSolutions
from service.input import InputService
from service.solver import SolverService

if __name__ == "__main__":
    solver_service = SolverService()
    input_service = InputService()
    try:
        coefficients = input_service.get_coefficients()
        print(solver_service.solve(coefficients))
    except WrongInput:
        print("Неверный ввод")
    except NoSolutions:
        print("Нет решений")
```

Файл service/input.py

```python
import sys
from typing import Optional, Callable

from entity.coefficients import Coefficients
```

```python
from entity.exceptions import WrongInput


class InputService:
    @staticmethod
    def parse_float_from_string(inp: str, validator: Optional[Callable[[float],
bool]] = None) -> float:
        digit = float(inp)
        if validator and not validator(digit):
            raise ValueError()

        return digit

    def get_float_from_terminal(self, prompt: str, validator:
Optional[Callable[[float], bool]] = None) -> float:
        while True:
            try:
                print(prompt)
                inp = input()

                return self.parse_float_from_string(inp, validator)
            except ValueError:
                print("Неверный ввод")

    def get_coefficients(self) -> Coefficients:
        if len(sys.argv) < 3:
            a = self.get_float_from_terminal("Введите коэффициент A", lambda x: x
!= 0)
            b = self.get_float_from_terminal("Введите коэффициент B")
            c = self.get_float_from_terminal("Введите коэффициент C")
        else:
            try:
                args = sys.argv[1:]
                a = self.parse_float_from_string(args[0], lambda x: x != 0)
                b = self.parse_float_from_string(args[1])
                c = self.parse_float_from_string(args[2])
            except ValueError:
                raise WrongInput()

        return Coefficients(
            A=a,
            B=b,
            C=c,
        )
```

Файл service/solver.py

```python
from math import sqrt
from typing import Set

from entity.coefficients import Coefficients
```

```python
from entity.exceptions import NoSolutions


class SolverService:
    @staticmethod
    def solve(coefficients: Coefficients) -> Set[float]:
        d = coefficients.B ** 2 - 4 * coefficients.A * coefficients.C
        if d < 0:
            raise NoSolutions()
        roots: Set[float] = set()

        possible_roots = [
            (-coefficients.B - sqrt(d)) / (2 * coefficients.A),
            (-coefficients.B + sqrt(d)) / (2 * coefficients.A),
        ]

        for root in possible_roots:
            if root >= 0:
                roots.add(sqrt(root))
                roots.add(-sqrt(root))

        return roots
```

Файл entity/coefficients.py

```python
from dataclasses import dataclass


@dataclass
class Coefficients:
    A: float
    B: float
    C: float
```

Файл entity/exceptions.py

```python
class WrongInput(Exception):
    pass


class NoSolutions(Exception):
    pass
```

Python, процедурная парадигма

```python
import sys
from dataclasses import dataclass
from math import sqrt
from typing import Callable, Optional, Set
```

```python
class WrongInput(Exception):
    pass


class NoSolutions(Exception):
    pass


@dataclass
class Coefficients:
    A: float
    B: float
    C: float


def parse_float_from_string(inp: str, validator: Optional[Callable[[float], bool]]
= None) -> float:
    digit = float(inp)
    if validator and not validator(digit):
        raise ValueError()

    return digit


def get_float_from_terminal(prompt: str, validator: Optional[Callable[[float],
bool]] = None) -> float:
    while True:
        try:
            print(prompt)
            inp = input()

            return parse_float_from_string(inp, validator)
        except ValueError:
            print("Неверный ввод")


def get_coefficients() -> Coefficients:
    if len(sys.argv) < 3:
        a = get_float_from_terminal("Введите коэффициент A", lambda x: x != 0)
        b = get_float_from_terminal("Введите коэффициент B")
        c = get_float_from_terminal("Введите коэффициент C")
    else:
        try:
            args = sys.argv[1:]
            a = parse_float_from_string(args[0], lambda x: x != 0)
            b = parse_float_from_string(args[1])
            c = parse_float_from_string(args[2])
        except ValueError:
            print("Неверные аргументы командной строки")
            exit()

    return Coefficients(
```

```
        A=a,
        B=b,
        C=c,
    )


def solve(coefficients: Coefficients) -> Set[float]:
    d = coefficients.B ** 2 - 4 * coefficients.A * coefficients.C
    if d < 0:
        raise NoSolutions()
    roots: Set[float] = set()

    possible_roots = [
        (-coefficients.B - sqrt(d)) / (2 * coefficients.A),
        (-coefficients.B + sqrt(d)) / (2 * coefficients.A),
    ]

    for root in possible_roots:
        if root >= 0:
            roots.add(sqrt(root))
            roots.add(-sqrt(root))

    return roots


if __name__ == "__main__":
    coefficients = get_coefficients()
    try:
        print(solve(coefficients))
    except NoSolutions:
        print("Нет решений")
```

Golang, объектно-ориентированная парадигма

Файл main.go

