

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 «Программирование на Python»

Выполнил студент группы ИВТ-б-о-21-1

Уланбекова Айканыш.

«19» декабря 2022г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

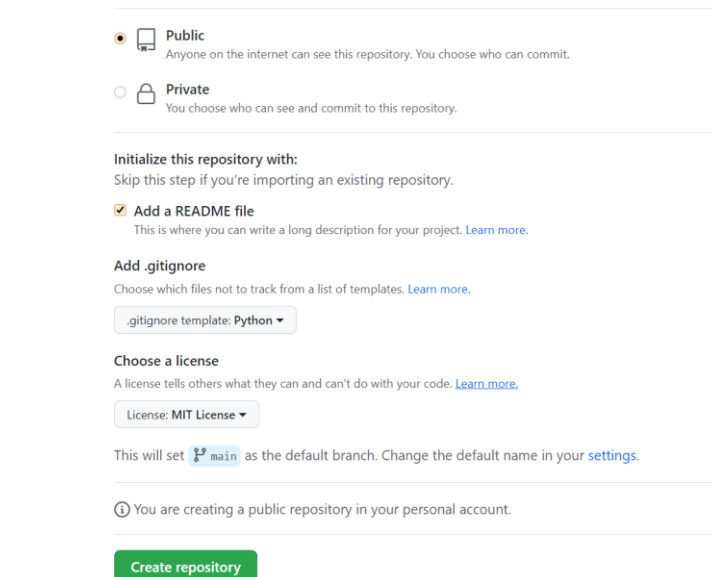
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2022

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

Порядок выполнения работы:

1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.



☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: Python ▼

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: MIT License ▼

This will set `main` as the default branch. Change the default name in your [settings](#).

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

Create repository

Рисунок 1 - Создание репозитория

2. Выполните клонирование созданного репозитория.

```
C:\Users\User>cd C:\Users\User\Desktop\2 кypc Python\lab 21
C:\Users\User\Desktop\2 кypc Python\lab 21>git clone https://github.com/aikanyshkauanbekova/2.18.git
Cloning into '2.18'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 4.51 KiB | 660.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.
C:\Users\User\Desktop\2 кypc Python\lab 21>
```

Рисунок 2 - Клонирование репозитория

3. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>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/User/Desktop/2 кypc Python/lab 10/lab-10/.git/hooks]
C:\Users\User\Desktop\2 кypc Python\lab 10\lab-10>
```

Рисунок 3 - Ветвление по модели git-flow

4. Формирование файла requirements.txt.

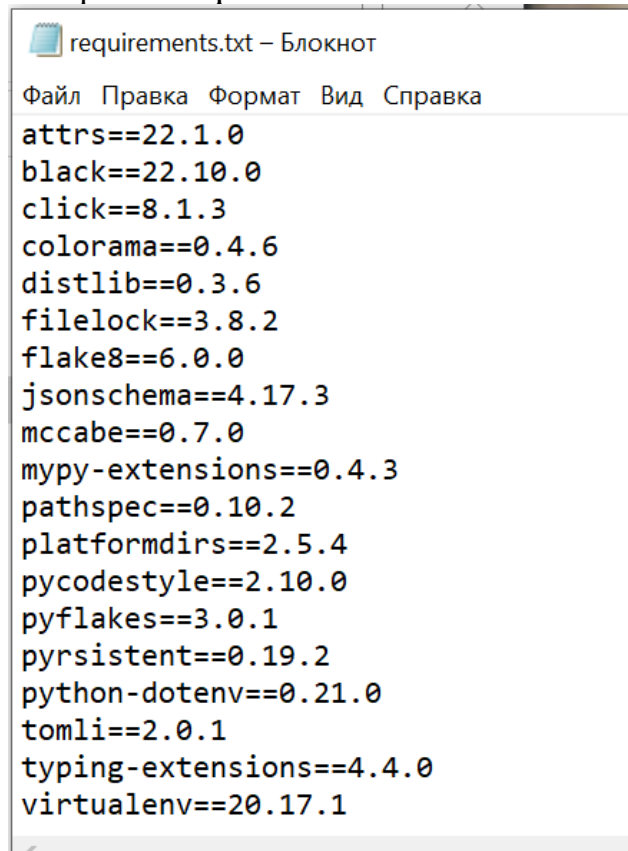


Рисунок 4 - Файл requirements.txt

5. Проработать примеры лабораторной работы.

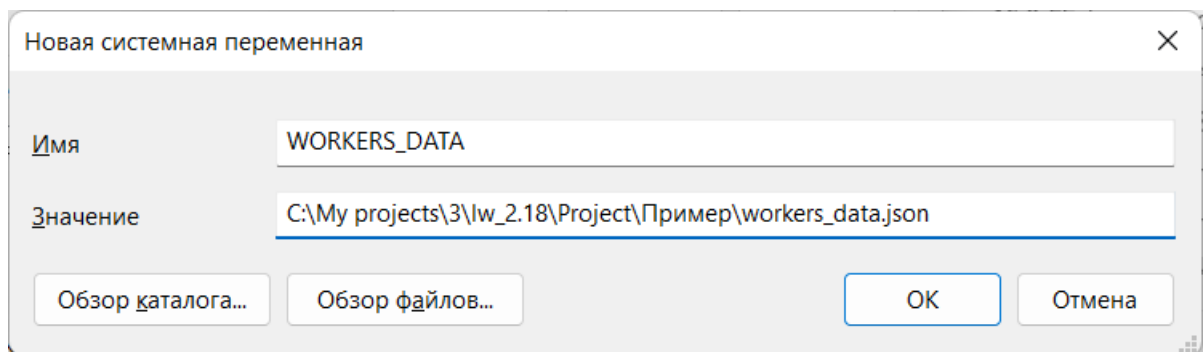


Рисунок 5 - Создание системной переменной WORKERS DATA

