

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 №28
дисциплины «Основы программной инженерии»

Выполнил:
Плугатырев Владислав Алексеевич
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 _____

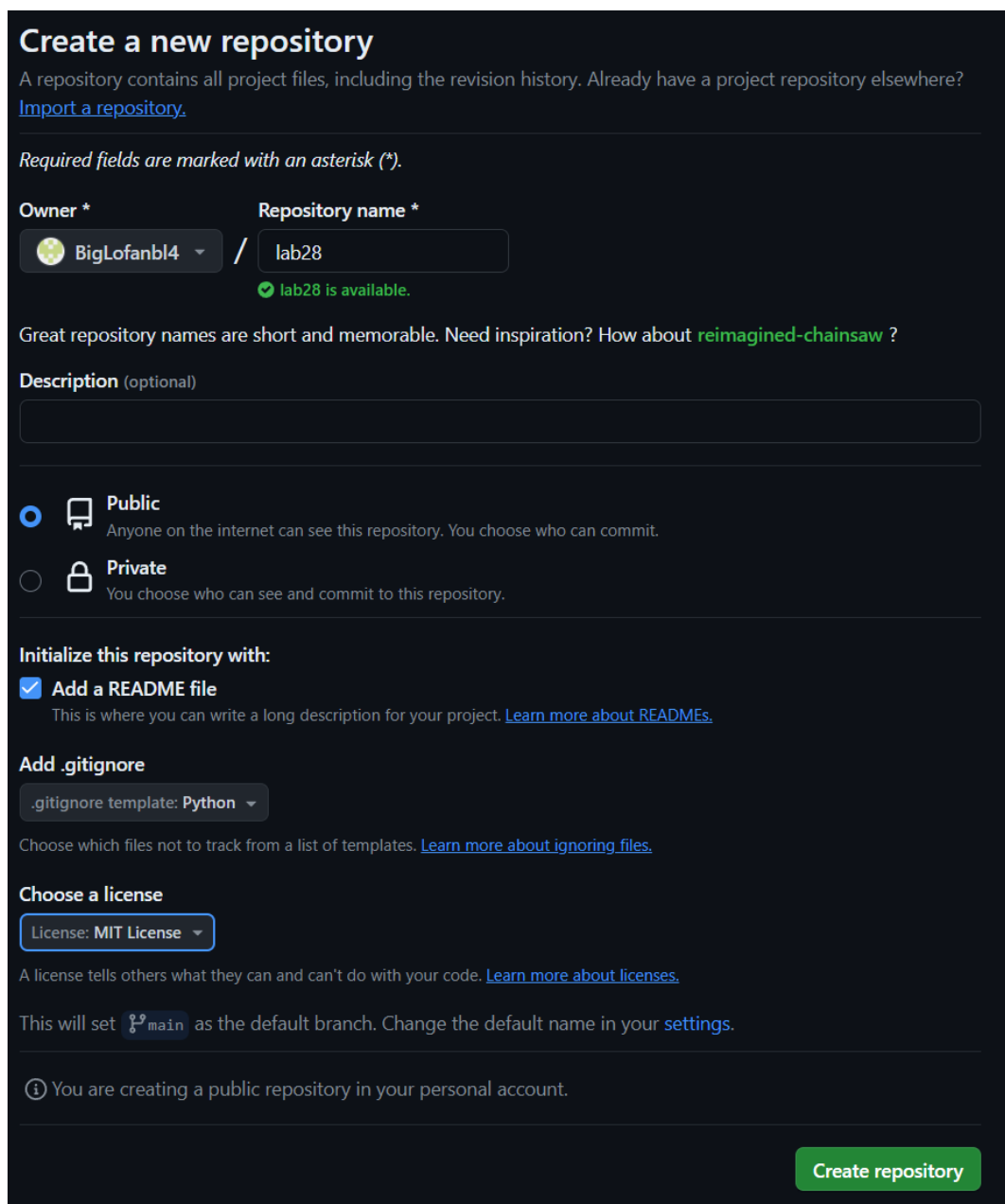
Ставрополь, 2024 г.

Тема: 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 туру для анализа Python кода.

Ход работы.

1. Создание репозитория.




Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).


Owner * / Repository name *


 BigLofanbl4 / lab28

✔ lab28 is available.

Great repository names are short and memorable. Need inspiration? How about [reimagined-chainsaw](#) ?

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore


.gitignore template: **Python**


Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **MIT License**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

[Create repository](#)

Рисунок 1.1 – Создание репозитория

2. Выполнение примеров из лабораторной работы.

