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 №2.18
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 переменными окружения в Python3.

Цель работы: приобретение навыков по работе с переменными окружения 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:

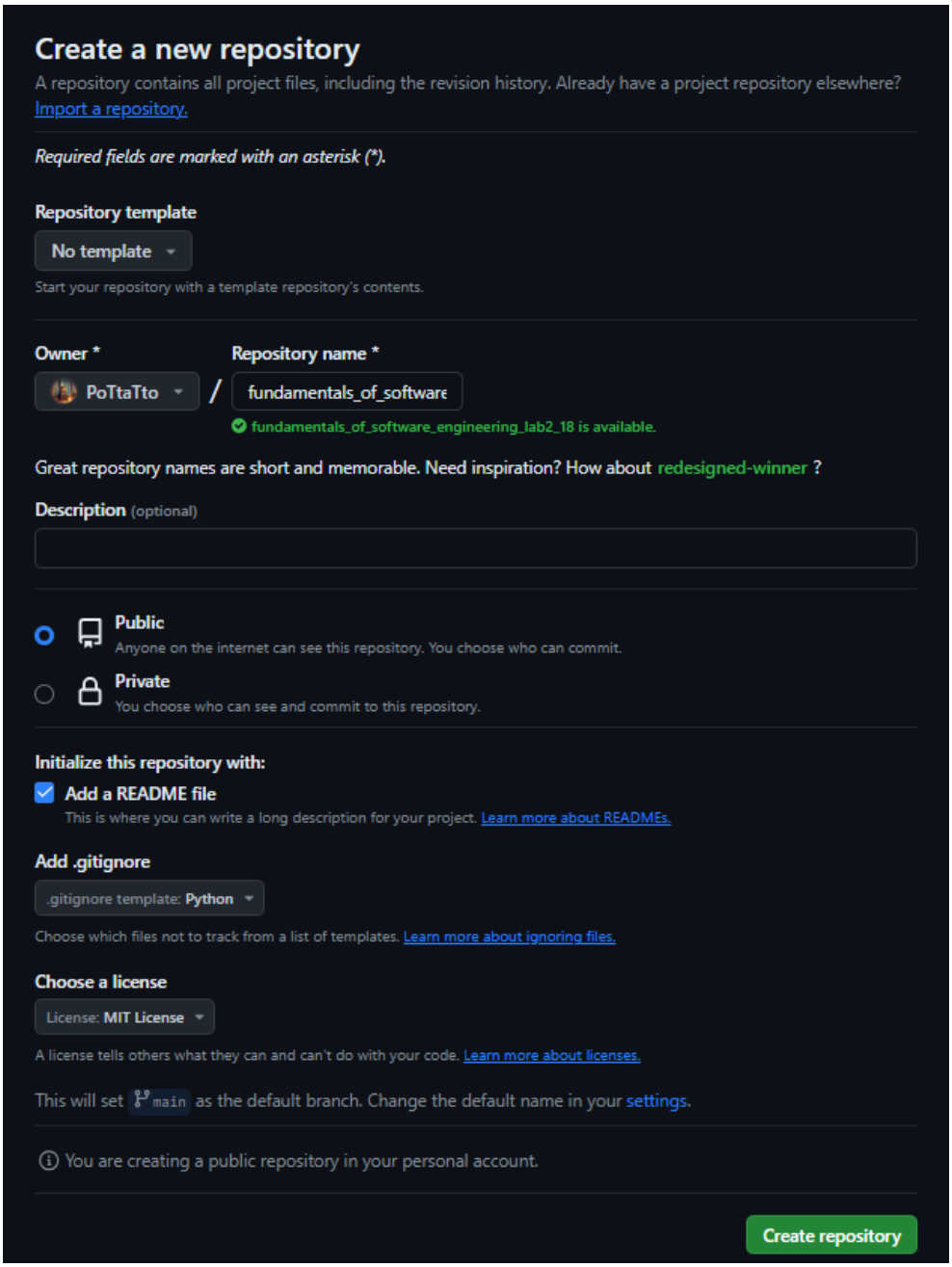

Рисунок 1 – Создание репозитория с заданными настройками

