

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

Отчет по лабораторной работе № 2.18

«Работа с переменными окружения в Python3»

по дисциплине «Основы программной инженерии»

Выполнила:

Образцова Мария Дмитриевна,

2 курс, группа ПИЖ-б-о-21-1,

Проверил:

Доцент кафедры

инфокоммуникаций, Воронкин Р.А.

Ставрополь, 2023 г.

1. Создала репозиторий в GitHub, дополнила правила в .gitignore для работы с IDE PyCharm с ЯП Python, выбрал лицензию MIT, клонировала его на компьютер и организовала в соответствии с моделью ветвления git-flow

A repository contains all project files, including the revision history. Already I [Import a repository.](#)

Owner *

Repository name *



obrMaria ▾



OPI_2.18



Great repository names are short and memorable. Need inspiration? How ab

Description (optional)



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



You are creating a public repository in your personal account.

Create repository

Рисунок – Создание репозитория

```
M@DESKTOP-UVM9NOL MINGW64 ~/desktop (master)
$ cd OPI_2.18

M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_2.18 (main)
$ 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/M/desktop/OPI_2.18/.git/hooks]

M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_2.18 (develop)
```

Рисунок – Клонирование репозитория

2. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python.

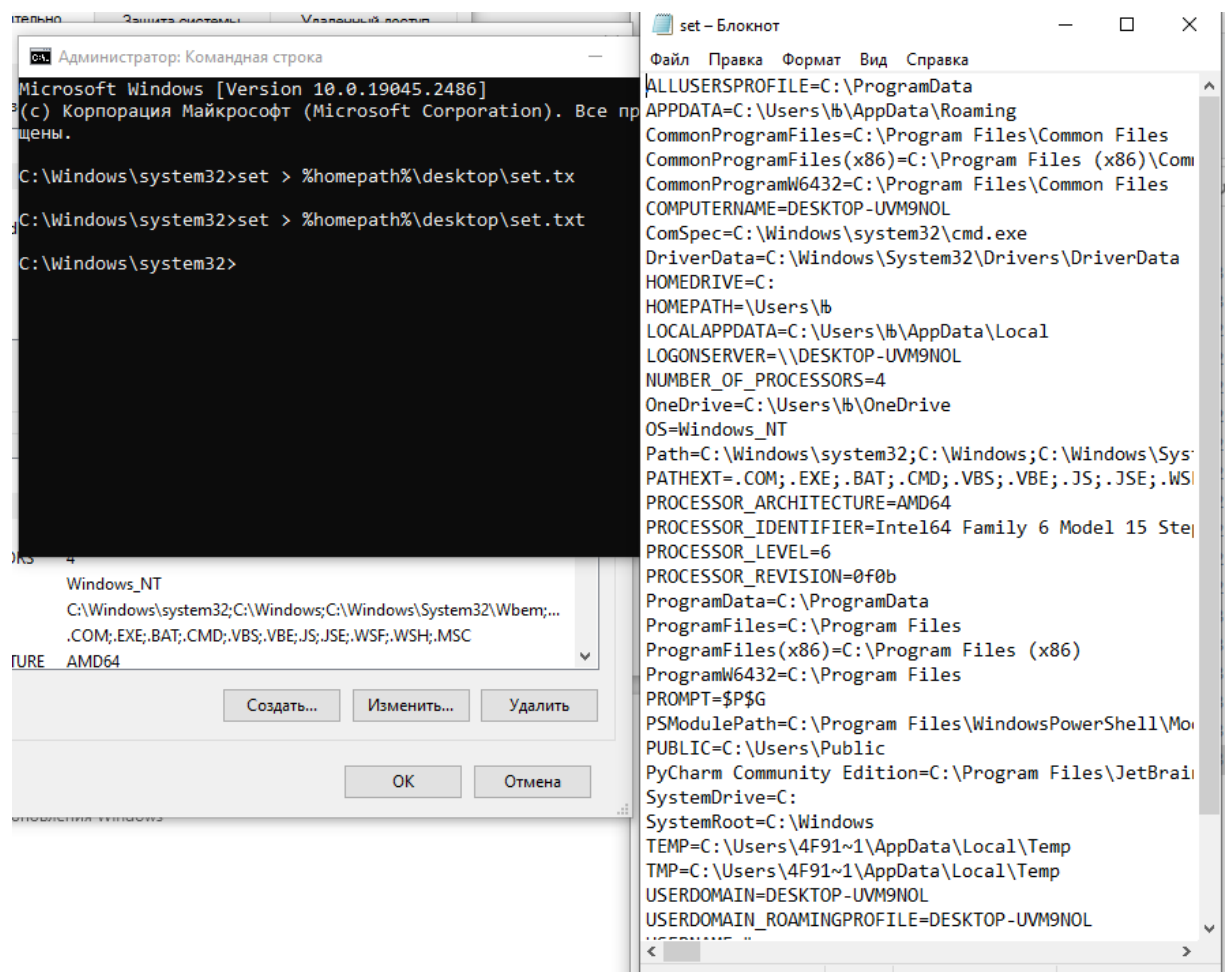
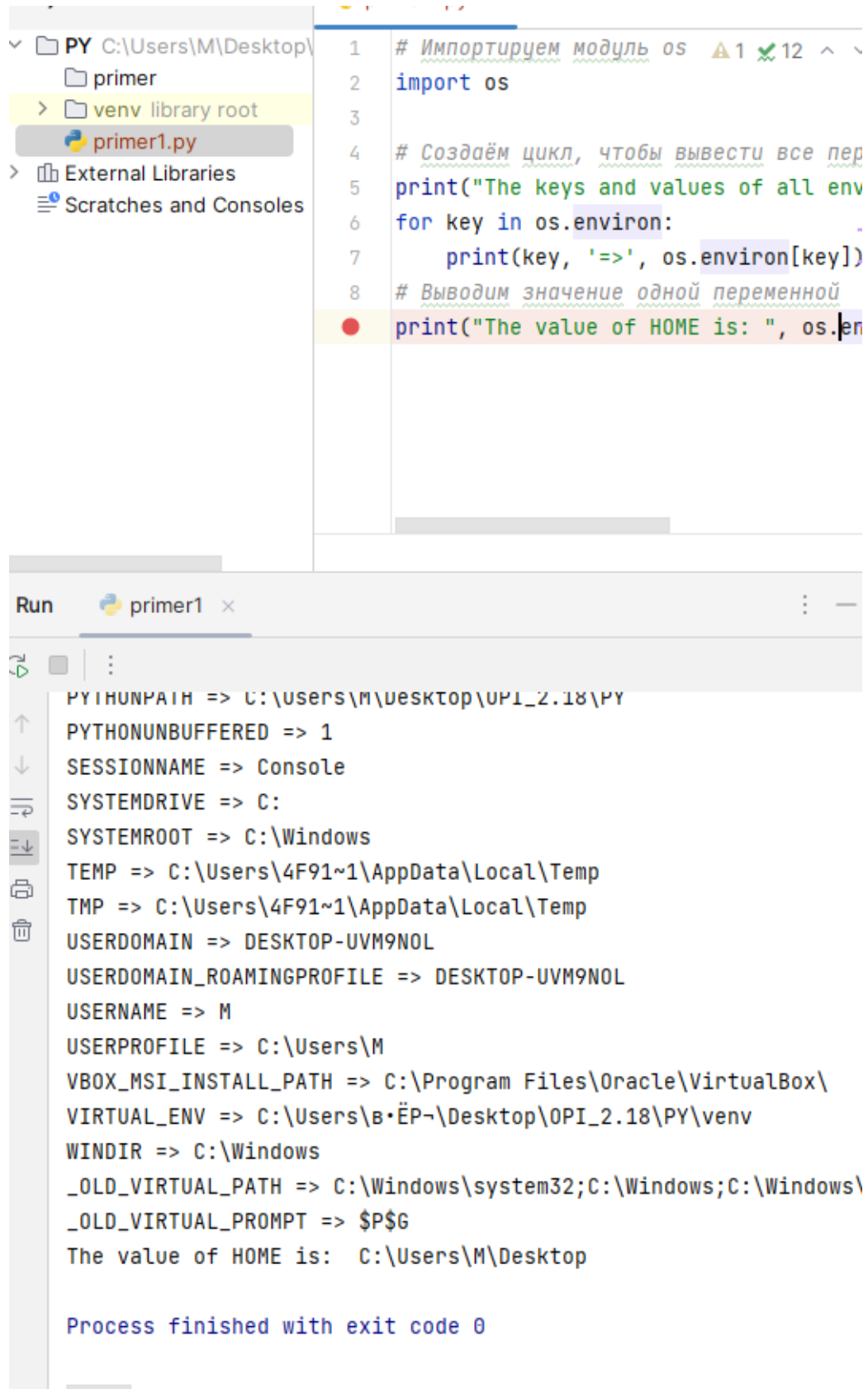


Рисунок – файл «set.txt» с указанием всех переменных окружения в системе.

Чтение переменных окружения на Python

Считываем одну или все переменные окружения



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'PY' located at 'C:\Users\M\Desktop\'. Inside the project, there is a folder 'primer' and a file 'primer1.py'. The code editor shows the following Python code:

