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 «Анализ данных»
Вариант 13

Выполнил:
Иващенко Олег Андреевич
2 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и вычислительные машины»,
направленность (профиль)
«Программное обеспечение средств вычислительной техники и автоматизированных систем»

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович,
доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: «Работа с переменными окружения в Python3»

Цель: Приобретение навыков по работе с переменными окружения с помощью языка программирования Python версии 3.x.

Порядок выполнения работы

Пример 1. Для примера 1 лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответствующую переменную окружения.

Таблица 1 – Код программы example.py

```
#!/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)
```

```python
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
    os.environ.setdefault('WORKERS_DATA', 'example.json')
    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__":
    main()
```

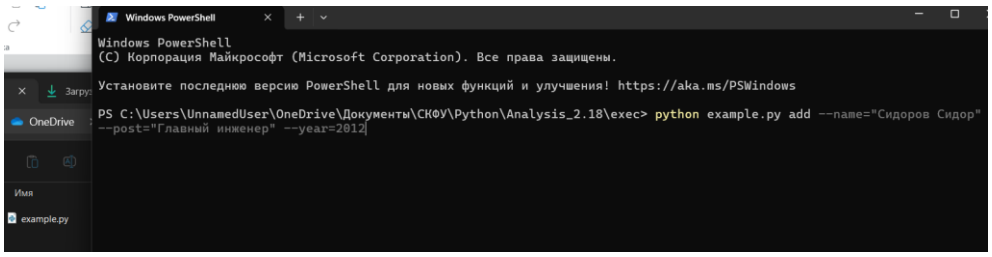

Рисунок 1.1 – Содержимое директории до выполнения команды

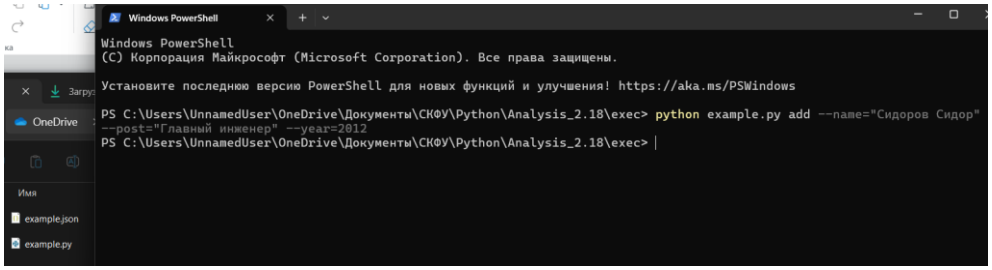

Рисунок 1.2 – Содержимое директории после выполнения команды

После выполнения программы был создан файл example.json, при этом в команде не указывалось имя файла. Имя файла было взято из переменной WORKERS_DATA как имя по умолчанию.

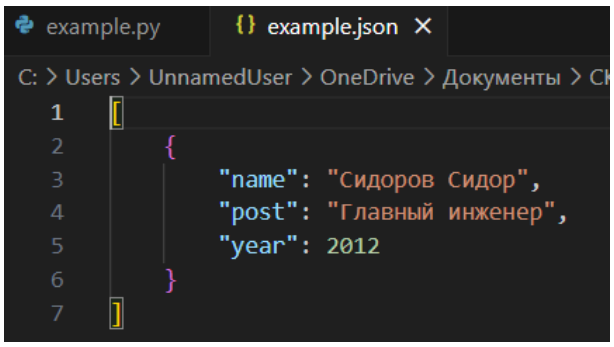

Рисунок 1.3 – Содержимое файла example.json

Индивидуальное задание 1. Для своего варианта лабораторной работы 2.17 добавить возможность получения имени файла данных, используя соответствующую переменной окружения.

