

**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

Классы данных в Python

Цель работы: приобретение навыков по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.

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 *

 aikanyshkaukanbekova

Repository name *

OOP-lab6

✓ OOP-lab6 is available.

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

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:

☐ README.md

Рисунок 1. Создание репозитория

2. Выполните клонирование созданного репозитория.

```
C:\Users\User>cd C:\Users\User\Documents\3,1 курс\объектно-ориентированное\лаб 6
C:\Users\User\Documents\3,1 курс\объектно-ориентированное\лаб 6>git clone https://github.com/aikanyshkaukanbekova/OOP-lab6.git
Cloning into 'OOP-lab6'...
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 6>
```

Рисунок 2. Клонирование репозитория

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

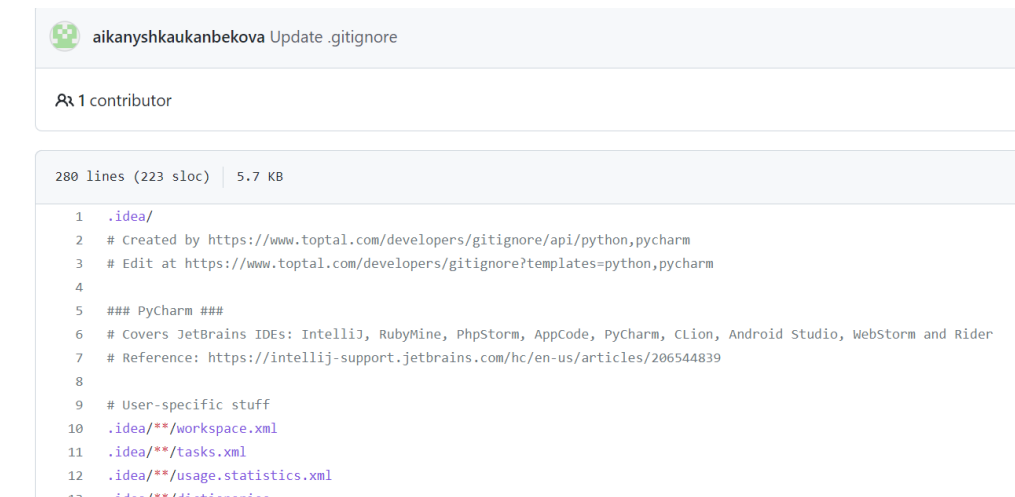


Рисунок 3. Дополнение файла .gitignore

Практическая часть:

Индивидуальное задание:

Выполнить индивидуальное задание лабораторной работы 4.5, используя классы данных, а также загрузку и сохранение данных в формат XML.

Код программы:

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

import argparse
import os.path
import sys
import xml.etree.ElementTree as ET
from dataclasses import dataclass

@dataclass(frozen=True)
class Route:
    start: str
    finish: str
    number: int

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

```

        routes.append(
            Route(
                start=start,
                finish=finish,
                number=number
            )
        )
    return routes

def display_route(routes: list[Route]) -> 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, route in enumerate(routes, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    route.start,
                    route.finish,
                    route.number
                )
            )
        print(line)
    else:
        print("Список маршрутов пуст.")

def select_route(routes: list[Route], period: int) -> list[Route]:
    """
    Выбрать маршрут
    """

```

```

result = []
for employee in routes:
    if employee.number == period:
        result.append(employee)

return result

def save_routes(file_name: str, routes: list[Route]) -> None:
    """
    Сохранить всех работников в файл JSON.
    """
    root = ET.Element('routes')
    for route in routes:
        element = ET.Element('route')

        ET.SubElement(element, 'start', text=route.start)
        ET.SubElement(element, 'finish', text=route.finish)
        ET.SubElement(element, 'number', text=str(route.number))

        root.append(element)

    tree = ET.ElementTree(root)
    with open(file_name, "wb") as fout:
        tree.write(fout, encoding='utf8', xml_declaration=True)

def load_routes(file_name: str) -> list[Route]:
    """
    Загрузить всех работников из файла JSON.
    """
    with open(file_name, "r", encoding="utf-8") as fin:
        xml = fin.read()

        parser = ET.XMLParser(encoding="utf8")
        tree = ET.fromstring(xml, parser=parser)

        routes = []
        for element in tree.findall("route"):
            start = element.find("start").attrib.get("text")
            finish = element.find("finish").attrib.get("text")
            number = element.find("number").attrib.get("text")

            route = Route(start, finish, int(number))
            routes.append(route)

        return routes

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="% (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()

```

Контрольные вопросы:

1. Как создать класс данных в языке Python?