```
#!/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:
        # Заголовки 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 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(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 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(data_file, workers)

if __name__ == "__main__":
    main()

```

Рисунок 8 - Результат выполнения примера

6. Выполнить индивидуальные задания.

Задание 1

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

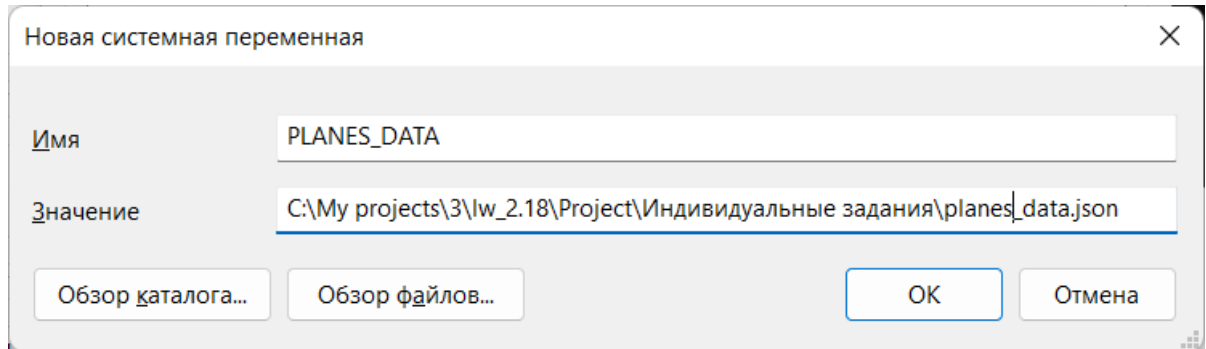


Рисунок 9 - Создание переменной окружения PLANES_DATA

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

import argparse
import json
import os.path
import sys

def add_route(routes, start, finish, number):
    """
    Добавить данные о маршруте
    """
    routes.append(
        {
            'start': start,
            'finish': finish,
            'number': number
        }
    )
    return routes

def display_route(routes):
    """
    Отобразить список маршрутов
    """
    if routes:
        line = '+-{}--{}--{}--{}--+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Начальный пункт",
                "Конечный пункт",
                "Номер маршрута"
            )
        )
```

```

    )
    print(line)

    for idx, worker in enumerate(routes, 1):
        print(
            '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                idx,
                worker.get('start', ''),
                worker.get('finish', ''),
                worker.get('number', 0)
            )
        )
    print(line)
else:
    print("Список маршрутов пуст.")

def select_route(routes, period):
    """
    Выбрать маршрут
    """
    result = []
    for employee in routes:
        if employee.get('number') == period:
            result.append(employee)

    return result

def save_routes(file_name, routes):
    """
    Сохранить всех работников в файл JSON.
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(routes, fout, ensure_ascii=False, indent=4)

def load_routes(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    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("routes")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new route"
    )

```



```

)
add.add_argument(
    "-s",
    "--start",
    action="store",
    required=True,
    help="The start of the route"
)
add.add_argument(
    "-f",
    "--finish",
    action="store",
    help="The finish of the route"
)
add.add_argument(
    "-n",
    "--number",
    action="store",
    type=int,
    required=True,
    help="The number of the route"
)
# Создать субпарсер для отображения всех маршрутов.
_ = subparsers.add_parser(
    "display",
    parents=[file_parser],
    help="Display all routes"
)
# Создать субпарсер для выбора маршрута.
select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Select the route"
)
select.add_argument(
    "-N",
    "--numb",
    action="store",
    type=int,
    required=True,
    help="The route"
)
# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)
# Загрузить все маршруты из файла, если файл существует.
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):
    routes = load_routes(data_file)
else:
    routes = []
# Добавить маршрут.
if args.command == "add":
    routes = add_route(
        routes,
        args.start,
        args.finish,
        args.number
    )
    is_dirty = True
# Отобразить все маршруты.
elif args.command == "display":

```

```

        display_route(routes)
# Выбрать требуемые маршруты.
elif args.command == "select":
    selected = select_route(routes, args.numb)
    display_route(selected)
# Сохранить данные в файл, если список маршрутов был изменен.
if is_dirty:
    save_routes(data_file, routes)

if __name__ == '__main__':
    main()

```

Рисунок 10 - Результат выполнения задания 1

Задание 2

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

```

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

import click
import json
import os
import sys
from dotenv import load_dotenv

@click.group()
def cli():
    pass

@cli.command()
@click.argument('data')
@click.option("-s", "--stay")
@click.option("-v", "--value")
@click.option("-n", "--number")
def adding(data, stay, number, value):
    """
    Добавление нового рейса
    """
    if os.path.exists(data):
        load_dotenv()
        dotenv_path = os.getenv("WORKERS_DATA")
        if not dotenv_path:
            click.secho('Такого файла нет', fg='red')
            sys.exit(1)
        if os.path.exists(dotenv_path):
            flights = opening(dotenv_path)
        else:
            flights = []
        flights.append(
            {
                'stay': stay,
                'number': number,
                'value': value
            }
        )
        with open(dotenv_path, "w", encoding="utf-8") as file_out:
            json.dump(flights, file_out, ensure_ascii=False, indent=4)
        click.secho("Рейс добавлен", fg='green')

```

```

    else:
        click.secho('Такого файла нет', fg='red')

@cli.command()
@click.argument('filename')
def table(filename):
    if os.path.exists(filename):
        load_dotenv()
        dotenv_path = os.getenv("WORKERS_DATA")
        if not dotenv_path:
            click.secho('Такого файла нет', fg='red')
            sys.exit(1)
        if os.path.exists(dotenv_path):
            flights = opening(dotenv_path)
        else:
            flights = []
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 20,
            '-' * 15,
            '-' * 16
        )
        """Вывод скиска рейсов"""
        print(line)
        print(
            '| {:^4} | {:^20} | {:^15} | {:^16} |'.format(
                "№",
                "Место прибытия",
                "Номер самолёта",
                "Тип")
        )
        print(line)
        for i, num in enumerate(flights, 1):
            print(
                '| {:<4} | {:<20} | {:<15} | {:<16} |'.format(
                    i,
                    num.get('stay', ''),
                    num.get('number', ''),
                    num.get('value', 0)
                )
            )
        print(line)

@cli.command()
@click.argument('filename')
@click.option("-t", "--typing")
def selecting(filename, typing):
    if os.path.exists(filename):
        load_dotenv()
        dotenv_path = os.getenv("WORKERS_DATA")
        if not dotenv_path:
            click.secho('Такого файла нет', fg='red')
            sys.exit(1)
        if os.path.exists(dotenv_path):
            flights = opening(dotenv_path)
        else:
            flights = []
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 20,
            '-' * 15,
            '-' * 16
        )
        """Выбор рейсов по типу самолёта"""
        count = 0
        print(line)
        print(

