

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

**Отчет по лабораторной работе №4.6
Классы данных в Python
по дисциплине «Объектно-ориентированное программирование»**

Выполнил студент группы ИВТ-б-о-21-1

Лысенко И.А. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2024

Цель работы: приобретение навыков по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

1. Создал репозиторий на GitHub:

https://github.com/IsSveshuD/OOP_Lab_4.6.git .

2. Выполнил индивидуальное задание.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from __future__ import annotations
from dataclasses import dataclass
import xml.etree.ElementTree as ET

from abc import ABC, abstractmethod

@dataclass
class Triad(ABC):
    a: int
    b: int
    c: int

    @abstractmethod
    def increase(self) -> None:
        pass

    @abstractmethod
    def display(self) -> None:
        pass

    @classmethod
    def from_xml(cls, file: str) -> Triad:
        with open(file, 'r') as file:
            xml_string = file.read()
            root = ET.fromstring(xml_string)
            return cls(int(root.find('a').text), int(root.find('b').text),
int(root.find('c').text))

    def to_xml(self, file: str) -> None:
        root = ET.Element('Triad')
        ET.SubElement(root, 'a').text = str(self.a)
        ET.SubElement(root, 'b').text = str(self.b)
        ET.SubElement(root, 'c').text = str(self.c)
        with open(file, 'w') as file:
            file.write(ET.tostring(root).decode('utf-8'))

@dataclass
class Date(Triad):
    day: int
    month: int
    year: int

    def __init__(self, day: int, month: int, year: int):
        super().__init__(day, month, year)

    def increase(self) -> None:
```

```

        self.a += 1
        self.b += 1
        self.c += 1

    def display(self) -> None:
        print(f"Дата: {self.a}/{self.b}/{self.c}")

    @classmethod
    def from_xml(cls, file: str) -> Date:
        with open(file, 'r') as file:
            string = file.read()
            root = ET.fromstring(string)
            return cls(int(root.find('day').text),
int(root.find('month').text), int(root.find('year').text))

        def to_xml(self, file: str) -> None:
            root = ET.Element('Date')
            ET.SubElement(root, 'day').text = str(self.a)
            ET.SubElement(root, 'month').text = str(self.b)
            ET.SubElement(root, 'year').text = str(self.c)
            with open(file, 'w') as file:
                file.write(ET.tostring(root).decode('utf-8'))

@dataclass
class Time(Triad):
    hour: int
    minute: int
    second: int

    def __init__(self, hour: int, minute: int, second: int):
        super().__init__(hour, minute, second)

    def increase(self) -> None:
        self.a += 1
        self.b += 1
        self.c += 1

    def display(self) -> None:
        print(f"Время: {self.a}:{self.b}:{self.c}")

    @classmethod
    def from_xml(cls, file: str) -> Time:
        with open(file, 'r') as file:
            string = file.read()
            root = ET.fromstring(string)
            return cls(int(root.find('hour').text),
int(root.find('minute').text), int(root.find('second').text))

        def to_xml(self, file: str) -> None:
            root = ET.Element('Time')
            ET.SubElement(root, 'hour').text = str(self.a)
            ET.SubElement(root, 'minute').text = str(self.b)
            ET.SubElement(root, 'second').text = str(self.c)
            with open(file, 'w') as file:
                file.write(ET.tostring(root).decode('utf-8'))

if __name__ == '__main__':
    date: Date = Date(17, 3, 2001)
    time: Time = Time(12, 30, 45)
    date.to_xml('date.xml')
    time.to_xml('time.xml')

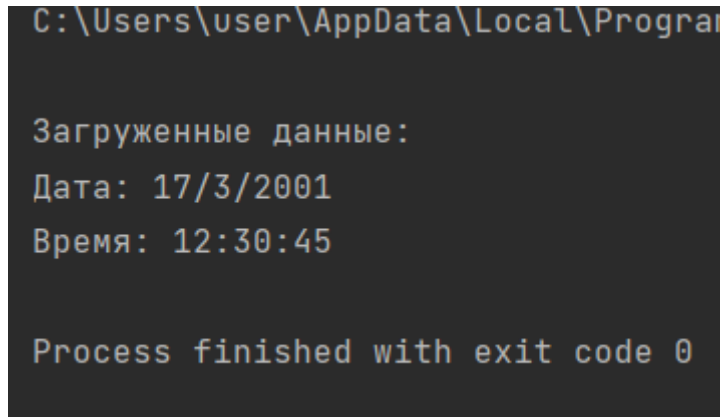
```

```
load_date = Date.from_xml('date.xml')
load_time = Time.from_xml('time.xml')

print("\nЗагруженные данные:")
load_date.display()
load_time.display()
```

Рисунок 2 – Индивидуальное задание 1

3. Получил следующий результат работы индивидуального задания.



