

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

Институт цифрового развития
Кафедра инфокоммуникаций

**«Лабораторная работа 2.18. Работа с
переменными окружения в Python3»**

**ОТЧЕТ
по лабораторной работе №21
дисциплины
«Основы программной инженерии»**

Выполнил:

Луценко Дмитрий Андреевич
2 курс, группа ПИЖ-б-о-21-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

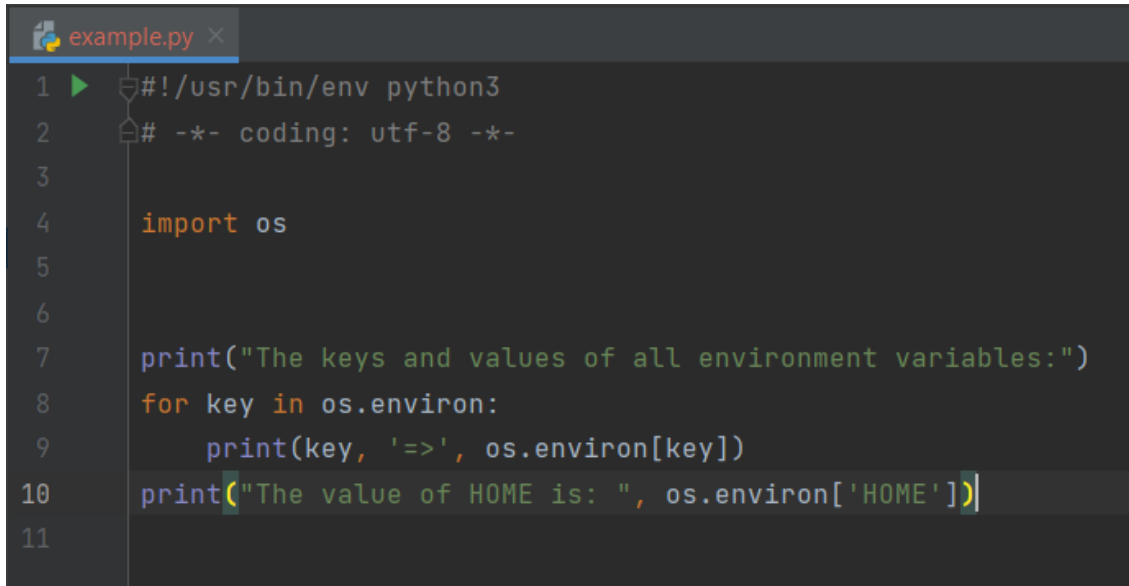
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Лабораторная работа 2.18. Работа с переменными окружения в Python3

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

Ход работы:



```
example.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      import os
5
6
7      print("The keys and values of all environment variables:")
8      for key in os.environ:
9          print(key, '=>', os.environ[key])
10     print("The value of HOME is: ", os.environ['HOME'])
11
```

Рисунок 1 – Пример 1



```
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      import os
5      import sys
6
7      while True:
8          key_value = input("Enter the key of the environment variable: ")
9
10         try:
11             if os.environ[key_value]:
12                 print(
13                     "The value of",
14                     key_value,
15                     " is ",
16                     os.environ[key_value]
17                 )
18
19         except KeyError:
20             print(key_value, 'environment variable is not set.')
21             sys.exit(1)
22
```

Рисунок 2 – Пример 2

```
example.py x
1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4      import os
5
6      os.environ.setdefault('DEBUG', 'True')
7      if os.environ.get('DEBUG') == 'True':
8          print('Debug mode is on')
9      else:
10         print('Debug mode is off')
11
```

Рисунок 3 – Пример 3

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

import argparse
import json
import os
import sys
import jsonschema
from jsonschema import validate
from prod_schema import schema

def add_product(products, prod, shop, cost):
    """
    Ввод информации о товарах.
    """
    products.append(
        {
            "product": prod,
            "shop": shop,
            "cost": cost
        }
    )
    return products

def save_products(file_name, products):
    """
    Сохранить список всех товаров в формате JSON
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(products, fout, ensure_ascii=False, indent=4)

def load_products(file_name):
    """
```

```

Загрузить список всех товаров из файла JSON
"""
with open(file_name, "r", encoding="utf-8") as fin:
    return json.load(fin)

def product_list(products):
    """
    Вывод списка товаров
    """
    if products:
        line = '+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20
        )
        print(line)
        print(
            '| {:^25} | {:^15} | {:^14} |'.format(
                "Товар",
                "Магазин",
                "Стоимость"
            )
        )
        print(line)

        for product in products:
            print(
                '| {:^25} | {:^15} | {:^14} |'.format(
                    product.get('product', ''),
                    product.get('shop', ''),
                    product.get('cost', 0)
                )
            )
            print(line)

def select(products, shop):
    """
    Выбрать товары из конкретного магазина.
    """
    result = []
    for product in products:
        if product.get('shop', '') == shop:
            result.append(product)
    return result

def validation(json_data):
    """
    Валидация данных
    """
    try:
        validate(instance=json_data, schema=schema)
    except:
        raise jsonschema.exceptions.ValidationError("Данные недействительны")

    msg = "Данные успешно загружены"
    return True, msg

def main(command_line=None):
    """
    Главная функция программы.

```

```

"""
file_parser = argparse.ArgumentParser(add_help=False)
file_parser.add_argument(
    "-d",
    "--data",
    action="store",
    required=False,
    help="Имя файла"
)

parser = argparse.ArgumentParser("products")
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 новый продукт"
)
add.add_argument(
    "-n",
    "--name",
    action="store",
    required=True,
    help="Название продуктов"
)
add.add_argument(
    "-s",
    "--shop",
    action="store",
    help="Название магазина"
)
add.add_argument(
    "-c",
    "--cost",
    action="store",
    type=float,
    required=True,
    help="Стоимость товара"
)

list = subparsers.add_parser(
    "list",
    parents=[file_parser],
    help="Отобразить список товаров"
)

select = subparsers.add_parser(
    "select",
    parents=[file_parser],
    help="Выбрать товары из магазина"
)
select.add_argument(
    "-s",
    "--shop",
    action="store",
    type=str,
    required=True,
    help="Название магазина"
)

```