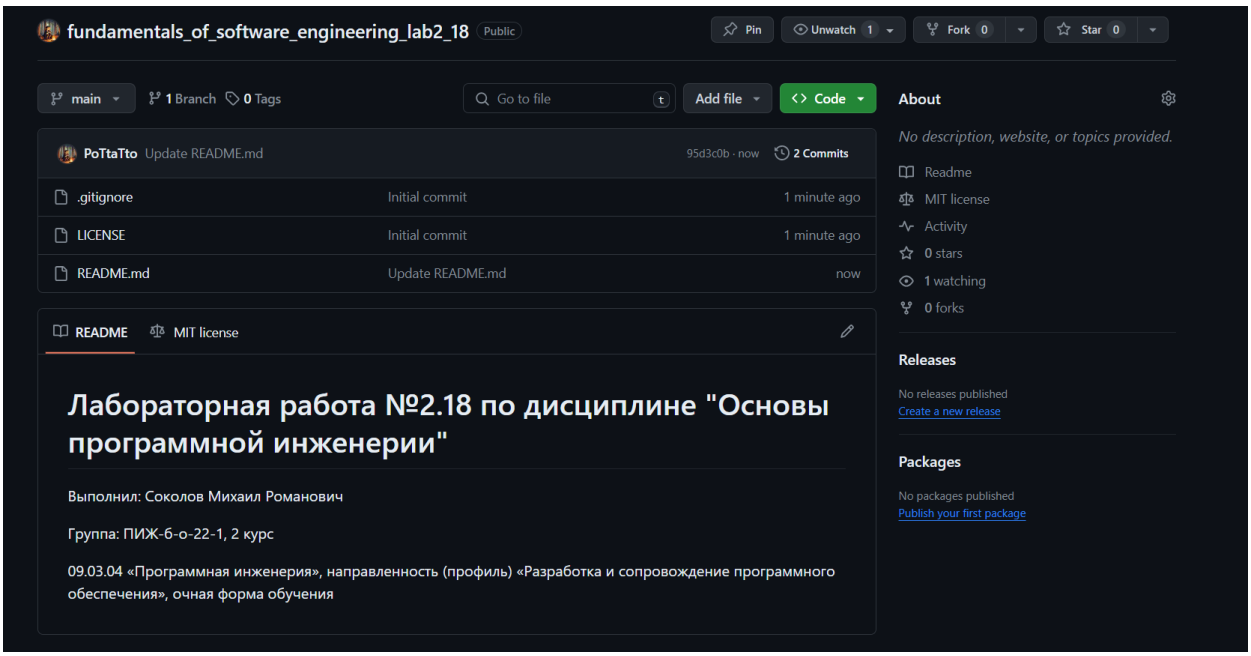

Рисунок 2 – Созданный репозиторий

```
PS C:\Study\programming_eng\2.18> git clone https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_18
Cloning into 'fundamentals_of_software_engineering_lab2_18'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 4.14 KiB | 848.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
PS C:\Study\programming_eng\2.18>
```

Рисунок 3 – Клонирование репозитория

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> git checkout -b develop
Switched to a new branch 'develop'
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18>
```

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

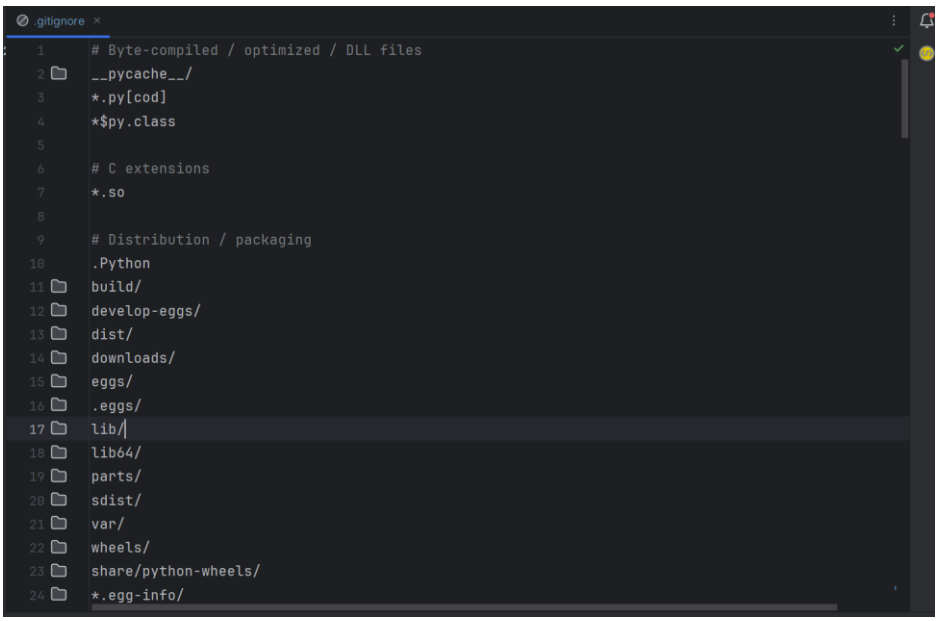

Рисунок 5 – Часть .gitignore, созданного GitHub

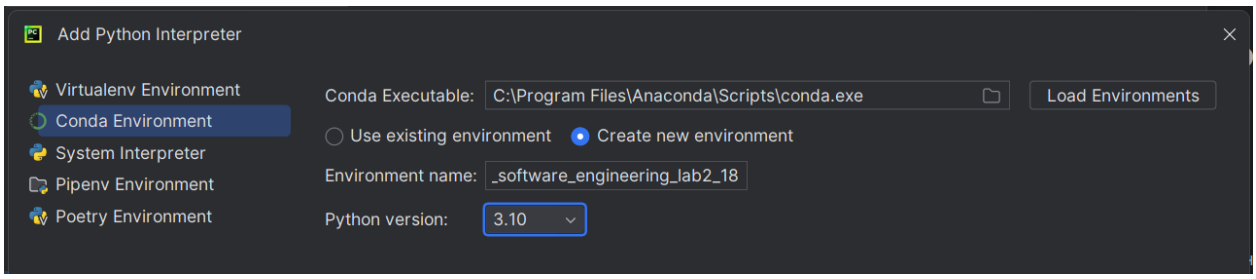

Рисунок 6 – Создание виртуального окружения с помощью Anaconda

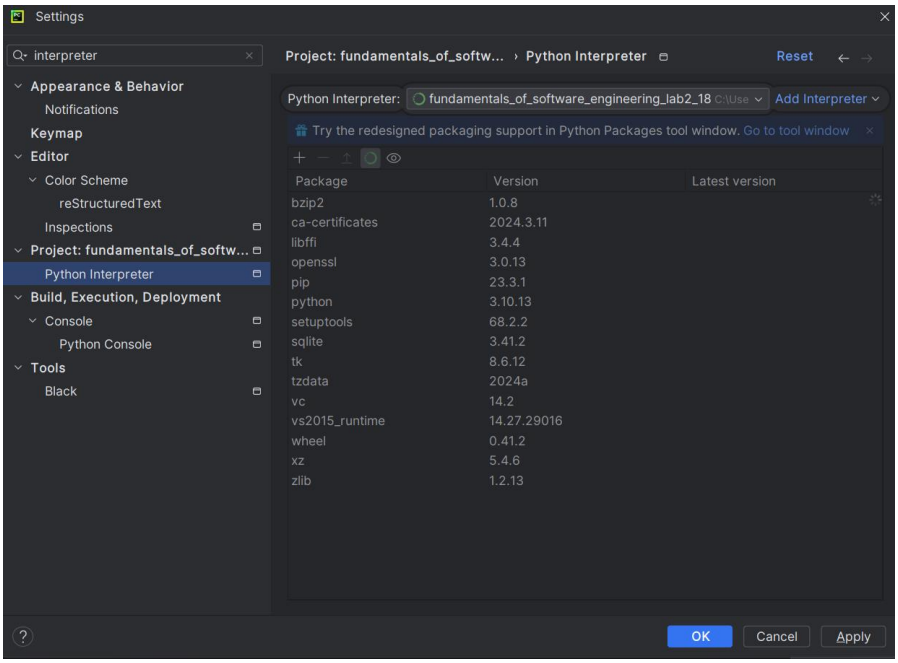

Рисунок 7 – Созданное окружение

2. Проработаем пример лабораторной работы. Для примера №1 лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответствующую переменную окружения:

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

import argparse
import json
import os
import sys
from datetime import date


def add_worker(staff, name, post, year):
    """
    Добавить данные о работнике.
    """
    staff.append(
        {
            "name": name,
            "post": post,
            "year": year
        }
    )
    return staff


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 кирилицы 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(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("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.1"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления работника.
    add = subparsers.add_parser(
        "add",
```

```
        parents=[file_parser],
        help="Add a new worker"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
    add.add_argument(
        "-p",
        "--post",
        action="store",
        help="The worker's post"
    )
    add.add_argument(
        "-y",
        "--year",
        action="store",
        type=int,
        required=True,
        help="The year of hiring"
    )

    # Создать субпарсер для отображения всех работников.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all workers"
    )

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-P",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )
    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Получить имя файла
    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):
        workers = load_workers(data_file)
    else:
        workers = []
```

```
    # Добавить работника.
    if args.command == "add":
        workers = add_worker(
            workers,
            args.name,
            args.post,
            args.year
        )
        is_dirty = True
    # Отобразить всех работников.
    elif args.command == "display":
        display_workers(workers)
        # Выбрать требуемых рааботников.
    elif args.command == "select":
        selected = select_workers(workers, args.period)
        display_workers(selected)
    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_workers(data_file, workers)


if __name__ == "__main__":
    os.environ.setdefault('WORKERS_DATA', 'data.json')
    main()
```

