МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №2.18 по дисциплине основы программной инженерии

Выполнил:
Шальнев Владимир Сергеевич,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:
Доцент кафедры
прикладной математики и
компьютерной безопасности,
Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2022 г.

Выполнение:

```
#!/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.")
```

```python
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()
```

Пример 1

```
F:\pythonProject\lab_2_6\example>python ex1.py add --name="Проверка Проверкович" --post="Стажер" --year=2015

F:\pythonProject\lab_2_6\example>python ex1.py display
+------+-----------------------------+---------------------+----------+
|  No  |           Ф.И.О.            |      Должность      |   Год    |
+------+-----------------------------+---------------------+----------+
|    1 | Проверка Проверкович        | Стажер              |     2015 |
+------+-----------------------------+---------------------+----------+
```

Результат работы примера 1

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


import sys
import argparse
import os
from datetime import datetime
import json
import jsonschema
from jsonschema import validate


def main_loop():
    trains = []
    while True:
        command = get_command()
        if command == 'exit':
            break
        elif command == 'add':
            trains.append(add_train())
            if len(trains) > 1:
                trains.sort(key=lambda item: item.get('destination', ''))
        elif command == 'list':
            print_list(trains)
        elif command.startswith('select '):
            select_train(command, trains)
        elif command.startswith("save "):
            parts = command.split(maxsplit=1)
            file_name = parts[1]
            save_trains(file_name, trains)

        elif command.startswith("load"):
            parts = command.split(maxsplit=1)
            if len(parts) > 1:
                file_name = parts[1]
```

```
            else:
                os.environ.setdefault('TRAIN_DATA', 'data.json')
                file_name = os.environ.get("TRAIN_DATA")
            if not file_name:
                print("The data file name is absent", file=sys.stderr)
                sys.exit(1)
            trains = load_trains(file_name)
        elif command == 'help':
            print_help()
        else:
            print(f"Неизвестная команда {command}", file=sys.stderr)


def get_command():
    return input(">>> ").lower()


def add_train(destination=None, number=None, time=None):
    print(destination)
    print(number)
    print(time)
    if destination is None \
            and number is None \
            and time is None:
        destination = input("Название пункта назначения? ")
        number = input("Номер поезда? ")
        time = input("Время отправления ЧЧ:ММ? ")
    train = {
        'destination': destination,
        'number': int(number),
        'time': datetime.strptime(time, '%H:%M'),
    }
    return train


def print_list(trains):
    line = '+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 28,
        '-' * 14,
        '-' * 19
    )
    print(line)
    print(
        '| {:^4} | {:^28} | {:^14} | {:^19} |'.format(
            "No",
            "Название пункта назначения",
            "Номер поезда",
            "Время отправления"
        )
    )
    print(line)
    for idx, train in enumerate(trains, 1):
        print(
            '| {:>4} | {:<28} | {:<14} | {:>19} |'.format(
                idx,
                train.get('destination', ''),
                train.get('number', ''),
                train.get('time', 0).strftime("%H:%M")
            )
        )
    print(line)
```

```
def print_help():
    print("Список команд:\n")
    print("add - добавить отправление;")
    print("list - вывести список отправлений;")
    print("select <ЧЧ:ММ> - вывод на экран информации о "
          "поездах, отправляющихся после этого времени;")
    print("save <имя файла.json> - сохранить в файл")
    print("load <имя файла.json> - загрузить из файла")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")


def select_train(command, trains):
    count = 0
    parts = command.split(' ', maxsplit=1)
    time = datetime.strptime(parts[1], '%H:%M')
    for train in trains:
        if train.get("time") > time:
            count += 1
            print(
                '{:>4}: {} {}'.format(
                    count,
                    train.get('destination', ''),
                    train.get("number")
                )
            )
    if count == 0:
        print("Отправлений позже этого времени нет.")


def save_trains(file_name, trains):
    if file_name.split('.', maxsplit=1)[-1] != "json":
        print("Несоответствующий формат файла", file=sys.stderr)
        return False
    try:
        list_of_files = os.listdir(os.path.split(os.getcwd())[0])
        index = list_of_files.index('.gitignore')
        flag = True
    except ValueError:
        flag = False
        print("Файл .gitignore не найден", file=sys.stderr)

    if flag:
        file = f"{os.path.split(os.getcwd())[0]}/.gitignore"
        with open(file, 'a', encoding="utf-8") as git_file:
            git_file.write(f"{file_name}\n")

    for i in trains:
        i['time'] = i['time'].time().strftime("%H:%M")
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(trains, fout, ensure_ascii=False, indent=4)
    return True


def validate_json(json_data):
    with open("schema.json", "r", encoding="utf-8") as json schema:
        schema = json.load(json_schema)
    try:
        validate(instance=json_data, schema=schema)
    except jsonschema.exceptions.ValidationError as err:
        return False
    return True
```

