

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 №2.16
дисциплины «Основы программной инженерии»

Выполнил:
Соколов Михаил Романович
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Руководитель практики:
Богданов С.С., ассистент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

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, в котором будет использована лицензия MIT и .gitignore файл для языка 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 ().*

Repository template

No template

Start your repository with a template repository's contents.

Owner * PoTtaTto / **Repository name *** fundamentals_of_software

fundamentals_of_software_engineering_lab2_16 is available.

Great repository names are short and memorable. Need inspiration? How about [didactic-octo-spoon](#) ?

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](#).

① You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория с заданными настройками

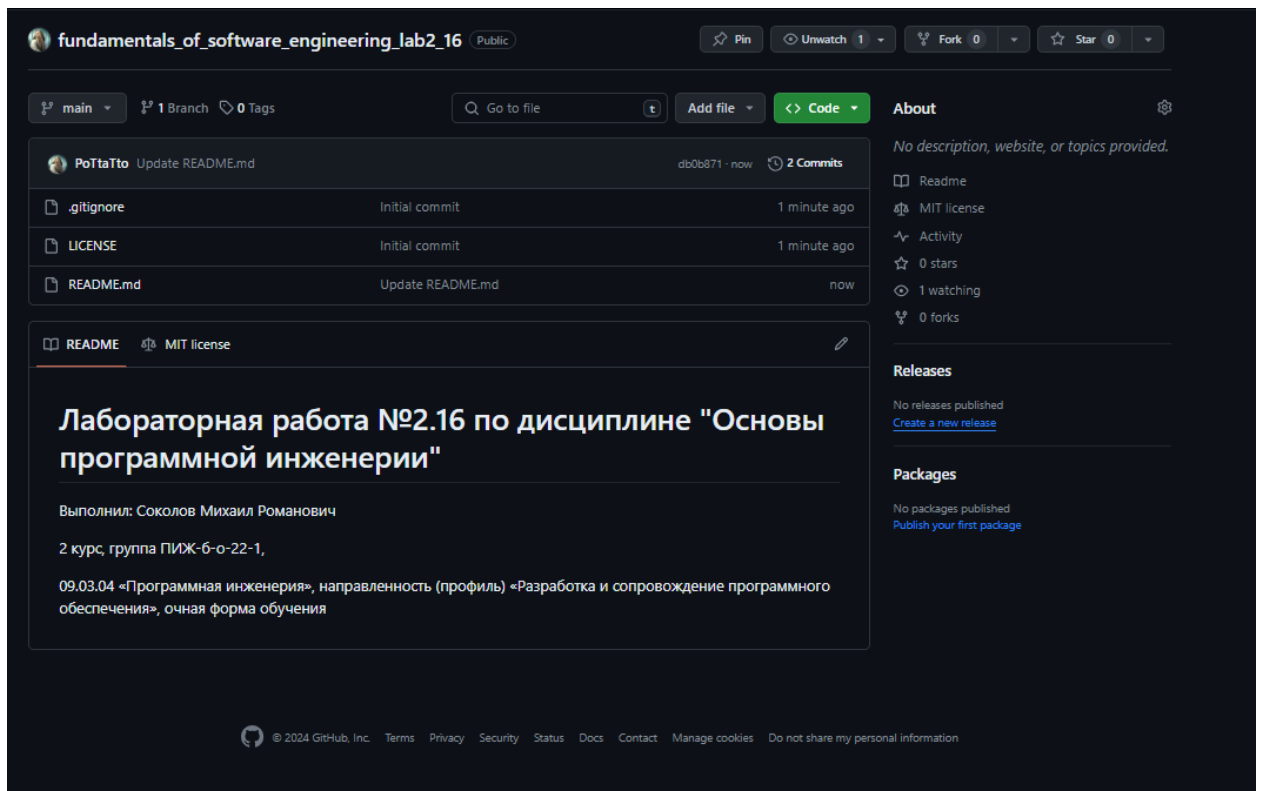


Рисунок 2 – Созданный репозиторий

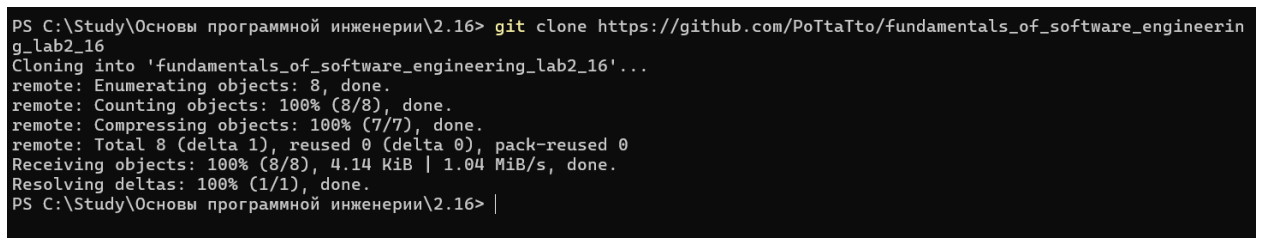


Рисунок 3 – Клонирование репозитория

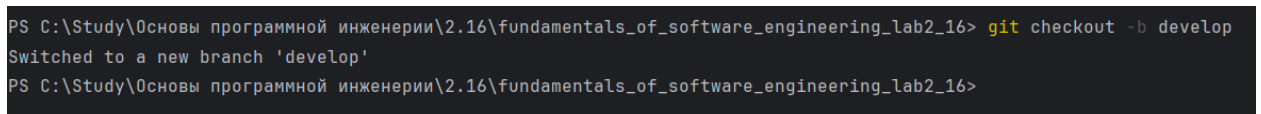


Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза

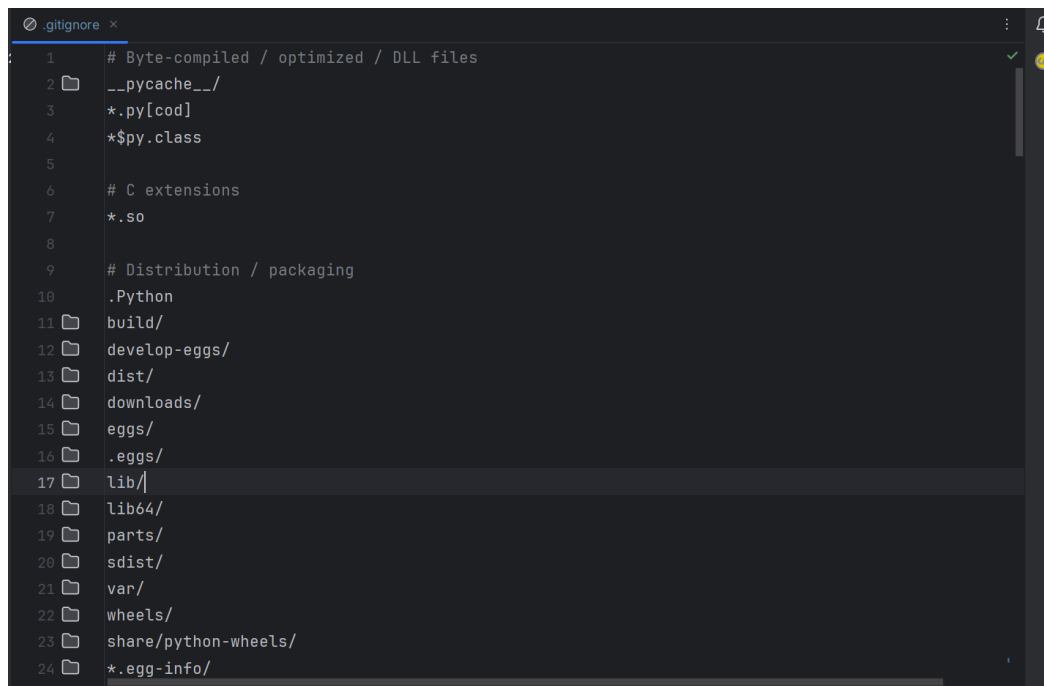


Рисунок 5 – Часть .gitignore, созданного GitHub

2. Проработаем пример лабораторной работы. Модифицируем пример №1 из лабораторной работы №2.8 и добавим возможность сохранения списка:

```
#!/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 соответствии 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()

