МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

«Работа с переменными окружения в Python3»

Отчет по лабораторной работе № 2.18

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

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

Ход работы:

1. Изучить теоретический материал работы.

2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.

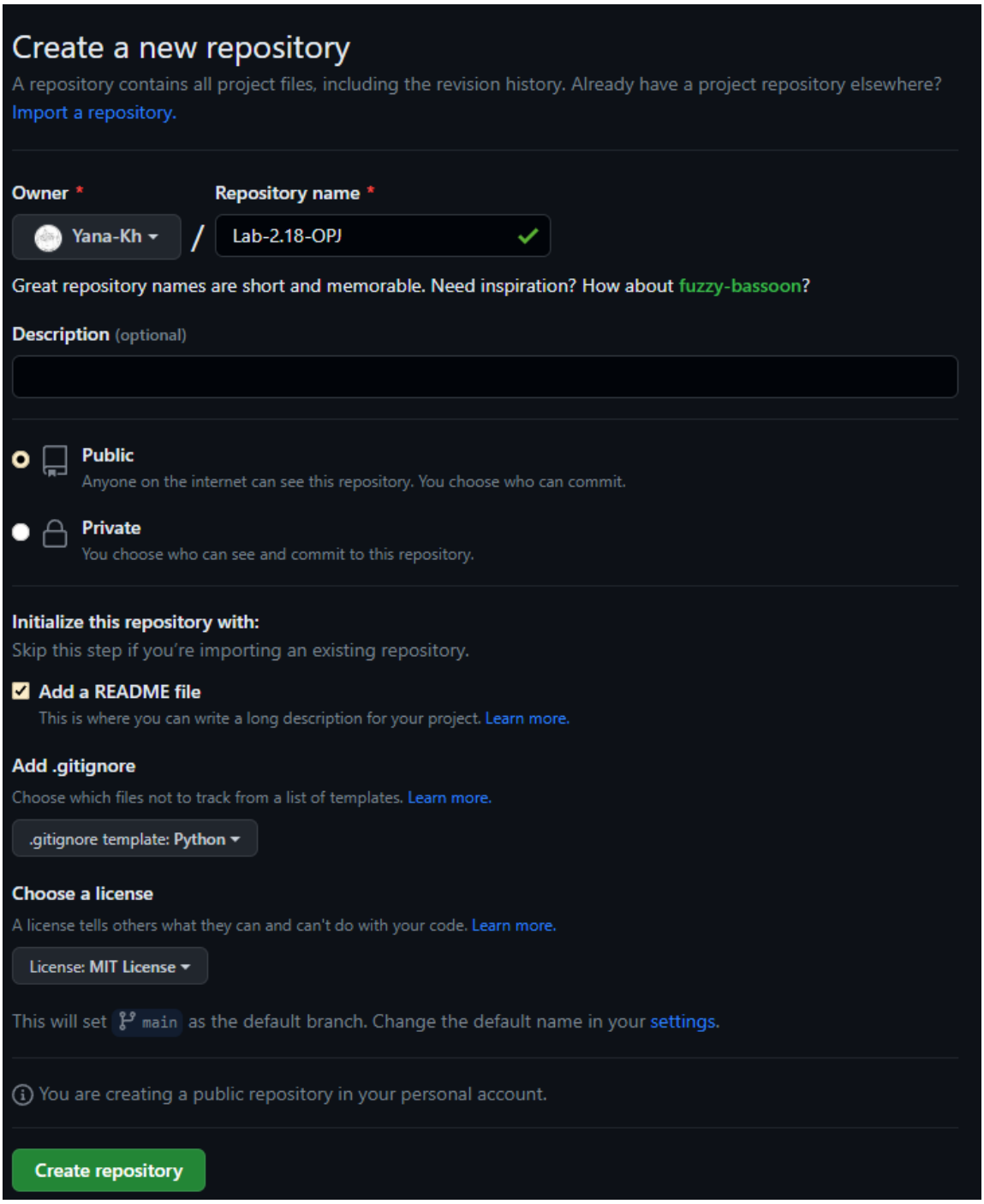


Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/
Yana-Kh/Lab-2.18-OPJ.git
Cloning into 'Lab-2.18-OPJ'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
```

Рисунок 2 – Клонирование репозитория

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.18-OPJ>git add .

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.18-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore


C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.18-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.18-OPJ>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/ynakh/OneDrive/Рабочий стол/Git/Lab-2.18-OPJ/.git/hooks]

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.18-OPJ>
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

Рисунок 5 – Создание проекта PyCharm

7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Зафиксируйте изменения в репозитории.

Пример 1. Для примера 1 лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответстсвующую переменную окружения. Для хранения имени файла данных будем использовать переменную окружения WORKERS_DATA. При этом сохраним возможность передавать имя файла данных через именной параметр --data. Иными словами, если при запуске программы в командной строке не задан параметр --data , то имя файла данных должно быть взято из переменной окружения WORKERS_DATA .

Код:

```
#!/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(
                "No",
                "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 0.1.0"
    )

    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(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()
```

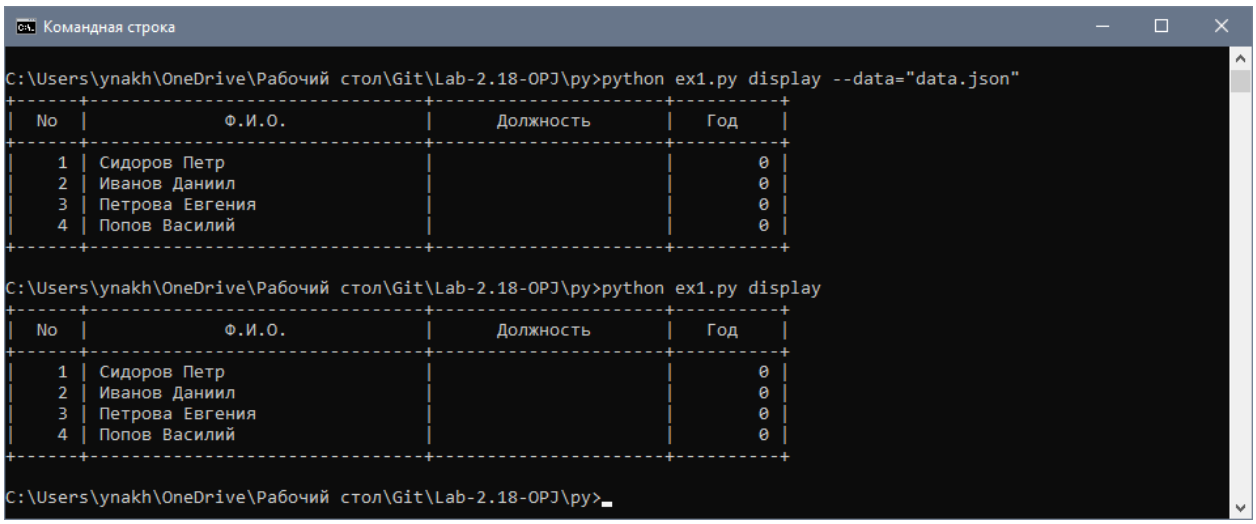


Рисунок 6 – Результат работы программы

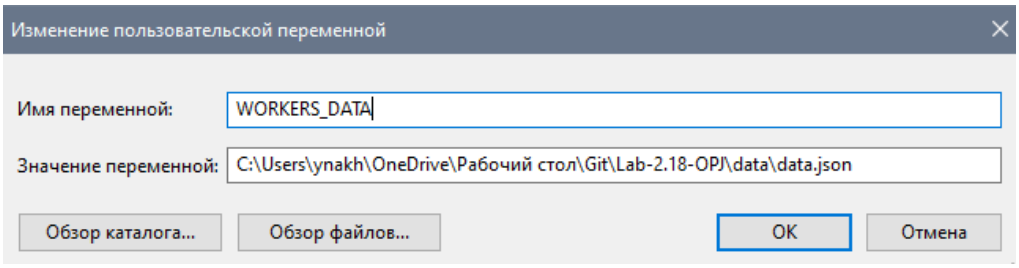
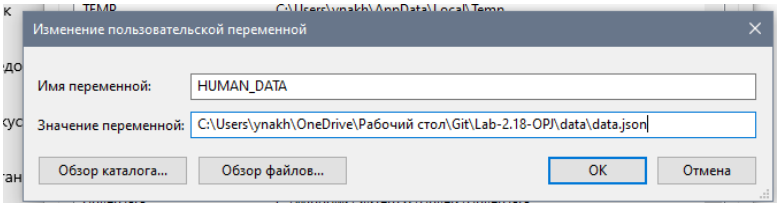


Рисунок 7 – Фиксирование изменений в репозитории

8. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.

Задание 1:

Для своего варианта лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответстсвующую переменную окружения.



Код:

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

import json
import sys
import datetime
```

```
import os.path
import argparse


def add_human(staff, name, phone, str_bday):
    """
    Запросить данные о человеке.
    """
    b_day = list(map(int, str_bday.split(".")))
    date_bday = datetime.date(b_day[2], b_day[1], b_day[0])
    staff.append(
        {
            "name": name,
            "phone": phone,
            "birthday": date_bday
        }
    )

    return staff


def display_human(staff):
    """
    Отобразить список людей.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4,
            "-" * 30,
            "-" * 15,
            "-" * 15
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^15} | {:^15} |".format(
                "№",
                "Фамилия и имя",
                "Телефон",
                "День рождения"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, human in enumerate(staff, 1):
            print(
                f"| {idx:>4} |"
                f' {human.get("name", ""):<30} |'
                f' {human.get("phone", 0):<15} |'
                f' {human.get("birthday")}      |'
            )
            print(line)

    else:
        print("Список пуст.")


def find_human(staff, fname):
    """
    Выбрать людей с заданным стажем.
    """
    # Сформировать список людей.
```

```
    result = []
    for h in staff:
        if fname in str(h.values()):
            result.append(h)

    # Проверка на наличие записей
    if len(result) == 0:
        return print("Запись не найдена")

    # Возвратить список выбранных работников.
    return result


def json_deserial(obj):
    """
    Деериализация объектов datetime
    """
    for h in obj:
        if isinstance(h["birthday"], str):
            # print(datetime.strptime(h['birthday'], '%Y-%m-%d'))
            bday = list(map(int, h["birthday"].split("-")))
            h["birthday"] = datetime.date(bday[0], bday[1], bday[2])


def load_humans(file_name):
    """
    Загрузить всех людей 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 json_serial(obj):
    """
    Сериализация объектов datetime
    """

    if isinstance(obj, (datetime.datetime, datetime.date)):
        return obj.isoformat()


def save_humans(file_name, staff):
    """
    Сохранить всех людей 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,
default=json_serial)


def main(command_line=None):
    """
    Главная функция программы.
    """
    # 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("people")
    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",
        "--phone",
        type=int,
        action="store",
        help="The worker's post"
    )

    add.add_argument(
        "-bd",
        "--bday",
        action="store",
        required=True,
        help="The year of hiring"
    )

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

    # Создать субпарсер для поиска людей по фамилии.
    find = subparsers.add_parser(
        "find",
        parents=[file_parser],
        help="Find the people"
    )

    find.add_argument(
        "-sn",
        "--surname",
        action="store",
```

```
        required=True,
        help="Required surname"
    )

    # 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("HUMAN_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):
        people = load_humans(data_file)
    else:
        people = []

    # Добавить челоека.
    if args.command == "add":
        people = add_human(
            people,
            args.name,
            args.phone,
            args.bday
        )
        is_dirty = True