```
C:\Users\user\AppData\Local\Program

Загруженные данные:
Дата: 17/3/2001
Время: 12:30:45

Process finished with exit code 0
```

Рисунок 3 – Результат работы задания 1

Ответы на вопросы:

1. Как создать класс данных в языке Python?

В Python создание класса данных осуществляется с использованием ключевого слова `class`. Вот пример простого класса данных:

```
class Person:
    def init (self, name, age):
        self.name = name
        self.age = age
# Создание экземпляра класса
person1 = Person("Иван", 25)
# Доступ к атрибутам экземпляра класса
print(person1.name) # Выведет: Иван
print(person1.age) # Выведет: 25
```

В этом примере мы создаем класс `Person`, который имеет атрибуты `name` и `age`. Метод `init` является конструктором класса и используется для инициализации атрибутов при создании экземпляра класса. При создании экземпляра класса `Person` мы передаем значения для атрибутов `name` и `age`.

Доступ к атрибутам экземпляра класса осуществляется с использованием точки (например, `person1.name`).

Это только простейший пример класса данных. В Python классы могут содержать методы (функции, связанные с классом), наследование, статические методы, свойства и многое другое.

2. Какие методы по умолчанию реализует класс данных?

В Python класс данных может реализовывать несколько встроенных методов по умолчанию, которые позволяют определить специальное поведение объекта. Некоторые из этих методов включают:

1) `init (self, ...)`: Конструктор класса, который вызывается при создании нового экземпляра класса.

2) `str (self)`: Метод, который возвращает строковое представление объекта. Он вызывается, когда объект передается функции `str()` или когда объект используется в строковом контексте.

3) `repr (self)`: Метод, который возвращает представление объекта, которое может быть использовано для его воссоздания. Он вызывается, когда объект передается функции `repr()` или когда объект используется в интерактивной оболочке Python.

4) `eq (self, other)`: Метод для сравнения объектов на равенство (используется оператор `==`).

5) `lt (self, other)`, `le (self, other)`, `gt (self, other)`, `ge (self, other)`: Методы для сравнения объектов (используются операторы `<`, `<=`, `>`, `>=`).

6) `hash (self)`: Метод для вычисления хэш-значения объекта, используемого в словарях и множествах.

7) `getattr (self, name)`, `setattr (self, name, value)`: Методы для перехвата доступа к атрибутам объекта.

8) `del (self)`: Метод, который вызывается при удалении объекта.

3. Как создать неизменяемый класс данных?

В Python неизменяемый класс данных можно создать, используя неизменяемые типы данных в качестве атрибутов класса, и предоставляя только методы для чтения значений атрибутов, но не для их изменения. Вот пример создания неизменяемого класса данных:

```
class ImmutableData:
    def init (self, value1, value2):
        self._value1 = value1 # Префикс "_" обозначает "приватный"
        self._value2 = value2
    def get_value1(self):
        return self._value1
    def get_value2(self):
        return self._value2
```

В этом примере атрибуты value1 и value2 являются приватными (по соглашению обозначены префиксом _), и доступ к ним осуществляется только через методы get_value1 и get_value2. Таким образом, значения атрибутов не могут быть изменены напрямую извне.

Пример использования:

```
data = ImmutableData(10, 20)
print(data.get_value1()) # Выведет: 10
print(data.get_value2()) # Выведет: 20
# Попытка изменить значение атрибута вызовет ошибку
data._value1 = 100 # AttributeError: can't set attribute
```

Этот подход позволяет создать неизменяемый класс данных, в котором значения атрибутов не могут быть изменены после создания экземпляра класса.

Вывод: в результате выполнения лабораторной работы были приобретены навыки по работе с классами данных при написании программ с помощью языка программирования Python версии.