

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 №4**  
**Дисциплины «Анализ данных»**

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и  
вычислительная техника (профиль)  
«Программное обеспечение средств  
вычислительной  
техники и автоматизированных  
систем», очная форма обучения

---

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических  
наук, доцент, доцент кафедры  
инфокоммуникаций

---

(подпись)

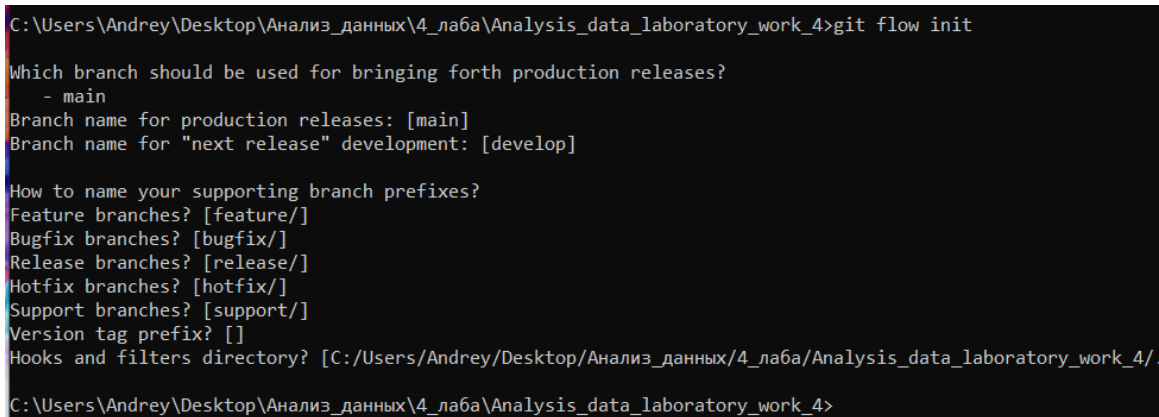
Ставрополь, 2024 г.

Тема: Работа с переменными окружениями в Python 3.

Цель: приобрести навыки по работе с переменными окружениями с помощью языка программирования Python версии 3.x.

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git-flow» (рис. 1).



```
C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4>git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/Andrey/Desktop/Анализ_данных/4_лаба/Analysis_data_laboratory_work_4/]

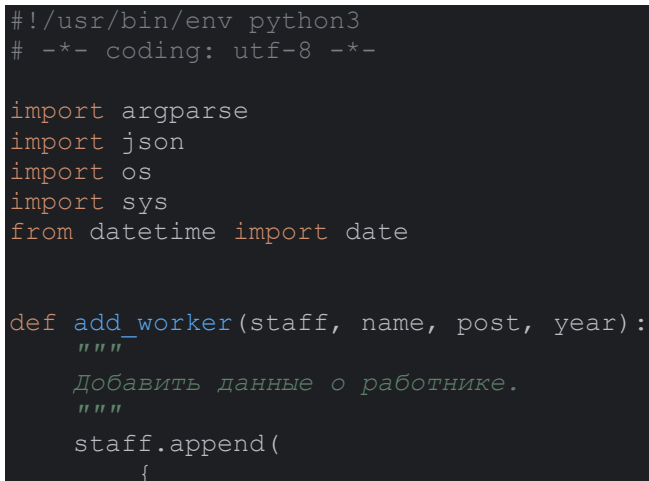
C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4>
```

Рисунок 1 – Организация модели ветвления «git-flow»

Проработка примеров лабораторной работы:

Пример 1.

Необходимо для примера лабораторной работы 2.17 добавить возможность получения имени файла данных, используя соответствующую переменную окружения. Переменная окружения должна иметь имя «WORKERS\_DATA». Код программы для решения данной задачи (рис. 2).



```
#!/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 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(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 0.1.0"
    )

    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(args.filename):
        workers = load_workers(args.filename)
    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 работников.
    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(args.filename, workers)

if __name__ == "__main__":
    main()

```

Рисунок 2 – Код программы примера №1

Создадим переменную окружения для данной программы (рис. 3).

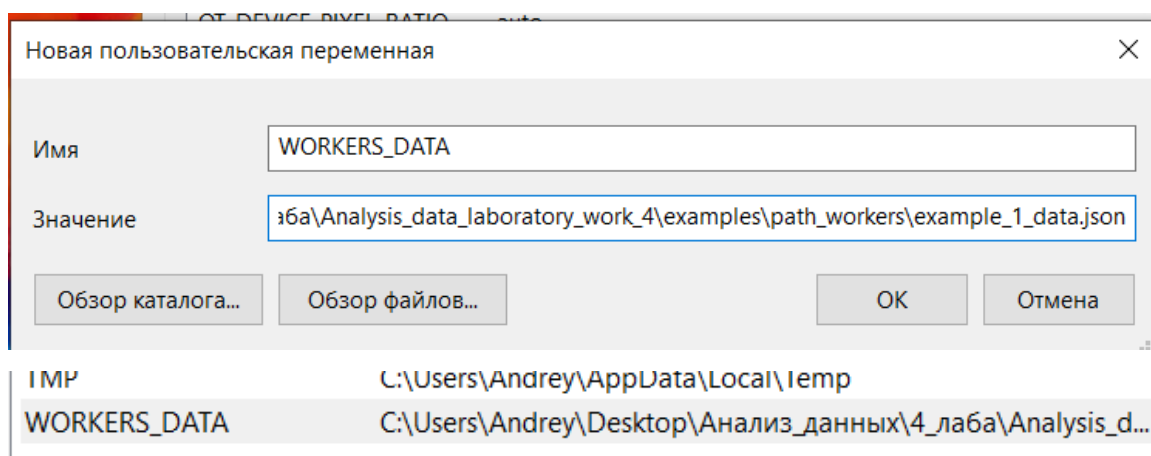


Рисунок 3 – Создание переменной окружения

Результаты работы данной программы без некоторых аргументов в командной строке (программа должна сама взять их по умолчанию из созданной переменной) (рис. 4).

```

C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4\examples>python example_1.py add --name="Иван Иванов" --post="Инженер" --year=2012

```

```

C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4\examples>python example_1.py display

```

№	Ф.И.О.	Должность	Год
1	Сидоров Сидор	Главный инженер	2012
2	Иван Иванов	Инженер	2012

Рисунок 4 – Результаты работы программы примера №1

Выполнение индивидуальных заданий:

Задание 1.

Необходимо для своего варианта лабораторной работы 2.17 дополнительно реализовать возможность получения имени файла данных,

используя соответствующую переменную окружения. Переменная окружения должна иметь имя «TRAINS\_DATA».

Необходимо использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение (Вариант 26 (7), работа 2.8). Код программы индивидуального задания (рис. 5).

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

import argparse
import json
import os.path
import sys

# Необходимо использовать словарь, содержащий следующие ключи: название
# пункта назначения;
# номер поезда; время отправления. Написать программу, выполняющую следующие
# действия: ввод
# с клавиатуры данных в список, состоящий из словарей заданной структуры;
# записи должны быть
# упорядочены по времени отправления поезда; вывод на экран информации о
# поездах, направляющихся
# в пункт, название которого введено с клавиатуры; если таких поездов нет,
# выдать на дисплей
# соответствующее сообщение (Вариант 26 (7), работа 2.8).

def add_train(trains, departure_point, number_train, time_departure,
              destination):
    """
    Добавить данные о поезде.
    """
    trains.append(
        {
            "departure_point": departure_point,
            "number_train": number_train,
            "time_departure": time_departure,
            "destination": destination
        }
    )

    return trains

def display_trains(trains):
```

```

"""
Отобразить список поездов со станциями.
"""
# Проверить, что список поездов не пуст.
if trains:
    # Заголовок таблицы.
    line = '+-{}-+-{}-+-{}-+-{}-+-+'.format(
        '-' * 4,
        '-' * 30,
        '-' * 13,
        '-' * 18,
        '-' * 14
    )
    print(line)
    print(
        '| {:^4} | {:^30} | {:^13} | {:^18} | {:^14} |'.format(
            "№",
            "Пункт отправления",
            "Номер поезда",
            "Время отправления",
            "Пункт назначения"
        )
    )
    print(line)

    # Вывести данные о всех поездах со станциями.
    for idx, train in enumerate(trains, 1):
        print(
            '| {:>4} | {:<30} | {:<13} | {:>18} | {:^16} |'.format(
                idx, train.get('departure_point', ''),
                train.get('number_train', ''),
                train.get('time_departure', ''),
                train.get('destination', '')
            )
        )
        print(line)

else:
    print("Список поездов пуст.")

def select_trains(trains, point_user):
    """
    Выбрать поезда по пункту назначения.
    """
    # Сформировать список поездов.
    result = []
    for train in trains:
        if point_user == str.lower(train['destination']):
            result.append(train)

    # Возвратить список выбранных поездов, направляющихся в пункт.
    return result

def save_trains(file_name, trains):
    """
    Сохранить все поезда со станциями 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(trains, fout, ensure_ascii=False, indent=4)

def load_trains(file_name):

```



```

"""
Загрузить все поезда со станциями 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("trains")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )

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

    # Создать субпарсер для добавления поезда.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new train"
    )
    add.add_argument(
        "-dep",
        "--departure_point",
        action="store",
        required=True,
        help="The train's departure point"
    )
    add.add_argument(
        "-n",
        "--number_train",
        action="store",
        required=True,
        help="The train's number"
    )
    add.add_argument(
        "-t",
        "--time_departure",
        action="store",
        required=True,
        help="The time departure of train"
    )
    add.add_argument(
        "-des",
        "--destination",
        action="store",
        required=True,
        help="The destination of train"
    )

    # Создать субпарсер для отображения всех поездов.

```

```

_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all trains"
)

# Создать субпарсер для выбора поездов по пунктам назначения.
select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the trains"
)
select.add_argument(
    "-p",
    "--point_user",
    action="store",
    required=True,
    help="The required point"
)

# 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("TRAINS_DATA")
if not data_file:
    print("The data file name is absent", file=sys.stderr)
    sys.exit(1)

# Загрузить все поезда со станциями из файла, если файл существует.
is_dirty = False
if os.path.exists(data_file):
    trains = load_trains(data_file)
else:
    trains = []

# Добавить поезд со станциями.
if args.command == "add":
    trains = add_train(
        trains,
        args.departure_point,
        args.number_train,
        args.time_departure,
        args.destination
    )
    is_dirty = True

# Отобразить все поезда со станциями.
elif args.command == "display":
    display_trains(trains)

# Выбрать требуемые поезда.
elif args.command == "select":
    selected = select_trains(trains, args.point_user)
    display_trains(selected)

# Сохранить данные в файл, если список поездов был изменен.
if is_dirty:
    save_trains(data_file, trains)

```

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

Рисунок 5 – Код программы индивидуального задания 1

Создание переменной окружения для данной программы (рис. 6).

Изменение пользовательской переменной ×

Имя	TRAINS_DATA
Значение	top\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4\individual\data.json

Рисунок 6 – Создание виртуальной среды для задания 1

Результаты работы программы с использованием команд без имени файла (имя и путь к файлу программа берет из переменной окружения) (рис. 7).

```
C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4\individual>python individual_1.py display
```

№	Пункт отправления	Номер поезда	Время отправления	Пункт назначения
1	Калининград Ю.	001	10:00	Калининград С.

```
C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4\individual>_
```

Рисунок 7 – Результаты работы программы индивидуального задания 1

## Задание 2.

Необходимо изучить работу с пакетом «python-dotenv» и модифицировать задание 1 таким образом, чтобы значения необходимых переменных окружения считывались из файла «.env».

Установка пакета в виртуальное окружение (рис. 8).

```
(lab_3) C:\Users\Andrey>conda install -c conda-forge python-dotenv
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 23.7.4
  latest version: 24.5.0
```

Рисунок 8 – Установка пакета «python-dotenv»

Код программы задания 2 (рис. 9).

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

import argparse
import json
import os.path
import sys
from dotenv import load_dotenv

# Необходимо использовать словарь, содержащий следующие ключи: название
# пункта назначения;
# номер поезда; время отправления. Написать программу, выполняющую следующие
# действия: ввод
# с клавиатуры данных в список, состоящий из словарей заданной структуры;
# записи должны быть
# упорядочены по времени отправления поезда; вывод на экран информации о
# поездах, направляющихся
# в пункт, название которого введено с клавиатуры; если таких поездов нет,
# выдать на дисплей
# соответствующее сообщение (Вариант 26 (7), работа 2.8).

def add_train(trains, departure_point, number_train, time_departure,
destination):
    """
    Добавить данные о поезде.
    """
    trains.append(
        {
            "departure_point": departure_point,
            "number_train": number_train,
            "time_departure": time_departure,
            "destination": destination
        }
    )

    return trains

def display_trains(trains):
    """
    Отобразить список поездов со станциями.
    """
    # Проверить, что список поездов не пуст.
    if trains:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+-+{}-+-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 13,
            '-' * 18,
            '-' * 14
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^13} | {:^18} | {:^14} |'.format(
                "№",
                "Пункт отправления",
                "Номер поезда",
                "Время отправления",
                "Пункт назначения"
            )
        )
    )
)

```

```

        print(line)

        # Вывести данные о всех поездах со станциями.
        for idx, train in enumerate(trains, 1):
            print(
                '| {:>4} | {:<30} | {:<13} | {:>18} | {:^16} |'.format(
                    idx, train.get('departure_point', ''),
                    train.get('number_train', ''),
                    train.get('time_departure', ''),
                    train.get('destination', '')
                )
            )
            print(line)

    else:
        print("Список поездов пуст.")

def select_trains(trains, point_user):
    """
    Выбрать поезда по пункту назначения.
    """
    # Сформировать список поездов.
    result = []
    for train in trains:
        if point_user == str.lower(train['destination']):
            result.append(train)

    # Возвратить список выбранных поездов, направляющихся в пункт.
    return result

def save_trains(file_name, trains):
    """
    Сохранить все поезда со станциями 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(trains, fout, ensure_ascii=False, indent=4)

def load_trains(file_name):
    """
    Загрузить все поезда со станциями 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):
    load_dotenv() # Загрузить переменные окружения из файла .env

    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("trains")
parser.add_argument(
    "--version",
    action="version",
    version="% (prog)s 0.1.0"
)

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

# Создать субпарсер для добавления поезда.
add = subparsers.add_parser(
    "add",
    parents=[file_parser],
    help="Add a new train"
)
add.add_argument(
    "-dep",
    "--departure_point",
    action="store",
    required=True,
    help="The train's departure point"
)
add.add_argument(
    "-n",
    "--number_train",
    action="store",
    required=True,
    help="The train's number"
)
add.add_argument(
    "-t",
    "--time_departure",
    action="store",
    required=True,
    help="The time departure of train"
)
add.add_argument(
    "-des",
    "--destination",
    action="store",
    required=True,
    help="The destination of train"
)

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

# Создать субпарсер для выбора поездов по пунктам назначения.
select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the trains"
)
select.add_argument(
    "-p",
    "--point_user",
    action="store",
    required=True,
    help="The required point"
)

```

```

)

# 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("TRAINS_DATA")
if not data_file:
    print("The data file name is absent", file=sys.stderr)
    sys.exit(1)

# Загрузить все поезда со станциями из файла, если файл существует.
is_dirty = False
if os.path.exists(data_file):
    trains = load_trains(data_file)
else:
    trains = []

# Добавить поезд со станциями.
if args.command == "add":
    trains = add_train(
        trains,
        args.departure_point,
        args.number_train,
        args.time_departure,
        args.destination
    )
    is_dirty = True

# Отобразить все поезда со станциями.
elif args.command == "display":
    display_trains(trains)

# Выбрать требуемые поезда.
elif args.command == "select":
    selected = select_trains(trains, args.point_user)
    display_trains(selected)

# Сохранить данные в файл, если список поездов был изменен.
if is_dirty:
    save_trains(data_file, trains)

if __name__ == "__main__":
    main()

```

Рисунок 9 – Код программы задания 2

Создание файла с информацией о переменных окружения (рис. 10).

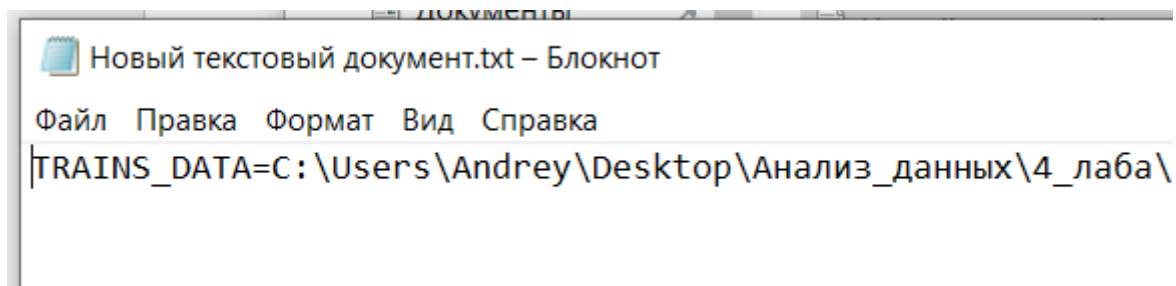


Рисунок 10 – Файл «.env»

Результаты работы программы задания 2 с использованием файла «.env» (рис. 11).

```
C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4\individual>python individual
+-----+-----+-----+-----+-----+
| № | Пункт отправления | Номер поезда | Время отправления | Пункт назначения |
+-----+-----+-----+-----+-----+
| 1 | Калининград Ю. | 001 | 10:00 | Калининград С. |
+-----+-----+-----+-----+-----+
C:\Users\Andrey\Desktop\Анализ_данных\4_лаба\Analysis_data_laboratory_work_4\individual>_
```

Рисунок 11 – Результаты работы программы задания 2

Ответы на контрольные вопросы:

1. Каково назначение переменных окружения?