```

Листинг 1 – Пример с новыми командами и методами (сохранение и выгрузка информации о работниках)

```

C:\Users\MrPot\AppData\Local\Programs\Python\Python312\python.exe "C:\Study\0
>>> add
Фамилия и инициалы? Соколов М.Р.
Должность? Студент
Год поступления? 2022
>>> add
Фамилия и инициалы? Юрьев И.Е.
Должность? Студент
Год поступления? 2022
>>> list
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |   Год   |
+-----+-----+-----+-----+
|  1 | Соколов М.Р.             | Студент              |  2022   |
|  2 | Юрьев И.Е.               | Студент              |  2022   |
+-----+-----+-----+-----+
>>> save students.json
>>>

```

Рисунок 6 – Добавление и сохранение информации о студентах в файле students.json

```

ex1.py  students.json x
1  [
2      {
3          "name": "Соколов М.Р.",
4          "post": "Студент",
5          "year": 2022
6      },
7      {
8          "name": "Юрьев И.Е.",
9          "post": "Студент",
10         "year": 2022
11     }
12 ]

```

Рисунок 7 – Полученный json файл

```
C:\Users\MrPot\AppData\Local\Programs\Python\Python312\python.exe "C:\Study\0сн
>>> list
Список работников пуст.
>>> load students.json
>>> list
+-----+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+-----+
| 1 | Соколов М.Р. | Студент | 2022 |
| 2 | Юрьев И.Е. | Студент | 2022 |
+-----+-----+-----+-----+-----+
>>>
```

Рисунок 8 – При перезапуске программы появляется возможность загрузить информацию из json файла

3. Выполним индивидуальные задания:

3.1. 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:

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

# Standard
import sys
from random import randint
import json

def add_train(trains):
    """
    Добавляет информацию о поезде в список trains.

    Args:
    - trains (list): Список поездов.

    """
    train_num = int(input('Введите номер поезда: '))
    destination = input('Введите пункт назначения: ')
    start_time = input('Введите время выезда: ')
    trains.append({'num': train_num, 'destination': destination,
'start_time': start_time})
    if len(trains) > 1:
        trains.sort(key=lambda item: item['start_time'])

def save_trains(file_name, trains):
```



```

"""
Сохраняет список поездов в файл в формате JSON.

Args:
- file_name (str): Имя файла.
- trains (list): Список поездов.

"""
with open(file_name, "w", encoding="utf-8") as fout:
    json.dump(trains, fout, ensure_ascii=False, indent=4)

def load_trains(file_name):
    """
    Загружает список поездов из файла в формате JSON.

    Args:
    - file_name (str): Имя файла.

    Returns:
    - trains (list): Список поездов.

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

def list_trains(trains):
    """
    Выводит список поездов на экран.

    Args:
    - trains (list): Список поездов.

    """
    line = f'+-{"-" * 15}+-{"-" * 30}+-{"-" * 25}+-'
    print(line)
    header = f"| {'№ поезда':^15} | {'Пункт назначения':^30} | {'Время отъезда':^25} |"
    print(header)
    print(line)
    for train in trains:
        num = train.get('num', randint(1000, 10000))
        destination = train.get('destination', 'None')
        start_time = train.get('start_time', 'None')
        recording = f"| {num:^15} | {destination:^30} | {start_time:^25} |"
        print(recording)
    print(line)

def select_train(trains, cmd_parts):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - cmd_parts (list): Список команды и параметра.