Листинг ex1.py

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> py ex1.py display
The data file name is absent
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18>
```

Рисунок 8 – Вывод программы при отсутствии переменной окружения WORKERS_DATA

```
if __name__ == "__main__":
    os.environ.setdefault( key: 'WORKERS_DATA', value: 'data.json')
    main()
```

Рисунок 9 – Установим значение переменной окружения WORKERS_DATA

ex1.py {} data.json

```
[
    {
        "name": "Соколов Михаил",
        "post": "Студент",
        "year": 2022
    }
]
```

Рисунок 10 – Содержимое data.json

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> py ex1.py display
+------+--------------------------------+----------------------+----------+
|  №   |             Ф.И.О.             |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Соколов Михаил                 | Студент              |     2022 |
+------+--------------------------------+----------------------+----------+
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18>
```

Рисунок 11 – Пример использования консольной команды display

3. Выполним индивидуальное задание. Для своего варианта лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответствующую переменную окружения:

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


# Standard
import os
import sys
import argparse
from random import randint
import json
import jsonschema


def add_train(trains, num, destination, start_time):
    """
    Добавляет информацию о поезде в список trains.

    Args:
    - trains (list): Список поездов.
    - num (int): Номер поезда.
    - destination (str): Пункт назначения.
    - start_time (str): Время отправки

    Returns:
    - trains (list): Список поездов.
    """
    trains.append({
        'num': num,
        'destination': destination,
        'start_time': start_time
    })
    if len(trains) > 1:
        trains.sort(key=lambda item: item['start_time'])

    return trains


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:
        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


def display_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_trains(trains, destination):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - destination (list): Пункт назначения.
```

```
    Returns:
    - trains (list): Список поездов.

