МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра

инфокоммуникаций

Институт цифрового

развития

ОТЧЁТ

по лабораторной работе №2.18

Дисциплина: «Основы программной инженерии»

Тема: «Работа с переменными окружения в Python3»

Выполнила:
студентка 2 курса
группы Пиж-б-о-21-1
Джолдошова Мээрим
Бекболотовна

Ставрополь 2023

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

1. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python. Зафиксируйте изменения в репозитории.

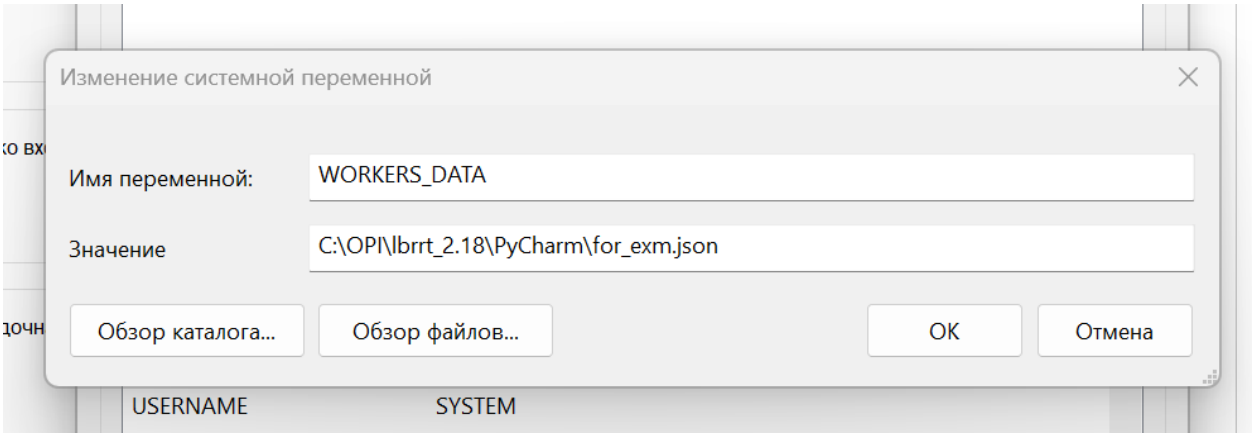

Рисунок 1 – Создание переменной окружения

```
C:\OPI\lbrrt_2.18\PyCharm>python example1.py display
+------+--------------------------------+----------------------+----------+
|  No  |             Ф.И.О.             |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Mike                           | ceo                  |     2003 |
|    2 | Mia                            | helper               |     2005 |
|    3 | Mei                            | ceo helper           |     2022 |
+------+--------------------------------+----------------------+----------+

C:\OPI\lbrrt_2.18\PyCharm>
```

Рисунок 2 – Результат работы программы

```
C:\OPI\lbrrt_2.18\PyCharm>python example1.py display
+------+--------------------------------+----------------------+----------+
|  No  |             Ф.И.О.             |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Mike                           | ceo                  |     2003 |
|    2 | Mia                            | helper               |     2005 |
|    3 | Mei                            | ceo helper           |     2022 |
|    4 | Noah                           | Gen Dir              |     2004 |
+------+--------------------------------+----------------------+----------+

C:\OPI\lbrrt_2.18\PyCharm>
```

Рисунок 3 – Результат работы функции select

2. Приведите в отчете скриншоты работы программ решения

индивидуальных заданий.

Задание 1

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

Код программы:

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

import sys
import json
import os
import argparse
import os.path


def add_flight(flights, dst, nmb, tpe):
    flights.append(
        {
            "destination": dst,
            "number_flight": nmb,
            "type_plane": tpe
        }
    )
    return flights


def display_flights(flights):
    """
    displaying the given information
    """
    if flights:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 18
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^18} |'.format(
                "№",
                "Destination",
                "NumberOfTheFlight",
                "TypeOfThePlane"
            )
        )
        print(line)

        for idx, flight in enumerate(flights, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>18} |'.format(
                    idx,
                    flight.get('destination', ''),
                    flight.get('number_flight', ''),
                    flight.get('type_plane', 0)
                )
```

