Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4
дисциплины «Анализ данных»

Выполнил:
Кожуховский Виктор Андреевич
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Работа с переменными окружения в Python3

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

Порядок выполнения работы:

1. Изучил теоретический материал работы.

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

3. Выполнил клонирование созданного репозитория.

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

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

6. Создал проект в папке репозитория.

7. Проработал примеры лабораторной работы.

8. Выполнил индивидуальное задание.

Задание 1

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

Код:

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

import argparse
import json
import os
from datetime import datetime
import jsonschema

person_schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "surname": {"type": "string"},
        "date_of_birth": {"type": "string", "format": "date"},
        "zodiac_sign": {"type": "string"}
    },
    "required": ["name", "surname", "date_of_birth", "zodiac_sign"]
}
def validate_person(person_data, schema):
    try:
        jsonschema.validate(person_data, schema)
        return True
    except jsonschema.exceptions.ValidationError as e:
        print(f"Данные человека не соответствуют схеме: {e}")
        return False
```

```
def add_person(people, name, surname, date_of_birth, zodiac_sign):
    """
    Добавление нового человека в список.
    Список сортируется по знаку зодиака после добавления нового элемента.
    """
    date_of_birth = datetime.strptime(date_of_birth, '%d.%m.%Y')

    person = {
        'name': name,
        'surname': surname,
        'date_of_birth': date_of_birth,
        'zodiac_sign': zodiac_sign
    }

    people.append(person)
    people.sort(key=lambda item: item.get('zodiac_sign', ''))
    return people

def list_people(people):
    """
    Вывод таблицы людей.
    """
    line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 20,
        '-' * 20,
        '-' * 15,
        '-' * 13
    )
    print(line)
    print(
        '| {:^4} | {:^20} | {:^20} | {:^15} | {:^12} |'.format(
            "№",
            "Имя",
            "Фамилия",
            "Знак Зодиака",
            "Дата рождения"
        )
    )
    print(line)

    for idx, person in enumerate(people, 1):
        birth_date_str = person.get('date_of_birth').strftime('%d.%m.%Y')
        print(
            '| {:^4} | {:<20} | {:<20} | {:<15} | {:<13} |'.format(
                idx,
                person.get('name', ''),
                person.get('surname', ''),
                person.get('zodiac_sign', ''),
                birth_date_str
            )
        )
    print(line)

def select_people(people, month):
    """
    Вывести список людей, родившихся в заданном месяце.
    """
    count = 0
    for person in people:
        if person.get('date_of_birth').month == month:
            count += 1
            print('{:>4}: {} {}'.format(count, person.get(
                'name', ''), person.get('surname', '')))

    if count == 0:
        print("Люди, родившиеся в указанном месяце, не найдены.")

def save_people(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    staff_formatted = [{**person, 'date_of_birth': person.get(
        'date_of_birth').strftime('%d.%m.%Y')} for person in staff]
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        json.dump(staff_formatted, fout, ensure_ascii=False, indent=4)
```

```
def load_people(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:
        staff_loaded = json.load(fin)
        result_people = []
        cnt = 0
        for person in staff_loaded:
            cnt += 1
            if validate_person(person, person_schema):
                try:
                    person['date_of_birth'] = datetime.strptime(
                        person['date_of_birth'], '%d.%m.%Y')
                    result_people.append(person)
                except:
                    print(
                        f"Ошибка при разборе даты в записи, пропуск записи {cnt}.")
            else:
                print("Неверные данные человека, пропуск записи.")
        return result_people

def main():
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "-f", "--filename",
        default=os.environ.get("DATA_ANALYSIS", "default.json"),
        help="The data file name (default: %(default)s)"
    )
    # Создание основного парсера.
    parser = argparse.ArgumentParser(description="Управление списком людей")
subparsers = parser.add_subparsers(dest="command")

    # Создание парсера для добавления человека.
    parser_add = subparsers.add_parser(
        'add', parents=[file_parser], help="Добавить человека")
    parser_add.add_argument("-n", "--name", help="Имя человека",
                            required=True)
    parser_add.add_argument("-s", "--surname", help="Фамилия человека",
                            required=True)
    parser_add.add_argument(
        "-d", "--date_of_birth", help="Дата рождения (формат ДД.ММ.ГГГГ)",
        required=True)
    parser_add.add_argument("-z", "--zodiac_sign", help="Знак зодиака",
                            required=True)

    # Создание парсера для вывода списка людей.
    _ = subparsers.add_parser(
        'list', parents=[file_parser], help="Вывести список людей")

    # Создание парсера для выбора человека по месяцу рождения.
    parser_select = subparsers.add_parser(
        'select', parents=[file_parser], help="Выбрать людей по месяцу рождения")
    parser_select.add_argument(
        "-m", "--month", type=int, help="Месяц рождения",
        required=True)

