МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе №4.5

Аннотация типов

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 аннотациями типов при написании программ с помощью языка программирования Python версии 3.x. Рассмотрен вопрос контроля типов переменных и функций с использованием комментариев и аннотаций. Приведено описание PEP‘ов, регламентирующих работу с аннотациями, и представлены примеры работы с инструментом mypy для анализа Python кода

Ход работы:

1. Создал репозиторий на GitHub: https://github.com/IsSveshuD/OOP_Lab_4.5.git .
2. Выполнил пример:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from __future__ import annotations

from abc import ABC, abstractmethod


class Triad(ABC):
    def __init__(self, a: int, b: int, c: int):
        self.a = a
        self.b = b
        self.c = c

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

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

    @abstractmethod
    def compare(self, other: Triad) -> str:
        pass


class Date(Triad):
    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}")

    def compare(self, other: Triad) -> str:
        if self.a == other.a and self.b == other.b and self.c ==
other.c:
```

```
            return "Даты равны"
        else:
            return "Даты не равны"

    def __eq__(self, other: Date) -> bool:
        return self.a == other.a and self.b == other.b and self.c ==
other.c

    def __gt__(self, other: Date) -> bool:
        return (self.c, self.b, self.a) > (other.c, other.b, other.a)

    def __lt__(self, other: Date) -> bool:
        return (self.c, self.b, self.a) < (other.c, other.b, other.a)

    def __ge__(self, other: Date) -> bool:
        return (self.c, self.b, self.a) >= (other.c, other.b, other.a)

    def __le__(self, other: Date) -> bool:
        return (self.c, self.b, self.a) <= (other.c, other.b, other.a)

    def __ne__(self, other: Date) -> bool:
        return not self.__eq__(other)


if __name__ == '__main__':
    date1 = Date(17, 3, 2001)

    date1.display()
    date1.increase()
    date1.display()

    date2 = Date(17, 3, 2001)
    print(date1.compare(Date(17, 3, 2001)))

    print(date1 == date2)
    print(date1 > date2)
    print(date1 < date2)
    print(date1 >= date2)
    print(date1 <= date2)
    print(date1 != date2)
```

Рисунок 1 – Индивидуальное задание

3. Результат работы задания 1

```
Дата: 17/3/2001
Дата: 18/4/2002
Даты не равны
False
True
False
True
False
True
```

Рисунок 2 – Работа задания 1

4. Проверил на соответствие стандартам mypy.

```
(tools) PS C:\Users\user\Documents\Учёба\3 Курс\5-й семестр\ООП\Laboratory\Lab_4.5> mypy Task_1.py
Success: no issues found in 1 source file
```

Рисунок 3 – Проверка mypy

Ответы на вопросы:

1. Для чего нужны аннотации типов в языке Python?

1) Аннотации типов в языке Python представляют собой способ указать ожидаемый тип данных для аргументов функций, возвращаемых значений функций и переменных. Вот несколько причин, по которым аннотации типов могут быть полезны:

2) Документация: Аннотации типов могут служить документацией для кода, помогая другим разработчикам понять ожидаемые типы данных в функциях и методах.

3) Поддержка инструментов статического анализа: Аннотации типов могут использоваться инструментами статического анализа кода, такими как Mypy, Pyre или Pyright, чтобы проверять соответствие типов данных во время компиляции или анализа кода.

4) Улучшение читаемости: Аннотации типов могут помочь улучшить читаемость кода, особенно в случае сложных или больших проектов, где явное указание типов данных может помочь понять назначение переменных и результатов функций.

5) Интеграция с IDE: Некоторые интегрированные среды разработки (IDE), такие как PyCharm, могут использовать аннотации типов для предоставления подсказок о типах данных и автоматической проверки соответствия типов.

2. Как осуществляется контроль типов в языке Python?

В языке Python контроль типов данных может осуществляться несколькими способами:

1) Аннотации типов: Как уже упоминалось, в Python можно использовать аннотации типов для указания ожидаемых типов данных для

аргументов функций, возвращаемых значений функций и переменных. Это позволяет документировать ожидаемые типы данных и использовать инструменты статического анализа кода для проверки соответствия типов.

2) Использование инструментов статического анализа: Существуют сторонние инструменты, такие как Mypy, Pyre и Pyright, которые могут использоваться для статической проверки соответствия типов данных в python-коде. Эти инструменты могут обнаруживать потенциальные ошибки типов данных и предоставлять рекомендации по улучшению кода.

3) Вручную проверять типы данных: В Python можно вручную выполнять проверку типов данных с помощью условных операторов и функций, таких как isinstance(). Например, можно написать условие для проверки типа данных перед выполнением определенной операции.

4) Использование аннотаций типов в комбинации с декораторами: В Python можно использовать декораторы, такие как @overload из модуля functools, для реализации перегрузки функций с разными типами аргументов.

3. Какие существуют предложения по усовершенствованию Python для работы с аннотациями типов?

Предложения по усовершенствованию работы с аннотациями типов в Python включают расширение поддержки аннотаций типов, улучшение интеграции с инструментами статического анализа, улучшение документации и рекомендаций, а также разработку стандартной библиотеки типов. Эти изменения могут сделать работу с аннотациями типов более мощной и удобной для разработчиков.

4. Как осуществляется аннотирование параметров и возвращаемых значений функций?

В Python аннотирование параметров и возвращаемых значений функций осуществляется с использованием двоеточия и указания типа данных после имени параметра или перед знаком "->" для возвращаемого значения.

Например:

```
def greet(name: str) -> str:
return "Hello, " + name
```

В этом примере name: str указывает, что параметр name должен быть строкой, а -> str указывает, что функция возвращает строку.

5. Как выполнить доступ к аннотациям функций?

В Python можно получить доступ к аннотациям функций с помощью специального атрибута annotations . Этот атрибут содержит словарь, в котором ключами являются имена параметров или "return" (для возвращаемого значения), а значениями - указанные типы данных.

Пример:

```
def greet(name: str) -> str:
return "Hello, " + name
print(greet. annotations )
```

Этот код выведет на экран словарь с аннотациями функции greet:

{'name': <class 'str'>, 'return': <class 'str'>}

Таким образом, вы можете получить доступ к аннотациям функции и использовать их в своем коде, например, для проверки типов данных или для документирования функций.

6. Как осуществляется аннотирование переменных в языке Python?

В Python переменные можно аннотировать с использованием синтаксиса аннотаций типов. Это позволяет указать ожидаемый тип данных для переменной, хотя интерпретатор Python не выполняет никакой проверки типов во время выполнения.

7. Для чего нужна отложенная аннотация в языке Python?

Отложенная аннотация в Python (Delayed Evaluation Annotation) позволяет создавать аннотации типов, используя строковые литералы вместо ссылок на фактические классы. Это может быть полезно в случаях, когда требуется аннотировать типы данных, которые еще не определены или недоступны в момент написания аннотации.

Отложенные аннотации могут быть полезны при работе с циклическими зависимостями между классами или модулями, при использовании динамически загружаемых модулей или при аннотации типов в коде, который будет выполняться на разных версиях Python.

Вывод: в результате выполнения лабораторной работы были приобретены навыки по работе с аннотациями типов при написании программ с помощью языка программирования Python версии 3.x. Рассмотрен вопрос контроля типов переменных и функций с использованием комментариев и аннотаций. Приведено описание PEP‘ов, регламентирующих работу с аннотациями, и представлены примеры работы с инструментом mypy для анализа Python кода.