Институт цифрового развития
Кафедра инфокоммуникаций

«Работа с переменными окружения в Python3»

ОТЧЕТ
по лабораторной работе №2.18
дисциплины
«Основы программной инженерии»

Выполнил:
Борсуков Владислав Олегович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Проверил:
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Примеры:

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

import os


if __name__ == '__main__':
    print("The keys and values of all environment variables:")
    for key in os.environ:
        print(key, '=>', os.environ[key])
    print("The value of HOME is: ", os.environ['home'])

```

Рисунок 1 – Пример 1

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

import os
import sys


if __name__ == '__main__':
    while True:
        key_value = input("Enter the key of the environment variable:")
        try:
            if os.environ[key_value]:
                print("The value of", key_value, " is ", os.environ[key_value])
        except KeyError:
            print(key_value, 'environment variable is not set.')
            sys.exit(1)
```

Рисунок 2 – Пример 2

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

import os


if __name__ == '__main__':
    if os.environ.get('DEBUG') == 'True':
        print('Debug mode is on')
    else:
        print('Debug mode is off')
```

Рисунок 3 – Пример 3

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

import os


if __name__ == '__main__':
    os.environ.setdefault('DEBUG', 'True')
    if os.environ.get('DEBUG') == 'True':
        print('Debug mode is on')
    else:
        print('Debug mode is off')

```

Рисунок 4 – Пример 4

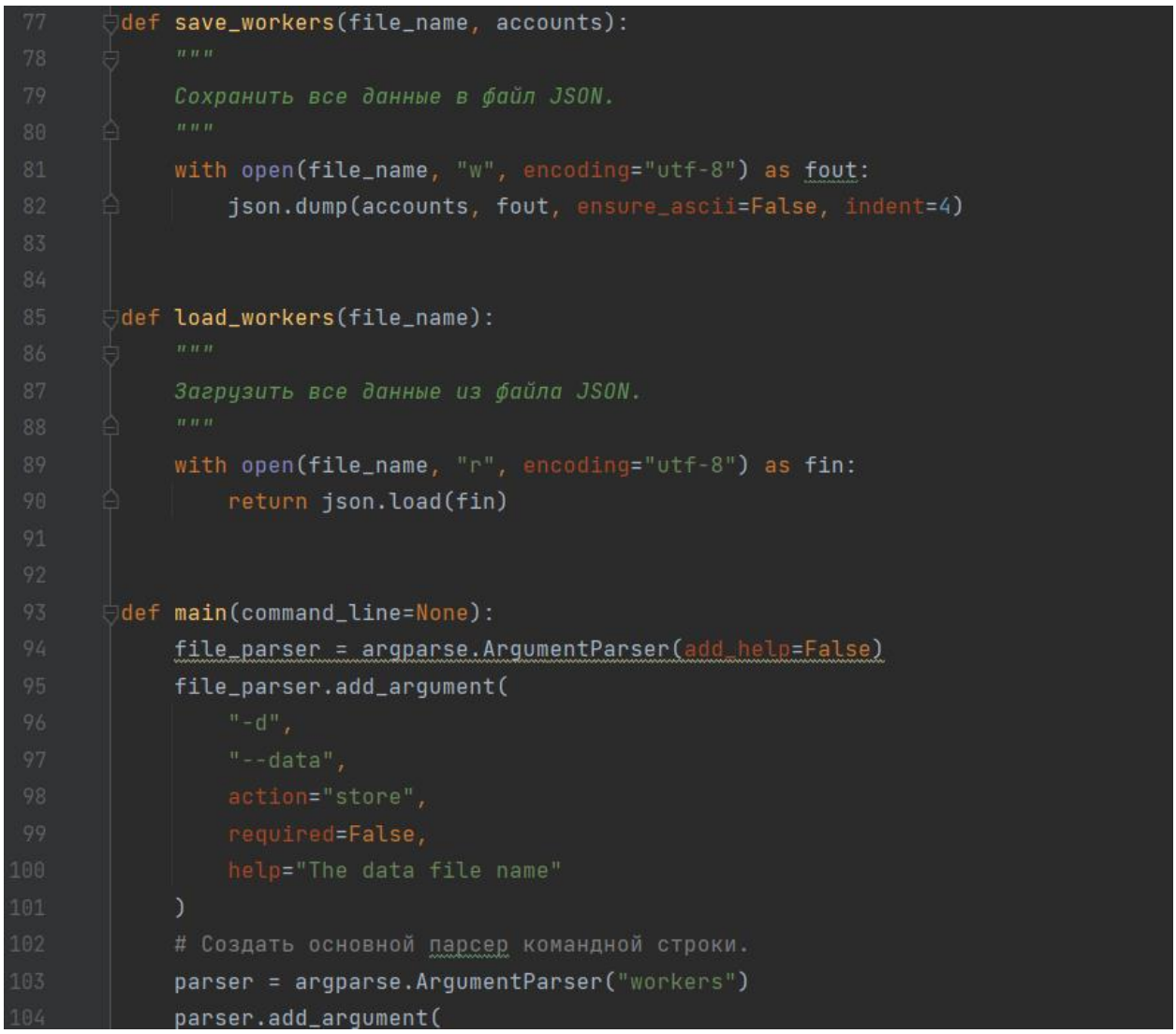
```python
def save_workers(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кирилицы установим 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"
```