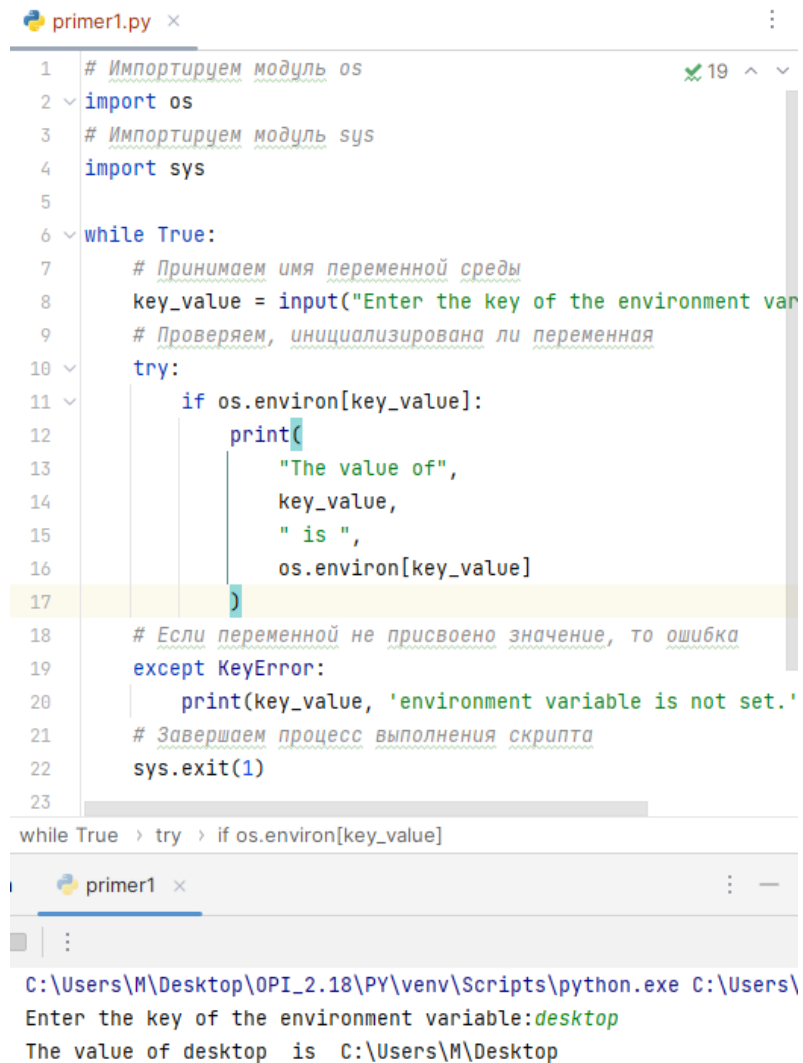
```
1 # Импортируем модуль os
2 import os
3
4 # Создаём цикл, чтобы вывести все пер
5 print("The keys and values of all env
6 for key in os.environ:
7     print(key, '=>', os.environ[key])
8 # Выводим значение одной переменной
9 print("The value of HOME is: ", os.en
```

Below the code editor is a 'Run' window titled 'Run primer1'. It shows the output of the script, which lists various environment variables and their values, followed by the specific value of the 'HOME' variable.

```
PYTHONPATH => C:\Users\M\Desktop\UPI_2.18\PY
PYTHONUNBUFFERED => 1
SESSIONNAME => Console
SYSTEMDRIVE => C:
SYSTEMROOT => C:\Windows
TEMP => C:\Users\4F91~1\AppData\Local\Temp
TMP => C:\Users\4F91~1\AppData\Local\Temp
USERDOMAIN => DESKTOP-UVM9N0L
USERDOMAIN_ROAMINGPROFILE => DESKTOP-UVM9N0L
USERNAME => M
USERPROFILE => C:\Users\M
VBOX_MSI_INSTALL_PATH => C:\Program Files\Oracle\VirtualBox\
VIRTUAL_ENV => C:\Users\B·ËP~\Desktop\UPI_2.18\PY\venv
WINDIR => C:\Windows
_OLD_VIRTUAL_PATH => C:\Windows\system32;C:\Windows;C:\Windows\
_OLD_VIRTUAL_PROMPT => $P$G
The value of HOME is: C:\Users\M\Desktop

Process finished with exit code 0
```

Проверяем, присвоено ли значение переменной окружения



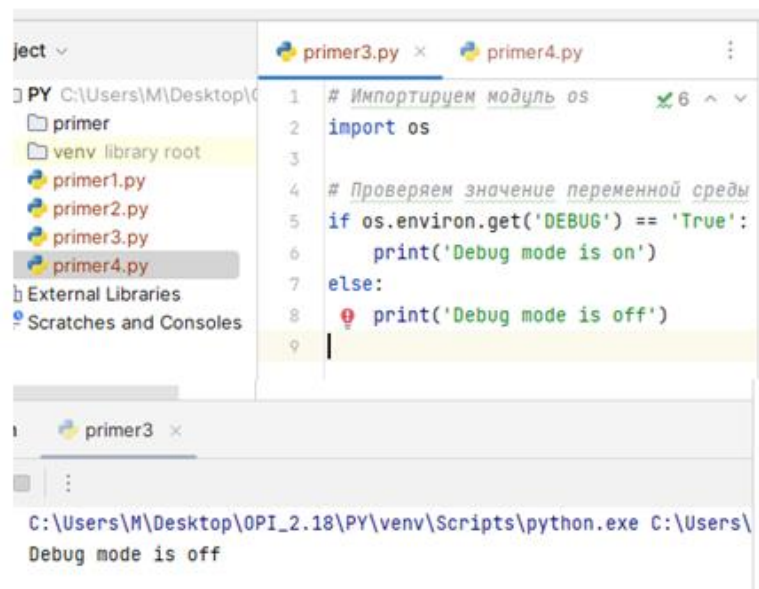
The screenshot shows a Python IDE with a file named `primer1.py`. The code is as follows:

```
1 # Импортируем модуль os
2 import os
3 # Импортируем модуль sys
4 import sys
5
6 while True:
7     # Принимаем имя переменной среды
8     key_value = input("Enter the key of the environment variable: ")
9     # Проверяем, инициализирована ли переменная
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    except KeyError:
19        print(key_value, 'environment variable is not set.')
20    # Завершаем процесс выполнения скрипта
21    sys.exit(1)
22
23
```