```
            )
        print(line)
    else:
        print('list is empty')


def select_flights(flights, t):
    result = []
    for flight in flights:
        if t in str(flight.values()):
            result.append(flight)
    return result


def save_flights(file_name, flights):
    with open(file_name, "w", encoding="utf-8", errors="ignore") as fout:
        json.dump(flights, fout, ensure_ascii=False, indent=4)


def load_flights(file_name):
    with open(file_name, "r", encoding="utf-8", errors="ignore") 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(
        "--data",
        action="store",
        required=False,
        help="The data file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("flights")
    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 flight"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination of the flight"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="Number of the flight"
```

```
    )
    add.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="Type of the plane"
    )

    # Создать субпарсер для отображения всех работников.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all flights"
    )

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the flights"
    )
    select.add_argument(
        "-s",
        "--select",
        action="store",
        required=True,
        help="The required select"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    data_file = args.data
    if not data_file:
        data_file = os.environ.get("FLIGHTS_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):
        flights = list(load_flights(data_file))
    else:
        flights = []


    # Добавить работника.
    if args.command == "add":
        flights = add_flight(
            flights,
            args.destination,
            args.number,
            args.type
        )
        is_dirty = True

    # Отобразить всех работников.
    elif args.command == "display":
        display_flights(flights)

    # Выбрать требуемых рааботников.
    elif args.command == "select":
```

```
        selected = select_flights(flights, args.select)
        display_flights(selected)

    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_flights(data_file, flights)