Рисунок 5 – Пример 5

Индивидуальное задание:



```
77  def save_workers(file_name, accounts):
78      """
79      Сохранить все данные в файл JSON.
80      """
81      with open(file_name, "w", encoding="utf-8") as fout:
82          json.dump(accounts, fout, ensure_ascii=False, indent=4)
83
84
85  def load_workers(file_name):
86      """
87      Загрузить все данные из файла JSON.
88      """
89      with open(file_name, "r", encoding="utf-8") as fin:
90          return json.load(fin)
91
92
93  def main(command_line=None):
94      file_parser = argparse.ArgumentParser(add_help=False)
95      file_parser.add_argument(
96          "-d",
97          "--data",
98          action="store",
99          required=False,
100         help="The data file name"
101     )
102     # Создать основной парсер командной строки.
103     parser = argparse.ArgumentParser("workers")
104     parser.add_argument(
```

Рисунок 6 – Индивидуальное задание №1

```
    )
    select.add_argument(
        "-t",
        "--t_a",
        action="store",
        type=int,
        required=True,
        help="Аккаунт для которого необходимо получить общую сумму трансферов"
    )

    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)

    data_file = args.data
    if not data_file:
        data_file = os.environ.get("REQS_2")
    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):
        requisites = load_workers(data_file)
    else:
        requisites = []

```

Рисунок 7 – Индивидуальное задание №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?

Значением такой переменной может быть, например, место размещения исполняемых файлов в системе, имя предпочитаемого текстового редактора или настройки системной локали.

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

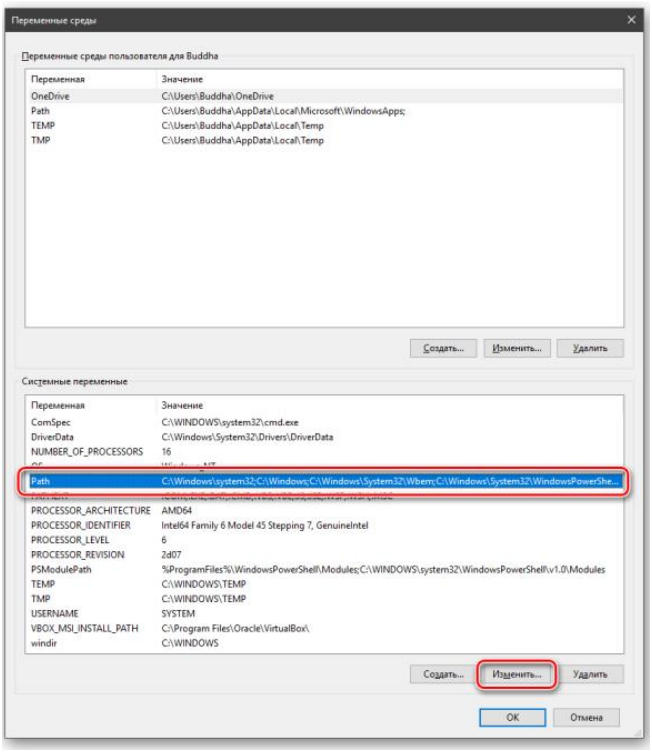
Для этого следует в Проводнике щелкнуть правой кнопкой мыши по иконке компьютера («Этот компьютер» в Windows 10, «Мой компьютер» в Windows 7) и выбрать «Свойства». Далее следует открыть «Дополнительные параметры системы», а в появившемся окне «Свойства системы» — «Переменные среды».

4. Каково назначение переменных PATH и PATHEXT?

«PATH» позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. 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?



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 можно разделить на две основные категории: 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?

команда printenv — выводит список всех переменных окружения (или какую-то отдельно заданную переменную);

9. Какие переменные окружения Linux Вам известны?

Ниже приведены некоторые из наиболее распространенных **переменных окружения**:

- `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?

Наиболее распространенные **переменные оболочки**:

- `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?

Для создания переменной окружения экспортируем нашу недавно созданную переменную оболочки:

```
$ export NEW_VAR
```

13. Для чего необходимо делать переменные окружения Linux постоянными?

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

14. Для чего используется переменная окружения PYTHONHOME ?

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…

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

Следующий код позволяет прочитать и вывести все переменные окружения, а также определенную переменную. Для вывода имен и значений всех переменных используется цикл `for`. Затем выводится значение переменной `HOME`.

```
# Импортируем модуль os
import os


# Создаём 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?

в программах на языке программирования Python?

Проверяем, присвоено ли значение переменной окружения

Давайте создадим Python-файл со следующим скриптом для проверки переменных. Для чтения значений переменных мы используем модуль os, а модуль sys — для прекращения работы приложения.

Бесконечный цикл `while` непрерывно принимает от пользователя имена переменных и проверяет их значения до тех пор, пока пользователь не введёт имя переменной, которой не присвоено значение.

Если пользователь вводит имя переменной окружения, которой присвоено значение, это значение выводится, если же нет — выводится соответствующее сообщение и процесс останавливается.

```
# 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?

Присваиваем значение переменной окружения

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

Давайте напишем код, чтобы с помощью функции `setdefault()` изменить значение переменной `DEBUG` на True (по умолчанию установлено False). После установки значения мы проверим его функцией `get()`.

Если мы сделали всё правильно, выведется сообщение «Режим отладки включен», в противном случае – «Режим отладки выключен».

```
# 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')
```