Таблица 2 – Код программы individual_1.py

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


import json
import sys
from datetime import datetime
import argparse
import os.path


def print_help():
    """
    Функция вывода доступных пользователю команд
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")



def add_worker(workers, surname, name, phone, date):
    """
    Функция добавления новой записи, возвращает запись
    """

    workers.append(
        {
            "surname": surname,
            'name': name,
            'phone': phone,
            'date': date
        }
    )

    return workers
```

```python
def print_list(list):
    """
    Функция выводит на экран список всех существующих записей
    """

    for member in list:
        print(f"{member['surname']} {member['name']} | "
              f"{member['phone']} | {member['date']}")



def find_member(workers, period):
    """
    Функция для вывода на экран всех записей, чьи фамилии совпадают
    с введённой (не возвращает никаких значений)
    """

    count = 0
    members = []


    for member in workers:
        year = datetime.strptime(member['date'], "%d.%m.%Y").year
        if datetime.now().year - period >= year:
            members.append(member)
            count += 1

    if count == 0:
        print("Записи не найдены")
    else:
        return members



def save_file(filename, data):
    """
    Сохранение списка сотрудников в файл формата JSON
    """
    with open(filename, "w", encoding="utf-8") as file:
        json.dump(data, file, ensure_ascii=False, indent=4)



def load_file(filename):
    """
    Загрузка данных о сотрудниках из указанного JSON-файла
    """


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



def parse_datetime(value):
    try:
        return datetime.strptime(value, "%d.%m.%Y")
    except ValueError:
        print("Error")
```

```python
def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        nargs='?',
        action="store",
        help="The data file name"
    )


    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )


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


    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new worker"
    )
    add.add_argument(
        "-s",
        "--surname",
        action="store",
        required=True,
        help="The worker's surname"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
    add.add_argument(
        "-p",
        "--phone",
        action="store",
        help="The worker's phone"
    )
    add.add_argument(
        "-d",
        "--date",
        action="store",
        required=True,
        help="The date of hiring"
    )