The console output shows the execution of the script:

```
C:\Users\M\Desktop\OPI_2.18\PY\venv\Scripts\python.exe C:\Users\
Enter the key of the environment variable: desktop
The value of desktop is C:\Users\M\Desktop
```

Проверяем переменную на истинность



The screenshot shows a Python IDE with two files: `primer3.py` and `primer4.py`. The code in `primer3.py` is as follows:

```
1 # Импортируем модуль os
2 import os
3
4 # Проверяем значение переменной среды
5 if os.environ.get('DEBUG') == 'True':
6     print('Debug mode is on')
7 else:
8     print('Debug mode is off')
```

The console output shows the execution of the script:

```
C:\Users\M\Desktop\OPI_2.18\PY\venv\Scripts\python.exe C:\Users\
Debug mode is off
```

3. Создадим переменные среды необходимые для выполнения лабораторной работы.

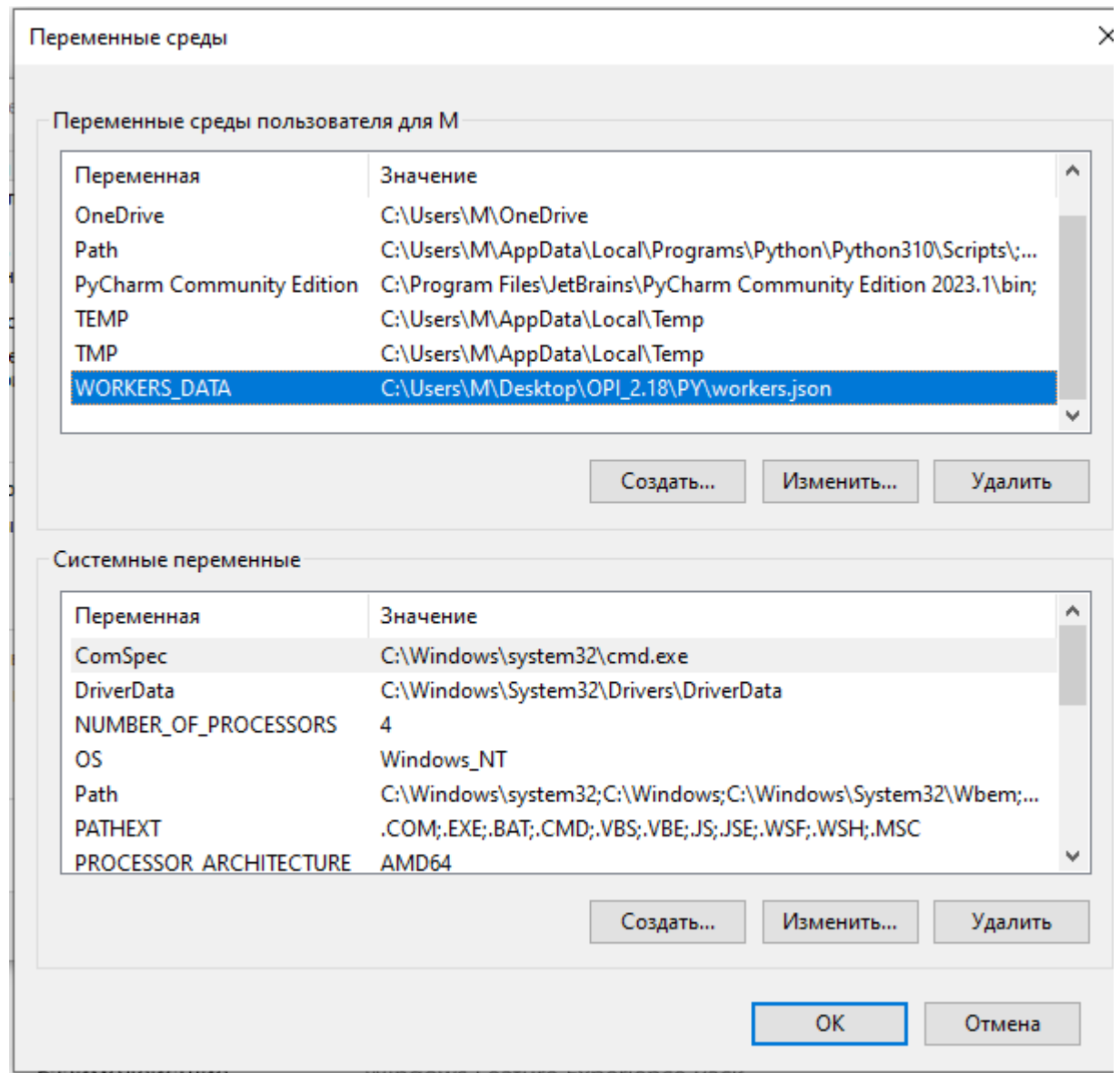


