

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 №4.5
дисциплины «Объектно-ориентированное программирование»

Выполнил:
Магомедов Имран Борисович
3 курс, группа «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Р.А., доцент департамента
цифровых, робототехнических систем и
электроники

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

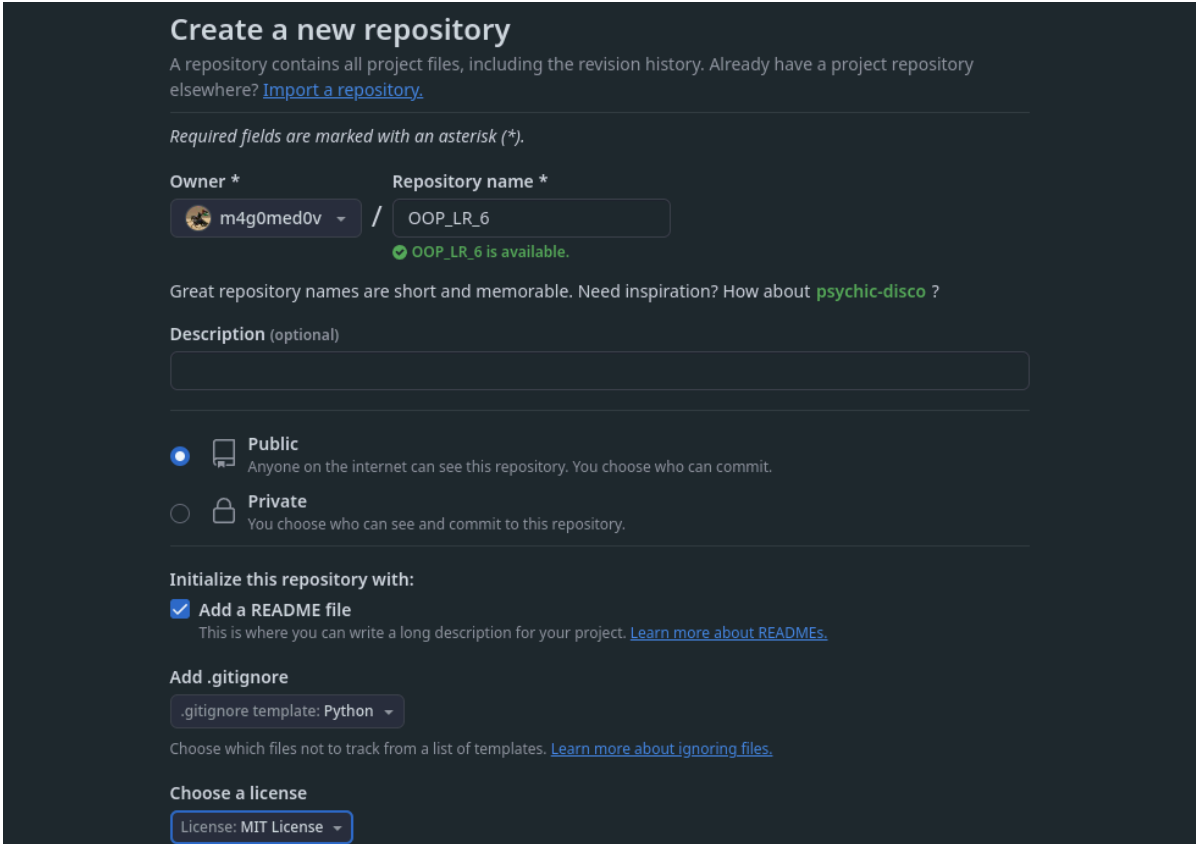
Ставрополь, 2024 г.

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.

Методика выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and explains that a repository contains all project files. Below this, it asks if the user already has a project repository elsewhere and provides a link to 'Import a repository'. A note states 'Required fields are marked with an asterisk (*)'. The form has two main sections: 'Owner' and 'Repository name'. The 'Owner' is set to 'm4g0med0v' with a dropdown arrow. The 'Repository name' is 'OOP_LR_6' with a dropdown arrow and a green checkmark indicating it is available. Below this, there is a suggestion for repository names: 'Great repository names are short and memorable. Need inspiration? How about **psychic-disco** ?'. The 'Description' field is optional and empty. The 'Visibility' section has two options: 'Public' (selected with a radio button) and 'Private'. The 'Initialize this repository with:' section has a checked checkbox for 'Add a README file'. The 'Add .gitignore' section has a dropdown menu set to '.gitignore template: Python'. The 'Choose a license' section has a dropdown menu set to 'License: MIT License'.

