Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ
№220дисциплины «Основы программной инженерии»

Выполнил:
Джараян Арег Александрович
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и сопровождение программного обеспечения», очная форма обучения

_____
(подпись)

Проверил Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Лабораторная работа 4.5 Аннотация типов.

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.

1. Создание нового репозитория с лицензией MIT.

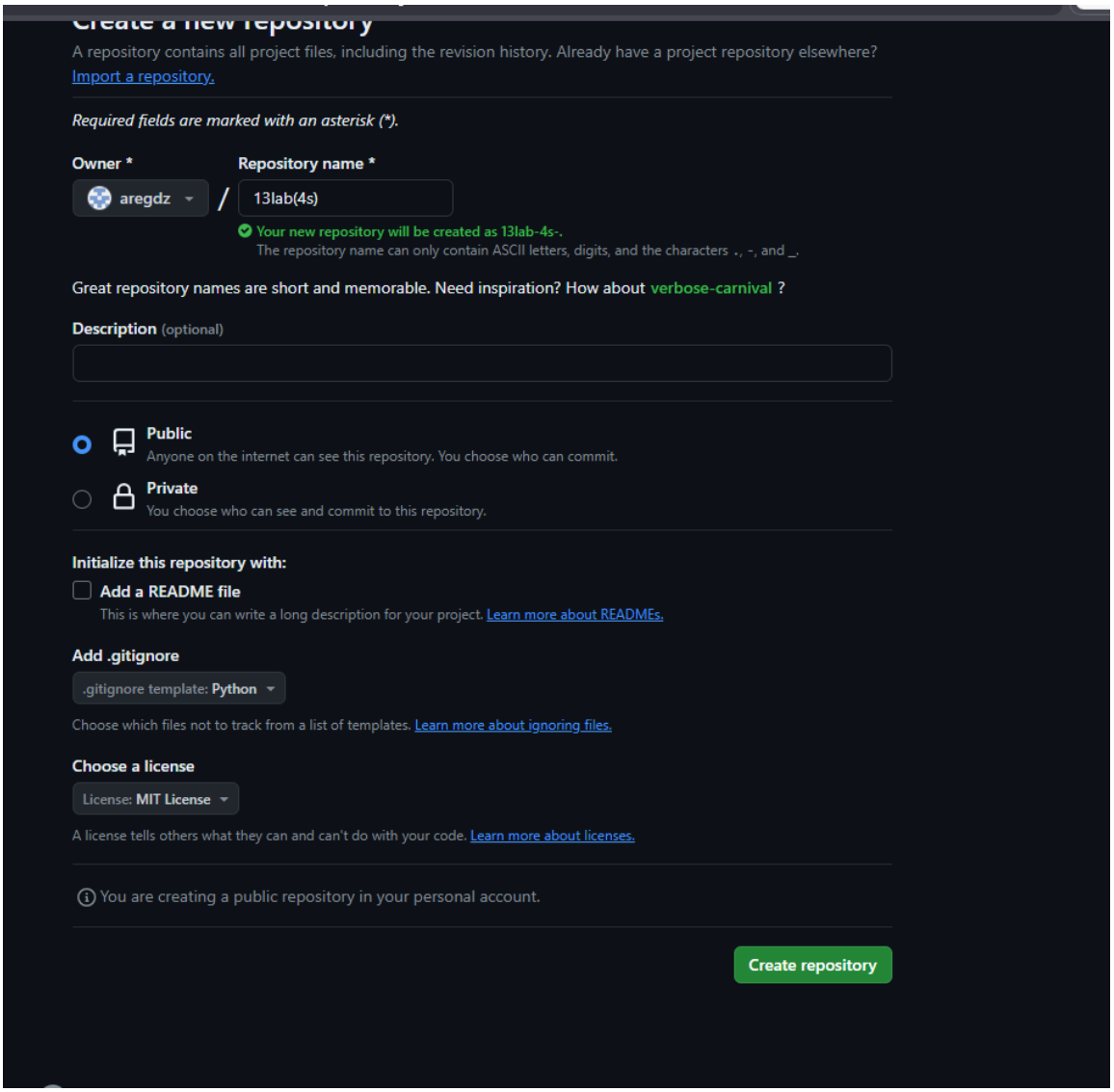

Рисунок 1 – создание репозитория

2. Клонировал репозиторий на рабочий ПК.

