МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ПО-12

Выполнил
А. В. Ишимов,
студент группы ПО-12

Проверил
А. А. Крощенко,
ст. преп. кафедры ИИТ,
«19» февраль 2026г.

Брест 2026

Вариант 7

Цель работы: закрепить навыки объектно-ориентированного программирования на языке Python.

Задание 1. Множество символов ограниченной мощности – предусмотреть возможность объединения двух множеств, вывода на консоль элементов множества, а также метод, определяющий, принадлежит ли указанное значение множеству. Класс должен содержать методы, позволяющие добавлять и удалять элемент в/из множества. Конструктор должен позволять создавать объекты с начальной инициализацией. Мощность множества задается при создании объекта. Реализацию множества осуществить на базе списка. Переопределить метод __eq__, выполняющий сравнение объектов данного типа.

Выполнение:

Код программы

```
class LimitedSet:
    def __init__(self, capacity, initial=None):
        self.capacity = capacity
        self._data = []

        if initial:
            for item in initial:
                self.add(item)

    def items(self):
        return list(self._data)

    def add(self, item):
        if len(self._data) >= self.capacity:
            raise ValueError("Множество заполнено")
        if item not in self._data:
            self._data.append(item)

    def remove(self, item):
        if item in self._data:
            self._data.remove(item)

    def contains(self, item):
        return item in self._data
```

```
        def union(self, other):
            if not isinstance(other, LimitedSet):
                raise TypeError("Можно объединять только
с LimitedSet")

            new_capacity = self.capacity + other.capacity
            result = LimitedSet(new_capacity)

            for item in self.items():
                result.add(item)

            for item in other.items():
                result.add(item)

            return result

        def __str__(self):
            return "{" + ", ".join(self._data) + "}"

        def __eq__(self, other):
            if not isinstance(other, LimitedSet):
                return False
            return          sorted(self._data)          ==
sorted(other._data)


    s1 = LimitedSet(5, ["a", "b", "c"])
    s2 = LimitedSet(5, ["c", "d", "e"])

    print("Set 1:", s1)
    print("Set 2:", s2)
    print("Union:", s1.union(s2))
    print("Contains 'b':", s1.contains("b"))
```

Спецификация ввода

>python lab2_1.py <мощность> <элемент1> <элемент2> <элемент3> ...

Пример

>python lab2_1.py 5 a b c

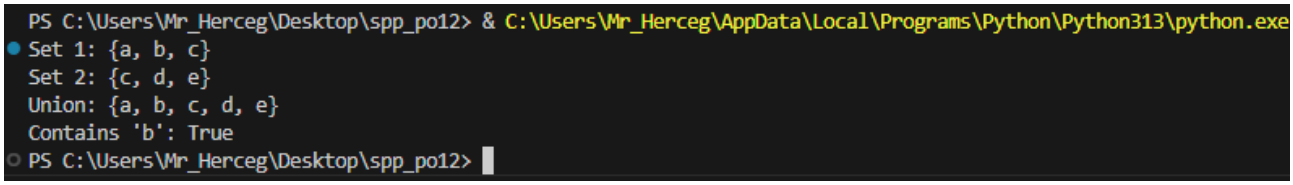
Спецификация вывода

Set = <строковое представление множества>
Contains '<символ>' = <True/False>
Union = <результат объединения>

Пример

Set = {b, c, d}
Contains 'b' = True
Union = {b, c, d, x, y}

Рисунки с результатами работы программы



Задание 2. Построить модель программной системы с применением отношений (обобщения, агрегации, ассоциации, реализации) между классами. Задать атрибуты и методы классов. Реализовать (если необходимо) дополнительные классы. Продемонстрировать работу разработанной системы. Система Автобаза. Диспетчер распределяет заявки на Рейсы между Водителями и назначает для этого Автомобиль. Водитель может сделать заявку на ремонт. Диспетчер может отстранить Водителя от работы. Водитель делает отметку о выполнении Рейса и состоянии Автомобиля.