Рисунок – Созданная новая переменная среда
для примера

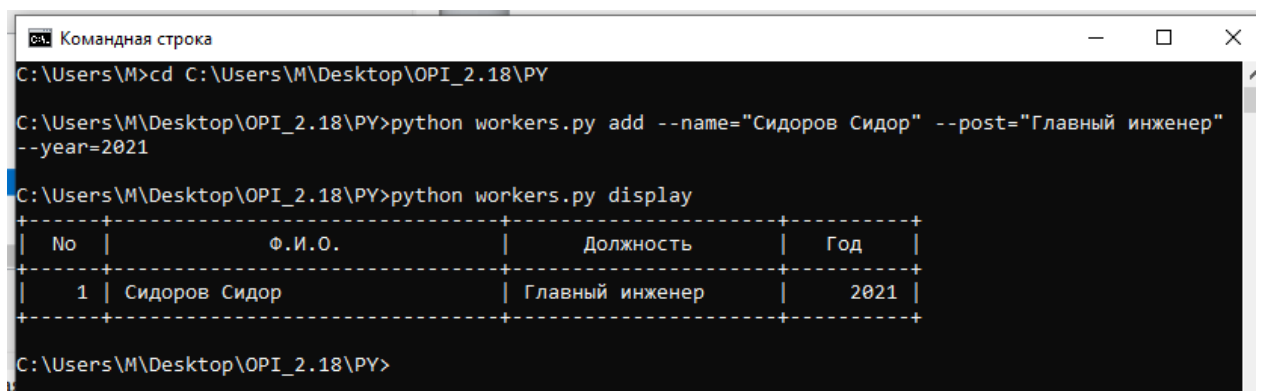


Рисунок – Вызов команды add и display
без указания файла с данными

Задание 1. Для своего варианта лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответствующую переменную окружения.

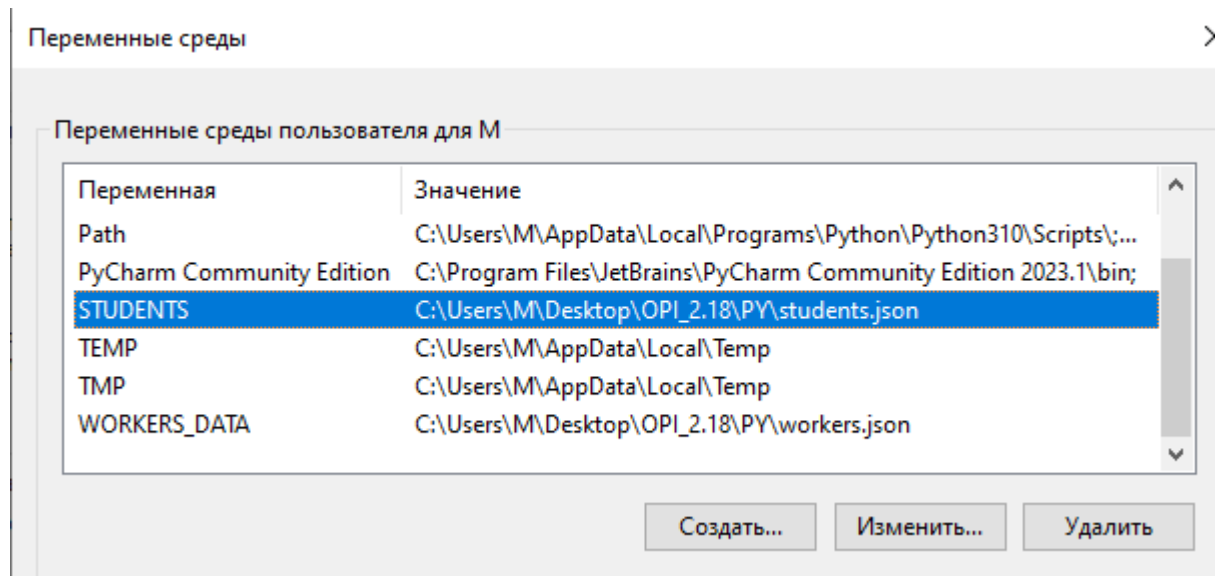


Рисунок – Создание новой переменной среды
для индивидуального задания

```
C:\Users\M\Desktop\OPI_2.18\PY>python ind_1.py add --name="maria" --group="123" --marks=
"1 2 3 4 5"
[{'name': 'maria', 'group': '123', 'marks': [1, 2, 3, 4, 5]}]

C:\Users\M\Desktop\OPI_2.18\PY>python ind_1.py display
+-----+-----+-----+-----+
| № | Ф.И.О. | Группа | Оценки |
+-----+-----+-----+-----+
| 1 | maria | 123 | 1,2,3,4,5 |
+-----+-----+-----+-----+

C:\Users\M\Desktop\OPI_2.18\PY>python ind_1.py find
список студентов пуст
```

