

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №5
дисциплины
«Объектно-ориентированное программирование»
Вариант 20

Выполнила:
Михеева Елена Александровна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Проверил:
Воронкин Р.А.-доцент департамента
цифровых, робототехнических систем и
электроники института перспективной
инженерии

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Задание:

Выполнить индивидуальное задание 2 лабораторной работы 2.19, добавив аннотации типов. Выполнить проверку программы с помощью утилиты туру.

Для проверки корректности работы программы были разработаны и реализованы тесты с использованием библиотеки pytest. Основное внимание уделялось проверке функции `display_tree` и корректной обработке аргументов командной строки.

Тестовые сценарии включали: проверку вывода дерева каталогов на различных уровнях вложенности; тестирование параметра `-a` для отображения скрытых файлов; проверку корректной обработки глубины отображения при использовании параметра `-s`; валидацию обработки несуществующих путей и выдачу соответствующих ошибок.

Результаты тестирования показали, что программа корректно обрабатывает все сценарии.

```
import argparse
import pathlib
from typing import List, Optional

def display_tree(
    directory: pathlib.Path,
    args: argparse.Namespace,
    prefix: str = "",
    current_depth: int = 0,
) -> None:
    """
    Рекурсивная функция для отображения дерева каталогов и файлов.

    :param directory: Путь к директории, которую нужно отображать.
    :param args: Объект аргументов командной строки.
    :param prefix: Префикс для отступов при отображении дерева.
    :param current_depth: Текущий уровень глубины для отображения дерева
    """
    if args.s is not None and current_depth > args.s:
        return

    items = list(directory.iterdir())
    items.sort()

    for idx, item in enumerate(items):
        connector = "├─ " if idx < len(items) - 1 else "└─ "
        new_prefix = prefix + (connector if idx < len(items) - 1 else " ")

        # Вывести дерево директорий
        if item.is_dir():
            if not args.f:
                print(prefix + connector + item.name + "/")
                display_tree(item, args, new_prefix, current_depth + 1)
        # Вывести дерево только с файлами
        elif item.is_file() and not args.d:
            # Если указан ключ -a, то учитываются скрытые файлы
            if args.a or not item.name.startswith("."):
                size = item.stat().st_size
                # Если ключ -t, то указывать полный путь
                if args.t:
                    print(f"{prefix}{connector}{item.name} ({size} bytes)")
                else:
                    print(f"{prefix}{connector}{item.name} ({size} bytes)

65 def main(command_line: Optional[List[str]] = None) -> None:
66     """
67     Главная функция программы.
68
69     :param command_line: Аргументы командной строки (по умолчанию None, если
70     вызывается непосредственно из командной строки).
71     """
72     # Создать основной парсер командной строки.
73     parser = argparse.ArgumentParser()
74     parser.add_argument(
75         "--version", action="version", version=f"%(prog)s 0.0.1"
76     )
77     parser.add_argument("directory", type=str, help="The directory to list.")
78     # Выводятся даже скрытые файлы.
79     parser.add_argument(
80         "-a", action="store_true", help="All files are listed."
81     )
82     # -f и -d взаимоисключающие, их нужно запретить вводить одновременно.
83     choose = parser.add_mutually_exclusive_group()
84     # Отображать каталоги.
85     choose.add_argument(
86         "-d", action="store_true", help="List directories only."
87     )
88     # Отображать файлы.
89     choose.add_argument("-f", action="store_true", help="List files only.")
90     # Максимальная глубина отображения дерева
91     parser.add_argument(
92         "-s", type=int, help="Max display depth of the directory tree."
93     )
94     # Не просто имя файла, а полное имя.
95     parser.add_argument(
96         "-t",
97         action="store_true",
98         help="Print the full path prefix for each file.",
99     )
100     # Выполнить разбор аргументов командной строки.
101     args = parser.parse_args(command_line)
102     directory = pathlib.Path(args.directory).resolve(strict=True)
103     display_tree(directory, args)
104
105
106 if __name__ == "__main__":
107     main()
108
```