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

    def add(self, name: str, post: str, year: int) -> None:
        today = date.today()
        if year < 0 or year > today.year:
            raise IllegalYearError(year)
        self.workers.append(Worker(name=name, post=post, year=year))

    (method) def __str__(self: Self@Staff) -> str: e)

def __str__(self) -> str:
    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)
    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]:
    today = date.today()
    result: List[Worker] = []
    for worker in self.workers:
        if today.year - worker.year >= period:
            result.append(worker)
    return result
```

Рисунок 2.1 – Код примера

3. Выполнение индивидуального задания: выполнить индивидуальное задание 2 лабораторной работы 2.19, добавив аннотации типов. Выполнить проверку программы с помощью утилиты туру.

```
def validation(instance: list) -> bool:
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "surname": {"type": "string"},
                "name": {"type": "string"},
                "zodiac": {"type": "string"},
                "birthday": {
                    "type": "array",
                    "items": {"type": "string"},
                    "minitems": 3,
                },
            },
            "required": ["surname", "name", "birthday"],
        },
    }
    try:
        validate(instance, schema=schema)
        return True
    except ValidationError as err:
        print(err.message)
        return False

def load_people(file_name: str) -> list:
    with open(file_name, "r") as f:
        people = json.load(f)

    if validation(people):
        return people

def save_people(file_name: str, people_list: list) -> None:
    with open(file_name, "w", encoding="utf-8") as f:
        json.dump(people_list, f, ensure_ascii=False, indent=4)

def add_person(people: list, surname: str, name: str, zodiac: str, birthday: str) -> list:
    people.append(
        {
            "surname": surname,
            "name": name,
            "zodiac": zodiac,
            "birthday": birthday.split("."),
        }
    )
    return people
```

Рисунок 3.1 – Код программы

```
(lab28) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\28\lab28>туру individuals/ind1.py
Success: no issues found in 1 source file

(lab28) C:\Users\vladi\OneDrive\Рабочий стол\Основы программной инженерии\28\lab28>
```

Рисунок 3.2 – Проверка с помощью туру

Ответы на контрольные вопросы

1. Аннотации типов в Python используются для указания ожидаемых типов переменных, аргументов функций и возвращаемых значений. Это добавляет ясность и документирует ожидаемое использование функций и переменных. Хотя Python остается динамически типизированным языком и аннотации не влияют на выполнение программы напрямую, они могут использоваться статическими анализаторами кода, IDE и другими инструментами для обеспечения типовой безопасности и предотвращения ошибок.

2. Контроль типов в Python, как правило, осуществляется во время выполнения (runtime). Для статической проверки типов могут использоваться внешние инструменты, такие как MyPy, Pyright, Pyre и другие. Эти инструменты анализируют аннотации типов и предоставляют разработчикам информацию об ошибках типизации до запуска программы.

3. Предложения по усовершенствованию Python для работы с аннотациями типов включают PEP (Python Enhancement Proposals), такие как:

- PEP 484 – введение аннотаций типов в Python.
- PEP 526 – синтаксис для аннотирования переменных.
- PEP 561 – поддержка пакетов распределения аннотаций типов.
- PEP 563 – отложенная оценка аннотаций, которая становится поведением по умолчанию в Python 3.10.

- PEP 585 – введение универсальных типизированных коллекций.
- PEP 589 – типизированные словари.
- PEP 593 – гибкие аннотации функций с помощью «typing.Annotated».

4. Аннотирование параметров и возвращаемых значений функций:

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

В этом примере параметр «name» аннотирован как строковый тип («str»), и функция указывает, что возвращает строку («-> str»).

5. Доступ к аннотациям функций можно получить через атрибут «__annotations__» функции:

```
print(greet.__annotations__)  
# Выведет: {'name': <class 'str'>, 'return': <class 'str'>}
```

6. Аннотирование переменных:

```
age: int = 25
```

Здесь переменная «age» аннотирована как целочисленный тип («int»), что помогает разработчикам и статическим анализаторам понять ожидаемый тип значения переменной.

7. Отложенная аннотация в Python (PEP 563) позволяет исправить проблему циклических импортов и улучшить производительность за счет того, что строки с аннотациями типов не оцениваются в момент определения класса. Вместо этого они оцениваются только тогда, когда это действительно необходимо, например при использовании инструментов статической проверки типов. Для активации отложенных аннотаций необходимо поместить «from __future__ import annotations» в начало файла: