

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**
**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе

Дисциплина: «Объектно – ориентированное программирование»

Выполнил: студент 3 курса

группы ИВТ-б-о-21-1

Уланбекова Айканыш Уланбековна

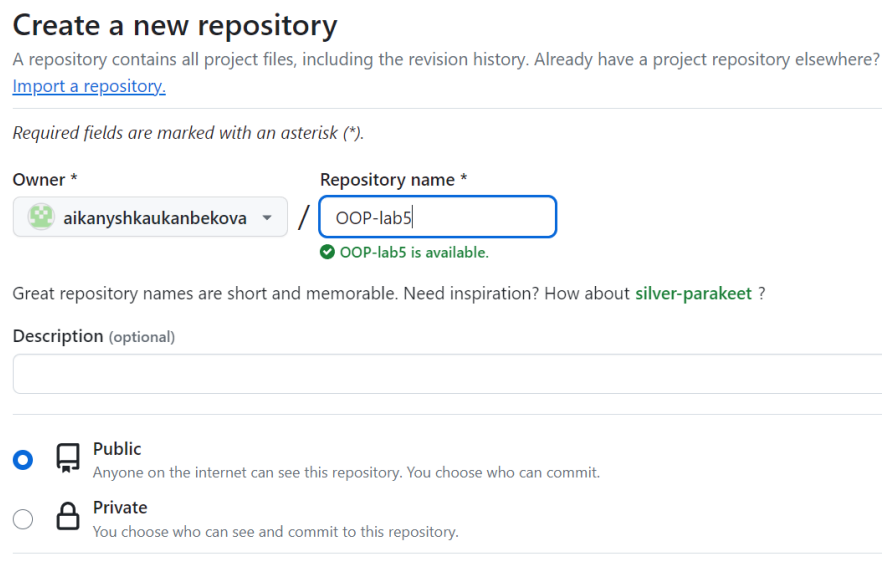
Ставрополь 2023

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.

Порядок выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.




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 *

 aikanyshkaukanbekova / OOP-lab5

✔ OOP-lab5 is available.

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

Description (optional)

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


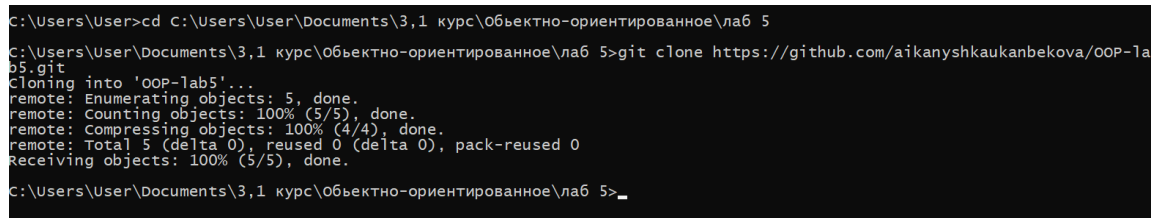
☐  **Private**
You choose who can see and commit to this repository.

Рисунок 1. Создание репозитория

2. Выполните клонирование созданного репозитория.



```
C:\Users\User>cd C:\Users\User\Documents\3,1 курс\Объектно-ориентированное\лаб 5
C:\Users\User\Documents\3,1 курс\Объектно-ориентированное\лаб 5>git clone https://github.com/aikanyshkaukanbekova/OOP-lab5.git
Cloning into 'OOP-lab5'...
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
Receiving objects: 100% (5/5), done.
C:\Users\User\Documents\3,1 курс\Объектно-ориентированное\лаб 5>
```

Рисунок 2. Клонирование репозитория

3. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

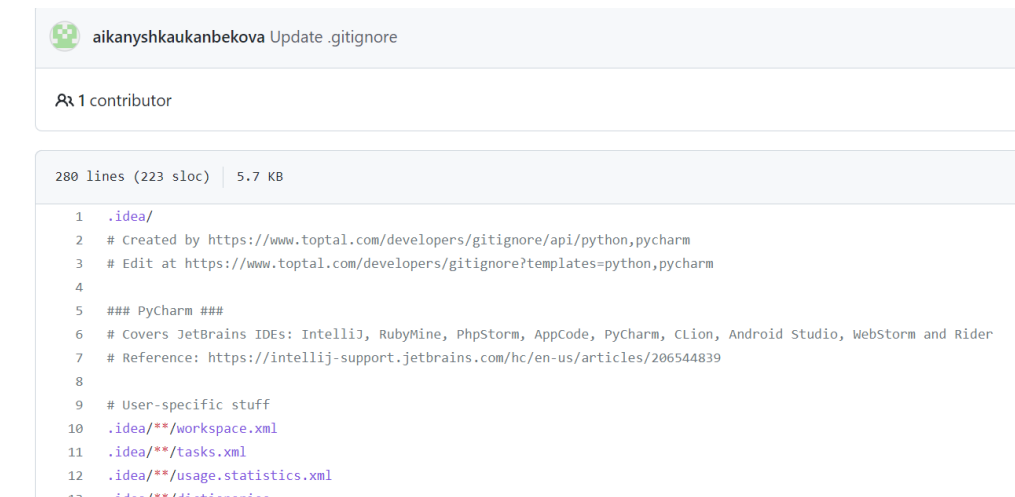


Рисунок 3. Дополнение файла .gitignore

Практическая часть:

Индивидуальное задание:

Выполнить индивидуальное задание 2 лабораторной работы 2.19, добавив аннотации типов.

