

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 2.16  
по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-22-1  
Душин Александр Владимирович.

Подпись студента \_\_\_\_\_

Работа защищена « » \_\_\_\_\_ 20\_\_ г.

Проверил Воронкин Р.А. \_\_\_\_\_  
(подпись)

Ставрополь 2024

Тема: Работа с данными формата JSON в языке Python.

Цель работы: приобретение навыков по работе с данными формата JSON с помощью языка программирования Python версии 3.x.

Ход выполнения работы:

1. Создать общедоступный репозиторий на GitHub с использованием лицензии 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 \*

 MrPlatynum ▾

Repository name \*

ProgrammEngineering19

✓ ProgrammEngineering19 is available.

Great repository names are short and memorable. Need inspiration? How about **crispy-adventure** ?

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 – Создание общедоступного репозитория на GitHub с заданными настройками

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents
$ git clone https://github.com/MrPlatynum/ProgrammEngineering19.git
Cloning into 'ProgrammEngineering19'...
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.
```

Рисунок 2 – Клонирование созданного репозитория на локальный компьютер

```
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/
```

Рисунок 3 – файл .gitignore

```
Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering19 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering19 (develop)
$ |
```

Рисунок 4 – организация репозитория в соответствии с моделью ветвления git flow

2. Проработайте примеры лабораторной работы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import json
import sys
from datetime import date

def get_worker():
    """
    Запросить данные о работнике.
    """
```

```

name = input("Фамилия и инициалы? ")
post = input("Должность? ")
year = int(input("Год поступления? "))
# Создать словарь.
return {
    'name': name,
    'post': post,
    'year': year,
}

```

```

def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)
        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )
            print(line)
    else:
        print("Список работников пуст.")

```

```

def select_workers(staff, period):

```

```

"""
Выбрать работников с заданным стажем.
"""
# Получить текущую дату.
today = date.today()
# Сформировать список работников.
result = []
for employee in staff:
    if today.year - employee.get('year', today.year) >= period:
        result.append(employee)

# Возвратить список выбранных работников.
return result

def save_workers(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кириллицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)

def load_workers(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)

def main():
    """
    Главная функция программы.
    """
    # Список работников.
    workers = []
    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()
        # Выполнить действие в соответствие с командой.
        if command == "exit":
            break
        elif command == "add":
            # Запросить данные о работнике.
            worker = get_worker()
            # Добавить словарь в список.

```

```

workers.append(worker)
# Отсортировать список в случае необходимости.
if len(workers) > 1:
    workers.sort(key=lambda item: item.get('name', ''))
elif command == "list":
    # Отобразить всех работников.
    display_workers(workers)
elif command.startswith("select "):
    # Разбить команду на части для выделения стажа.
    parts = command.split(maxsplit=1)
    # Получить требуемый стаж.
    period = int(parts[1])
    # Выбрать работников с заданным стажем.
    selected = select_workers(workers, period)
    # Отобразить выбранных работников.
    display_workers(selected)
elif command.startswith("save "):
    # Разбить команду на части для выделения имени файла.
    parts = command.split(maxsplit=1)
    # Получить имя файла.
    file_name = parts[1]

    # Сохранить данные в файл с заданным именем.
    save_workers(file_name, workers)

elif command.startswith("load "):
    # Разбить команду на части для выделения имени файла.
    parts = command.split(maxsplit=1)
    # Получить имя файла.
    file_name = parts[1]

    # Сохранить данные в файл с заданным именем.
    workers = load_workers(file_name)
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("help - отобразить справку;")
    print("load - загрузить данные из файла;")
    print("save - сохранить данные в файл;")
    print("exit - завершить работу с программой.")
else:
    print(f"Неизвестная команда {command}", file=sys.stderr)

if __name__ == '__main__':
    main()