```
package main

import (
    "fmt"
    "errors"
    "lab1/service"
    "lab1/domain"
)

func main() {
    inputService := service.NewInputService()

    var coefficients domain.Coefficients
    var err error
```

```
    for {
        coefficients, err = inputService.Get()
        if err != nil {
            fmt.Println(err)
            if errors.Is(err, domain.ErrWrongConsoleInput) {
                break
            }
        } else {
            break
        }
    }


    solverService := service.NewSolverService()
    roots, err := solverService.Solve(coefficients)
    if err != nil {
        fmt.Println(err)
        return
    }

    fmt.Printf("Корни: %+v\n", roots)
}
```

Файл domain/coefficients.go

```
package domain

type Coefficients struct {
    A float64
    B float64
    C float64
}
```

Файл domain/errors.go

```
package domain

import "errors"

var (
    ErrWrongInput  = errors.New("Неверный ввод")
    ErrWrongConsoleInput  = errors.New("Неверный ввод")
    ErrNoSolutions = errors.New("Нет решений")
)
```

Файл domain/input.go

```
package domain

type Input struct {
    A string
```

```
    B string
    C string
}
```

Файл service/input.go

```
package service

import (
    "fmt"
    "lab1/domain"
    "os"
    "strconv"
)

type InputService struct{}

func NewInputService() *InputService {
    return &InputService{}
}

func (_ *InputService) parseFloatFromString(input string) (float64, error) {
    value, err := strconv.ParseFloat(input, 64)

    if err != nil {
        return 0, domain.ErrWrongInput
    }

    return value, nil
}

func (s *InputService) getCoefficients(input domain.Input) (output
domain.Coefficients, err error) {
    output.A, err = s.parseFloatFromString(input.A)
    if err != nil {
        return
    }

    if output.A == 0 {
        err = domain.ErrWrongInput
        return
    }

    output.B, err = s.parseFloatFromString(input.B)
    if err != nil {
        return
    }

    output.C, err = s.parseFloatFromString(input.C)
    if err != nil {
        return
```

```
    }

    return
}

func getMethodBasedErr(isConsole bool) error {
    if (isConsole) {
        return domain.ErrWrongConsoleInput
    } else {
        return domain.ErrWrongInput
    }
}

func (s *InputService) Get() (domain.Coefficients, error) {
    var input domain.Input
    isConsole := len(os.Args) > 3

    if isConsole {
        input.A = os.Args[1]
        input.B = os.Args[2]
        input.C = os.Args[3]
    } else {
        fmt.Print("Введите коэффициенты через пробел: ")
        _, err := fmt.Scanf("%s %s %s", &input.A, &input.B, &input.C)
        if err != nil {
            return domain.Coefficients{}, getMethodBasedErr(isConsole)
        }
    }

    coefficients, err := s.getCoefficients(input)
    if err != nil {
        return domain.Coefficients{}, getMethodBasedErr(isConsole)
    }
    return coefficients, nil
}
```

Файл service/solver.go

```
package service

import (
    "lab1/domain"
    "math"
)

type SolverService struct{}

func NewSolverService() *SolverService {
    return &SolverService{}
}
```

```go
func (s *SolverService) Solve(coefficients domain.Coefficients) ([]float64, error)
{
    rootSet := map[float64]struct{}{}

    d := math.Pow(coefficients.B, 2) - 4*coefficients.A*coefficients.C
    if d < 0 {
        return []float64{}, domain.ErrNoSolutions
    }

    possibleRoots := []float64{
        (-coefficients.B - math.Sqrt(d)) / (2 * coefficients.A),
        (-coefficients.B + math.Sqrt(d)) / (2 * coefficients.A),
    }

    for _, root := range possibleRoots {
        if root < 0 {
            continue
        }

        rootSet[-math.Sqrt(root)] = struct{}{}
        rootSet[math.Sqrt(root)] = struct{}{}
    }

    roots := make([]float64, 0, len(rootSet))
    for root := range rootSet {
        roots = append(roots, root)
    }

    return roots, nil
}
```

Примеры выполнения программы

1. **Python, объектно-ориентированная парадигма**

```
❯ python3 main.py 0 0 0
Неверный ввод
❯ python3 main.py
Введите коэффициент A
0
Неверный ввод
Введите коэффициент A
1
Введите коэффициент B
-5
Введите коэффициент C
4
{1.0, 2.0, -1.0, -2.0}
❯ python3 main.py 1 -5 0
{0.0, 2.23606797749979, -2.23606797749979}
```

2. **Python, процедурная парадигма**

```
❯ python3 functional.py 0 0 0
Неверные аргументы командной строки
❯ python3 functional.py
Введите коэффициент A
0
Неверный ввод
Введите коэффициент A
1
Введите коэффициент B
-5
Введите коэффициент C
4
{1.0, 2.0, -1.0, -2.0}
❯ python3 functional.py 1 -5 0
{0.0, 2.23606797749979, -2.23606797749979}
```

3. **Golang, объектно-ориентированная парадигма**

```
❯ go run main.go
Введите коэффициенты через пробел: 0 0 0
Неверный ввод
Введите коэффициенты через пробел: 1 -5 4
Корни: [-1 1 -2 2]
❯ go run main.go 1 -5 4
Корни: [-1 1 -2 2]
❯ go run main.go 1 -5 0
Корни: [-2.23606797749979 2.23606797749979 0]
```