    # Разбираем аргументы командной строки.
    args = parser.parse_args()

    is_dirty = False

    filename = os.path.join("data", args.filename)

    if os.path.exists(filename):
        people = load_people(filename)
    else:
        people = []

    # Определяем, какую команду нужно выполнить.
    if args.command == 'add':
        people = add_person(people, args.name, args.surname,
                            args.date_of_birth, args.zodiac_sign)
        is_dirty = True

    elif args.command == 'list':
        list_people(people)
```

```
    elif args.command == 'select':
        select_people(people, args.month)

    if is_dirty:
        save_people(filename, people)

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

```
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 2.py list
+-----+----------------------+----------------------+----------------+---------------+
|  №  |         Имя          |       Фамилия        |  Знак Зодиака  | Дата рождения |
+-----+----------------------+----------------------+----------------+---------------+
|  1  | test                 | testd                | asd            | 23.02.2000    |
|  2  | test                 | testd                | asd            | 23.02.2000    |
|  3  | test                 | testd                | asd            | 23.02.2000    |
+-----+----------------------+----------------------+----------------+---------------+
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 3.py list -f default2.json
+-----+----------------------+----------------------+----------------+---------------+
|  №  |         Имя          |       Фамилия        |  Знак Зодиака  | Дата рождения |
+-----+----------------------+----------------------+----------------+---------------+
|  1  | test                 | testd                | asd            | 23.02.2000    |
+-----+----------------------+----------------------+----------------+---------------+
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 2.py add -n test -s testd -d 23.02.2000 -z asd -f default2.json
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 2.py list -f default2.json
+-----+----------------------+----------------------+----------------+---------------+
|  №  |         Имя          |       Фамилия        |  Знак Зодиака  | Дата рождения |
+-----+----------------------+----------------------+----------------+---------------+
|  1  | test                 | testd                | asd            | 23.02.2000    |
|  2  | test                 | testd                | asd            | 23.02.2000    |
+-----+----------------------+----------------------+----------------+---------------+
```

Рисунок 1. Выполнение кода 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. Код:

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

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

person_schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "surname": {"type": "string"},
        "date_of_birth": {"type": "string", "format": "date"},
        "zodiac_sign": {"type": "string"}
    },
    "required": ["name", "surname", "date_of_birth", "zodiac_sign"]
}

def validate_person(person_data, schema):
    try:
        jsonschema.validate(person_data, schema)
        return True
    except jsonschema.exceptions.ValidationError as e:
        print(f"Данные человека не соответствуют схеме: {e}")
        return False

def add_person(people, name, surname, date_of_birth, zodiac_sign):
    """
    Добавление нового человека в список.
```

```
    Список сортируется по знаку зодиака после добавления нового элемента.
    """

date_of_birth = datetime.strptime(date_of_birth, '%d.%m.%Y')

    person = {
        'name': name,
        'surname': surname,
        'date_of_birth': date_of_birth,
        'zodiac_sign': zodiac_sign
    }

    people.append(person)
    people.sort(key=lambda item: item.get('zodiac_sign', ''))
    return people

def list_people(people):
    """
    Вывод таблицы людей.
    """
    line = '+-{}-+-{}-+-{}-+-{}-+-{}-+'.format(
        '-' * 4,
        '-' * 20,
        '-' * 20,
        '-' * 15,
        '-' * 13
    )
    print(line)
    print(
        '| {:^4} | {:^20} | {:^20} | {:^15} | {:^12} |'.format(
            "№",
            "Имя",
            "Фамилия",
            "Знак Зодиака",
            "Дата рождения"
        )
    )
    print(line)

    for idx, person in enumerate(people, 1):
        birth_date_str = person.get('date_of_birth').strftime('%d.%m.%Y')
        print(
            '| {:^4} | {:<20} | {:<20} | {:<15} | {:<13} |'.format(
                idx,
                person.get('name', ''),
                person.get('surname', ''),
                person.get('zodiac_sign', ''),
                birth_date_str
            )
        )

    print(line)

def select_people(people, month):
    """
    Вывести список людей, родившихся в заданном месяце.
    """
    count = 0
    for person in people:
        if person.get('date_of_birth').month == month:
            count += 1
            print('{:>4}: {} {}'.format(count, person.get(
                'name', ''), person.get('surname', '')))

    if count == 0:
        print("Люди, родившиеся в указанном месяце, не найдены.")

def save_people(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    staff_formatted = [{**person, 'date_of_birth': person.get(
        'date_of_birth').strftime('%d.%m.%Y')} for person in staff]
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        json.dump(staff_formatted, fout, ensure_ascii=False, indent=4)

def load_people(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:
        staff_loaded = json.load(fin)
        result_people = []
        cnt = 0
        for person in staff_loaded:
            cnt += 1
            if validate_person(person, person_schema):
                try:
                    person['date_of_birth'] = datetime.strptime(
                        person['date_of_birth'], '%d.%m.%Y')
                    result_people.append(person)
                except:
                    print(
                        f"Ошибка при разборе даты в записи, пропуск записи {cnt}.")
            else:
                print("Неверные данные человека, пропуск записи.")
        return result_people

def main():
    load_dotenv()
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "-f", "--filename",
        default=os.environ.get("DATA_ANALYSIS", "default.json"),
        help="The data file name (default: %(default)s)"
    )
    # Создание основного парсера.
    parser = argparse.ArgumentParser(description="Управление списком людей")
    subparsers = parser.add_subparsers(dest="command")

    # Создание парсера для добавления человека.
    parser_add = subparsers.add_parser(
        'add', parents=[file_parser], help="Добавить человека")
    parser_add.add_argument("-n", "--name", help="Имя человека",
                            required=True)
    parser_add.add_argument("-s", "--surname", help="Фамилия человека",
                            required=True)
    parser_add.add_argument(
        "-d", "--date_of_birth", help="Дата рождения (формат ДД.ММ.ГГГГ)",
        required=True)
    parser_add.add_argument("-z", "--zodiac_sign", help="Знак зодиака",
                            required=True)

    # Создание парсера для вывода списка людей.
    _ = subparsers.add_parser(
        'list', parents=[file_parser], help="Вывести список людей")

    # Создание парсера для выбора человека по месяцу рождения.
    parser_select = subparsers.add_parser(
        'select', parents=[file_parser], help="Выбрать людей по месяцу рождения")
    parser_select.add_argument(
        "-m", "--month", type=int, help="Месяц рождения",
        required=True)

    # Разбираем аргументы командной строки.
    args = parser.parse_args()

    is_dirty = False

    filename = os.path.join("data", args.filename)

    if os.path.exists(filename):
        people = load_people(filename)
    else:
        people = []

    # Определяем, какую команду нужно выполнить.
    if args.command == 'add':
        people = add_person(people, args.name, args.surname,
                            args.date_of_birth, args.zodiac_sign)
        is_dirty = True

    elif args.command == 'list':
        list_people(people)
```

```
    elif args.command == 'select':
        select_people(people, args.month)

    if is_dirty:
        save_people(filename, people)

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

```
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 3.py list -f default2.json
+-----+----------------------+----------------------+----------------+--------------+
|  №  |         Имя          |       Фамилия        |  Знак Зодиака  | Дата рождения |
+-----+----------------------+----------------------+----------------+--------------+
|  1  | test                 | testd                | asd            | 23.02.2000   |
|  2  | test                 | testd                | asd            | 23.02.2000   |
+-----+----------------------+----------------------+----------------+--------------+
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 2.py add -n tet -s ted -d 23.05.2000 -z sd -f default2.json
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 3.py add -n st -s tstd -d 23.03.2000 -z ad -f default2.json
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4> python 3.py list -f default2.json
+-----+----------------------+----------------------+----------------+--------------+
|  №  |         Имя          |       Фамилия        |  Знак Зодиака  | Дата рождения |
+-----+----------------------+----------------------+----------------+--------------+
|  1  | st                   | tstd                 | ad             | 23.03.2000   |
|  2  | test                 | testd                | asd            | 23.02.2000   |
|  3  | test                 | testd                | asd            | 23.02.2000   |
|  4  | tet                  | ted                  | sd             | 23.05.2000   |
+-----+----------------------+----------------------+----------------+--------------+
(da4-py3.11) PS C:\Users\viktor\Desktop\скфу\DataAnalysis\Data4>
```

Рисунок 2. Выполнение кода задания 2

9. Зафиксировал сделанные изменения в репозитории.

10. Добавил отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксировал изменения.

11. Выполнил слияние ветки для разработки с веткой master/main.

12. Отправил сделанные изменения на сервер GitHub.

Контрольные вопросы:

1. Каково назначение переменных окружения?

Переменные окружения используются для передачи информации процессам, которые запущены в оболочке.

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?

Нужно открыть окно свойства системы и нажать на 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 дает возможность не указывать даже расширение файла, если оно прописано в ее значениях.

5. Как создать или изменить переменную окружения в Windows?

В окне “Переменные среды” нужно нажать на кнопку “Создать”, затем ввести имя переменной и 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.

HOME – домашняя директория текущего пользователя.

SHELL – путь к оболочке текущего пользователя.

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.

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.

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.

PS1 – определяет внешний вид строки приглашения ввода новых команд.

PS2 – вторичная строка приглашения.

UID – идентификатор текущего пользователя.

11. Как установить переменные оболочки в Linux?

$ NEW_VAR='значение'

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.

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

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

value = os.environ.get('MY_ENV_VARIABLE')

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

if os.environ[key_value]:

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

Для присвоения значения любой переменной среды используется функция os.environ.setdefault(«Переменная», «Значение»)