```

```

"C:\Program Files\Python312\python.exe" "C:/Users/Alexander/Desktop/Универ/4 сем
>>> add
Фамилия и инициалы? Petrov A.V.
Должность? Worker
Год поступления? 2008
>>> add
Фамилия и инициалы? Philinov
Должность? 2017
Год поступления? 2018
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Petrov A.V.              | Worker              |      2008     |
|  2 | Philinov                 | 2017                |      2018     |
+-----+-----+-----+-----+

>>> save workers.json
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Petrov A.V.              | Worker              |      2008     |
|  2 | Philinov                 | 2017                |      2018     |
+-----+-----+-----+-----+

```

Рисунок 5 – Проработанные команды

```

workers.json ×
C: > Users > Alexander > Desktop > Универ > 4 семестр > Основы програ
1  [
2      {
3          "name": "Petrov A.V.",
4          "post": "Worker",
5          "year": 2008
6      },
7      {
8          "name": "Philinov",
9          "post": "2017",
10         "year": 2018
11     }
12 ]

```

Рисунок 6 – Файл workers.json

### 3. Выполнить индивидуальные задания (вариант 7):

Задание 1. Для своего варианта лабораторной работы 2.8 необходимо дополнительно реализовать сохранение и чтение данных из файла формата

JSON. Необходимо также проследить за тем, чтобы файлы генерируемый этой программой не попадали в репозиторий лабораторной работы.

Листинг кода:

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

import json

def add_train(trains):
    """Добавляет информацию о поезде."""
    destination = input("Название пункта назначения: ")
    train_number = input("Номер поезда: ")
    departure_time = input("Время отправления (в формате ЧЧ:ММ): ")

    train = {
        'название пункта назначения': destination,
        'номер поезда': train_number,
        'время отправления': departure_time,
    }

    trains.append(train)
    trains.sort(key=lambda x: x['название пункта назначения'])

def list_trains(trains):
    """Выводит список всех поездов."""
    line = f'+-{"-" * 35}-+-{"-" * 15}-+-{"-" * 25}-+'
    print(line)
    print(f"| {'Название пункта назначения':^35} | {'Номер поезда':^15} | {'Время отправления':^25} |")

    for train in trains:
        print(line)
        print(
            f"| {train['название пункта назначения':^35]} | {train['номер поезда':^15]} | {train['время отправления':^25]} |")
        print(line)

def select_trains(trains, search_time):
    """Выводит поезда, отправляющиеся после указанного времени."""
    found = False
    result = []

    print(f"Поезда, отправляющиеся после {search_time}:")
    for train in trains:
        train_time = train['время отправления']
        if train_time >= search_time:
```



```

        result.append(train)
        found = True
    if found:
        return result

    if not found:
        return "Нет поездов, отправляющихся после указанного времени."

def display_help():
    """Выводит справку о доступных командах."""
    print("Список команд:\n")
    print("add - добавить информацию о поезде;")
    print("list - вывести список всех поездов;")
    print("select <время> - вывести поезда, отправляющиеся после указанного времени;")
    print("save <file_name> - сохранить информацию о поездах в файл JSON;")
    print("load <file_name> - загрузить информацию о поездах из файла JSON;")
    print("exit - завершить работу с программой.")

def save_trains(file_name, trains):
    """
    Сохранить информацию о поездах в файл JSON.
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кириллицы установим ensure_ascii=False
        json.dump(trains, fout, ensure_ascii=False, indent=4)
        print(f'Данные о поездах сохранены в файл {file_name}')

def load_trains(file_name):
    """
    Загрузить информацию о поездах из файла JSON.
    """
    with open(file_name, "r", encoding="utf-8") as fin:
        # Выполнить десериализацию данных из формата JSON.
        return json.load(fin)

def main():
    """Основная функция управления программой."""
    trains = []

    while True:
        command = input(">>> ").lower().split()

        if not command:
            continue

        if command[0] == 'exit':
            break

```

```
elif command[0] == 'add':
    add_train(trains)
elif command[0] == 'list':
    list_trains(trains)
elif command[0] == 'select':
    if len(command) != 2:
        print("Некорректное количество аргументов для команды 'select'.")
        continue
    selected = select_trains(trains, command[1])
    list_trains(selected)
elif command[0] == 'save':
    if len(command) != 2:
        print("Некорректное количество аргументов для команды 'save'.")
        continue
    save_trains(command[1], trains)
elif command[0] == 'load':
    if len(command) != 2:
        print("Некорректное количество аргументов для команды 'load'.")
        continue
    trains = load_trains(command[1])
elif command[0] == 'help':
    display_help()
else:
    print(f"Неизвестная команда {command[0]}")