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

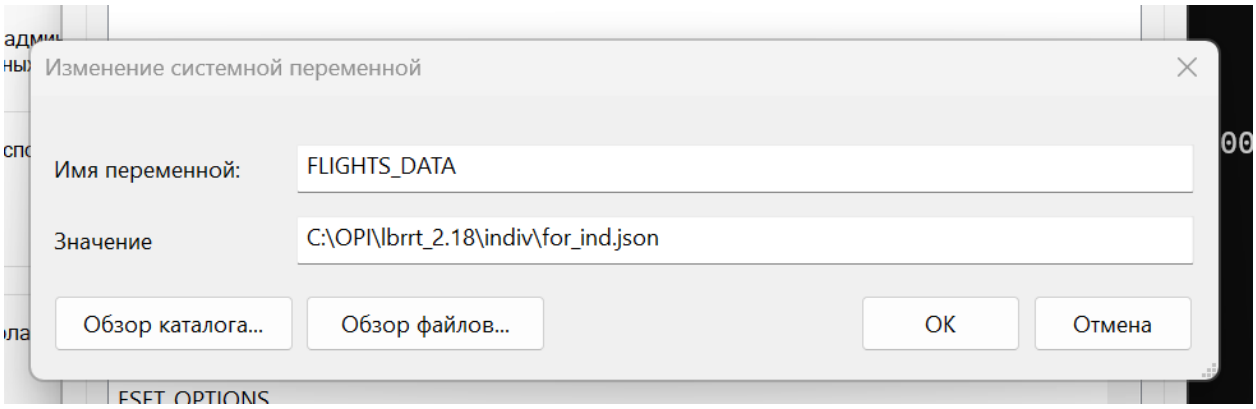

Рисунок 4 – Создание переменной окружения

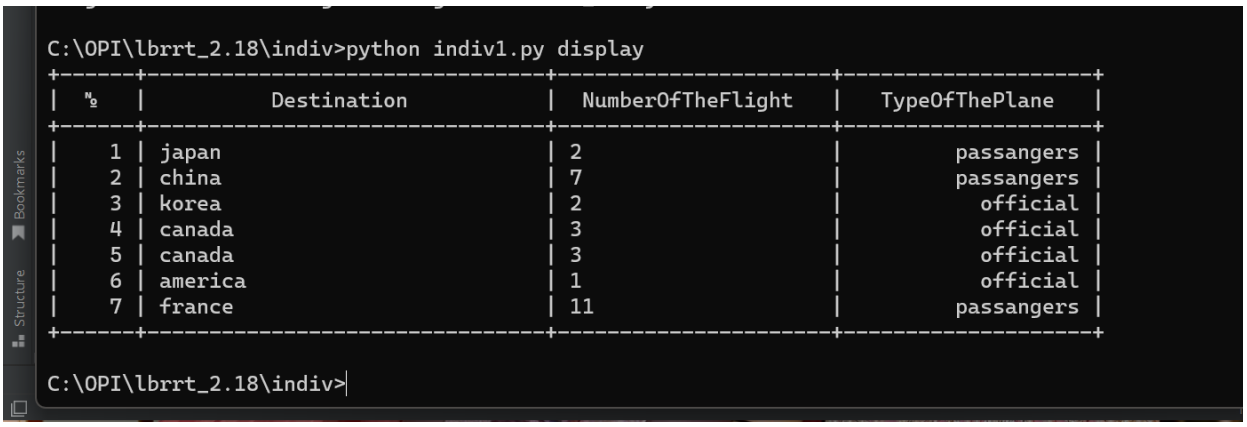

Рисунок 5 – Результат работы программы

```
C:\OPI\lbrrt_2.18\indiv>python indiv1.py add -d "Indonesia" -n 9 -t "passangers"

C:\OPI\lbrrt_2.18\indiv>python indiv1.py display
+------+--------------------------------+----------------------+--------------------+
|  №   |          Destination           |  NumberOfTheFlight   |   TypeOfThePlane   |
+------+--------------------------------+----------------------+--------------------+
|    1 | japan                          | 2                    |         passangers |
|    2 | china                          | 7                    |         passangers |
|    3 | korea                          | 2                    |           official |
|    4 | canada                         | 3                    |           official |
|    5 | canada                         | 3                    |           official |
|    6 | america                        | 1                    |           official |
|    7 | france                         | 11                   |         passangers |
|    8 | Indonesia                      | 9                    |         passangers |
+------+--------------------------------+----------------------+--------------------+

C:\OPI\lbrrt_2.18\indiv>
```

Рисунок 6 – Результат работы функции add

```
C:\OPI\lbrrt_2.18\indiv>python indiv1.py select -s official
+------+--------------------------------+----------------------+--------------------+
|  №   |          Destination           |  NumberOfTheFlight   |   TypeOfThePlane   |
+------+--------------------------------+----------------------+--------------------+
|    1 | korea                          | 2                    |           official |
|    2 | canada                         | 3                    |           official |
|    3 | canada                         | 3                    |           official |
|    4 | america                        | 1                    |           official |
+------+--------------------------------+----------------------+--------------------+

C:\OPI\lbrrt_2.18\indiv>
```

Рисунок 7 – Результат работы функции select

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 программы:

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

import sys
import json
import os
import argparse
import os.path
from dotenv import load_dotenv


def add_flight(flights, dst, nmb, tpe):
    flights.append(
        {
            "destination": dst,
```

```
            "number_flight": nmb,
            "type_plane": tpe
        }
    )
    return flights


def display_flights(flights):
    """
    displaying the given information
    """
    if flights:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 18
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^18} |'.format(
                "№",
                "Destination",
                "NumberOfTheFlight",
                "TypeOfThePlane"
            )
        )
        print(line)

        for idx, flight in enumerate(flights, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>18} |'.format(
                    idx,
                    flight.get('destination', ''),
                    flight.get('number_flight', ''),
                    flight.get('type_plane', 0)
                )
            )
        print(line)
    else:
        print('list is empty')


def select_flights(flights, t):
    result = []
    for flight in flights:
        if t in str(flight.values()):
            result.append(flight)
    return result


def save_flights(file_name, flights):
    with open(file_name, "w", encoding="utf-8", errors="ignore") as fout:
        json.dump(flights, fout, ensure_ascii=False, indent=4)


def load_flights(file_name):
    with open(file_name, "r", encoding="utf-8", errors="ignore") 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(
        "--data",
        action="store",
        required=False,
        help="The data file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("flights")
    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 flight"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination of the flight"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="Number of the flight"
    )
    add.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="Type of the plane"
    )

    # Создать субпарсер для отображения всех работников.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all flights"
    )

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the flights"
    )
    select.add_argument(
        "-s",
        "--select",
        action="store",
        required=True,
```

```
        help="The required select"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    data_file = args.data
    dotenv_path = os.path.join(os.path.dirname(__file__), ".env")
    if os.path.exists(dotenv_path):
        load_dotenv(dotenv_path)
    if not data_file:
        data_file = os.getenv("FLIGHTS_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):
        flights = list(load_flights(data_file))
    else:
        flights = []