Переменные окружения используются для хранения информации, доступной для всех процессов, запущенных в операционной системе. Они предоставляют программам и системе информацию о конфигурации, путях поиска файлов, доступе к ресурсам, языковых настройках и многом другом.

2. Какая информация может храниться в переменных окружения?

Пути к исполняемым файлам (например, переменная PATH). Конфигурационные настройки программ. Языковые параметры (например, LANG, LC\_ALL). Данные о временных директориях, пользователях и системе. Параметры, управляющие поведением операционной системы и программ.

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

Для получения доступа к переменным окружения в Windows можно использовать команду «echo %VARIABLE\_NAME%» в командной строке, где VARIABLE\_NAME - имя переменной. В окне "Свойства системы" можно просмотреть и изменить переменные окружения через панель управления.



4. Каково назначение переменных PATH и PATHNEXT?

PATH: Переменная, хранящая пути к исполняемым файлам. Она определяет, где операционная система будет искать исполняемые файлы, когда команда вводится в командной строке. PATHNEXT: Список расширений файлов, который интерпретируется как исполняемые файлы в Windows.

5. Как создать или изменить переменную окружения в Windows?

Для создания или изменения переменной окружения в Windows можно использовать "Свойства системы" -> "Дополнительные параметры системы" -> "Переменные окружения". Можно добавить новую переменную или изменить значение существующей.

6. Что представляют собой переменные окружения в ОС Linux?

В Linux переменные окружения представляют собой параметры, хранящиеся в системе, доступные для всех процессов. Они определяют окружение, в котором запускаются процессы, включая пути поиска, языковые настройки и другие параметры.

7. В чем отличие переменных окружения от переменных оболочки?

Переменные оболочки (shell variables) – это переменные, специфичные для конкретной оболочки и доступные только для этой оболочки. Переменные окружения (environment variables) – это переменные, доступные для всех процессов, запущенных в операционной системе, их значения наследуются от родительских процессов.

8. Как вывести значение переменной окружения в Linux?

В командной строке Linux можно использовать команду «echo \$VARIABLE\_NAME», где «VARIABLE\_NAME» - имя переменной.

9. Какие переменные окружения Linux Вам известны?

PATH, HOME, USER, LANG, SHELL, PWD и другие.

10. Какие переменные оболочки Linux Вам известны?

PS1, PS2, HISTSIZE, HISTFILE и другие, специфичные для определенных оболочек (например, BASH, Zsh).

11. Как установить переменные оболочки в Linux?

Для установки переменных оболочки в Linux используются команды экспорта переменной с ключевым словом «export» (например, «export VARIABLE\_NAME=value»).

12. Как установить переменные окружения в Linux?

Переменные окружения устанавливаются в Linux также, как и переменные оболочки, но они будут доступны для всех процессов. Эти переменные часто устанавливаются в файлах конфигурации системы, таких как «.bashrc», «.bash\_profile», «/etc/environment», и т. д.

13. Для чего необходимо делать переменные окружения Linux постоянными?

Переменные окружения могут быть установлены постоянно, добавив их в файлы инициализации оболочки, такие как «.bashrc» или «.bash\_profile» в домашнем каталоге пользователя.

14. Для чего используется переменная окружения PYTHONHOME?

PYTHONHOME – это переменная окружения Python, которая определяет базовый каталог установки Python.

15. Для чего используется переменная окружения PYTHONPATH?

PYTHONPATH – это переменная окружения Python, определяющая пути поиска Python для модулей.

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

PYTHONHOME, PYTHONPATH, PYTHONSTARTUP, PYTHONCASEOK, PYTHONIOENCODING и другие.

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

В Python переменные окружения можно читать с помощью модуля «os» с функцией «os.getenv()».

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

Для проверки установленного значения переменной окружения в Python используйте функцию «os.getenv('VARIABLE\_NAME')».

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

Для установки значения переменной окружения в Python можно использовать «os.putenv('VARIABLE')».

Вывод: в ходе выполнения лабораторной работы была более подробно изучена работа с переменными окружения, были рассмотрены способы создания переменных окружения и способы их использования в модулях Python.