    """
    cmd_destination = cmd_parts[1]

```

```

        select_trains = [train for train in trains if
train['destination'].strip() == cmd_destination]
        if len(select_trains) >= 1:
            for train in select_trains:
                print(f'{train["num"]:^15}: {train["start_time"]:^25}')
        else:
            print('Нет поездов едущих в данное место!', file=sys.stderr)

def show_help():
    """
    Выводит список доступных команд на экран.

    """
    print("Список команд:\n")
    print("add - добавить поезд;")
    print("list - вывести список поездов;")
    print("select <пункт назначения> - запросить поезда с пунктом
назначения;")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")

if __name__ == '__main__':
    trains = []
    while True:
        cmd = input('>>> ')
        cmd_parts = cmd.split(maxsplit=1)
        match cmd_parts[0]:
            case 'add':
                add_train(trains)
            case 'list':
                list_trains(trains)
            case 'select':
                select_train(trains, cmd_parts)
            case 'help':
                show_help()
            case 'exit':
                break
            case 'save':
                if len(cmd_parts) == 2:
                    save_trains(cmd_parts[1], trains)
                else:
                    print("Использование: save <имя файла>",
file=sys.stderr)
            case 'load':
                if len(cmd_parts) == 2:
                    trains = load_trains(cmd_parts[1])
                else:
                    print("Использование: load <имя файла>",
file=sys.stderr)
            case _:
                print(f'Неизвестная команда {cmd}', file=sys.stderr)

```

Листинг 2 – Переделанный пример индивидуального задания лабораторной работы №2.8. Добавлены команды сохранения и загрузки данных из json файла

```
>>> add
Введите номер поезда: 1
Введите пункт назначения: Ставрополь
Введите время выезда: 14:00
>>> add
Введите номер поезда: 2
Введите пункт назначения: Москва
Введите время выезда: 16:00
>>> list
+-----+-----+-----+
|   № поезда   |   Пункт назначения   |   Время отъезда   |
+-----+-----+-----+
|         1     |   Ставрополь         |       14:00        |
|         2     |   Москва             |       16:00        |
+-----+-----+-----+
>>> save C:/Projects/trains.json
>>> |
```

Рисунок 9 – Добавление информации о поездах и сохранение в файл trains.json

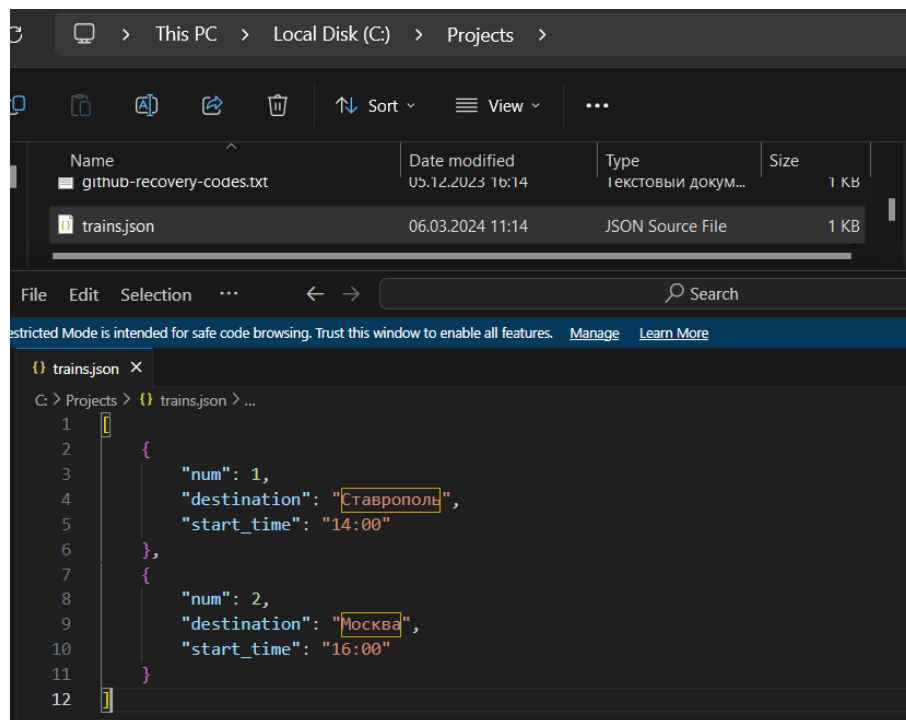


Рисунок 10 – Сохраненные данные в trains.json