3. Выполните клонирование созданного репозитория.

```
~/ncfu > git clone https://github.com/m4g0med0v/00P_LR_6.git
Клонирование в «00P_LR_6»...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Получение объектов: 100% (5/5), готово.
```

4. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```
~/00P_LR_6 > git branch develop
~/00P_LR_6 > git switch develop
Переключились на ветку «develop»
```

5. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

Индивидуальное задание 1.

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

```

35 def connect_db(db_name: str) → sqlite3.Connection:
36     """Создание соединения с базой данных и создание таблиц."""
37     try:
38         if not Path("data/").exists():
39             os.mkdir("data")
40         conn = sqlite3.connect(f"data/{db_name}.db")
41         cursor = conn.cursor()
42
43         cursor.execute("""
44             CREATE TABLE IF NOT EXISTS trains (
45                 id INTEGER PRIMARY KEY AUTOINCREMENT,
46                 destination TEXT NOT NULL,
47                 number TEXT NOT NULL UNIQUE,
48                 time TEXT NOT NULL
49             )
50         """)
51
52         cursor.execute("""
53             CREATE TABLE IF NOT EXISTS stations (
54                 id INTEGER PRIMARY KEY AUTOINCREMENT,
55                 station_name TEXT NOT NULL,
56                 train_id INTEGER,
57                 FOREIGN KEY (train_id) REFERENCES trains(id)
58             )
59         """)
60
61         conn.commit()
62         logging.info("Соединение с базой данных успешно установлено.")
63         return conn
64     except Exception as e:
65         logging.error("Ошибка при подключении к базе данных: %s", e)
66         raise ConnectError()

```

```

69 def add_train(
70     conn: sqlite3.Connection,
71     destination: str,
72     number: str,
73     time: str,
74     station_name: str,
75 ) → Optional[Tuple[str, str, str, str]]:
76     """Добавить новый поезд и связанную станцию в базу данных."""
77     try:
78         cursor = conn.cursor()
79         cursor.execute(
80             "INSERT INTO trains (destination, number, time) VALUES (?, ?, ?)",
81             (destination, number, time),
82         )
83         train_id = cursor.lastrowid
84
85         cursor.execute(
86             "INSERT INTO stations (station_name, train_id) VALUES (?, ?)",
87             (station_name, train_id),
88         )
89
90         conn.commit()
91         logging.info(
92             "Добавлен поезд №%s, пункт назначения: %s, время отправления: %s.",
93             number,
94             destination,
95             time,
96         )
97         return find_train(conn, number)
98     except sqlite3.IntegrityError:
99         logging.error("Поезд с номером %s уже существует.", number)
100         print(f"Ошибка: поезд с номером {number} уже существует.")
101     except Exception as e:
102         logging.error("Ошибка при добавлении поезда: %s", e)
103         print(f"Ошибка при добавлении поезда: {e}")
104     return None

```

```

107 def list_trains(
108     conn: sqlite3.Connection,
109 ) → List[Tuple[str, str, str, str]]:
110     """Вывести все поезда и их станции."""
111     try:
112         cursor = conn.cursor()
113         cursor.execute("""
114             SELECT trains.number, trains.destination, trains.time, stations.station_name
115             FROM trains
116             LEFT JOIN stations ON trains.id = stations.train_id
117         """)
118
119         trains = cursor.fetchall()
120         if trains:
121             logging.info("Список поездов успешно извлечен.")
122             return trains
123         else:
124             logging.info("Список поездов пуст.")
125             return []
126     except Exception as e:
127         logging.error("Ошибка при получении списка поездов: %s", e)
128         print(f"Ошибка при получении списка поездов: {e}")
129     return []

```

```

132 def find_train(
133     conn: sqlite3.Connection, number: str
134 ) → Optional[Tuple[str, str, str, str]]:
135     """Найти и вывести информацию о поезде по его номеру."""
136     try:
137         cursor = conn.cursor()
138         cursor.execute(
139             """
140             SELECT trains.number, trains.destination, trains.time, stations.station_name
141             FROM trains
142             LEFT JOIN stations ON trains.id = stations.train_id
143             WHERE trains.number = ?
144         """,
145             (number,),
146         )
147
148         train = cursor.fetchone()
149         if isinstance(train, tuple) and len(train) == 4:
150             logging.info("Поиск поезда №%s завершен.", number)
151             return train # Успешный результат
152         else:
153             logging.info("Поезд №%s не найден.", number)
154             return None
155     except Exception as e:
156         logging.error("Ошибка при поиске поезда №%s: %s", number, e)
157         print(f"Ошибка при поиске поезда: {e}")
158     return None

```

```

~/ncfu/OOP_LR_6  git  develop ?4  mypy src/individual_task_1.py
Success: no issues found in 1 source file

```

6. Зафиксируйте сделанные изменения в репозитории.