```python
def load_trains(file_name):
    if file_name.split('.', maxsplit=1)[-1] != "json":
        print("Несоответствующий формат файла", file=sys.stderr)
        return []

    if not os.path.exists(f"{os.getcwd()}/{file_name}"):
        print("Файл не существует", file=sys.stderr)
        return []

    with open(file_name, "r", encoding="utf-8") as fin:
        data = []
        try:
            data = json.load(fin)
            flag = validate_json(data)
        except Exception as e:
            print("Некоректный файл", file=sys.stderr)
            flag = False
        if flag:
            for i in data:
                i['time'] = datetime.strptime(i['time'], '%H:%M')
            return data
        else:
            return []


def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--filename",
        action="store",
        help="Имя файла для хранения данных"
    )
    parser = argparse.ArgumentParser("trains")
    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="Добавить новое отправление"
    )
    add.add_argument(
        "-trd",
        "--train_dest",
        action="store",
        required=True,
        help="Пункт назначения поезда"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        help="Номер поезда"
    )
    add.add_argument(
        "-t",
        "--time",
        action="store",
        required=True,
        help="Время отправления поезда"
    )
```

```
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Вывод информации о всех поездах"
    )
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Выбрать поезда, отправляющиеся "
             "позже указанного времени"
    )
    select.add_argument(
        "-t",
        "--time",
        action="store",
        required=True,
        help="Время отправления"
    )
    args = parser.parse_args(command_line)
    try:
        data_file = args.filename
        if not data_file:
            os.environ.setdefault('TRAIN_DATA', 'data.json')
            data_file = os.environ.get("TRAIN_DATA")
        if not data_file:
            print("The data file name is absent", file=sys.stderr)
            sys.exit(1)
        is_dirty = False
        if os.path.exists(data_file):
            trains = load_trains(data_file)
        else:
            trains = []
        if args.command == "add":
            trains.append(add_train(
                args.train_dest,
                args.number,
                args.time
            ))
            is_dirty = True
        elif args.command == "display":
            print_list(trains)
        elif args.command == "select":
            select_train(f"select {args.time}", trains)
        if is_dirty:
            save_trains(data_file, trains)
    except Exception as e:
        main_loop()


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

Индивидуальное задание №1

```
F:\pythonProject\lab_2_6\task_1>python task.py display
+------+-------------------------------+---------------+---------------------+
|  No  |  Название пункта назначения   |  Номер поезда |  Время отправления  |
+------+-------------------------------+---------------+---------------------+
|    1 | Moscow                        | 123           |               12:55 |
+------+-------------------------------+---------------+---------------------+

F:\pythonProject\lab_2_6\task_1>python task.py add -h
usage: trains add [-h] [--filename FILENAME] -trd TRAIN_DEST [-n NUMBER] -t TIME

optional arguments:
  -h, --help            show this help message and exit
  --filename FILENAME   Имя файла для хранения данных
  -trd TRAIN_DEST, --train_dest TRAIN_DEST
                        Пункт назначения поезда
  -n NUMBER, --number NUMBER
                        Номер поезда
  -t TIME, --time TIME  Время отправления поезда

F:\pythonProject\lab_2_6\task_1>python task.py add -trd="Stavropol" -n=250 -t=12:00
Stavropol
250
12:00

F:\pythonProject\lab_2_6\task_1>python task.py display
+------+-------------------------------+---------------+---------------------+
|  No  |  Название пункта назначения   |  Номер поезда |  Время отправления  |
+------+-------------------------------+---------------+---------------------+
|    1 | Moscow                        | 123           |               12:55 |
|    2 | Stavropol                     | 250           |               12:00 |
+------+-------------------------------+---------------+---------------------+
```

Пример работы индивидуального заданий №1

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


import sys
import argparse
import os
from datetime import datetime
import json
import jsonschema
from jsonschema import validate
from dotenv import load_dotenv


def main_loop():
    trains = []
    while True:
        command = get_command()
        if command == 'exit':
```