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

```

"C:\Program Files\Python312\python.exe" "C:/Users/Alexander/Desktop/Универ/4 семестр/Основы
>>> add
Название пункта назначения: Moscow
Номер поезда: 123
Время отправления (в формате ЧЧ:ММ): 12:30
>>> add
Название пункта назначения: Stavropol
Номер поезда: 34
Время отправления (в формате ЧЧ:ММ): 14:30
>>> list
+-----+-----+-----+
| Название пункта назначения | Номер поезда | Время отправления |
+-----+-----+-----+
| Moscow                     | 123          | 12:30              |
+-----+-----+-----+
| Stavropol                  | 34           | 14:30              |
+-----+-----+-----+
>>> select 13:00
Поезда, отправляющиеся после 13:00:
+-----+-----+-----+
| Название пункта назначения | Номер поезда | Время отправления |
+-----+-----+-----+
| Stavropol                  | 34           | 14:30              |
+-----+-----+-----+
>>> save trains.json
Данные о поездах сохранены в файл trains.json
>>> load trains.json
>>>

```

Рисунок 7 – Выполнение команд

```

{} trains.json X
C: > Users > Alexander > Desktop > Универ > 4 семестр > Основы программной
1  [
2      {
3          "название пункта назначения": "Moscow",
4          "номер поезда": "123",
5          "время отправления": "12:30"
6      },
7      {
8          "название пункта назначения": "Stavropol",
9          "номер поезда": "34",
10         "время отправления": "14:30"
11     }
12 ]

```

Рисунок 8 – Файл trains.json

```

.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
# JetBrains specific template is maintained in a separate JetBrains.gitignore that can
# be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
# and can be added to the global gitignore or merged into this file. For a more nuclear
# option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/
*.json

```

Рисунок 9 – Изменённый файл .gitignore

Задание повышенной сложности. В индивидуальном задании никак не проверяет правильность загружаемых данных формата JSON. В следствие чего, необходимо после загрузки из файла JSON выполнять валидацию загруженных данных. Валидацию данных необходимо производить с использованием спецификации JSON Schema, описанной на сайте <https://jsonschema.org/>. Одним из возможных вариантов работы с JSON Schema является использование пакета jsonschema, который не является частью стандартной библиотеки Python. Таким образом, необходимо реализовать валидацию загруженных данных с помощью спецификации JSON Schema.

Листинг кода:

```

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

import json
from jsonschema import validate

train_schema = {
    "type": "object",
    "properties": {
        "название пункта назначения": {"type": "string"},
        "номер поезда": {"type": "string"},
        "время отправления": {"type": "string", "pattern": "^\\d{2}:\\d{2}$"}
    },
    "required": ["название пункта назначения", "номер поезда", "время отправления"]
}

```

```

def add_train(trains):
    """Добавляет информацию о поезде."""
    destination = input("Название пункта назначения: ")
    train_number = input("Номер поезда: ")
    departure_time = input("Время отправления (в формате ЧЧ:ММ): ")

    train = {
        'название пункта назначения': destination,
        'номер поезда': train_number,
        'время отправления': departure_time,
    }

    trains.append(train)
    trains.sort(key=lambda x: x['название пункта назначения'])

def list_trains(trains):
    """Выводит список всех поездов."""
    line = f'+-{"-" * 35}-+-{"-" * 15}-+-{"-" * 25}-+'
    print(line)
    print(f'| {"Название пункта назначения":^35} | {"Номер поезда":^15} | {"Время отправления":^25} |')

    for train in trains:
        print(line)
        print(
            f'| {train['название пункта назначения']:^35} | {train['номер поезда']:^15} | {train['время отправления']:^25} |')
        print(line)

def select_trains(trains, search_time):
    """Выводит поезда, отправляющиеся после указанного времени."""
    found = False
    result = []

    print(f"Поезда, отправляющиеся после {search_time}:")
    for train in trains:
        train_time = train['время отправления']
        if train_time >= search_time:
            result.append(train)
            found = True
    if found:
        return result

    if not found:
        return "Нет поездов, отправляющихся после указанного времени."

def display_help():
    """Выводит справку о доступных командах."""
    print("Список команд:\n")