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

```
    )


    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"
    )


    args = parser.parse_args(command_line)


    data_file = args.filename
    os.environ.setdefault('DEFAULT_FILE', 'individual_1.json')


    if data_file:
        if os.path.exists(data_file):
            workers = load_file(data_file)
        else:
            print("Файл не существует")
    else:
        data_file = os.environ.get('DEFAULT_FILE')
        workers = load_file(data_file)


    is_dirty = False
    if os.path.exists(data_file):
        workers = load_file(data_file)
    else:
        workers = []


    if args.command == "add":
        workers = add_worker(
            workers,
            args.surname,
            args.name,
            args.phone,
            args.date
        )
        is_dirty = True

    elif args.command == "display":
        print_list(workers)


    elif args.command == "select":
        selected = find_member(workers, args.period)
        print_list(selected)


    if is_dirty:
```

```
        data_file = args.filename
        os.environ.setdefault('DEFAULT_FILE', 'individual_1.json')


        if not data_file:
            data_file = os.environ.get('DEFAULT_FILE')
        if not data_file:
            print("Ошибка загрузки/сохранения данных")
            sys.exit(1)
        else:
            save_file(data_file, workers)



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

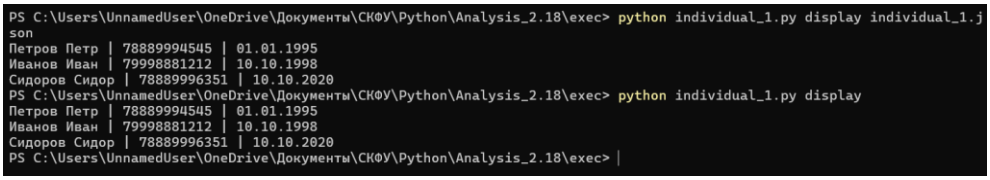

Рисунок 2.1 – Вывод записей из файла с использованием имени файла и без (DEFAULT_FILE='individual_1.json')

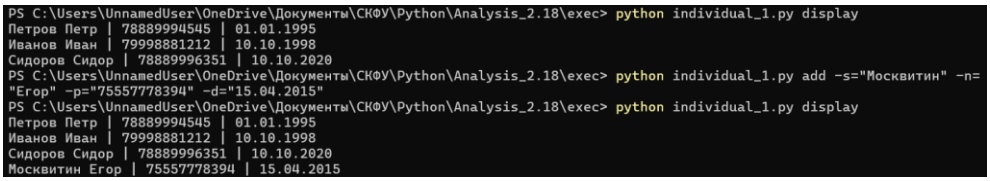

Рисунок 2.2 – Добавление новой записи в файл без указания имени файла

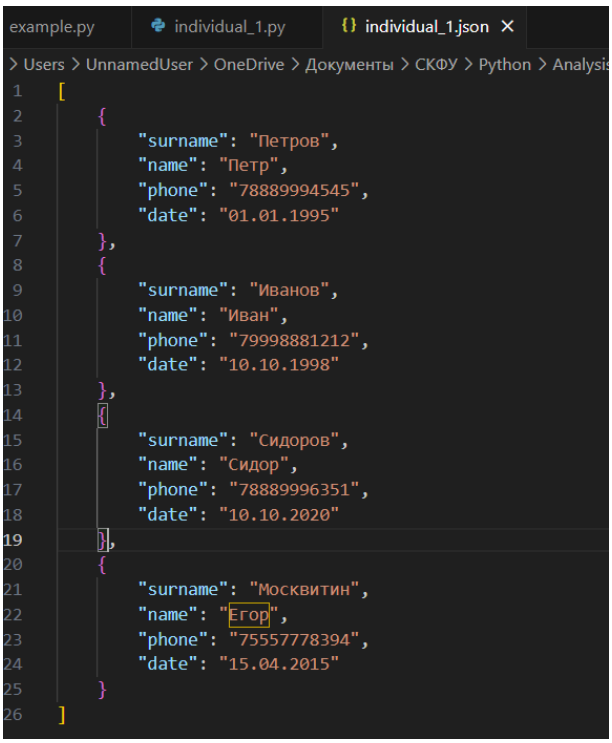

Рисунок 2.3 – Содержимое файла individual_1.json

Индивидуальное задание 2. Самостоятельно изучите работу с пакетом python-dotenv. Модифицируйте программу задания 1 таким образом, чтобы значения необходимых переменных окружения считывались из файла .env.

Таблица 3 – Код программы individual_2.py

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


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


dotenv.load_dotenv(dotenv_path="individual_2.env")


def print_help():
    """
    Функция вывода доступных пользователю команд
    """

    print("list - вывод всех добавленных записей")
    print("add - добавление новых записей")
    print("find - найти запись по фамилии")
    print("exit - завершение работы программы")
```

```python
def add_worker(workers, surname, name, phone, date):
    """
    Функция добавления новой записи, возвращает запись
    """

    workers.append(
        {
            "surname": surname,
            'name': name,
            'phone': phone,
            'date': date
        }
    )

    return workers



def print_list(list):
    """
    Функция выводит на экран список всех существующих записей
    """

    for member in list:
        print(f"{member['surname']} {member['name']} | "
              f"{member['phone']} | {member['date']}")



def find_member(workers, period):
    """
    Функция для вывода на экран всех записей, чьи фамилии совпадают
    с введённой (не возвращает никаких значений)
    """

    count = 0
    members = []


    for member in workers:
        year = datetime.strptime(member['date'], "%d.%m.%Y").year
        if datetime.now().year - period >= year:
            members.append(member)
            count += 1

    if count == 0:
        print("Записи не найдены")
    else:
        return members



def save_file(filename, data):
    """