Код программы:

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

import argparse
import json
import os.path
import pathlib
import sys

def add_route(routes: list[dict[str, object]], start: str, finish: str, number:
int) -> list[dict[str, object]]:
    """
    Добавить данные о маршруте
    """

    routes.append(
        {
            'start': start,
            'finish': finish,
            'number': number
        }
    )
    return routes
```

```

def display_route(routes: list[dict[str, object]]) -> None:
    """
    Отобразить список маршрутов
    """
    if routes:
        line = '+-{}--{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Начальный пункт",
                "Конечный пункт",
                "Номер маршрута"
            )
        )
        print(line)

        # Вывести данные о всех рейсах.
        for idx, worker in enumerate(routes, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('start', ''),
                    worker.get('finish', ''),
                    worker.get('number', 0)
                )
            )
            print(line)
    else:
        print("Список маршрутов пуст.")

def select_route(routes: list[dict[str, object]], period: int) -> list[dict[str, object]]:
    """
    Выбрать маршрут
    """
    result = []
    for employee in routes:
        if employee.get('number') == period:
            result.append(employee)

    return result

def save_routes(file_name: str, routes: list[dict[str, object]]) -> None:

```

```

"""
Сохранить всех работников в файл JSON.
"""

with open(file_name, "w", encoding="utf-8") as fout:
    json.dump(routes, fout, ensure_ascii=False, indent=4)
    directory = pathlib.Path.cwd().joinpath(file_name)
    directory.replace(pathlib.Path.home().joinpath(file_name))

def load_routes(file_name: str) -> list[dict[str, object]]:
    """
    Загрузить всех работников из файла JSON.
    """

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

def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "-d",
        "--data",
        action="store",
        required=False,
        help="The data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("routes")
    parser.add_argument(
        "--version",
        action="version",
        version=f"%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new route"
    )
    add.add_argument(
        "-s",
        "--start",
        action="store",
        required=True,
        help="The start of the route"
    )
    add.add_argument(
        "-f",
        "--finish",

```

```

        action="store",
        help="The finish of the route"
    )
add.add_argument(
    "-n",
    "--number",
    action="store",
    type=int,
    required=True,
    help="The number of the route"
)
# Создать субпарсер для отображения всех маршрутов.
_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all routes"
)
# Создать субпарсер для выбора маршрута.
select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the route"
)
select.add_argument(
    "-N",
    "--numb",
    action="store",
    type=int,
    required=True,
    help="The route"
)
# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)
# Загрузить все маршруты из файла, если файл существует.
data_file = args.data
if not data_file:
    data_file = os.environ.get("WORKERS_DATA")
if not data_file:
    print("The data file name is absent", file=sys.stderr)
    sys.exit(1)
# Загрузить всех работников из файла, если файл существует.
is_dirty = False
if os.path.exists(data_file):
    routes = load_routes(data_file)
else:
    routes = []
# Добавить маршрут.
if args.command == "add":
    routes = add_route(
        routes,
        args.start,

```

```

        args.finish,
        args.number
    )
    is_dirty = True
    # Отобразить все маршруты.
    elif args.command == "display":
        display_route(routes)
    # Выбрать требуемые маршруты.
    elif args.command == "select":
        selected = select_route(routes, args.numb)
        display_route(selected)
    # Сохранить данные в файл, если список маршрутов был изменен.
    if is_dirty:
        save_routes(data_file, routes)

if __name__ == '__main__':
    main()

```

Выполнить проверку программы с помощью утилиты муру.

```

PS C:\Users\User\Documents\3,1 курс\Объектно-ориентированное\лаб 5> python.exe -m муру .\indiv.py
Success: no issues found in 1 source file
PS C:\Users\User\Documents\3,1 курс\Объектно-ориентированное\лаб 5>

```

Рисунок 4. Проверка программы

Контрольные вопросы:

1. Для чего нужны аннотации типов в языке Python?

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:

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

2. Поддержка инструментов статического анализа: Аннотации типов могут использоваться инструментами статического анализа кода, такими как Муру, Руре или Pyright, чтобы проверять соответствие типов данных во время компиляции или анализа кода.

3. 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.

4. 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.