```

```
print("add - добавить информацию о поезде;")
print("list - вывести список всех поездов;")
print("select <время> - вывести поезда, отправляющиеся после указанного времени;")
print("save <file_name> - сохранить информацию о поездах в файл JSON;")
print("load <file_name> - загрузить информацию о поездах из файла JSON;")
print("exit - завершить работу с программой.")
```

```
def save_trains(file_name, trains):
```

```
    """
```

```
    Сохранить информацию о поездах в файл JSON.
```

```
    """
```

```
    with open(file_name, "w", encoding="utf-8") as fout:
```

```
        # Выполнить сериализацию данных в формат JSON.
```

```
        # Для поддержки кириллицы установим ensure_ascii=False
```

```
        json.dump(trains, fout, ensure_ascii=False, indent=4)
```

```
        print(f"Данные о поездах сохранены в файл {file_name}")
```

```
def load_trains(file_name):
```

```
    """
```

```
    Загрузить информацию о поездах из файла JSON и выполнить валидацию данных.
```

```
    """
```

```
    try:
```

```
        # Открыть файл с заданным именем для чтения.
```

```
        with open(file_name, "r", encoding="utf-8") as fin:
```

```
            # Выполнить десериализацию данных из формата JSON.
```

```
            loaded_trains = json.load(fin)
```

```
            # Выполнить валидацию загруженных данных с использованием JSON Schema.
```

```
            for train in loaded_trains:
```

```
                validate(instance=train, schema=train_schema)
```

```
            return loaded_trains
```

```
    except FileNotFoundError:
```

```
        print(f"Файл {file_name} не найден.")
```

```
    return []
```

```
def main():
```

```
    """Основная функция управления программой."""
```

```
    trains = []
```

```
    while True:
```

```
        command = input(">>> ").lower().split()
```

```
        if not command:
```

```
            continue
```

```
        if command[0] == 'exit':
```

```
            break
```

```
        elif command[0] == 'add':
```

```
            add_train(trains)
```

```
        elif command[0] == 'list':
```

```

list_trains(trains)
elif command[0] == 'select':
    if len(command) != 2:
        print("Некорректное количество аргументов для команды 'select'.")
        continue
    selected = select_trains(trains, command[1])
    list_trains(selected)
elif command[0] == 'save':
    if len(command) != 2:
        print("Некорректное количество аргументов для команды 'save'.")
        continue
    save_trains(command[1], trains)
elif command[0] == 'load':
    if len(command) != 2:
        print("Некорректное количество аргументов для команды 'load'.")
        continue
    trains = load_trains(command[1])
elif command[0] == 'help':
    display_help()
else:
    print(f"Неизвестная команда {command[0]}")

if __name__ == '__main__':
    main()

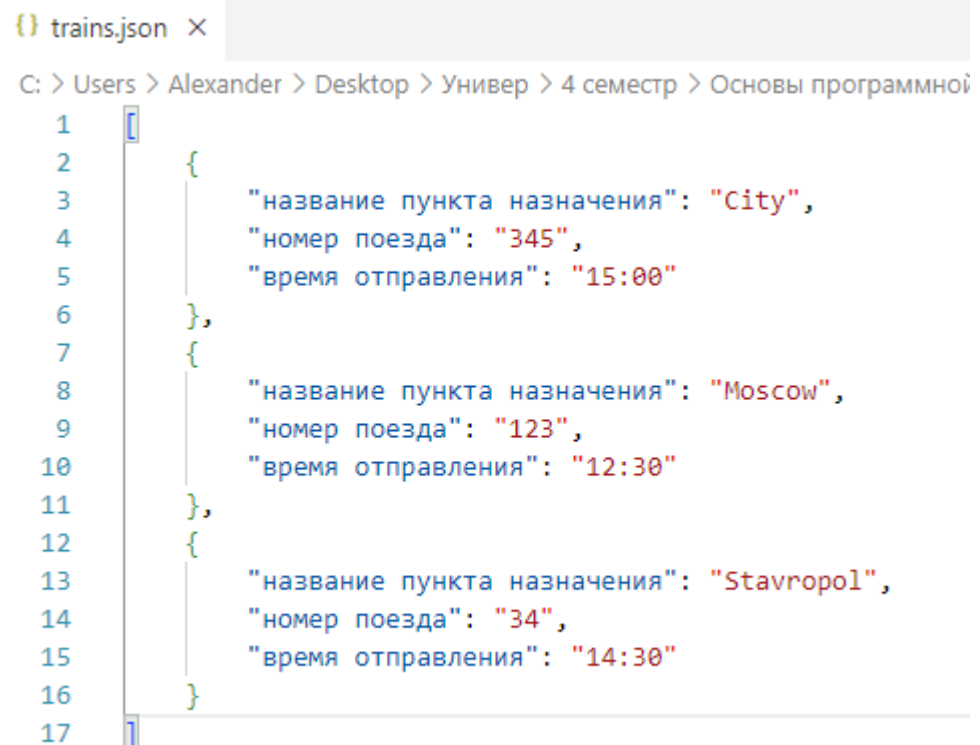
```

```

"C:\Program Files\Python312\python.exe" "C:/Users/Alexander/Desktop/Универ/4 семестр/Оср
>>> load trains.json
>>> list
+-----+-----+-----+
| Название пункта назначения | Номер поезда | Время отправления |
+-----+-----+-----+
| Moscow | 123 | 12:30 |
+-----+-----+-----+
| Stavropol | 34 | 14:30 |
+-----+-----+-----+
>>> select 14:00
Поезда, отправляющиеся после 14:00:
+-----+-----+-----+
| Название пункта назначения | Номер поезда | Время отправления |
+-----+-----+-----+
| Stavropol | 34 | 14:30 |
+-----+-----+-----+
>>> add
Название пункта назначения: City
Номер поезда: 345
Время отправления (в формате ЧЧ:ММ): 15:00
>>> save trains.json
Данные о поездах сохранены в файл trains.json