Рисунок – Вывод программы индивидуального задания №1

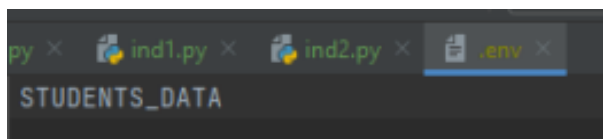
Задание 2. Самостоятельно изучите работу с пакетом python-dotenv. Модифицируйте программу задания 1 таким образом, чтобы значения необходимых переменных окружения считывались из файла .env. Дополнил код программы из индивидуального задания №1

```

13
14     # Выполнить разбор аргументов командной строки.
15     args = parser.parse_args(command_line)
16
17     # Подключение к окружению
18     data_file = args.data
19     dotenv_path = os.path.join(os.path.dirname(__file__), ".env")
20     if os.path.exists(dotenv_path):
21         load_dotenv(dotenv_path)
22     if not data_file:
23         data_file = os.getenv("STUDENTS")
24     if not data_file:
25         print("The data file name is absent", file=sys.stderr)
26         sys.exit(1)
27

```

Рисунок – Изменения в коде программы



```

py x ind1.py x ind2.py x .env x
STUDENTS_DATA

```

Рисунок – Содержимое файла env

```

C:\Users\M\Desktop\OPI_2.18\PY>python ind_2.py add --name="maria" --group="123" --marks="
1 2 3 4 5"
C:\Users\M\Desktop\OPI_2.18\PY>python ind_2.py display
+-----+-----+-----+-----+
| № | Ф.И.О. | Группа | Оценки |
+-----+-----+-----+-----+
| 1 | maria | 123 | 1,2,3,4,5 |
+-----+-----+-----+-----+
C:\Users\M\Desktop\OPI_2.18\PY>_

```

Рисунок – Вывод программы индивидуального задания №2

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? Она хранят в текстовом виде ссылки на определённые каталоги,

количество ядер процессора, доступные расширения и т.п.

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

Просмотреть все переменные окружения можно с помощью команды:

```
set > %homepath%\desktop\set.txt
```

Также доступ к ним можно получить через свойства системы.

4. Каково назначение переменных PATH и PATHNEXT?

«PATH» позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. Например, если ввести в «Командную строку» explorer.exe система осуществит поиск по папкам, указанным в значении переменной, найдет и запустит соответствующую программу.

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?

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 (посмотрите списки).

В поле Значение указываем путь до папки Рабочий стол:

```
C:\Users\Имя_пользователя\Desktop
```

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

Перезапускаем Проводник и консоль или целиком систему.

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. Переменные окружения применяются для настройки поведения приложений и работы самой системы. Например, переменная окружения может хранить информацию о путях к исполняемым файлам, заданном по умолчанию текстовом редакторе, браузере, языковых параметрах (локали) системы или настройках раскладки клавиатуры.

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`. Если команде в качестве аргумента передать имя переменной, то будет отображено значение только этой переменной. Если же вызвать `printenv` без аргументов, то выведется построчный список всех переменных окружения.

Пример: `$ 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?

Чтобы создать новую переменную оболочки с именем, например, `NEW_VAR` и значением `Ravesli.com`, просто введите:

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

Вы можете убедиться, что переменная действительно была создана, с помощью команды `echo`:

```
$ echo $NEW_VAR
```

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.

Для создания переменной окружения экспортируем нашу недавно созданную переменную оболочки:

```
$ export NEW_VAR
```

Вы также можете использовать и следующую конструкцию для создания переменной окружения:

```
$ export MY_NEW_VAR="My New Var"
```

Примечание: Созданные подобным образом переменные окружения доступны только в текущем сеансе. Если вы откроете новую оболочку или выйдете из системы, то все переменные будут потеряны.

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.

Чтобы установить постоянную переменную окружения для текущего пользователя, откройте файл `~/.bashrc`:

```
$ sudo nano ~/.bashrc
```

Для каждой переменной, которую вы хотите сделать постоянной, добавьте в конец файла строку, используя следующий синтаксис:

```
export [ИМЯ_ПЕРЕМЕННОЙ]=[ЗНАЧЕНИЕ_ПЕРЕМЕННОЙ]
```

13. Для чего необходимо делать переменные окружения Linux постоянными?

Для того, чтобы они сохранялись при перезапуске сессии.

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`. 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` ?

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` .

PYTHONVERBOSE :

Если значение переменной среды PYTHONVERBOSE непустая строка, то это эквивалентно указанию опции `-v` . Если установлено целое число, это эквивалентно многократному указанию `-v` .

PYTHONCASEOK :

Если значение переменной среды PYTHONCASEOK установлено, то Python игнорирует регистр символов в операторах импорта. Это работает только в Windows и OS X.

PYTHONDONTWRITEBYTECODE :

Если значение переменной среды PYTHONDONTWRITEBYTECODE непустая строка, то Python не будет пытаться писать файлы `.рус` при импорте исходных модулей. Это эквивалентно указанию параметра `-B` .

