

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

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе

Дисциплина: «Объектно – ориентированное программирование»

Выполнил студент группы

ИВТ-б-о-21-1

Богдашов А.В. « » _____ 20__ г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

Проверил доцент

Кафедры инфокоммуникаций, старший
преподаватель

Воронкин Р.А.

(подпись)

Классы данных в Python

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

Ход работа:

Индивидуальное задание.

Выполнить индивидуальное задание лабораторной работы 4.5, используя классы данных, а также загрузку и сохранение данных в формат XML.

Код программы:

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

from dataclasses import dataclass
import xml.etree.ElementTree as ET

@dataclass
class Money:
    rubles: int = 0
    kopecks: int = 0

    def read(self) -> None:
        self.rubles, self.kopecks = map(int, input("Введите количество рублей и копеек через пробел: ").split())

    def display(self) -> None:
        print(f"{self.rubles} руб. {self.kopecks:02d} коп.")

    def add(self, other: 'Money') -> 'Money':
        total_kopecks = self.rubles * 100 + self.kopecks + other.rubles * 100 + other.kopecks
        return Money(*divmod(total_kopecks, 100))

    def subtract(self, other: 'Money') -> 'Money':
        total_kopecks = self.rubles * 100 + self.kopecks - (other.rubles * 100 + other.kopecks)
        return Money(*divmod(total_kopecks, 100))

    def divide_sum(self, num: float) -> 'Money':
        total_kopecks = (self.rubles * 100 + self.kopecks) / num
        return Money(*divmod(int(total_kopecks), 100))

    def multiply_by_number(self, num: float) -> 'Money':
        total_kopecks = (self.rubles * 100 + self.kopecks) * num
        return Money(*divmod(int(total_kopecks), 100))

    def compare(self, other: 'Money') -> bool:
        return self.rubles == other.rubles and self.kopecks == other.kopecks

    def is_less_than(self, other: 'Money') -> bool:
        return (self.rubles * 100 + self.kopecks) < (other.rubles * 100 + other.kopecks)

    def to_xml_element(self) -> ET.Element:
        money_element = ET.Element('Money')
        ET.SubElement(money_element, 'Rubles').text = str(self.rubles)
        ET.SubElement(money_element, 'Kopecks').text = str(self.kopecks)
```

```

        return money_element

    @classmethod
    def from_xml_element(cls, element: ET.Element) -> 'Money':
        rubles = int(element.find('Rubles').text)
        kopecks = int(element.find('Kopecks').text)
        return cls(rubles, kopecks)

if __name__ == '__main__':
    money1 = Money()
    money1.read()
    money1.display()

    money2 = Money()
    money2.read()
    money2.display()

    sum_result = money1.add(money2)
    print("Сумма:")
    sum_result.display()

    diff_result = money1.subtract(money2)
    print("Разность:")
    diff_result.display()

    divide_sum_num = float(input("Введите число для деления суммы: "))
    div_sum_result = money1.divide_sum(divide_sum_num)
    print("Деление суммы на число:")
    div_sum_result.display()

    multiply_by_num = float(input("Введите число для умножения: "))
    mul_result = money1.multiply_by_number(multiply_by_num)
    print("Умножение на число:")
    mul_result.display()

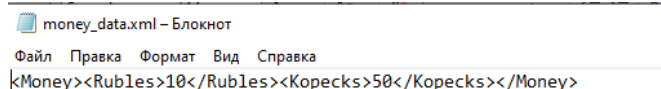
    comparison_result = money1.compare(money2)
    print(f"Сравнение: {comparison_result}")

    comparison_result_lt = money1.is_less_than(money2)
    print(f"Сравнение меньше: {comparison_result_lt}")

    # Сохранение в XML
    money1_element = money1.to_xml_element()
    tree = ET.ElementTree(money1_element)
    tree.write('money_data.xml')

    # Загрузка из XML
    loaded_tree = ET.parse('money_data.xml')
    loaded_money_element = loaded_tree.getroot()
    loaded_money = Money.from_xml_element(loaded_money_element)
    print("Данные, загруженные из XML:")
    loaded_money.display()

```



```

money_data.xml - Блокнот
Файл  Правка  Формат  Вид  Справка
<Money><Rubles>10</Rubles><Kopecks>50</Kopecks></Money>

```

Рисунок 1. Результат выполнения

Контрольные 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.

Это только небольшой набор методов по умолчанию, которые могут быть реализованы в классе данных. В Python есть еще много других "магических" методов, которые позволяют определить специальное поведение объектов.

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"
атрибут
        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 ходе выполнения данной лабораторной работы были приобретены навык по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.