    """

    return [train for train in trains if train['destination'].strip() ==
destination]


def main(command_line=None):
    # Creating file parser
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        '--data',
        action="store",
        required=False,
        help="The data file name"
    )

    # Main parser of command line
    parser = argparse.ArgumentParser("trains")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 1.0.0"
    )

    # Subparsers
    subparsers = parser.add_subparsers(dest="command")

    # Subparser for add command
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new train"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        required=True,
        type=int,
        help="The number of a train"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination point"
    )
    add.add_argument(
        "-st",
        "--start_time",
        action="store",
        required=True,
        help="Depart time"
    )

    # Subparser for display command
    display = subparsers.add_parser(
        "display",
        parents=[file_parser],
```

```
        help="Display all trains"
    )

    # Subparser for select command
    select = subparsers.add_parser(
        'select',
        parents=[file_parser],
        help='Select trains by destination'
    )

    select.add_argument(
        "-D",
        "--dest",
        action="store",
        required=True,
        help="The required destination"
    )

    args = parser.parse_args(command_line)

    # Получить имя файла
    data_file = args.data
    if not data_file:
        data_file = os.environ.get('TRAINS_DATA')
    if not data_file:
        print('The data file name is absent', file=sys.stderr)
        sys.exit(1)

    is_dirty = False
    if os.path.exists(data_file):
        trains = load_trains(data_file)
    else:
        trains = []

    match args.command:
        case 'add':
            trains = add_train(
                trains,
                args.number,
                args.destination,
                args.start_time
            )
            is_dirty = True
        case 'display':
            display_trains(trains)
        case 'select':
            selected = select_trains(trains, args.dest)
            display_trains(selected)

    # Save changes in file if data is changed
    if is_dirty:
        save_trains(data_file, trains)


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

Листинг task1.py – Добавление чтения файла данных из переменной окружения

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> py task1.py display
The data file name is absent
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18>
```

Рисунок 12 – Отображение при неустановленной переменной окружения TRAINS_DATA

```
if __name__ == '__main__':
    os.environ.setdefault( key: 'TRAINS_DATA',  value: 'trains.json')
    main()
```

Рисунок 13 – Установка значения переменной окружения

```
[
    {
        "num": 1,
        "destination": "Ставрополь",
        "start_time": "17:00"
    },
    {
        "num": 2,
        "destination": "Москва",
        "start_time": "18:40"
    }
]
```

Рисунок 14 – Содержимое файла trains.json

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> py task1.py display
+-----------------+--------------------------------+---------------------------+
|    № поезда     |        Пункт назначения        |       Время отъезда       |
+-----------------+--------------------------------+---------------------------+
|        1        |           Ставрополь           |           17:00           |
|        2        |             Москва             |           18:40           |
+-----------------+--------------------------------+---------------------------+
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18>
```

Рисунок 15 – Использование консольной команды display

4. Выполним задание повышенной сложности. Самостоятельно изучить работу с пакетом python-dotenv. Модифицировать программу задания №1

таким образом, чтобы значения необходимых переменных окружения считывались из файла .env:

python-dotenv – это библиотека для работы с переменными среды из файлов .env в приложениях на Python. Она позволяет загружать переменные окружения из файла .env в корне проекта и автоматически добавлять их в переменные среды Python при запуске приложения. Это особенно удобно для хранения конфиденциальной информации, такой как ключи API или пароли, которые не должны быть включены в исходный код приложения. Библиотека предоставляет простой и удобный способ управления конфигурацией приложения и облегчает работу с переменными окружения.

```
(fundamentals_of_software_engineering_lab2_18) PS C:\Users\MrPot> pip install python-dotenv
Collecting python-dotenv
  Downloading python_dotenv-1.0.1-py3-none-any.whl.metadata (23 kB)
Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.0.1
(fundamentals_of_software_engineering_lab2_18) PS C:\Users\MrPot>
```