    Сохранение списка сотрудников в файл формата JSON
    """
    with open(filename, "w", encoding="utf-8") as file:
        json.dump(data, file, ensure_ascii=False, indent=4)
```

```python
def load_file(filename):
    """
    Загрузка данных о сотрудниках из указанного JSON-файла
    """


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



def parse_datetime(value):
    try:
        return datetime.strptime(value, "%d.%m.%Y")
    except ValueError:
        print("Error")



def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        nargs='?',
        action="store",
        help="The data file name"
    )


    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )


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


    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new worker"
    )
    add.add_argument(
        "-s",
        "--surname",
        action="store",
        required=True,
        help="The worker's surname"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
```

```
    add.add_argument(
        "-p",
        "--phone",
        action="store",
        help="The worker's phone"
    )
    add.add_argument(
        "-d",
        "--date",
        action="store",
        required=True,
        help="The date of hiring"
    )


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


    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"
    )


    args = parser.parse_args(command_line)


    data_file = args.filename


    if data_file:
        if os.path.exists(data_file):
            workers = load_file(data_file)
        else:
            print("Файл не существует")
    else:
        data_file = os.getenv('DEFAULT_FILE')
        workers = load_file(data_file)


    is_dirty = False
    if os.path.exists(data_file):
        workers = load_file(data_file)
    else:
        workers = []


    if args.command == "add":
        workers = add_worker(
```

```
            workers,
            args.surname,
            args.name,
            args.phone,
            args.date
        )
        is_dirty = True

    elif args.command == "display":
        print_list(workers)


    elif args.command == "select":
        selected = find_member(workers, args.period)
        print_list(selected)


    if is_dirty:
        data_file = args.filename


        if not data_file:
            data_file = os.getenv('DEFAULT_FILE')
        if not data_file:
            print("Ошибка загрузки/сохранения данных")
            sys.exit(1)
        else:
            save_file(data_file, workers)



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

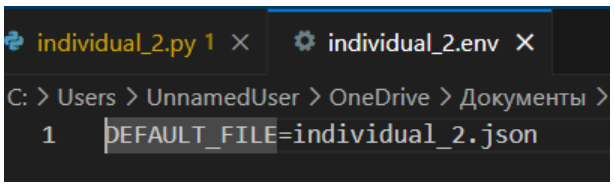

Рисунок 3.1 – Установленное стандартное имя файла

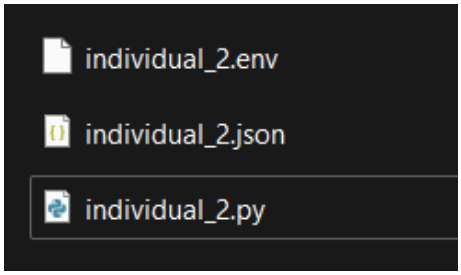

Рисунок 3.2 – Содержимое директории исполняемого файла

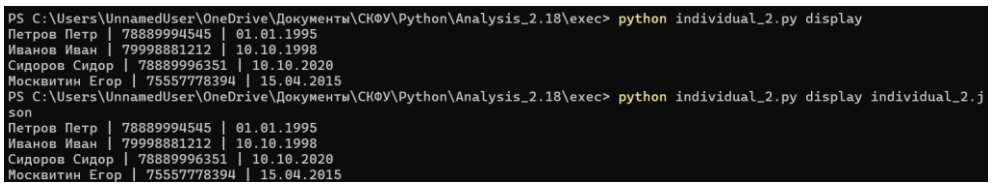

Рисунок 3.3 – Вывод содержимого файла без использования имени файла и с использованием

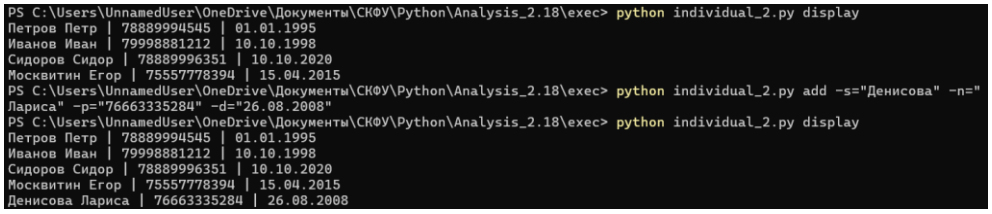

Рисунок 3.4 – Добавление новой записи в файл без использования имени файла

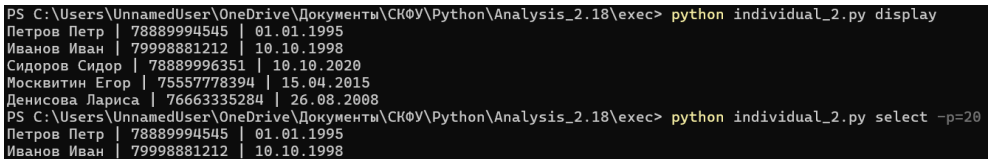

Рисунок 3.5 – Выборка записей из файла без использования имени файла

Контрольные вопросы

1. Каково назначение переменных окружения?

Переменные окружения предназначены для хранения информации, доступной для всех процессов в ОС или для конкретного пользователя. Они обеспечиваю способ передачи информации о конфигурации системы.

2. Какая информация может храниться в переменных окружения?

Имена пользователей, домашний каталог, текущий рабочий каталог, путь к исполняемым файлам, настройки локализации, параметры конфигурации программ и многое другое.