```
>>> list
+-----+-----+-----+
|   № поезда   |   Пункт назначения   |   Время отъезда   |
+-----+-----+-----+
>>> load C:/Projects/trains.json
>>> list
+-----+-----+-----+
|   № поезда   |   Пункт назначения   |   Время отъезда   |
+-----+-----+-----+
|         1     |   Ставрополь        |        14:00       |
|         2     |   Москва             |        16:00       |
+-----+-----+-----+
>>>
```

Рисунок 11 – Загрузка информации о поездах из файла trains.json

3.2 Необходимо производить валидацию данных с использованием спецификации JSON Schema, описанной на сайте <https://json-schema.org/>:

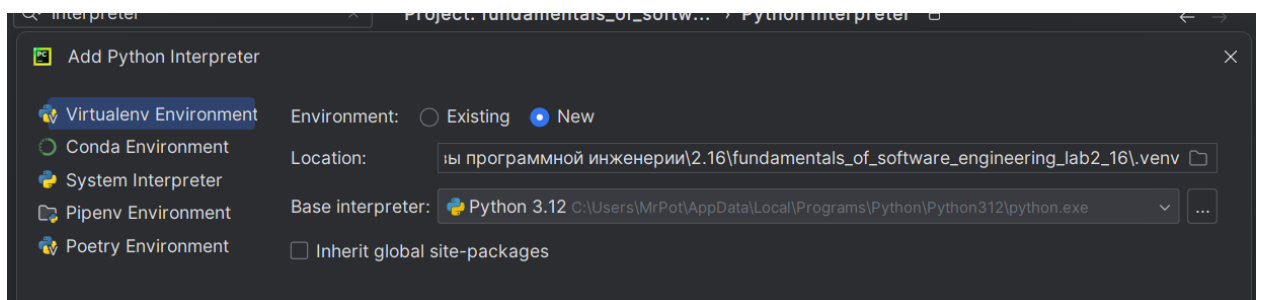


Рисунок 12 – Добавим виртуальное окружение

```
PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16> .\.venv\Scripts\activate
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16> pip install jsonschema
Collecting jsonschema
  Obtaining dependency information for jsonschema from https://files.pythonhosted.org/packages/39/9d/b035d024c62c85f2e2d4806a59ca7b8520307f34e0932fbc8cc75fe7b2d9/jsonschema-4.21.1-py3-none-any.whl.metadata
  Downloading jsonschema-4.21.1-py3-none-any.whl.metadata (7.8 kB)
Collecting attrs>=22.2.0 (from jsonschema)
  Obtaining dependency information for attrs>=22.2.0 from https://files.pythonhosted.org/packages/e0/44/827b2a91a5816512fcdf3cc4ebc465ccd5d598c45cefa6703fcf4a79018f/attrs-23.2.0-py3-none-any.whl.metadata
  Using cached attrs-23.2.0-py3-none-any.whl.metadata (9.5 kB)
Collecting jsonschema-specifications>=2023.03.6 (from jsonschema)
  Obtaining dependency information for jsonschema-specifications>=2023.03.6 from https://files.pythonhosted.org/packages/ee/07/44bd408781594c4d0a827666ef27fab1e441b109dc3b76b4f836f8fd04fe/jsonschema\_specifications-2023.12.1-py3-none-any.whl.metadata
```

Рисунок 13 – Установка пакета jsonschema

```
indiv_task1.py  scheme.json x
1  {
2      "$schema": "http://json-schema.org/draft-07/schema#",
3      "type": "array",
4      "items": {
5          "type": "object",
6          "properties": {
7              "num": {"type": "integer"},
8              "destination": {"type": "string"},
9              "start_time": {"type": "string", "format": "date-time"}
10         },
11         "required": ["num", "destination", "start_time"]
12     }
13 }
14
```

Рисунок 14 – Создание файла, определяющий структуру данных

```
def load_trains(file_name):
    """
    Загружает список поездов из файла в формате JSON.