```

args = parser.parse_args(command_line)

data_file = args.data
if not data_file:
    data_file = os.environ.get("PRODUCTS_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):
    products = load_products(data_file)
else:
    products = []

match args.command:
    case 'add':
        products = add_product(
            products,
            args.name,
            args.shop,
            args.cost
        )
        is_dirty = True
    case 'list':
        product_list(products)
    case 'select':
        selected = select(products, args.shop)
        product_list(selected)

if is_dirty:
    save_products(data_file, products)

if __name__ == '__main__':
    main()

```

Листинг 1 – Индивидуальное задание

```
E:\GitHub\laba21>individual.py add --name="картошка" --shop="рынок" --cost=120
```

```
E:\GitHub\laba21>individual.py list
```

Товар	Магазин	Стоимость
кола	магнит	100.0
макароны	пятерочка	44.0
котлеты	магнит	249
картошка	рынок	120.0

Рисунок 4 – Результат работы программы индивидуального задания

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

import argparse
import json
import os
import sys
import jsonschema
from dotenv import load_dotenv, find_dotenv
from jsonschema import validate
from prod_schema import schema

def add_product(products, prod, shop, cost):
    """
    Ввод информации о товарах.
    """
    products.append(
        {
            "product": prod,
            "shop": shop,
            "cost": cost
        }
    )
    return products

def save_products(file_name, products):
    """
    Сохранить список всех товаров в формате JSON
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(products, fout, ensure_ascii=False, indent=4)

def load_products(file_name):
    """
    Загрузить список всех товаров из файла JSON
    """
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)

def product_list(products):
    """
    Вывод списка товаров
    """
    if products:
        line = '+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20
        )
        print(line)
        print(
            '| {:^25} | {:^15} | {:^14} |'.format(
                "Товар",
                "Магазин",
                "Стоимость"
            )
        )
        print(line)

        for product in products:
```

```

        print(
            '| {:^25} | {:^15} | {:^14} |'.format(
                product.get('product', ''),
                product.get('shop', ''),
                product.get('cost', 0)
            )
        )
        print(line)

def select(products, shop):
    """
    Выбрать товары из конкретного магазина.
    """
    result = []
    for product in products:
        if product.get('shop', '') == shop:
            result.append(product)
    return result

def validation(json_data):
    """
    Валидация данных
    """
    try:
        validate(instance=json_data, schema=schema)
    except:
        raise jsonschema.exceptions.ValidationError("Данные недействительны")

    msg = "Данные успешно загружены"
    return True, msg

def main(command_line=None):
    """
    Главная функция программы.
    """
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "-d",
        "--data",
        action="store",
        required=False,
        help="Имя файла"
    )

    parser = argparse.ArgumentParser("products")
    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 новый продукт"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",

```



```

        required=True,
        help="Название продуктов"
    )
    add.add_argument(
        "-s",
        "--shop",
        action="store",
        help="Название магазина"
    )
    add.add_argument(
        "-c",
        "--cost",
        action="store",
        type=float,
        required=True,
        help="Стоимость товара"
    )

    list = subparsers.add_parser(
        "list",
        parents=[file_parser],
        help="Отобразить список товаров"
    )

    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Выбрать товары из магазина"
    )
    select.add_argument(
        "-s",
        "--shop",
        action="store",
        type=str,
        required=True,
        help="Название магазина"
    )

    args = parser.parse_args(command_line)

    data_file = args.data
    if not data_file:
        load_dotenv(find_dotenv())
        data_file = os.environ.get("PRODUCTS_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):
        products = load_products(data_file)
    else:
        products = []

    match args.command:
        case 'add':
            products = add_product(
                products,
                args.name,
                args.shop,
                args.cost
            )
            is_dirty = True
        case 'list':

```

```

        product_list(products)
    case 'select':
        selected = select(products, args.shop)
        product_list(selected)

    if is_dirty:
        save_products(data_file, products)

if __name__ == '__main__':
    main()

```

Листинг 2 – Индивидуальное задание с dotenv

Вывод: в ходе лабораторной работы приобретены навыки по работе с переменными окружения с помощью языка программирования Python версии 3.x.

Ответы на контрольные вопросы

1. Каково назначение переменных окружения? С помощью них 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?
Название переменных, пути для запуска приложений, расширения файлов.

3. Как получить доступ к переменным окружения в ОС Windows?
Нужной перейти во «свойства» «мой компьютер», в левой панели выбрать дополнительные параметры системы. В открывшемся окне перейти во вкладку «дополнительно» и нажать на кнопку «переменные среды».

4. Каково назначение переменных PATH и PATHNEXT? «PATH» позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. PATHNEXT, 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? В меню переменных среды нажать «создать» или «изменить».

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? Нужно ввести `printenv` и название переменной

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. 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.

13. Для чего необходимо делать переменные окружения Linux постоянными? Для того чтобы переменная сохранялась после завершения сеанса оболочки.

14. Для чего используется переменная окружения `PYTHONHOME`?

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?

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 zip-файлов.

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`.

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

```
# 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?

```
# 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')
```