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

В командной строке Windows используется команда echo «имя_переменной» для вывода значения переменной окружения. Для установки или изменения переменных окружения можно использовать панель управления или команду set «имя_переменной»=«значение» в командной строке.

4. Каково назначение переменных PATH и PATHEXT?

Переменная PATH определяет список каталогов, в которых ОС будет искать исполняемые файлы.

Переменная PATHEXT содержит список расширений файлов, которые ОС считает исполняемыми.

5. Как создать или изменить переменную окружения в Windows?

Чтобы создать или изменить переменную окружения в Windows, можно использовать панель управления или команду set «имя_переменной»=«значение» в командной строке.

6. Что представляю собой переменные окружения в ОС Linux?

В Linux переменные окружения используются для хранения информации о системной конфигурации и настройках пользовательской среды.

7. В чём отличие переменных окружения от переменных оболочки?

Переменные окружения доступны для всех процессов в системе, в то время как переменные оболочки принадлежат только определённой оболочке и доступны только в её контексте.

8. Как вывести значение переменной окружения в Linux?

В Linux значение переменной окружения можно вывести используя команду echo «имя_переменной».

9. Какие переменные окружения Linux Вам известны?

PATH, HOME, USER, LANG, TERM.

10. Какие переменные оболочки Linux Вам известны?

PS1-4, SHELL, PWD

11. Как установить переменные оболочки в Linux?

Переменные оболочки можно установить, присваивая им значения в файле настройки оболочки ~/.bashrc или ~/.bash_profile.

12. Как установить переменные окружения в Linux?

Переменные окружения можно установить в файлах настройки оболочки или в скриптах инициализации системы.

13. Для чего необходимо делать переменные окружения в Linux постоянными?

Делая переменные окружения постоянными, гарантируется их доступность для всех процессов, запущенных в системе, и после перезагрузки. Это полезно для установки системных настроек.

14. Для чего используется переменная окружения PYTHONHOME?

Переменная окружения PYTHONHOME используется для указания расположения корневого каталога Python при запуске интерпретатора.

15. Для чего используется переменная окружения PYTHONPATH?

Переменная окружения PYTHONPATH определяет список каталогов, в которых интерпретатор Python будет искать модули при выполнении программ.

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

Например, PYTHONSTARTUP для указания файла Python, которые будет выполнен при запуске интерпретатора.

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

В Python переменные окружения могут быть прочитаны с помощью модуля os, используя функцию os.getenv(имя_переменной).

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

В Python можно проверить, установлено ли значение переменной окружения, используя функцию os.getenv(имя_переменной) и проверку на None.

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

В Python значение переменной окружения можно установить с помощью функции os.putenv(имя_переменной, значение), однако лучше использовать os.environ[имя_переменной] = «значение», так как это обеспечивает синхронизацию с изменениями переменных окружения в текущем процессе и его дочерних процессах.

Выводы: В процессе выполнения лабораторной работы были приобретены навыки по работе с переменными окружения с помощью языка

программирования Python. Был проработан пример, выполнены индивидуальные задания и самостоятельно изучен пакет python-dotenv.