```

Рисунок 10 – Выполнение команд



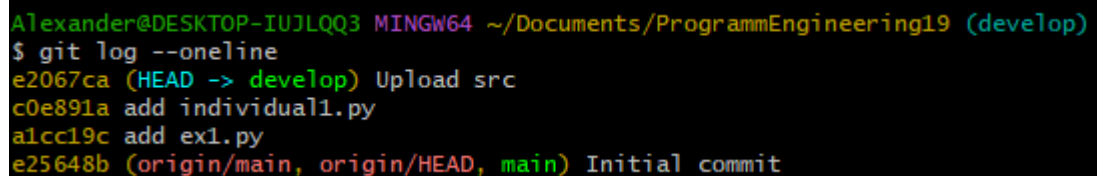
```

1  [
2      {
3          "название пункта назначения": "City",
4          "номер поезда": "345",
5          "время отправления": "15:00"
6      },
7      {
8          "название пункта назначения": "Moscow",
9          "номер поезда": "123",
10         "время отправления": "12:30"
11     },
12     {
13         "название пункта назначения": "Stavropol",
14         "номер поезда": "34",
15         "время отправления": "14:30"
16     }
17 ]

```

Рисунок 11 – Итоговый файл trains.json

4. Зафиксируем сделанные изменения, сольем ветки и отправим на удаленный репозиторий:



```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering19 (develop)
$ git log --oneline
e2067ca (HEAD -> develop) Upload src
c0e891a add individual1.py
a1cc19c add ex1.py
e25648b (origin/main, origin/HEAD, main) Initial commit