    # Отобразить всех людей.
    elif args.command == "display":
        display_human(people)

    # Выбрать требуемых людей.
    elif args.command == "find":
        selected = find_human(people, args.surname)
        display_human(selected)

    # Сохранить данные в файл, если список людей был изменен.
    if is_dirty:
        save_humans(args.filename, people)


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

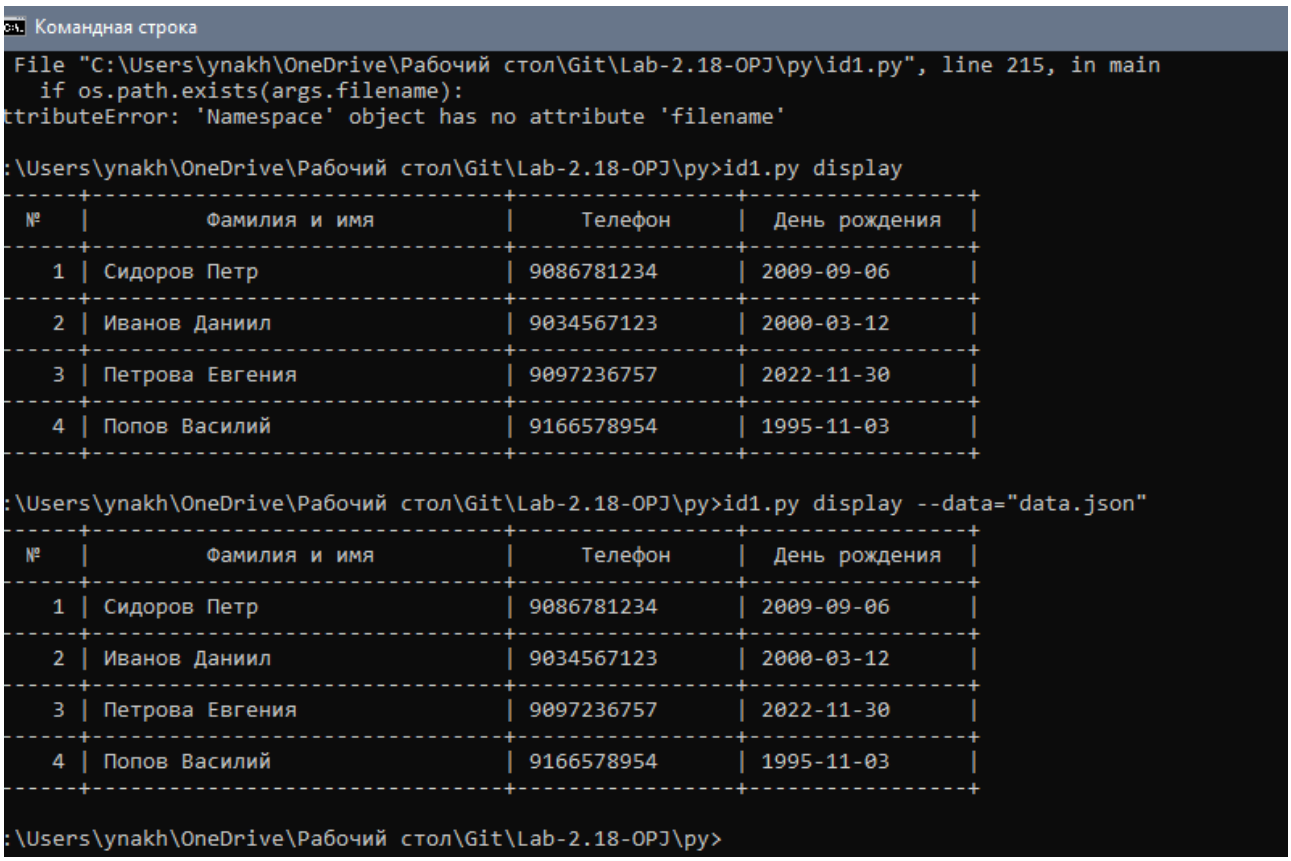


Рисунок 8 – Результат работы программы

Задание 2:

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

Код:

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

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


def add_human(staff, name, phone, str_bday):
    """
    Запросить данные о человеке.
    """
    b_day = list(map(int, str_bday.split(".")))
    date_bday = datetime.date(b_day[2], b_day[1], b_day[0])
    staff.append(
        {
            "name": name,
```

```
            "phone": phone,
            "birthday": date_bday
        }
    )

    return staff


def display_human(staff):
    """
    Отобразить список людей.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4,
            "-" * 30,
            "-" * 15,
            "-" * 15
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^15} | {:^15} |".format(
                "№",
                "Фамилия и имя",
                "Телефон",
                "День рождения"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, human in enumerate(staff, 1):
            print(
                f"| {idx:>4} |"
                f' {human.get("name", ""):<30} |'
                f' {human.get("phone", 0):<15} |'
                f' {human.get("birthday")}       |'
            )
            print(line)

    else:
        print("Список пуст.")


def find_human(staff, fname):
    """
    Выбрать людей с заданным стажем.
    """
    # Сформировать список людей.
    result = []
    for h in staff:
        if fname in str(h.values()):
            result.append(h)

    # Проверка на наличие записей
    if len(result) == 0:
        return print("Запись не найдена")

    # Возвратить список выбранных работников.
    return result
```

```python
def json_deserial(obj):
    """
    Деериализация объектов datetime
    """
    for h in obj:
        if isinstance(h["birthday"], str):
            # print(datetime.strptime(h['birthday'], '%Y-%m-%d'))
            bday = list(map(int, h["birthday"].split("-")))
            h["birthday"] = datetime.date(bday[0], bday[1], bday[2])


def load_humans(file_name):
    """
    Загрузить всех людей 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 json_serial(obj):
    """
    Сериализация объектов datetime
    """

    if isinstance(obj, (datetime.datetime, datetime.date)):
        return obj.isoformat()


def save_humans(file_name, staff):
    """
    Сохранить всех людей 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,
default=json_serial)


def main(command_line=None):
    """
    Главная функция программы.
    """
    # 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("people")
    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",
        "--phone",
        type=int,
        action="store",
        help="The worker's post"
    )

    add.add_argument(
        "-bd",
        "--bday",
        action="store",
        required=True,
        help="The year of hiring"
    )

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

    # Создать субпарсер для поиска людей по фамилии.
    find = subparsers.add_parser(
        "find",
        parents=[file_parser],
        help="Find the people"
    )

    find.add_argument(
        "-sn",
        "--surname",
        action="store",
        required=True,
        help="Required surname"
    )
    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Подключение к окружению
    data_file = args.data

    dotenv_path = os.path.join(os.path.dirname(__file__), '.env')
    if os.path.exists(dotenv_path):
        load_dotenv(dotenv_path)
    if not data_file:
```

```
        data_file = os.getenv("HUMAN_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):
        people = load_humans(data_file)
    else:
        people = []

    # Добавить челоека.
    if args.command == "add":
        people = add_human(
            people,
            args.name,
            args.phone,
            args.bday
        )
        is_dirty = True

    # Отобразить всех людей.
    elif args.command == "display":
        display_human(people)

    # Выбрать требуемых людей.
    elif args.command == "find":
        selected = find_human(people, args.surname)
        display_human(selected)

    # Сохранить данные в файл, если список людей был изменен.
    if is_dirty:
        save_humans(data_file, people)


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

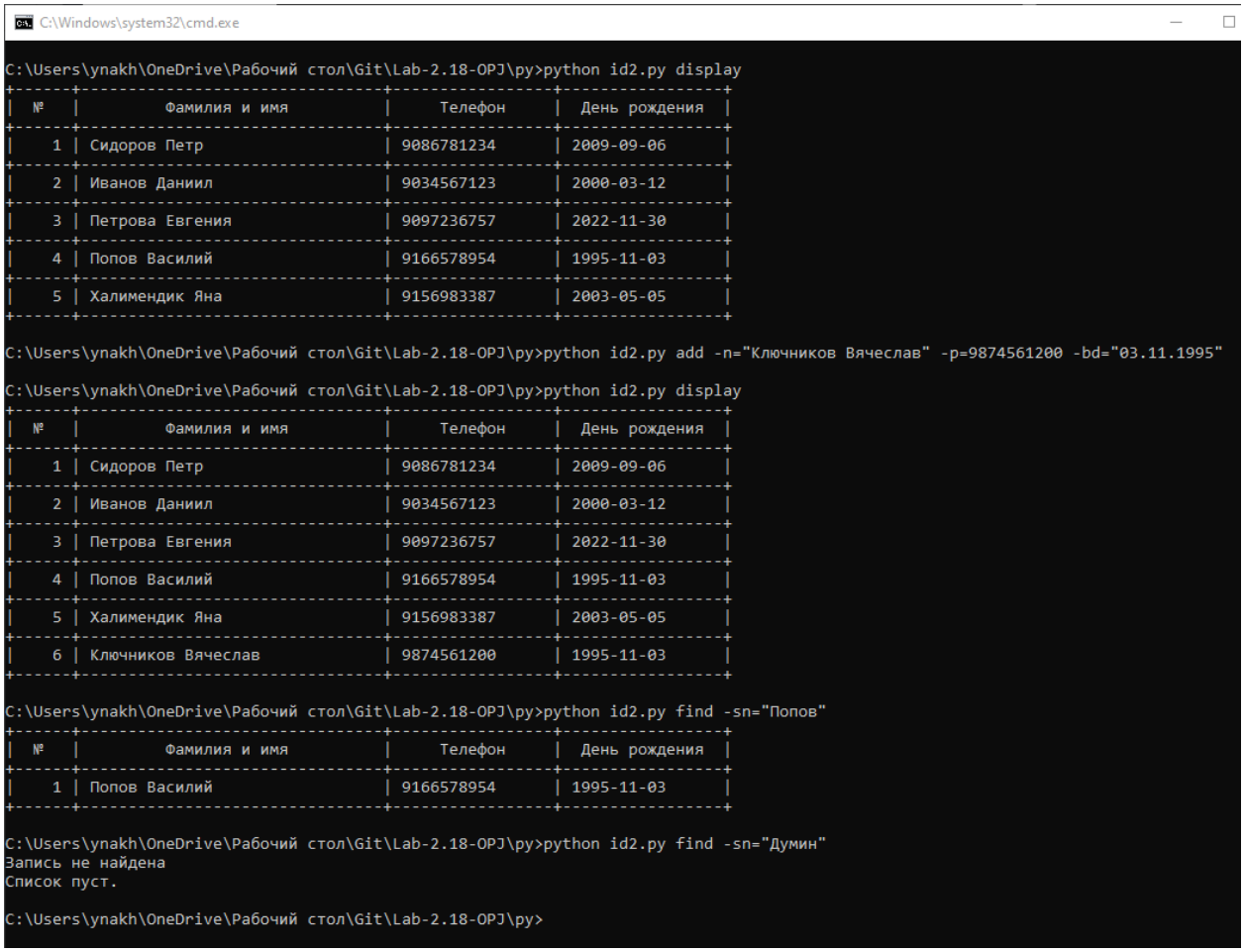


Рисунок 9 – Результат работы программы

10. Зафиксируйте сделанные изменения в репозитории.

Рисунок 10 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

1. Каково назначение переменных окружения?

Переменная среды — это динамический «объект» на компьютере, содержащий редактируемое значение, которое может использоваться одной или несколькими программами в Windows. Переменные среды помогают программам узнать, в какой каталог устанавливать файлы, где хранить временные файлы и где найти настройки профиля пользователя. Они помогают формировать среду, которую программы на вашем компьютере используют для запуска.

Использовать ее можно в скриптах, например, когда нужно каждому пользователю сделать какую-то настройку, для примера положить на рабочий стол новую папку с документами. Так как для каждого пользователя путь до рабочего стола свой, можно обратиться и по относительному пути за счет переменной %HOMEPATH%, если в проводнике ввести %HOMEPATH%\Desktop.

2. Какая информация может храниться в переменных окружения?

Переменные среды хранят информацию о среде операционной системы. Эта информация включает такие сведения, как путь к операционной системе, количество процессоров, используемых операционной системой, и расположение временных папок.

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

Получить информацию о существующих переменных можно в свойствах системы.

Если требуется просмотреть весь перечень, запускаем «Командную строку» от имении администратора и выполняем команду «set > %homepath%\desktop\set.txt»

4. Каково назначение переменных PATH и PATHEXT?

«PATH» позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. «PATHEXT», в свою очередь, дает возможность не указывать даже расширение файла, если оно прописано в ее значениях.

5. Как создать или изменить переменную окружения в Windows?

Нажимаем кнопку Создать. Сделать это можно как в пользовательском разделе, так и в системном. Вводим имя, например, desktop. Важно, чтобы такое название еще не было использовано. В поле Значение указываем путь (в

нашем случае до папки Рабочий стол: C:\Users\Имя_пользователя\Desktop). Нажимаем ОК. Повторяем это действие во всех открытых окнах. Перезапускаем Проводник и консоль или целиком систему. Готово, новая переменная создана, увидеть ее можно в соответствующем списке.

6. Что представляют собой переменные окружения в ОС LInux?

Переменные окружения в Linux представляют собой набор именованных значений, используемых другими приложениями.

Переменные окружения применяются для настройки поведения приложений и работы самой системы. Например, переменная окружения может хранить информацию о путях к исполняемым файлам, заданном по умолчанию текстовом редакторе, браузере, языковых параметрах (локали) системы или настройках раскладки клавиатуры.

7. В чем отличие переменных окружения от переменных оболочки?

Переменные окружения (или «переменные среды») — это переменные, доступные в масштабах всей системы и наследуемые всеми дочерними процессами и оболочками.

Переменные оболочки — это переменные, которые применяются только к текущему экземпляру оболочки. Каждая оболочка, например, bash или zsh, имеет свой собственный набор внутренних переменных.

8. Как вывести значение переменной окружения в Linux?

Наиболее часто используемая команда для вывода переменных окружения — printenv . Если команде в качестве аргумента передать имя переменной, то будет отображено значение только этой переменной. Если же вызвать printenv без аргументов, то выведется построчный список всех переменных окружения.

9. Какие переменные окружения Linux Вам известны?

USER — текущий пользователь.

PWD — текущая директория.

OLDPWD — предыдущая рабочая директория. Используется оболочкой для того, чтобы вернуться в предыдущий каталог при выполнении команды cd - .

HOME — домашняя директория текущего пользователя.

SHELL — путь к оболочке текущего пользователя (например, bash или zsh).

EDITOR — заданный по умолчанию редактор. Этот редактор будет вызываться в ответ на команду edit .

LOGNAME — имя пользователя, используемое для входа в систему.

PATH — пути к каталогам, в которых будет производиться поиск вызываемых команд. При выполнении команды система будет проходить по данным каталогам в указанном порядке и выберет первый из них, в котором будет находиться исполняемый файл искомой команды.

LANG — текущие настройки языка и кодировки.

TERM — тип текущего эмулятора терминала.

MAIL — место хранения почты текущего пользователя.

LS_COLORS — задает цвета, используемые для выделения объектов (например, различные типы файлов в выводе команды ls будут выделены разными цветами).

10. Какие переменные оболочки Linux Вам известны?

BASHOPTS — список задействованных параметров оболочки, разделенных двоеточием.

BASH_VERSION — версия запущенной оболочки bash.

COLUMNS — количество столбцов, которые используются для отображения выходных данных.

DIRSTACK — стек директорий, к которому можно применять команды pushd и popd .

HISTFILESIZE — максимальное количество строк для файла истории команд.

HISTSIZE — количество строк из файла истории команд, которые можно хранить в памяти.

HOSTNAME — имя текущего хоста.

IFS — внутренний разделитель поля в командной строке (по умолчанию используется пробел).

PS1 — определяет внешний вид строки приглашения ввода новых команд.

PS2 — вторичная строка приглашения.

SHELLOPTS — параметры оболочки, которые можно устанавливать с помощью команды set .

UID — идентификатор текущего пользователя.

11. Как установить переменные оболочки в Linux?

Чтобы создать новую переменную оболочки с именем, например, NEW_VAR и значением Ravesli.com, просто введите имя этой переменной потом знак равенства и указать значение новой переменной, например, $ NEW_VAR='Ravesli.com' Вы можете убедиться, что переменная действительно была создана, с помощью команды echo : $ echo $NEW_VAR

12. Как установить переменные окружения в Linux?

Команда export используется для задания переменных окружения. С помощью данной команды мы экспортируем указанную переменную, в результате чего она будет видна во всех вновь запускаемых дочерних командных оболочках. Переменные такого типа принято называть внешними.

Для создания переменной окружения экспортируем нашу недавно созданную переменную оболочки: $ export NEW_VAR

Вы также можете использовать и следующую конструкцию для создания переменной окружения: $ export MY_NEW_VAR="My New Var"

Однако, созданные подобным образом переменные окружения доступны только в текущем сеансе. Если вы откроете новую оболочку или выйдете из системы, то все переменные будут потеряны.

Если вы хотите, чтобы переменная сохранялась после закрытия сеанса оболочки, то необходимо прописать ее в специальном файле. Прописать переменную можно как для текущего пользователя, так и для всех пользователей.

Чтобы установить постоянную переменную окружения для текущего пользователя, откройте файл .**bashrc: $ sudo nano ~/.bashrc Для каждой переменной, которую вы хотите сделать постоянной, добавьте в конец файла строку, используя следующий синтаксис:

export [ИМЯ_ПЕРЕМЕННОЙ]=[ЗНАЧЕНИЕ_ПЕРЕМЕННОЙ]

13. Для чего необходимо делать переменные окружения Linux постоянными?

Для того, чтобы они сохранялись при перезапуске сессии.

14. Для чего используется переменная окружения PYTHONHOME ?

Переменная среды PYTHONHOME изменяет расположение стандартных библиотек Python. По умолчанию библиотеки ищутся в prefix/lib/pythonversion и exec_prefix/lib/pythonversion, где prefix и exec_prefix – это каталоги, зависящие от установки, оба каталога по умолчанию – /usr/local. Когда для PYTHONHOME задан один каталог, его значение заменяет prefix и exec_prefix . Чтобы указать для них разные значения, установите для PYTHONHOME значение prefix:exec_prefix .

15. Для чего используется переменная окружения PYTHONPATH ?

Переменная среды PYTHONPATH изменяет путь поиска по умолчанию для файлов модуля. Формат такой же, как для оболочки PATH : один или

несколько путей к каталогам, разделенных os.pathsep (например, двоеточие в Unix или точка с запятой в Windows). Несуществующие каталоги игнорируются. Помимо обычных каталогов, отдельные записи PYTHONPATH могут относиться к zip-файлам, содержащим чистые модули Python в исходной или скомпилированной форме. Модули расширения нельзя импортировать из zipфайлов. Путь поиска по умолчанию зависит от установки Python, но обычно начинается с префикса /lib/pythonversion . Он всегда добавляется к PYTHONPATH

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.

PYTHONUNBUFFERED: Если значение переменной среды PYTHONUNBUFFERED непустая строка, то это эквивалентно указанию параметра -u.

PYTHONVERBOSE: Если значение переменной среды PYTHONVERBOSE непустая строка, то это эквивалентно указанию опции -v. Если установлено целое число, это эквивалентно многократному указанию - v.

PYTHONCASEOK: Если значение переменной среды PYTHONCASEOK установлено, то Python игнорирует регистр символов в операторах импорта. Это работает только в Windows и OS X.

PYTHONDONTWRITEBYTECODE: Если значение переменной среды PYTHONDONTWRITEBYTECODE непустая строка, то Python не будет пытаться писать файлы .pyc при импорте исходных модулей. Это эквивалентно указанию параметра -B.

PYTHONPYCACHEPREFIX: Если значение переменной среды PYTHONPYCACHEPREFIX установлено, то Python будет записывать файлы .pyc в зеркальном дереве каталогов по этому пути, а не в каталогах pycache в исходном дереве. Это эквивалентно указанию параметра –X pycache_prefix=PATH.

PYTHONHASHSEED: Если значение переменной среды PYTHONHASHSEED не установлено или имеет значение random, то случайное значение используется для заполнения хэшей объектов str и bytes . Если для PYTHONHASHSEED задано целочисленное значение, то оно используется как фиксированное начальное число для генерации hash() типов,

охватываемых рандомизацией хэша. Цель - разрешить повторяемое хеширование, например, для самотестирования самого интерпретатора, или позволить кластеру процессов Python совместно использовать хеш- значения. Целое число должно быть десятичным числом в диапазоне [0,4294967295]. Указание значения 0 отключит рандомизацию хэша.

PYTHONIOENCODING: Если значение переменной среды PYTHONIOENCODING установлено до запуска интерпретатора, то оно переопределяет кодировку, используемую для stdin / stdout / stderr , в синтаксисе encodingname:errorhandler . И имя кодировки encodingname , и части :errorhandler являются необязательными и имеют то же значение, что и в функции str.encode() . Для stderr часть: errorhandler игнорируется, а обработчик всегда будет заменять обратную косую черту.

PYTHONNOUSERSITE: Если значение переменной среды PYTHONNOUSERSITE установлено, то Python не будет добавлять пользовательский каталог site-packages в переменную sys.path .

PYTHONUSERBASE: Переменная среды PYTHONUSERBASE определяет базовый каталог пользователя, который используется для вычисления пути к каталогу пользовательских пакетов сайта site-packages и путей установки Distutils для python setup.py install --user.

PYTHONWARNINGS: Переменная среды PYTHONWARNINGS эквивалентна опции -W. Если она установлена в виде строки, разделенной запятыми, то это эквивалентно многократному указанию -W, при этом фильтры, расположенные позже в списке, имеют приоритет над фильтрами ранее в списке. В простейших настройках определенное действие безоговорочно применяется ко всем предупреждениям, выдаваемым процессом (даже к тем, которые по умолчанию игнорируются):

PYTHONWARNINGS=default - предупреждает один раз для каждого вызова;

PYTHONWARNINGS=error - преобразовывает в исключения;

PYTHONWARNINGS=always - предупреждает каждый раз;