Рисунок 16 – Установка пакета python-dotenv

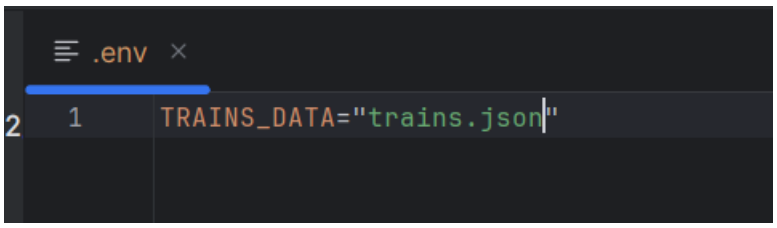

Рисунок 17 – Создание файла. env и запись переменной окружения

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


# Standard
import os
import argparse
from random import randint
import json
import jsonschema

# Third-party
from dotenv import load_dotenv


def add_train(trains, num, destination, start_time):
    """
    Добавляет информацию о поезде в список trains.
```

```
    Args:
    - trains (list): Список поездов.
    - num (int): Номер поезда.
    - destination (str): Пункт назначения.
    - start_time (str): Время отправки

    Returns:
    - trains (list): Список поездов.
    """
    trains.append({
        'num': num,
        'destination': destination,
        'start_time': start_time
    })
    if len(trains) > 1:
        trains.sort(key=lambda item: item['start_time'])

    return trains


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:
        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


def display_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_trains(trains, destination):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - destination (list): Пункт назначения.

    Returns:
    - trains (list): Список поездов.

    """

    return [train for train in trains if train['destination'].strip() ==
destination]


def main(command_line=None):
    # Creating file parser
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        '--data',
        action="store",
        required=False,
        help="The data file name"
    )

    # Main parser of command line
    parser = argparse.ArgumentParser("trains")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 1.0.0"
    )

    # Subparsers
    subparsers = parser.add_subparsers(dest="command")

    # Subparser for add command
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new train"
    )
```

```
    add.add_argument(
        "-n",
        "--number",
        action="store",
        required=True,
        type=int,
        help="The number of a train"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination point"
    )
    add.add_argument(
        "-st",
        "--start_time",
        action="store",
        required=True,
        help="Depart time"
    )

    # Subparser for display command
    display = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all trains"
    )

    # Subparser for select command
    select = subparsers.add_parser(
        'select',
        parents=[file_parser],
        help='Select trains by destination'
    )

    select.add_argument(
        "-D",
        "--dest",
        action="store",
        required=True,
        help="The required destination"
    )

    args = parser.parse_args(command_line)

    is_dirty = False
    data_file = os.getenv('TRAINS_DATA')
    if os.path.exists(data_file):
        trains = load_trains(data_file)
    else:
        trains = []

    match args.command:
        case 'add':
            trains = add_train(
                trains,
                args.number,
                args.destination,
                args.start_time
            )
            is_dirty = True
```

```
        case 'display':
            display_trains(trains)
        case 'select':
            selected = select_trains(trains, args.dest)
            display_trains(selected)

    # Save changes in file if data is changed
    if is_dirty:
        save_trains(data_file, trains)


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

Листинг task1.py – Загрузка переменных окружения из файла .env с помощью модуля python-dotenv

```
(fundamentals_of_software_engineering_lab2_18) PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab
2_18> python task1.py display
+-----------------+--------------------------------+---------------------------+
|    № поезда     |        Пункт назначения        |       Время отъезда       |
+-----------------+--------------------------------+---------------------------+
|        1        |           Ставрополь           |           17:00           |
|        2        |             Москва             |           18:40           |
+-----------------+--------------------------------+---------------------------+
(fundamentals_of_software_engineering_lab2_18) PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab
2_18>
```

Рисунок 18 – Вывод консольной команды display

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

Ссылка: https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_18

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> git log --oneline
ec175dc (HEAD -> develop) hard task is completed
b108cdb task 1 is completed
1c9665f example 1 is completed
95d3c0b (origin/main, origin/HEAD, main) Update README.md
ff9c537 Initial commit
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18>
```

Рисунок 19 – История коммитов

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> git merge develop
Updating 95d3c0b..ec175dc
Fast-forward
 .gitignore  |   2 +-
 data.json   |   7 ++
 ex1.py      | 206 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 scheme.json |  13 ++++
```

Рисунок 20 – Слияние веток main и develop

```
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18> git push origin main
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 12 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (17/17), 5.33 KiB | 5.33 MiB/s, done.
Total 17 (delta 6), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (6/6), completed with 1 local object.
To https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_18
   95d3c0b..68f523a  main -> main
PS C:\Study\programming_eng\2.18\fundamentals_of_software_engineering_lab2_18>
```

Рисунок 21 – Отправка изменений на удаленный репозиторий

Ответы на контрольные вопросы:

1. Каково назначение переменных окружения?