```
            break
        elif command == 'add':
            trains.append(add_train())
            if len(trains) > 1:
                trains.sort(key=lambda item: item.get('destination', ''))
        elif command == 'list':
            print_list(trains)
        elif command.startswith('select '):
            select_train(command, trains)
        elif command.startswith("save "):
            parts = command.split(maxsplit=1)
            file_name = parts[1]
            save_trains(file_name, trains)

        elif command.startswith("load"):
            parts = command.split(maxsplit=1)
            if len(parts) > 1:
                file_name = parts[1]
            else:
                os.environ.setdefault('TRAIN_DATA', 'data.json')
                file_name = os.environ.get("TRAIN_DATA")
            if not file_name:
                print("The data file name is absent", file=sys.stderr)
                sys.exit(1)
            trains = load_trains(file_name)
        elif command == 'help':
            print_help()
        else:
            print(f"Неизвестная команда {command}", file=sys.stderr)


def get_command():
    return input(">>> ").lower()


def add_train(destination=None, number=None, time=None):
    print(destination)
    print(number)
    print(time)
    if destination is None \
            and number is None \
            and time is None:
        destination = input("Название пункта назначения? ")
        number = input("Номер поезда? ")
        time = input("Время отправления ЧЧ:ММ? ")
    train = {
        'destination': destination,
        'number': int(number),
        'time': datetime.strptime(time, '%H:%M'),
    }
    return train


def print_list(trains):
    line = '+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 28,
        '-' * 14,
        '-' * 19
    )
    print(line)
    print(
        '| {:^4} | {:^28} | {:^14} | {:^19} |'.format(
            "No",
```

```
            "Название пункта назначения",
            "Номер поезда",
            "Время отправления"
        )
    )
    print(line)
    for idx, train in enumerate(trains, 1):
        print(
            '| {:>4} | {:<28} | {:<14} | {:>19} |'.format(
                idx,
                train.get('destination', ''),
                train.get('number', ''),
                train.get('time', 0).strftime("%H:%M")
            )
        )
    print(line)


def print_help():
    print("Список команд:\n")
    print("add - добавить отправление;")
    print("list - вывести список отправлений;")
    print("select <ЧЧ:ММ> - вывод на экран информации о "
          "поездах, отправляющихся после этого времени;")
    print("save <имя файла.json> - сохранить в файл")
    print("load <имя файла.json> - загрузить из файла")
    print("help - отобразить справку;")
    print("exit - завершить работу с программой.")


def select_train(command, trains):
    count = 0
    parts = command.split(' ', maxsplit=1)
    time = datetime.strptime(parts[1], '%H:%M')
    for train in trains:
        if train.get("time") > time:
            count += 1
            print(
                '{:>4}: {} {}'.format(
                    count,
                    train.get('destination', ''),
                    train.get("number")
                )
            )
    if count == 0:
        print("Отправлений позже этого времени нет.")


def save_trains(file_name, trains):
    if file_name.split('.', maxsplit=1)[-1] != "json":
        print("Несоответствующий формат файла", file=sys.stderr)
        return False
    try:
        list_of_files = os.listdir(os.path.split(os.getcwd())[0])
        index = list_of_files.index('.gitignore')
        flag = True
    except ValueError:
        flag = False
        print("Файл .gitignore не найден", file=sys.stderr)

    if flag:
        file = f"{os.path.split(os.getcwd())[0]}/.gitignore"
        with open(file, 'a', encoding="utf-8") as git_file:
            git_file.write(f"{file_name}\n")
```

```
    for i in trains:
        i['time'] = i['time'].time().strftime("%H:%M")
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(trains, fout, ensure_ascii=False, indent=4)
    return True


def validate_json(json_data):
    with open("schema.json", "r", encoding="utf-8") as json_schema:
        schema = json.load(json_schema)
    try:
        validate(instance=json_data, schema=schema)
    except jsonschema.exceptions.ValidationError as err:
        return False
    return True


def load_trains(file_name):
    if file_name.split('.', maxsplit=1)[-1] != "json":
        print("Несоответствующий формат файла", file=sys.stderr)
        return []

    if not os.path.exists(f"{os.getcwd()}/{file_name}"):
        print("Файл не существует", file=sys.stderr)
        return []

    with open(file_name, "r", encoding="utf-8") as fin:
        data = []
        try:
            data = json.load(fin)
            flag = validate_json(data)
        except Exception as e:
            print("Некоректный файл", file=sys.stderr)
            flag = False
        if flag:
            for i in data:
                i['time'] = datetime.strptime(i['time'], '%H:%M')
            return data
        else:
            return []


def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--filename",
        action="store",
        help="Имя файла для хранения данных"
    )
    parser = argparse.ArgumentParser("trains")
    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="Добавить новое отправление"
    )
    add.add_argument(
        "-trd",
```