Выполнение:

Код программы

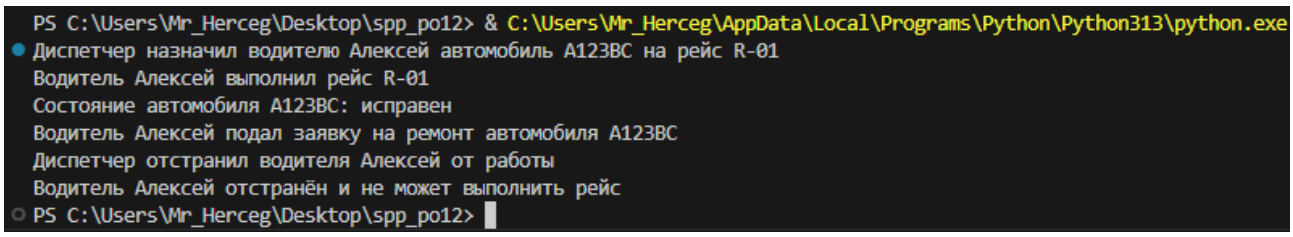
```
from abc import ABC, abstractmethod


# Обобщение
class Person(ABC):
    def __init__(self, name):
        self.name = name


# Интерфейс
class RepairRequester(ABC):
    @abstractmethod
    def request_repair(self, car):
        pass
```

```
    # Класс водителя
    class Driver(Person, RepairRequester):
        def __init__(self, name):
            super().__init__(name)
            self.suspended = False

        def request_repair(self, car):
            print(f"Водитель {self.name} подал заявку на
ремонт автомобиля {car.number}")
            car.needs_repair = True

        def complete_route(self, route, car):
            if self.suspended:
                print(f"Водитель {self.name} отстранён и
не может выполнить рейс")
                return

            print(f"Водитель {self.name} выполнил рейс
{route.route_id}")
            route.completed = True
            print(f"Состояние автомобиля {car.number}:
{'нужен ремонт' if car.needs_repair else 'исправен'}")


    # Класс автомобиля
    class Car:
        def __init__(self, number):
            self.number = number
            self.needs_repair = False

        def __str__(self):
            return f"Автомобиль {self.number}"


    # Класс рейса
    class Route:
        def __init__(self, route_id):
            self.route_id = route_id
            self.completed = False

        def __str__(self):
            return f"Рейс {self.route_id}"
```

```
    # Класс диспетчера
    class Dispatcher(Person):
        def __init__(self, name):
            super().__init__(name)
            self.drivers = []
            self.cars = []

        # Агрегация
        def add_driver(self, driver):
            self.drivers.append(driver)

        def add_car(self, car):
            self.cars.append(car)

        # Ассоциация
        def assign_route(self, driver, car, route):
            if driver.suspended:
                print(f"Диспетчер: водитель {driver.name}
отстранён, рейс не назначен")
                return

            print(f"Диспетчер      назначил      водителю
{driver.name}    автомобиль    {car.number}    на    рейс
{route.route_id}")

        def suspend_driver(self, driver):
            driver.suspended = True
            print(f"Диспетчер     отстранил     водителя
{driver.name} от работы")


    dispatcher = Dispatcher("Иван Петров")
    driver1 = Driver("Алексей")
    driver2 = Driver("Сергей")

    car1 = Car("A123BC")
    car2 = Car("B777BB")

    route1 = Route("R-01")

    dispatcher.add_driver(driver1)
    dispatcher.add_driver(driver2)
    dispatcher.add_car(car1)
    dispatcher.add_car(car2)
```

```
dispatcher.assign_route(driver1, car1, route1)

driver1.complete_route(route1, car1)

driver1.request_repair(car1)

dispatcher.suspend_driver(driver1)

driver1.complete_route(route1, car1)
```

Рисунки с результатами работы программы



Вывод: закрепил навыки объектно-ориентированного программирования на языке Python.