```

~/ncfu/OOP_LR_6 git develop +1 ?1 git commit -m "added individual_task"
check yaml.....(no files to check)Skipped
fix end of files.....Passed
trim trailing whitespace.....Passed
ruff.....Passed
ruff-format.....Passed
mypy.....Passed
[develop 87ec019] added individual_task
1 file changed, 214 insertions(+)
create mode 100644 src/individual_task_1.py

```

7. Выполните слияние ветки для разработки с веткой main / master.

```

~/ncfu/OOP_LR_6 git develop git switch main
Переключились на ветку «main»
Эта ветка соответствует «origin/main».

~/ncfu/OOP_LR_6 main git merge develop
Обновление a216389..c3f0454
Fast-forward
 .pre-commit-config.yaml | 24 ++++++
 requirements.txt         |  2 ++
 src/individual_task_1.py | 214 ++++++
 tests/run_tests.py       | 12 +++++
 tests/test_individual_task_1.py | 83 ++++++
 5 files changed, 335 insertions(+)
 create mode 100644 .pre-commit-config.yaml
 create mode 100644 requirements.txt
 create mode 100644 src/individual_task_1.py
 create mode 100644 tests/run_tests.py
 create mode 100644 tests/test_individual_task_1.py

```

8. Отправьте сделанные изменения на сервер GitHub.

```

~/ncfu/OOP_LR_6 main 13 git push
Перечисление объектов: 14, готово.
Подсчет объектов: 100% (14/14), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (11/11), готово.
Запись объектов: 100% (13/13), 4.77 КиБ | 1.19 МиБ/с, готово.
Total 13 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/m4g0med0v/OOP_LR_6.git
 a216389..c3f0454  main -> main

```

Контрольные вопросы

1. Для чего нужны аннотации типов в языке Python?

Аннотации типов в Python служат для указания ожидаемых типов данных переменных, параметров функций и их возвращаемых значений. Это позволяет:

- Делать код более читаемым и понятным для других разработчиков.
- Уменьшить количество ошибок, связанных с неправильным использованием типов.
- Использовать инструменты статического анализа, такие как `myru`, для проверки типов на этапе разработки.

2. Как осуществляется контроль типов в языке Python?

Python не является строго типизированным языком и не выполняет проверку типов во время выполнения (`runtime`). Вместо этого:

- Аннотации типов используются только для документации и проверки статическими анализаторами, такими как `myru`, `pyright` или `pylint`.
- В самом Python контроль типов можно реализовать вручную с помощью проверок в коде, например, через `isinstance`.

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

PEP 484: Ввёл систему аннотаций типов и модуль `typing`.

PEP 563: Предложил использовать "отложенные аннотации" (аннотации, представленные в виде строк), чтобы ускорить время компиляции и решить проблемы с циклическими импортами.

PEP 585: Обеспечил поддержку аннотаций типов встроенными типами (list, dict и т. д.) без необходимости использования typing.

PEP 649: Рассматривает более эффективный способ хранения аннотаций для случаев, когда они нужны во время выполнения.

4. Как осуществляется аннотирование параметров и возвращаемых значений функций?

Аннотирование параметров и возвращаемых значений функций выполняется с помощью синтаксиса:

```
def function_name(param1: Type1, param2: Type2) → ReturnType:
    ...
```

5. Как выполнить доступ к аннотациям функций?

Аннотации типов функций сохраняются в атрибуте `__annotations__`:

```
def greet(name: str) → str:
    return f"Hello, {name}!"

print(greet.__annotations__)
# Вывод: {'name': 'str', 'return': 'str'}
```


6. Как осуществляется аннотирование переменных в языке Python?

Для аннотирования переменных используется синтаксис:

```
variable_name: Type = value
```

7. Для чего нужна отложенная аннотация в языке Python?

Отложенная аннотация (PEP 563) заключается в том, что аннотации типов сохраняются в виде строк и вычисляются только при необходимости. Это полезно для:

- Решения проблем с циклическими импортами.
- Ускорения времени загрузки модулей.
- Обеспечения обратной совместимости.

```
from __future__ import annotations

def foo(a: 'MyClass') → 'AnotherClass':
    ...
```