В Python создание класса данных осуществляется с использованием ключевого слова `class`. Вот пример простого класса данных:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# Создание экземпляра класса
person1 = Person("Иван", 25)

# Доступ к атрибутам экземпляра класса
print(person1.name) # Выведет: Иван
print(person1.age)  # Выведет: 25
```

В этом примере мы создаем класс `Person`, который имеет атрибуты `name` и `age`. Метод `__init__` является конструктором класса и используется для инициализации атрибутов при создании экземпляра класса. При создании экземпляра класса `Person` мы передаем значения для атрибутов `name` и `age`.

Доступ к атрибутам экземпляра класса осуществляется с использованием точки (например, `person1.name`).

Это только простейший пример класса данных. В Python классы могут содержать методы (функции, связанные с классом), наследование, статические методы, свойства и многое другое.

2. Какие методы по умолчанию реализует класс данных?

В Python класс данных может реализовывать несколько встроенных методов по умолчанию, которые позволяют определить специальное поведение объекта. Некоторые из этих методов включают:

1. `__init__(self, ...)`: Конструктор класса, который вызывается при создании нового экземпляра класса.

2. `__str__(self)`: Метод, который возвращает строковое представление объекта. Он вызывается, когда объект передается функции `str()` или когда объект используется в строковом контексте.

3. `__repr__(self)`: Метод, который возвращает представление объекта, которое может быть использовано для его воссоздания. Он вызывается, когда объект передается функции `repr()` или когда объект используется в интерактивной оболочке Python.

4. `__eq__(self, other)`: Метод для сравнения объектов на равенство (используется оператор `==`).

5. `__lt__(self, other)`, `__le__(self, other)`, `__gt__(self, other)`, `__ge__(self, other)`: Методы для сравнения объектов (используются операторы `<`, `<=`, `>`, `>=`).

6. `__hash__(self)`: Метод для вычисления хэш-значения объекта, используемого в словарях и множествах.

7. `__getattr__(self, name)`, `__setattr__(self, name, value)`: Методы для перехвата доступа к атрибутам объекта.

8. `__del__(self)`: Метод, который вызывается при удалении объекта.

Это только небольшой набор методов по умолчанию, которые могут быть реализованы в классе данных. В Python есть еще много других "магических" методов, которые позволяют определить специальное поведение объектов.

3. Как создать неизменяемый класс данных?

В Python неизменяемый класс данных можно создать, используя неизменяемые типы данных в качестве атрибутов класса, и предоставляя только методы для чтения значений атрибутов, но не для их изменения. Вот пример создания неизменяемого класса данных:

```
class ImmutableData:
    def __init__(self, value1, value2):
```

```
self._value1 = value1 # Префикс "_" обозначает "приватный"
атрибут
```

```
self._value2 = value2
```

```
def get_value1(self):
    return self._value1
```

```
def get_value2(self):
    return self._value2
```

В этом примере атрибуты `value1` и `value2` являются приватными (по соглашению обозначены префиксом `_`), и доступ к ним осуществляется только через методы `get_value1` и `get_value2`. Таким образом, значения атрибутов не могут быть изменены напрямую извне.

Пример использования:

```
data = ImmutableData(10, 20)
print(data.get_value1()) # Выведет: 10
print(data.get_value2()) # Выведет: 20
```

```
# Попытка изменить значение атрибута вызовет ошибку
```

```
data._value1 = 100 # AttributeError: can't set attribute
```

Этот подход позволяет создать неизменяемый класс данных, в котором значения атрибутов не могут быть изменены после создания экземпляра класса.