Рисунок 3. Программа индивидуального задания

```

import io
import pathlib
import unittest
from unittest.mock import MagicMock, patch

from src.individual import display_tree

class TestDisplayTree(unittest.TestCase):

    @patch("pathlib.Path.iterdir")
    def test_display_tree_with_files(self, mock_iterdir):
        # Создаем фиктивную директорию с файлом
        fake_file = MagicMock()
        fake_file.is_dir.return_value = False
        fake_file.is_file.return_value = True
        fake_file.name = "file.txt"
        fake_file.stat.return_value.st_size = 1234

        mock_iterdir.return_value = [fake_file]

        # Патчим вывод
        with patch("sys.stdout", new_callable=io.StringIO) as mock_stdout:
            display_tree(
                pathlib.Path("/test_dir"),
                MagicMock(s=None, a=False, f=False, d=False, t=False),
                "",
            )
            output = mock_stdout.getvalue().strip()

        # Используем assert для проверки, что в выводе есть нужная информация
        assert (
            "file.txt (1234 bytes)" in output
        ), f"Expected 'file.txt (1234 bytes)' in output, but got {output}"

    @patch("pathlib.Path.iterdir")
    def test_display_tree_hidden_files(self, mock_iterdir):
        # Создаем фиктивную директорию с скрытым файлом
        fake_file = MagicMock()
        fake_file.is_dir.return_value = False
        fake_file.is_file.return_value = True
        fake_file.name = ".hidden_file"
        fake_file.stat.return_value.st_size = 1234

        mock_iterdir.return_value = [fake_file]

        # Патчим вывод с флагом -a для отображения скрытых файлов
        with patch("sys.stdout", new_callable=io.StringIO) as mock_stdout:
            display_tree(
                pathlib.Path("/test_dir"),
                MagicMock(s=None, a=True, f=True, d=False, t=False),
                "",
            )
            output = mock_stdout.getvalue().strip()

        # Проверка, что скрытый файл отображается
        assert (
            ".hidden_file (1234 bytes)" in output
        ), f"Expected '.hidden_file (1234 bytes)' in output, but got {output}"

if __name__ == "__main__":
    unittest.main()

```

Рисунок 4. Программа test_individual

```

(oop-5-py3.11) (base) elenamiheeva@MacBook-Pro-Elena Oop_5 % poetry run pytest
===== test session starts =====
platform darwin -- Python 3.11.7, pytest-8.3.4, pluggy-1.5.0
rootdir: /Users/elenamiheeva/Oop_5
configfile: pyproject.toml
plugins: mock-3.14.0
collected 2 items

test/test_individual.py ..

===== 2 passed in 0.03s =====
(oop-5-py3.11) (base) elenamiheeva@MacBook-Pro-Elena Oop_5 % mypy src/individual.py
Success: no issues found in 1 source file

```

Рисунок 4. Результат тестирования программы с помощью mypy и pytest

Ответы на контрольные вопросы:

1. Для чего нужны аннотации типов в языке Python?

Аннотации типов в Python предназначены для улучшения читаемости кода, облегчения его поддержки и документирования. Они позволяют явно указывать ожидаемые типы данных для переменных, параметров функций и возвращаемых значений. Это помогает разработчикам быстрее понимать, как

использовать функции и переменные. Кроме того, аннотации типов используются инструментами статического анализа, такими как `myru`, `pylint` или `pyright`, для выявления ошибок до выполнения кода.

2. Как осуществляется контроль типа в языке Python?

Python является языком с динамической типизацией, что означает, что контроль типов происходит во время выполнения программы. Однако разработчики могут использовать инструменты статического анализа, такие как `myru`, для проверки типов на этапе написания кода. Встроенные функции, такие как `isinstance()` и `type()`, позволяют вручную проверять типы данных. Если операция выполняется с неподходящим типом, Python выбрасывает исключение, например, `TypeError`.

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

Существует несколько предложений (PEP), направленных на улучшение работы с аннотациями типов. Например, PEP 563 вводит отложенную оценку аннотаций, что позволяет использовать типы, которые еще не определены. PEP 585 упрощает использование встроенных коллекций, таких как `list` и `dict`, для аннотаций. PEP 604 предлагает более удобный синтаксис для объединения типов с использованием оператора `|`. PEP 612 улучшает поддержку аннотаций для функций высшего порядка, а PEP 695 вводит новый синтаксис для параметров типов в Python 3.12.

4. Как осуществляется аннотирование параметров и возвращаемых значений функций?

Аннотации типов для параметров функций добавляются с использованием синтаксиса `: <тип>`, а для возвращаемых значений — с помощью `-> <тип>`.

5. Как выполнить доступ к аннотациям функций?

Аннотации функций хранятся в атрибуте `__annotations__`, который представляет собой словарь. В этом словаре ключи — это имена параметров, а значения — их типы.

6. Как осуществляется аннотирование переменных в языке Python?

Аннотации переменных добавляются с использованием синтаксиса :
<тип>.

7. Для чего нужна отложенная аннотация в языке Python?

Отложенная аннотация (PEP 563) позволяет использовать аннотации, которые зависят от типов, еще не определенных в коде. Это особенно полезно в случаях, когда класс ссылается на себя или когда аннотации находятся в модулях, которые еще не импортированы.

Вывод: были приобретены навыки по работе с аннотациями типов при написании программ с помощью языка программирования Python версии 3.x.