Переменные окружения используются для хранения конфигурационных данных и параметров, которые используются программами во время их выполнения. Они играют ключевую роль в настройке среды выполнения программного обеспечения и позволяют обеспечить гибкость приложений. Являются короткой ссылкой на какой-либо объект в системе.

2. Какая информация может храниться в переменных окружения?

В переменных окружения может храниться разнообразная информация, включая пути к исполняемым файлам, настройки локализации, конфигурационные параметры программ и секретные ключи. Они также могут содержать информацию о системных ресурсах, таких как пути к файлам и библиотекам.

3. Как получить доступ к переменным окружения в ОС Windows?

В операционной системе Windows доступ к переменным окружения осуществляется через специальный интерфейс управления системой, который позволяет просматривать, создавать и изменять переменные окружения. Для этого можно воспользоваться панелью управления или специальным окном настроек системы. Также переменные окружения могут быть установлены и изменены с помощью командной строки, используя команды типа set или setx.

4. Каково назначение переменных PATH и PATHEXT?

Переменная PATH в Windows содержит список директорий, в которых операционная система ищет исполняемые файлы при выполнении команд в командной строке. Переменная PATHEXT содержит список расширений файлов, которые интерпретируются как исполняемые файлы без указания их расширения в командной строке.

5. Как создать или изменить переменную окружения в Windows?

Для создания или изменения переменной окружения в Windows можно воспользоваться командой set, например: set MY_VARIABLE=value. Чтобы сделать переменную окружения постоянной, которая сохранится после

перезагрузки системы, можно использовать команду setx, например: setx MY_VARIABLE value. Или отредактировать переменные в контрольной панели.

6. Что представляют собой переменные окружения в ОС Linux?

Переменные окружения в операционной системе Linux представляют собой набор именованных строковых значений, которые используются программами во время их выполнения. Они определяют среду выполнения для пользовательских процессов и содержат информацию о различных аспектах системы, таких как пути к исполняемым файлам, настройки локализации и другие параметры.

7. В чем отличие переменных окружения от переменных оболочки?

Отличие между переменными окружения и переменными оболочки заключается в их области видимости. Переменные окружения доступны для всех процессов, запущенных в среде операционной системы, в то время как переменные оболочки доступны только для текущей оболочки или процесса, в котором они были определены.

8. Как вывести значение переменной окружения в Linux?

Для вывода значения переменной окружения в Linux можно использовать команду echo, например: echo $MY_VARIABLE. Или воспользоваться командой printenv.

9. Какие переменные окружения Linux Вам известны?

Некоторые известные переменные окружения в Linux включают PATH (список директорий для поиска исполняемых файлов), HOME (домашняя директория текущего пользователя), LANG (языковая настройка локали), TERM (тип терминала) и другие.

10. Какие переменные оболочки Linux Вам известны?

Некоторые известные переменные оболочки в Linux включают PS1 (приглашение командной строки), PS2 (вторичное приглашение), BASH_VERSION (версия оболочки Bash) и другие.

11. Как установить переменные оболочки в Linux?

Для установки переменных оболочки в Linux можно использовать команды экспорта, например: export MY_VARIABLE=value. Для постоянного сохранения переменных оболочки можно добавить их в файлы настройки оболочки, такие как ~/.bashrc или ~/.bash_profile. Или можно воспользоваться комндой set VARIABLE=value.

12. Как установить переменные окружения в Linux?