    # Добавить работника.
    if args.command == "add":
        flights = add_flight(
            flights,
            args.destination,
            args.number,
            args.type
        )
        is_dirty = True

    # Отобразить всех работников.
    elif args.command == "display":
        display_flights(flights)

    # Выбрать требуемых рааботников.
    elif args.command == "select":
        selected = select_flights(flights, args.select)
        display_flights(selected)

    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_flights(data_file, flights)


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

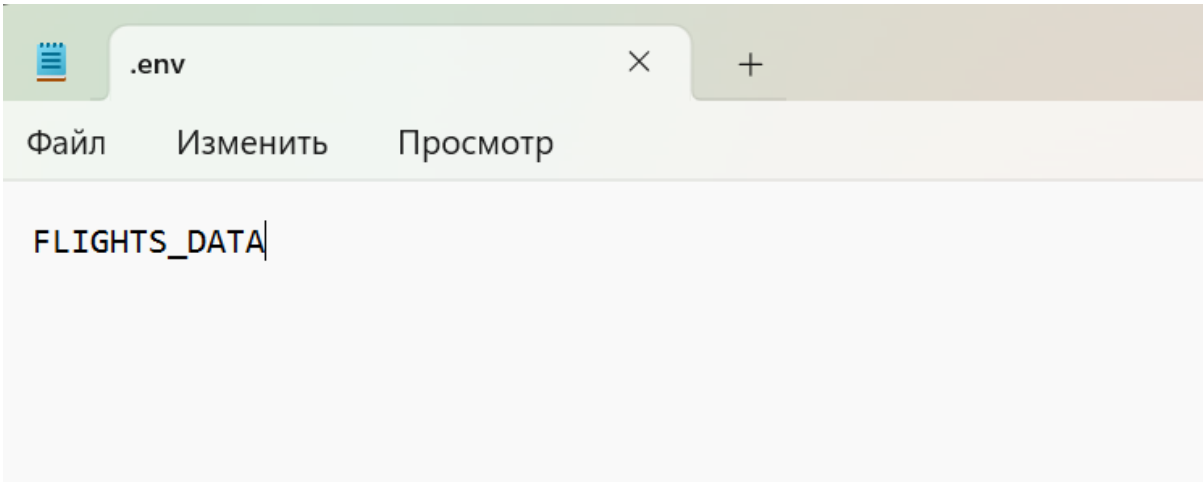

Рисунок 8 – Содержимое файла .env

```
C:\OPI\lbrrt_2.18\indiv>python indiv1.py select -s official
+------+--------------------------------+----------------------+--------------------+
|  №   |          Destination           |  NumberOfTheFlight   |   TypeOfThePlane   |
+------+--------------------------------+----------------------+--------------------+
|    1 | korea                          | 2                    |           official |
|    2 | canada                         | 3                    |           official |
|    3 | canada                         | 3                    |           official |
|    4 | america                        | 1                    |           official |
+------+--------------------------------+----------------------+--------------------+

C:\OPI\lbrrt_2.18\indiv>
```

Рисунок 9 – Результат работы программы

Контрольные вопросы

1. Каково назначение переменных окружения?

Переменные окружения используются для передачи информаци 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 толькок 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.

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.

PS1 – определяет внешний вид строки приглашения ввода новых команд.

PS2 – вторичная строка приглашения.

SHELLOPTS – параметры оболочки, которые можно устанавливать спомощью команды set.

UID – идентификатор текущего пользователя.

11. Как установить переменные оболочки в Linux?

Чтобы создать новую переменную оболочки с именем, нужно ввести имя этой переменной потом знак равенства и указать значение новой переменной

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
PYTHONVERBOSEPYTHONCASEOK
PYTHONDONTWRITEBYTECODE
PYTHONPYCACHEPREFIXPYTHONHASHSEED
PYTHONIOENCODING
PYTHONNOUSERSITE PYTHONUSERBASE PYTHONWARNINGS
PYTHONFAULTHANDLER

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

Путём использования модуля os, при помощи которого программист может получить и изменить значения всех переменных среды.

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

При помощи модуля os можно просмотреть все переменные окружения,

у которых есть значение.

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

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