PYTHONWARNINGS=module - предупреждает один раз для каждого вызванного модуля;

PYTHONWARNINGS=once - предупреждает один раз для каждого процесса Python;

PYTHONWARNINGS=ignore - никогда не предупреждает.

PYTHONFAULTHANDLER: Если значение переменной среды PYTHONFAULTHANDLER непустая строка, то при запуске вызывается faulthandler.enable() : устанавливается обработчик сигналов SIGSEGV , SIGFPE , SIGABRT , SIGBUS и SIGILL , чтобы вывести данные трассировки Python. Это эквивалентно опции обработчика ошибок -X.

PYTHONTRACEMALLOC: Если значение переменной среды PYTHONTRACEMALLOC непустая строка, то начнется отслеживание выделения памяти Python с помощью модуля tracemalloc. Значение переменной - это максимальное количество кадров, хранящихся в обратной трассировке trace . Например, PYTHONTRACEMALLOC=1 сохраняет только самый последний кадр.

PYTHONPROFILEIMPORTTIME: Если значение переменной среды PYTHONPROFILEIMPORTTIME непустая строка, то Python покажет, сколько времени занимает каждый импорт. Это в точности эквивалентно установке –X importtime в командной строке.

PYTHONASYNCIODEBUG: Если значение переменной среды PYTHONASYNCIODEBUG непустая строка, то включается режим отладки модуля asyncio .

PYTHONMALLOC: Переменная PYTHONMALLOC задает распределители памяти Python и/или устанавливает отладочные хуки. Задает семейство распределителей памяти, используемых Python:

default: использует распределители памяти по умолчанию.

malloc: использует функцию malloc() библиотеки C для всех доменов (PYMEM_DOMAIN_RAW, PYMEM_DOMAIN_MEM, PYMEM_DOMAIN_OBJ).

pymalloc: использует распределитель pymalloc для доменов PYMEM_DOMAIN_MEM и PYMEM_DOMAIN_OBJ и использует функцию malloc() для домена PYMEM_DOMAIN_RAW . Устанавливает хуки отладки:

debug: устанавливает хуки отладки поверх распределителей памяти по умолчанию.

malloc_debug: то же, что и malloc , но также устанавливает отладочные хуки.

pymalloc_debug: то же, что и pymalloc, но также устанавливает отладочные хуки.

PYTHONMALLOCSTATS: Если значение переменной среды PYTHONMALLOCSTATS непустая строка, то Python будет печатать статистику распределителя памяти pymalloc каждый раз, когда создается новая область объекта pymalloc, а также при завершении работы. Эта переменная игнорируется, если переменная среды PYTHONMALLOC используется для принудительного использования распределителя malloc() библиотеки C или если Python настроен без поддержки pymalloc .

PYTHONLEGACYWINDOWSFSENCODING: Если значение переменной среда Python PYTHONLEGACYWINDOWSFSENCODING непустая строка, то кодировка файловой системы по умолчанию и режим ошибок вернутся к своим значениям mbcs и replace до версии Python 3.6 соответственно. В противном случае используются новые значения по умолчанию utf-8 и surrogatepass .

PYTHONLEGACYWINDOWSSTDIO: Если значение переменной среды PYTHONLEGACYWINDOWSSTDIO непустая строка, то новые средства чтения и записи консоли не используются. Это означает, что символы Unicode будут закодированы в соответствии с активной кодовой страницей консоли, а не с использованием utf-8 . Эта переменная игнорируется, если стандартные потоки перенаправляются в файлы или каналы, а не ссылаются на буферы консоли.

PYTHONCOERCECLOCALE: Если значение переменной среды PYTHONCOERCECLOCALE установлено в значение 0, то это заставит основное приложение командной строки Python пропускать приведение устаревших локалей C и POSIX на основе ASCII к более функциональной альтернативе на основе UTF-8. Если эта переменная не установлена или имеет значение, отличное от 0 , то переменная среды переопределения локали LC_ALL также не задана, а текущая локаль, указанная для категории LC_CTYPE , является либо локалью C по умолчанию, либо локалью POSIX явно основанной на ASCII , то Python CLI попытается настроить следующие локали для категории LC_CTYPE в порядке, указанном перед загрузкой среды выполнения интерпретатора: C.UTF-8 , C.utf8 , UTF-8 . Если установка одной из этих категорий локали прошла успешно, то переменная среды LC_CTYPE также будет установлена соответствующим образом в текущей среде процесса д инициализации среды выполнения Python. Это гарантирует, что обновленный параметр будет виден как самому интерпретатору, так и другим компонентам, зависящим от локали, работающим в одном процессе (например, библиотеке GNU readline), и в субпроцессах (независимо от того, работают ли эти процессы на интерпретаторе Python или нет), а также в операциях, которые запрашивают среду, а не текущую локаль C (например, собственный locale.getdefaultlocale() Python). Настройка одного из этих языковых стандартов явно или с помощью указанного выше неявного принуждения языкового стандарта автоматически включает обработчик ошибок surrogateescape для sys.stdin и sys.stdout (sys.stderr продолжает использовать обратную косую черту, как и в любой другой локали). Это поведение обработки потока можно переопределить, используя PYTHONIOENCODING, как обычно. Для целей отладки, установка PYTHONCOERCECLOCALE=warn приведет к тому, что Python будет выдавать предупреждающие сообщения на stderr , если активируется принуждение языкового стандарта или если языковой стандарт, который мог бы вызвать приведение, все еще активен при инициализации среды

выполнения Python. Также обратите внимание, что даже когда принуждение языкового стандарта отключено или когда не удается найти подходящую целевую локаль, переменная среды PYTHONUTF8 все равно будет активироваться по умолчанию в устаревших локалях на основе ASCII. Чтобы для системных интерфейсов интерпретатор использовал ASCII вместо UTF-8 , необходимо обе переменные отключить.

PYTHONDEVMODE: Если значение переменной среды PYTHONDEVMODE непустая строка, то включится режим разработки Python, введя дополнительные проверки времени выполнения, которые слишком "дороги" для включения по умолчанию.

PYTHONUTF8: Если переменная среды PYTHONUTF8 установлена в значение 1, то это включает режим интерпретатора UTF-8, где UTF-8 используется как кодировка текста для системных интерфейсов, независимо от текущей настройки локали.

PYTHONWARNDEFAULTENCODING: Если для этой переменной среды задана непустая строка, то код будет выдавать EncodingWarning , когда используется кодировка по умолчанию, зависящая от локали.

PYTHONTHREADDEBUG: Если значение переменной среды PYTHONTHREADDEBUG установлено, то Python распечатает отладочную информацию о потоках. Нужен Python, настроенный с параметром сборки --with-pydebug.

PYTHONDUMPREFS: Если значение переменной среды PYTHONDUMPREFS установлено, то Python будет сбрасывать объекты и счетчики ссылок, все еще живые после завершения работы интерпретатора.

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

Для начала потребуется импортировать модуль os, чтобы считывать переменные. Для доступа к переменным среды в Python используется объект

os.environ . С его помощью программист может получить и изменить значения всех переменных среды.

```
# Импортируем модуль os
import os
# Создаѐм цикл, чтобы вывести все переменные среды
print("The keys and values of all environment variables:")
for key in os.environ:
print(key, '=>', os.environ[key])
# Выводим значение одной переменной
print("The value of HOME is: ", os.environ['HOME'])
```

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

```
# Импортируем модуль os
import os
# Импортируем модуль sys
import sys

while True:
    # Принимаем имя переменной среды
    key_value = input("Enter the key of the environment variable:")
    # Проверяем, инициализирована ли переменная
    try:
        if os.environ[key_value]:
        print(
                "The value of",
                key_value,
                " is ",
                os.environ[key_value]
        )
```

```
    # Если переменной не присвоено значение, то ошибка
    except KeyError:
        print(key_value, 'environment variable is not set.')
        # Завершаем процесс выполнения скрипта
        sys.exit(1)
```

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

Для присвоения значения любой переменной среды используется функция setdefault()

```
# Импортируем модуль os
import os

# Задаём значение переменной DEBUG
os.environ.setdefault('DEBUG', 'True')
# Проверяем значение переменной
if os.environ.get('DEBUG') == 'True':
    print('Debug mode is on')
else:
    print('Debug mode is off')
```