PYTHONPYCACHEDPREFIX :

Если значение переменной среды PYTHONPYCACHEDPREFIX установлено, то Python будет записывать файлы `.рус` в зеркальном дереве каталогов по этому пути, а не в каталогах `русache` в исходном дереве. Это эквивалентно указанию параметра `-X русache_prefix=PATH` .

PYTHONHASHSEED :

Если значение переменной среды PYTHONHASHSEED не установлено или имеет значение `random` , то случайное значение используется для заполнения хэшей объектов `str` и `bytes` .

Если для PYTHONHASHSEED задано целочисленное значение, то оно используется как фиксированное начальное число для генерации hash() типов, охватываемых рандомизацией хэша.

Цель - разрешить повторяемое хеширование, например, для самотестирования самого интерпретатора, или позволить кластеру процессов Python совместно использовать хеш- значения.

Целое число должно быть десятичным числом в диапазоне [0,4294967295]. Указание значения 0 отключит рандомизацию хэша.

PYTHONIOENCODING :

Если значение переменной среды PYTHONIOENCODING установлено до запуска интерпретатора, то оно переопределяет кодировку, используемую для stdin / stdout / stderr , в синтаксисе encodingname:errorhandler . И имя кодировки encodingname , и части :errorhandler являются необязательными и имеют то же значение, что и в функции str.encode() .

Для stderr часть :errorhandler игнорируется, а обработчик всегда будет заменять обратную косую черту.

PYTHONNOUSERSITE :

Если значение переменной среды PYTHONNOUSERSITE установлено, то Python не будет добавлять пользовательский каталог site-packages в переменную sys.path .

PYTHONUSERBASE :

Переменная среды PYTHONUSERBASE определяет базовый каталог пользователя, который используется для вычисления пути к каталогу пользовательских пакетов сайта site-packages и путей установки Distutils для python setup.py install --user .

PYTHONWARNINGS :

Переменная среды PYTHONWARNINGS эквивалентна опции -W . Если она установлена в виде строки, разделенной запятыми, то это эквивалентно многократному указанию -W , при этом фильтры, расположенные позже в списке, имеют приоритет над фильтрами ранее в

списке.

В простейших настройках определенное действие безоговорочно применяется ко всем предупреждениям, выдаваемым процессом (даже к тем, которые по умолчанию игнорируются):

`PYTHONWARNINGS=default` - предупреждает один раз для каждого вызова;

`PYTHONWARNINGS=error` - преобразовывает в исключения;
`PYTHONWARNINGS=always` - предупреждает каждый раз;

`PYTHONWARNINGS=module` - предупреждает один раз для каждого вызванного модуля;

`PYTHONWARNINGS=once` - предупреждает один раз для каждого процесса Python;

`PYTHONWARNINGS=ignore` - никогда не предупреждает.
`PYTHONFAULTHANDLER` :

Если значение переменной среды `PYTHONFAULTHANDLER` непустая строка, то при запуске вызывается `faulthandler.enable()` : устанавливается обработчик сигналов `SIGSEGV` , `SIGFPE` ,

`SIGABRT` , `SIGBUS` и `SIGILL` , чтобы вывести данные трассировки Python. Это эквивалентно опции обработчика ошибок `-X` .

`PYTHONTRACEMALLOC` :

Если значение переменной среды `PYTHONTRACEMALLOC` непустая строка, то начнется отслеживание выделения памяти Python с помощью модуля `tracemalloc` . Значение переменной - это максимальное количество кадров, хранящихся в обратной трассировке `trace` .

Например, `PYTHONTRACEMALLOC=1` сохраняет только самый последний кадр.

`PYTHONPROFILEIMPORTTIME` :

Если значение переменной среды `PYTHONPROFILEIMPORTTIME` непустая строка, то Python покажет, сколько времени занимает каждый

импорт. Это в точности эквивалентно установке `-X importtime` в командной строке.

`PYTHONASYNCIODEBUG :`

Если значение переменной среды `PYTHONASYNCIODEBUG` непустая строка, то включается режим отладки модуля `asyncio`.

`PYTHONMALLOC :`

Переменная `PYTHONMALLOC` задает распределители памяти Python и/или устанавливает отладочные хуки. Задает семейство распределителей памяти, используемых Python:

`default` : использует распределители памяти по умолчанию.

`malloc` : использует функцию `malloc()` библиотеки C для всех доменов (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).

`rumalloc` : использует распределитель `rumalloc` для доменов `PYMEM_DOMAIN_MEM` и `PYMEM_DOMAIN_OBJ` и использует функцию `malloc()` для домена `PYMEM_DOMAIN_RAW`.

Устанавливает хуки отладки:

`debug` : устанавливает хуки отладки поверх распределителей памяти по умолчанию.

`malloc_debug` : то же, что и `malloc`, но также устанавливает отладочные хуки.

`rumalloc_debug` : то же, что и `rumalloc`, но также устанавливает отладочные хуки.

`PYTHONMALLOCSTATS :`

Если значение переменной среды `PYTHONMALLOCSTATS` непустая строка, то Python будет печатать статистику распределителя памяти `rumalloc` каждый раз, когда создается новая область объекта `rumalloc`, а также при завершении работы.