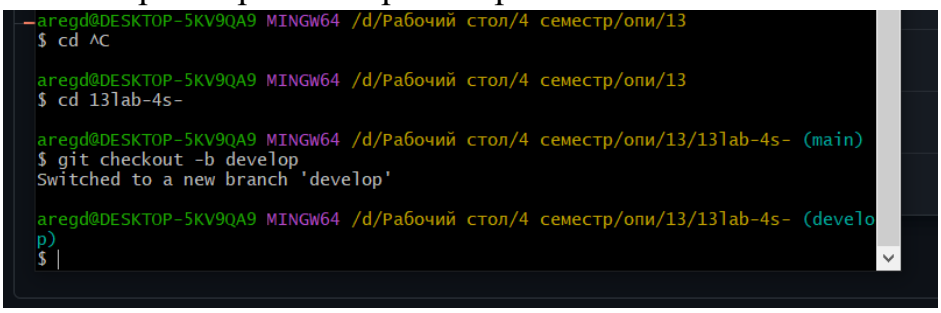

Рисунок 2 – клонирование репозитория

3. Дополнил файл .gitignore необходимыми инструкциями.

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
```

Рисунок 4 – Файл .gitignore

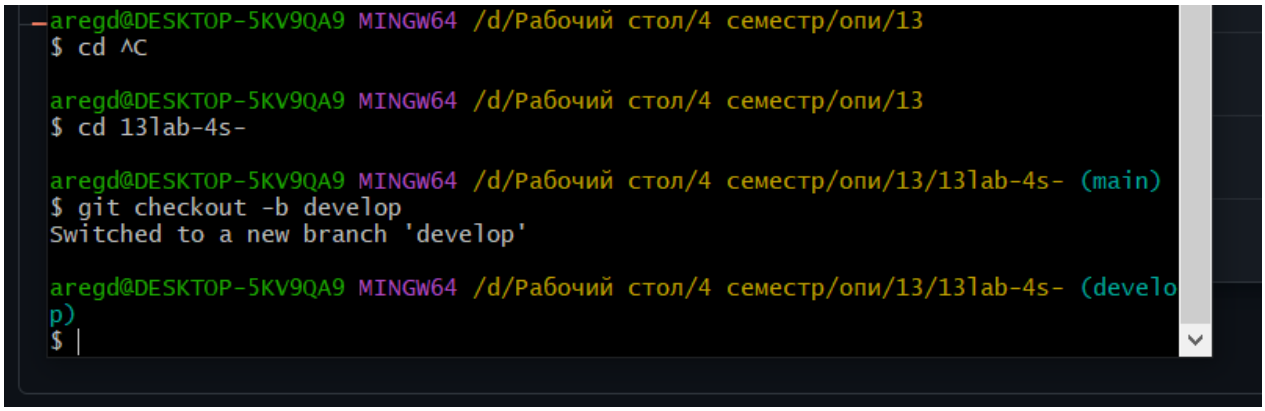

Рисунок 4 – организация ветки

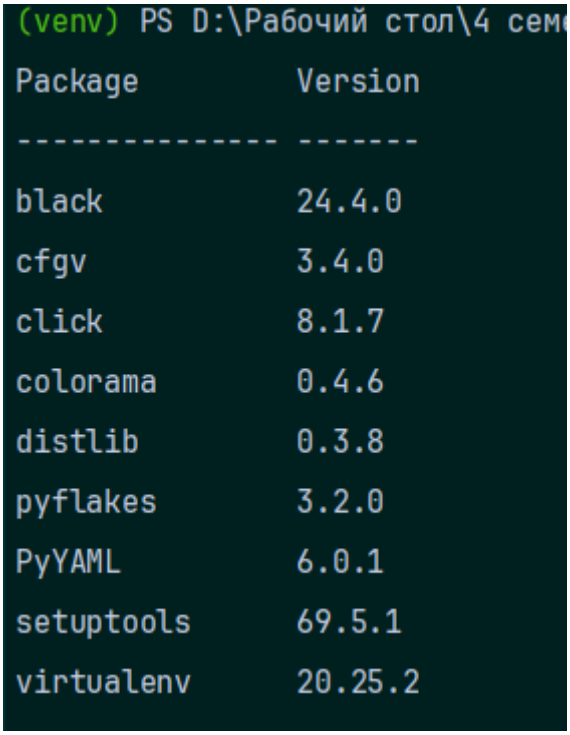

Рисунок 5 – создание виртуального окружения

4.Пример 1.

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

from dataclasses import dataclass, field
from datetime import date
import logging
import sys
from typing import List
import xml.etree.ElementTree as ET

# Класс пользовательского исключения в случае, если неверно
# введен номер года.
```

```python
class IllegalYearError(Exception):
    def __init__(self, year, message="Illegal year number"):
        self.year = year
        self.message = message
        super(IllegalYearError, self).__init__(message)

    def __str__(self):
        return f"{self.year} -> {self.message}"

# Класс пользовательского исключения в случае, если введенная
# команда является недопустимой.
class UnknownCommandError(Exception):
    def __init__(self, command, message="Unknown command"):
        self.command = command
        self.message = message
        super(UnknownCommandError, self).__init__(message)

    def __str__(self):
        return f"{self.command} -> {self.message}"

@dataclass(frozen=True)
class Worker:
    name: str
    post: str
    year: int

@dataclass
class Staff:
    workers: List[Worker] = field(default_factory=lambda: [])

    def add(self, name: str, post: str, year: int) -> None:
        # Получить текущую дату.
        today = date.today()
        if year < 0 or year > today.year:
            raise IllegalYearError(year)
        self.workers.append(
            Worker(
                name=name,
                post=post,
                year=year
            )
        )
        self.workers.sort(key=lambda worker: worker.name)

    def __str__(self) -> str:
        # Заголовок таблицы.
        table = []
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        table.append(line)
        table.append(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        table.append(line)
        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(self.workers, 1):
            table.append(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.name,
                    worker.post,
                    worker.year
```

```
                )
            )
            table.append(line)
        return '\n'.join(table)

    def select(self, period: int) -> List[Worker]:
        # Получить текущую дату.
        today = date.today()
        result: List[Worker] = []
        for worker in self.workers:
            if today.year - worker.year >= period:
                result.append(worker)
        return result

    def load(self, filename: str) -> None:
        with open(filename, 'r', encoding='utf8') as fin:
            xml = fin.read()
        parser = ET.XMLParser(encoding="utf8")
        tree = ET.fromstring(xml, parser=parser)
        self.workers = []
        for worker_element in tree:
            name, post, year = None, None, None
            for element in worker_element:
                if element.tag == 'name':
                    name = element.text
                elif element.tag == 'post':
                    post = element.text
                elif element.tag == 'year':
                    year = int(element.text)
            if name is not None and post is not None \
                    and year is not None:
                self.workers.append(
                    Worker(
                        name=name,
                        post=post,
                        year=year
                    )
                )

    def save(self, filename: str) -> None:
        root = ET.Element('workers')
        for worker in self.workers:
            worker_element = ET.Element('worker')
            name_element = ET.SubElement(worker_element, 'name')
            name_element.text = worker.name
            post_element = ET.SubElement(worker_element, 'post')
            post_element.text = worker.post
            year_element = ET.SubElement(worker_element, 'year')
            year_element.text = str(worker.year)
            root.append(worker_element)
        tree = ET.ElementTree(root)
        with open(filename, 'wb') as fout:
            tree.write(fout, encoding='utf8', xml_declaration=True)

if __name__ == '__main__':
    # Выполнить настройку логгера.
    logging.basicConfig(
        filename='workers4.log',
        level=logging.INFO
    )
    # Список работников.
    staff = Staff()
    # Организовать бесконечный цикл запроса команд.
    while True:
        try:
            # Запросить команду из терминала.
            command = input(">>> ").lower()
            # Выполнить действие в соответствие с командой.
            if command == 'exit':
                break
            elif command == 'add':
```

```
            # Запросить данные о работнике.
            name = input("Фамилия и инициалы? ")
            post = input("Должность? ")
            year = int(input("Год поступления? "))
            # Добавить работника.
            staff.add(name, post, year)
            logging.info(
                f"Добавлен сотрудник: {name}, {post}, "
                f"поступивший в {year} году."
            )
        elif command == 'list':
            # Вывести список.
            print(staff)
            logging.info("Отображен список сотрудников.")
        elif command.startswith('select '):
            # Разбить команду на части для выделения номера года.
            parts = command.split(maxsplit=1)
            # Запросить работников.
            selected = staff.select(parts[1])
            # Вывести результаты запроса.
            if selected:
                for idx, worker in enumerate(selected, 1):
                    print(
                        '{:>4}: {}'.format(idx, worker.name)
                    )
                    logging.info(
                        f"Найдено {len(selected)} работников со "
                        f"стажем более {parts[1]} лет."
                    )
            else:
                print("Работники с заданным стажем не найдены.")
                logging.warning(
                    f"Работники со стажем более {parts[1]} лет не найдены."
                )
        elif command.startswith('load '):
            # Разбить команду на части для имени файла.
            parts = command.split(maxsplit=1)
            # Загрузить данные из файла.
            staff.load(parts[1])
            logging.info(f"Загружены данные из файла {parts[1]}.")
        elif command.startswith('save '):
            # Разбить команду на части для имени файла.
            parts = command.split(maxsplit=1)
            # Сохранить данные в файл.
            staff.save(parts[1])
            logging.info(f"Сохранены данные в файл {parts[1]}.")
        elif command == 'help':
            # Вывести справку о работе с программой.
            print("Список команд:\n")
            print("add - добавить работника;")
            print("list - вывести список работников;")
            print("select <стаж> - запросить работников со стажем;")
            print("load <имя_файла> - загрузить данные из файла;")
            print("save <имя_файла> - сохранить данные в файл;")
            print("help - отобразить справку;")
            print("exit - завершить работу с программой.")
        else:
            raise UnknownCommandError(command)
    except Exception as exc:
        logging.error(f"Ошибка: {exc}")
        print(exc, file=sys.stderr)
```