```
        "--train_dest",
        action="store",
        required=True,
        help="Пункт назначения поезда"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        help="Номер поезда"
    )
    add.add_argument(
        "-t",
        "--time",
        action="store",
        required=True,
        help="Время отправления поезда"
    )
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Вывод информации о всех поездах"
    )
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Выбрать поезда, отправляющиеся "
             "позже указанного времени"
    )
    select.add_argument(
        "-t",
        "--time",
        action="store",
        required=True,
        help="Время отправления"
    )
    dotenv_path = os.path.join(os.path.dirname(__file__), '.env')
    if os.path.exists(dotenv_path):
        load_dotenv(dotenv_path)
    args = parser.parse_args(command_line)
    try:
        data_file = args.filename
        if not data_file:
            os.environ.setdefault('TRAIN_DATA', 'data.json')
            data_file = os.environ.get("TRAIN_DATA")
        if not data_file:
            print("The data file name is absent", file=sys.stderr)
            sys.exit(1)
        is_dirty = False
        if os.path.exists(data_file):
            trains = load_trains(data_file)
        else:
            trains = []
        if args.command == "add":
            trains.append(add_train(
                args.train_dest,
                args.number,
                args.time
            ))
            is_dirty = True
        elif args.command == "display":
            print_list(trains)
        elif args.command == "select":
            select_train(f"select {args.time}", trains)
```

```
        if is_dirty:
            save_trains(data_file, trains)
    except Exception as e:
        main_loop()


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

Индивидуальное задание №2

```
F:\pythonProject\lab_2_6\task_2>python task.py display
+------+------------------------------+----------------+----------------------+
|  No  |  Название пункта назначения  |  Номер поезда  |  Время отправления   |
+------+------------------------------+----------------+----------------------+
|    1 | MOscow                       | 125            |                12:55 |
|    2 | Stavropol                    | 126            |                13:40 |
+------+------------------------------+----------------+----------------------+

F:\pythonProject\lab_2_6\task_2>python task.py add -trd="Stavropol" -n=250 -t=12:00
Stavropol
250
12:00

F:\pythonProject\lab_2_6\task_2>python task.py display
+------+------------------------------+----------------+----------------------+
|  No  |  Название пункта назначения  |  Номер поезда  |  Время отправления   |
+------+------------------------------+----------------+----------------------+
|    1 | MOscow                       | 125            |                12:55 |
|    2 | Stavropol                    | 126            |                13:40 |
|    3 | Stavropol                    | 250            |                12:00 |
+------+------------------------------+----------------+----------------------+

F:\pythonProject\lab_2_6\task_2>
```

Пример работы индивидуального заданий №2

Ответы на 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?

Пути к различным файлам и папкам

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

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. Например, если ввести в «Командную строку»

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?

1. Нажимаем кнопку Создать. Сделать это можно как в пользовательском разделе, так и в системном.

2. Вводим имя, например, desktop. Обратите внимание на то, чтобы такое название еще не было использовано (просмотрите списки).

3. В поле Значение указываем путь до папки Рабочий стол:

C:\Users\Имя_пользователя\Desktop

4. Нажимаем ОК. Повторяем это действие во всех открытых окнах (см. выше).

5. Перезапускаем Проводник и консоль или целиком систему.

6. 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.

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?

printenv HOME

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?

NEW_VAR='Ravesli.com'
echo $NEW_VAR
Либо
set | grep NEW_VAR

12. Как установить переменные окружения в Linux?

export MY_NEW_VAR="My New Var"

13. Для чего необходимо делать переменные окружения Linux постоянными?

Если вы хотите, чтобы переменная сохранялась после закрытия сеанса оболочки

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.

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.

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
PYTHONLEGACYWINDOWSSTDIO
PYTHONCOERCECLOCALE
PYTHONDEVMODE
PYTHONUTF8
PYTHONWARNDEFAULTENCODING
PYTHONTHREADDEBUG
PYTHONDUMPREFS

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.

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

Создаём Python-файл со следующим кодом. Для проверки переменной DEBUG на истинность здесь используется функция get() . Программа выводит разные сообщения в зависимости от значения переменной.

os.environ.get('DEBUG')

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

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