```

Рисунок 12 – Коммиты ветки develop во время выполнения лабораторной работы



```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering19 (develop)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering19 (main)
$ git merge develop
Updating e25648b..e2067ca
Fast-forward
 .gitignore      | 3 +-
 ex1.py          | 164 ++++++
 individual1.py  | 144 ++++++
 3 files changed, 310 insertions(+), 1 deletion(-)
 create mode 100644 ex1.py
 create mode 100644 individual1.py

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering19 (main)
$ |

```

Рисунок 13 – Слияние ветки develop в ветку main

```

Alexander@DESKTOP-IUJLQQ3 MINGW64 ~/Documents/ProgrammEngineering19 (main)
$ git push origin main
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 12 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 4.34 KiB | 4.34 MiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
To https://github.com/MrPlatinum/ProgrammEngineering19.git
 e25648b..e2067ca  main -> main

```

Рисунок 14 – Отправка на удаленный репозиторий

Ответы на контрольные вопросы:

1. Для чего используется JSON?

JSON (JavaScript Object Notation) используется для обмена данными между сервером и клиентом в веб-приложениях, для хранения и передачи структурированных данных в различных программах и сервисах, а также для конфигурации приложений.

2. Какие типы значений используются в JSON?

В JSON используются следующие типы значений:

Строки (например, "hello")

Числа (например, 42 или 3.14)

Логические значения (true или false)

Массивы (например, [1, 2, 3])

Объекты (например, {"key": "value"})

Значение null (например, null)

3. Как организована работа со сложными данными в JSON?

В JSON можно организовать работу со сложными данными, используя массивы и объекты. Массивы позволяют хранить упорядоченные списки данных, а объекты позволяют хранить пары ключ-значение.

4. Самостоятельно ознакомьтесь с форматом данных JSON5? В чем отличие этого формата от формата данных JSON?

Формат данных JSON5 - это расширение стандартного JSON, которое добавляет некоторые дополнительные возможности, такие как комментарии, необязательные запятые в конце списка элементов массива или объекта, а также одинарные кавычки для строк. Одно из основных отличий от стандартного JSON - это возможность добавления комментариев, что упрощает чтение и поддержку файлов с данными.

5. Какие средства языка программирования Python могут быть использованы для работы с данными в формате JSON5?

Для работы с данными в формате JSON5 на языке Python можно использовать сторонние библиотеки, такие как json5, которая предоставляет

функционал для чтения и записи данных в формате JSON5.

6. Какие средства предоставляет язык Python для сериализации данных в формате JSON?

В языке Python для сериализации данных в формат JSON используется модуль `json`. Он предоставляет функции для преобразования данных Python в формат JSON.

7. В чем отличие функций `json.dump()` и `json.dumps()`?

Основное отличие между `json.dump()` и `json.dumps()` заключается в том, что `json.dump()` записывает данные JSON в файл, а `json.dumps()` возвращает строку JSON.

8. Какие средства предоставляет язык Python для десериализации данных из формата JSON?

Для десериализации данных из формата JSON в языке Python также используется модуль `json`. Он предоставляет функции для преобразования данных JSON в данные Python.

9. Какие средства необходимо использовать для работы с данными формата JSON, содержащими кириллицу?

Для работы с данными формата JSON, содержащими кириллицу, в языке Python не требуется никаких дополнительных средств. Стандартная библиотека `json` поддерживает работу с данными в формате JSON, включая кириллические символы.

10. Самостоятельно ознакомьтесь со спецификацией JSON Schema? Что такое схема данных? Приведите схему данных для примера 1

JSON Schema - это спецификация, которая описывает формат данных JSON. С помощью JSON Schema можно определить структуру и типы данных в JSON-документе. Пример схемы данных для примера 1 (например, для объекта с полями "name", "age" и "email") может выглядеть так:

```
{  
  "type": "array",  
  "items": {
```

```
"type": "object",
"properties": {
  "name": {"type": "string"},
  "post": {"type": "string"},
  "year": {"type": "integer"}
},
"required": ["name", "post", "year"]
}
```