Рисунок 6 – пример 1

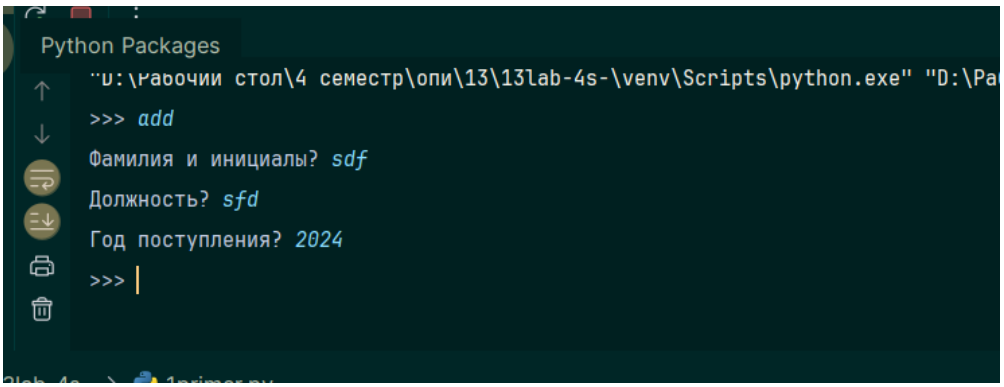

Рисунок 7 – выполнение примера 1

5.Задание 1. Выполнить индивидуальное задание 2 лабораторной работы 2.19, добавив аннтотации типов. Выполнить проверку программы с помощью утилиты mypy.

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

import argparse
import json
from pathlib import Path
from typing import List, Dict, Any


def add_flight(flights: List[Dict[str, Any]], destination: str, departure_date: int,
aircraft_type: str) -> List[Dict[str, Any]]:

    flights.append(
        {
            "destination": destination,
            "departure_date": departure_date,
            "aircraft_type": aircraft_type,
        }
    )

    return flights


def display_flights(flights: List[Dict[str, Any]]) -> None:

    if flights:
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4, "-" * 30, "-" * 20, "-" * 8
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^20} | {:^8} |".format(
                "No", "Destination", "Departure Date", "Aircraft Type"
            )
        )
        print(line)
        for idx, flight in enumerate(flights, 1):
            print(
                "| {:>4} | {:<30} | {:<20} | {:>8} |".format(
```

```
                    idx,
                    flight.get("destination", ""),
                    flight.get("departure_date", ""),
                    flight.get("aircraft_type", ""),
                )
            )
            print(line)
    else:
        print("List of flights is empty.")


def select_flights(flights: List[Dict[str, Any]], date: int) -> List[Dict[str, Any]]:

    result = []
    for flight in flights:
        if flight.get("departure_date") == date:
            result.append(flight)
    return result


def save_flights(file_path: Path, flights: List[Dict[str, Any]]) -> None:

    with open(file_path, "w", encoding="utf-8") as fout:
        json.dump(flights, fout, ensure_ascii=False, indent=4)


def load_flights(file_path: Path) -> List[Dict[str, Any]]:

    with open(file_path, "r", encoding="utf-8") as fin:
        return json.load(fin)


def main(command_line=None) -> None:
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename", action="store", help="The data file name"
    )

    parser = argparse.ArgumentParser("flights")
    parser.add_argument("--version", action="version", version="%(prog)s 0.1.0")
    subparsers = parser.add_subparsers(dest="command")