Эта переменная игнорируется, если переменная среды `PYTHONMALLOC` используется для принудительного использования

распределителя malloc() библиотеки C или если Python настроен без поддержки pymalloc .

PYTHONLEGACYWINDOWSFSENCODING :

Если значение переменной среда Python PYTHONLEGACYWINDOWSFSENCODING непустая строка, то кодировка файловой системы по умолчанию и режим ошибок вернутся к своим значениям mbcs и replace до версии Python 3.6 соответственно. В противном случае используются новые значения по умолчанию utf-8 и surrogatepass .

PYTHONLEGACYWINDOWSSTDIO :

Если значение переменной среды PYTHONLEGACYWINDOWSSTDIO непустая строка, то новые средства чтения и записи консоли не используются. Это означает, что символы Unicode будут закодированы в соответствии с активной кодовой страницей консоли, а не с использованием utf-8 .

Эта переменная игнорируется, если стандартные потоки перенаправляются в файлы или каналы, а не ссылаются на буферы консоли.

PYTHONCOERCECLOCALE :

Если значение переменной среды PYTHONCOERCECLOCALE установлено в значение 0 , то это заставит основное приложение командной строки Python пропускать приведение устаревших локалей C и POSIX на основе ASCII к более функциональной альтернативе на основе UTF-8 . Если эта переменная не установлена или имеет значение, отличное от 0 , то переменная среды переопределения локали LC_ALL также не задана, а текущая локаль, указанная для категории

LC_STYPE , является либо локалью C по умолчанию, либо локалью POSIX явно основанной на ASCII , то Python CLI попытается настроить следующие локали для категории LC_STYPE в порядке, указанном перед загрузкой среды выполнения интерпретатора: C.UTF-8 , C.utf8 , UTF-8 .

Если установка одной из этих категорий локали прошла успешно, то переменная среды

LC_STYPE также будет установлена соответствующим образом в текущей среде процесса и инициализации среды выполнения Python. Это гарантирует, что обновленный параметр будет виден как самому интерпретатору, так и другим компонентам, зависящим от локали, работающим в одном процессе (например, библиотеке GNU readline), и в subprocesses (независимо от того, работают ли эти процессы на интерпретаторе Python или нет), а также в операциях, которые запрашивают среду, а не текущую локаль C (например, собственный

`locale.getdefaultlocale()` Python).

Настройка одного из этих языковых стандартов явно или с помощью указанного выше неявного принуждения языкового стандарта автоматически включает обработчик ошибок surrogateescape для `sys.stdin` и `sys.stdout` (`sys.stderr` продолжает использовать обратную косую черту, как и в любой другой локали). Это поведение обработки потока можно переопределить, используя `PYTHONIOENCODING` , как обычно.

Для целей отладки, установка `PYTHONCOERCECLOCALE=warn` приведет к тому, что Python будет выдавать предупреждающие сообщения на `stderr` , если активируется принуждение языкового стандарта или если языковой стандарт, который мог бы вызвать приведение, все еще активен при инициализации среды выполнения Python. Также обратите внимание, что даже когда принуждение языкового стандарта отключено или когда не удастся найти подходящую целевую локаль, переменная среды `PYTHONUTF8` все равно будет активироваться по умолчанию в устаревших локалях на основе ASCII . Чтобы для системных интерфейсов интерпретатор использовал ASCII вместо UTF-8 , необходимо обе переменные отключить.

PYTHONDEVMODE :

Если значение переменной среды `PYTHONDEVMODE` непустая строка, то включится режим разработки Python, введя дополнительные проверки времени выполнения, которые слишком "дороги" для включения по умолчанию.

PYTHONUTF8 :

Если переменная среды PYTHONUTF8 установлена в значение 1, то это включает режим интерпретатора UTF-8 , где UTF-8 используется как кодировка текста для системных интерфейсов, независимо от текущей настройки локали.

PYTHONWARNDEFAULTENCODING :

Если для этой переменной среды задана непустая строка, то код будет выдавать EncodingWarning , когда используется кодировка по умолчанию, зависящая от локали.

PYTHONTHREADDEBUG :

Если значение переменной среды PYTHONTHREADDEBUG установлено, то Python распечатает отладочную информацию о потоках. Нужен Python, настроенный с параметром сборки --with-pydebug .

PYTHONDUMPPREFS :

Если значение переменной среды PYTHONDUMPPREFS установлено, то Python будет сбрасывать объекты и счетчики ссылок, все еще живые после завершения работы интерпретатора.

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. Далее мы рассмотрим различные способы чтения, проверки и присвоения значения переменной среды.

```
# 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?

```
key_value = input("Enter the key of the environment variable:") 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 `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')
```

Значения переменных окружения можно считывать и изменять при помощи объекта `environ[]` модуля `os` либо путем использования функций `setdefault()` и `get()`. В качестве ключа, по которому можно обратиться и получить либо присвоить значение переменной, в `environ[]` используется имя переменной окружения. Функция `get()` используется для получения значения определённой переменной, а `setdefault()` – для инициализации.