    Args:
    - file_name (str): Имя файла.

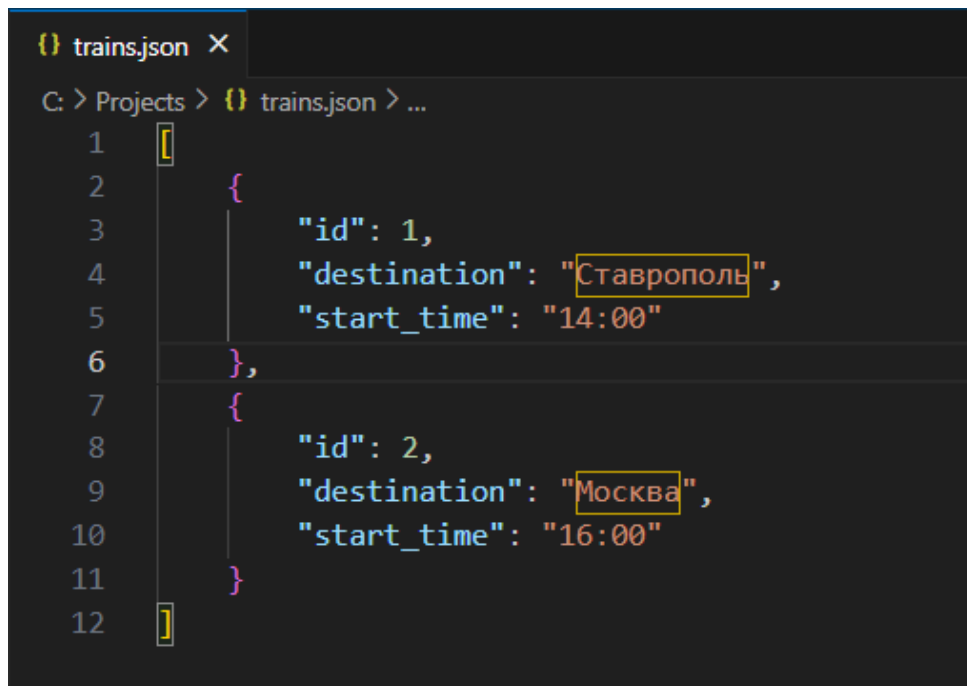
    Returns:
    - trains (list): Список поездов.

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

    with open('scheme.json', 'r', encoding='utf-8') as scheme_file:
        scheme = json.load(scheme_file)

    try:
        jsonschema.validate(loaded_data, scheme)
        return loaded_data
    except jsonschema.exceptions.ValidationError as e:
        print('Ошибка валидации данных:', e)
        return None
```

Рисунок 15 – Изменения в методе загрузки данных из файла



```
{
  "id": 1,
  "destination": "Ставрополь",
  "start_time": "14:00"
},
{
  "id": 2,
  "destination": "Москва",
  "start_time": "16:00"
}
```

Рисунок 16 – В качестве примера изменим поле “num” на “id” в train.json



```
>>> load C:/Projects/trains.json
Ошибка валидации данных: 'num' is a required property

Failed validating 'required' in schema['items']:
  {'properties': {'destination': {'type': 'string'},
                  'num': {'type': 'integer'},
                  'start_time': {'format': 'date-time',
                                'type': 'string'}},
   'required': ['num', 'destination', 'start_time'],
   'type': 'object'}

On instance[0]:
  {'destination': 'Ставрополь', 'id': 1, 'start_time': '14:00'}
>>>
```

Рисунок 17 – Попытка загрузки неправильных данных

4. Сольем ветки develop и main/master и отправим изменения на удаленный репозиторий:

```
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16> git log --oneline
bacd38b (HEAD -> develop) individual_task hard
7486676 indiv_task usual
6faa52c example 1 of lab
db0b871 (origin/main, origin/HEAD, main) Update README.md
fa69c7d Initial commit
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16>
```

Рисунок 18 – История коммитов

```
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16> git merge develop
Updating db0b871..bacd38b
Fast-forward
 .gitignore      |    2 +-
 ex1.py          | 162 +++++
 indiv_task1.py  | 148 +++++
 scheme.json     |   13 +
 students.json   |   12 +
 5 files changed, 336 insertions(+), 1 deletion(-)
 create mode 100644 ex1.py
 create mode 100644 indiv_task1.py
 create mode 100644 scheme.json
 create mode 100644 students.json
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16>
```

Рисунок 19 – Слияние веток main и develop

```
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16> git push origin main
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 12 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 4.51 KiB | 4.51 MiB/s, done.
Total 12 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/PoItaIto/fundamentals\_of\_software\_engineering\_lab2\_16
 db0b871..bacd38b  main -> main
(.venv) PS C:\Study\programming_eng\2.16\fundamentals_of_software_engineering_lab2_16>
```

Рисунок 20 – Отправка изменений на удаленный репозиторий

Ответы на контрольные вопросы:

1. Для чего используется JSON?

JSON (JavaScript Object Notation) используется для обмена данными между приложениями. Он предоставляет простой и удобный формат для хранения и передачи структурированных данных.

2. Какие типы значений используются в JSON?

В JSON могут использоваться следующие типы значений: строки, числа, булевы значения, массивы, объекты, null.

3. Как организована работа со сложными данными в JSON?

В JSON сложные данные организованы в виде вложенных объектов и массивов. Это позволяет представлять структурированные данные различной сложности, включая вложенные структуры и списки объектов.

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

JSON5 – это расширение JSON, которое добавляет некоторые дополнительные возможности, такие как поддержка комментариев, необязательные запятые в конце списков и ключей объектов, а также поддержка многострочных строк. Основное отличие от JSON заключается в том, что JSON5 более гибок и удобен для чтения и написания людьми.

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

Для работы с данными в формате JSON5 в Python можно использовать сторонние библиотеки, такие как `json5` или `json5-data`, которые позволяют работать с данными в формате JSON5 так же, как и с данными в формате JSON.

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

В Python для сериализации данных в формат JSON используется модуль `json`. Он предоставляет функцию `json.dump()` для записи данных в файл и функцию `json.dumps()` для преобразования данных в строку JSON.

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

`json.dump()` записывает данные в файл, в то время как `json.dumps()` возвращает строку JSON. Таким образом, `json.dump()` принимает два аргумента: данные и файловый объект для записи, в то время как `json.dumps()` принимает только один аргумент - данные, которые нужно преобразовать в JSON строку.

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

В Python для десериализации данных из формата JSON используется модуль `json`. Он предоставляет функцию `json.load()` для чтения данных из файла и функцию `json.loads()` для преобразования строки JSON в объект Python.

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

Для работы с данными формата JSON, содержащими кириллицу, в Python можно использовать параметр `ensure_ascii=False` при сериализации данных с помощью `json.dump()` или `json.dumps()`. Это позволяет сохранить кириллические символы в их оригинальном виде без преобразования в ASCII.

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

JSON Schema – это язык описания структуры данных в формате JSON. С помощью JSON Schema можно определить ожидаемую структуру данных, их типы и валидацию. Схема данных представляет собой формальное описание структуры данных, которое позволяет проверять данные на соответствие этой структуре. В примере 1 схема данных могла бы выглядеть так:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "name": {"type": "string"},
      "post": {"type": "string"},
      "year": {"type": "integer"}
    },
    "required": ["name", "post", "year"]
  }
}
```