    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Add a new flight"
    )
    add.add_argument(
        "filename", action="store", help="The data file name"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination of the flight",
    )
    add.add_argument(
        "-dd",
        "--departure_date",
        action="store",
        required=True,
        help="Departure date of the flight",
    )
    add.add_argument(
        "-at",
        "--aircraft_type",
        action="store",
        required=True,
        help="Aircraft type of the flight",
    )

    _ = subparsers.add_parser(
```

```
        "display", parents=[file_parser], help="Display all flights"
    )

    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select flights by departure date",
    )
    select.add_argument(
        "-D",
        "--date",
        action="store",
        required=True,
        help="Departure date to select flights",
    )

    args = parser.parse_args(command_line)

    home_dir = Path.home()  # Получаем домашний каталог пользователя
    file_path = (
        home_dir / args.filename
    )  # Путь к файлу данных в домашнем каталоге

    is_dirty = False
    if file_path.exists():
        flights = load_flights(file_path)
    else:
        flights = []

    if args.command == "add":
        flights = add_flight(
            flights, args.destination, args.departure_date, args.aircraft_type
        )
        is_dirty = True

    elif args.command == "display":
        display_flights(flights)

    elif args.command == "select":
        selected = select_flights(flights, args.date)
        display_flights(selected)

    if is_dirty:
        save_flights(file_path, flights)


if __name__ == "__main__":
    main()
```

Рисунок 8 – Задание 1

```
(venv) PS D:\Рабочий стол\4 семестр\опи\13\13lab-4s-> mypy 1zadanie.py
Success: no issues found in 1 source file
(venv) PS D:\Рабочий стол\4 семестр\опи\13\13lab-4s->
```

Рисунок 9 – пример выполнения примера 2

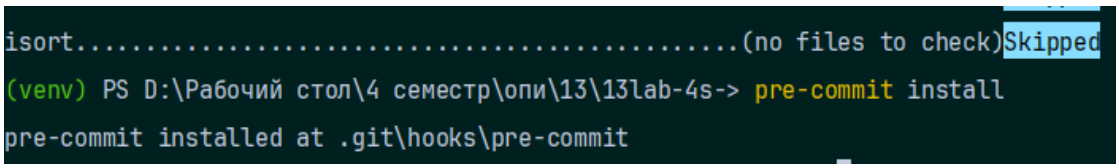

Рисунок 10 - фиксация изменений

Контрольные вопросы:

1.Для чего нужны аннотации типов в языке Python? Аннотации типов в Python предоставляют способ явно указывать типы данных параметров функций и их возвращаемых значений. Это не влияет на выполнение программы, но улучшает читаемость кода, облегчает отладку и позволяет инструментам статического анализа кода проводить более точную проверку.

2.Как осуществляется контроль типов в языке Python? В Python контроль типов не осуществляется во время выполнения программы, поскольку это динамически типизированный язык. Однако, с помощью аннотаций типов и сторонних инструментов, таких как mypy, можно осуществлять статический анализ кода и проверять соответствие типов до выполнения программы.

3.Какие существуют предложения по усовершенствованию Python для работы с аннотациями типов? Одно из значительных усовершенствований — PEP 484, представляющий синтаксис для аннотаций типов и стандартный модуль typing, который предоставляет множество стандартных типов для аннотации. PEP 563 предлагает отложенную оценку аннотаций типов, что позволяет использовать типы, определенные ниже по тексту модуля.

4.Как осуществляется аннотирование параметров и возвращаемых значений функций? Аннотирование параметров функции производится путем добавления : <Тип> после имени параметра. Для указания типа возвращаемого значения используется -> <Тип> перед двоеточием в конце объявления функции.

5.Как выполнить доступ к аннотациям функций? Аннотации функций хранятся в атрибуте __annotations__ функции. Можно обратиться к этому атрибуту, чтобы получить словарь аннотаций.

6. Как осуществляется аннотирование переменных в языке Python?Аннотации переменных можно задавать с помощью синтаксиса имя_переменной: Тип = значение.

7. Для чего нужна отложенная аннотация в языке Python? Отложенные аннотации (PEP 563) позволяют ссылаться на типы, которые еще не определены на момент аннотации. Такое бывает необходимо, когда аннотация типа в функции использует класс, который объявлен позже в коде, или для аннотаций внутри самого класса. Отложенное оценивание аннотаций избавляет от необходимости использовать строковые литералы для таких ссылок и позволяет Python "ожидать" с определением аннотации до тех пор, пока все необходимые типы не будут доступны.