Для установки переменных окружения в Linux можно использовать команды экспорта, например: export MY_VARIABLE=value. Также переменные окружения могут быть установлены во время запуска сессии оболочки, путем добавления команд установки в файлы инициализации оболочки, такие как ~/.bashrc или ~/.bash_profile.

13. Для чего необходимо делать переменные окружения Linux постоянными?

Постоянные переменные окружения в Linux необходимы для сохранения пользовательских настроек и конфигураций между сеансами работы. Это особенно важно для переменных, используемых программами или скриптами, которые запускаются автоматически или в фоновом режиме, и требуют доступа к определенным ресурсам или настройкам при каждом запуске.

14. Для чего используется переменная окружения PYTHONHOME?

Переменная окружения PYTHONHOME используется для указания директории, в которой находится корневой каталог установки интерпретатора Python. Она может использоваться для настройки среды выполнения Python и указания местоположения различных библиотек и файлов, используемых интерпретатором.

15. Для чего используется переменная окружения PYTHONPATH?

Переменная окружения PYTHONPATH используется для указания путей поиска модулей Python. Она позволяет добавлять дополнительные директории, в которых интерпретатор Python будет искать модули при выполнении программ.

16. Какие еще переменные окружения используются для управления работой интерпретатора Python?

PYTHONSTARTUP: если переменная среды PYTHONSTARTUP это имя файла, то команды Python в этом файле выполняются до отображения первого приглашения в интерактивном режиме. Файл выполняется в том же пространстве имен, в котором выполняются интерактивные команды, так что определенные или импортированные в нем объекты можно использовать без квалификации в интерактивном сеансе. При запуске вызывает событие аудита cpython.run_startup с именем файла в качестве аргумента.

PYTHONOPTIMIZE: если в переменной среды PYTHONOPTIMIZE задана непустая строка, это эквивалентно указанию параметра -O. Если установлено целое число, то это эквивалентно указанию -OO.

PYTHONBREAKPOINT: если переменная среды PYTHONBREAKPOINT установлена, то она определяет вызываемый объект с помощью точечной нотации. Модуль, содержащий вызываемый объект, будет импортирован, а затем вызываемый объект будет запущен реализацией по умолчанию sys.breakpointhook() , которая сама вызывается встроенной функцией breakpoint() . Если PYTHONBREAKPOINT не задан или установлен в пустую строку, то это эквивалентно значению pdb.set_trace . Установка этого значения в строку 0 приводит к тому, что стандартная реализация sys.breakpointhook() ничего не делает, кроме немедленного возврата.

PYTHONDEBUG: если значение переменной среды PYTHONDEBUG непустая строка, то это эквивалентно указанию опции -d. Если установлено целое число, то это эквивалентно многократному указанию -dd.

PYTHONINSPECT: если значение переменной среды PYTHONINSPECT непустая строка, то это эквивалентно указанию параметра -i. Эта переменная также может быть изменена кодом Python с помощью os.environ для принудительного режима проверки при завершении программы.

С остальными переменными можно ознакомится в официальной документации.

17. Как осуществляется чтение переменных окружения в программах на языке программирования Python?

Для чтения переменных окружения в программах на языке программирования Python можно использовать модуль os. Например, для получения значения переменной окружения MY_VARIABLE можно использовать os.environ['MY_VARIABLE'].

18. Как проверить, установлено или нет значение переменной окружения в программах на языке программирования Python?

Для проверки установлено ли значение переменной окружения в программах на языке программирования Python можно использовать метод os.environ.get('MY_VARIABLE'). Если значение переменной существует, метод вернет его, в противном случае вернется None.

19. Как присвоить значение переменной окружения в программах на языке программирования Python?

Для присвоения значения переменной окружения в программах на языке программирования Python можно использовать модуль os. Например, для установки значения переменной окружения MY_VARIABLE можно использовать os.environ['MY_VARIABLE'] = 'value'. Либо воспользоваться методом os.environ.setdefault(‘MY_VARIABLE’, ‘value’).