```

```

        '| {:^4} | {:^20} | {:^15} | {:^16} |'.format(
            "№",
            "Место прибытия",
            "Номер самолёта",
            "Тип"))
    print(line)
    for i, num in enumerate(flights, 1):
        if typing == num.get('value', ''):
            count += 1
            print(
                '| {:<4} | {:<20} | {:<15} | {:<16} |'.format(
                    count,
                    num.get('stay', ''),
                    num.get('number', ''),
                    num.get('value', 0)))
    print(line)

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

def main():
    cli()

if __name__ == '__main__':
    main()

```

Рисунок 11 - Результат выполнения задания 2

Контрольные вопросы:

1. Каково назначение переменных окружения?

Переменная среды (переменная окружения) – это короткая ссылка на какой-либо объект в системе. С помощью таких сокращений, например, можно создавать универсальные пути для приложений, которые будут работать на любых ПК, независимо от имен пользователей и других параметров.

2. Какая информация может храниться в переменных окружения?

Переменная окружения может хранить информацию о путях к исполняемым файлам, заданном по умолчанию текстовом редакторе, браузере, языковых параметрах (локали) системы или настройках раскладки клавиатуры.

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

Получить информацию о существующих переменных можно в

свойствах системы. Для этого кликаем по ярлыку Компьютера на рабочем столе правой кнопкой мыши и выбираем соответствующий пункт.

Переходим в «Дополнительные параметры».

В открывшемся окне с вкладкой «Дополнительно» нажимаем кнопку «Переменные среды».

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. Обратите внимание на то, чтобы такое название еще не было использовано (просмотрите списки).

В поле Значение указываем путь.

6. Что представляют собой переменные окружения в ОС Linux?

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

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`.

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:

```
$ NEW_VAR='Ravesli.com'
```

12. Как установить переменные окружения в Linux?

Команда `export` используется для задания переменных окружения.

13. Для чего необходимо делать переменные окружения Linux постоянными?

Чтобы переменная сохранялась после закрытия сеанса оболочки.

14. Для чего используется переменная окружения PYTHONHOME?

Переменная среды `PYTHONHOME` изменяет расположение стандартных библиотек Python.

15. Для чего используется переменная окружения PYTHONPATH?

Переменная среды `PYTHONPATH` изменяет путь поиска по умолчанию для файлов модуля.

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

`PYTHONSTARTUP`, `PYTHONOPTIMIZE`, `PYTHONBREAKPOINT`,
`PYTHONDEBUG`, `PYTHONINSPECT`, `PYTHONUNBUFFERED`,

PYTHONVERBOSE, PYTHONCASEOK, PYTHONDONTWRITEBYTECODE,
PYTHONPYCACHEPREFIX, PYTHONHASHSEED, PYTHONIOENCODING,
PYTHONNOUSERSITE, PYTHONUSERBASE, PYTHONWARNINGS,
PYTHONFAULTHANDLER, PYTHONTRACEMALLOC,
PYTHONPROFILEIMPORTTIME, PYTHONASYNCIODEBUG,
PYTHONMALLOC, PYTHONMALLOCSTATS,
PYTHONLEGACYWINDOWSFSENCODING,
PYTHONLEGACYWINDOWSTDIO, PYTHONCOERCECLOCALE,
PYTHONDEVMODE, PYTHONUTF8, PYTHONWARNDEFAULTENCODING,
PYTHONTHREADDEBUG, PYTHONDUMPPREFS.

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

Для доступа к переменным среды в Python используется объект `os.environ`.

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

Для чтения значений переменных мы используем модуль `os`, а модуль `sys` — для прекращения работы приложения.

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

Для присвоения значения любой переменной среды используется функция `setdefault()`.

Вывод: были приобретены навыки по работе с переменными окружения с помощью языка программирования Python версии 3.x.