CERTIFIED ASSOCIATE IN PYTHON PROGRAMMING

BY: IMRAN

Module 4:
MISCELLANEOUS
Part 3: The os module

Introduction to the os module

- In this section, you'll learn about a module called *os*, which lets you **interact with the operating system using Python**.
- It provides functions that are available on Unix and/or Windows systems. If you're familiar with the command console, you'll see that some functions give the same results as the commands available on the operating systems.
- A good example of this is the mkdir function, which allows you to create a directory just like the *mkdir* command in Unix and Windows. If you don't know this command, don't worry.
-

ImranNust
imran_muet
muhammad-imran-b7865495

Getting information about the operating system

- Before you create your first directory structure, you'll see how you can get information about the current operating system. This is really easy because the *os* module provides a function called *uname*, which returns an object containing the following attributes:
  - **systemname** — stores the name of the operating system;
  - **nodename** — stores the machine name on the network;
  - **release** — stores the operating system release;
  - **version** — stores the operating system version;
  - **machine** — stores the hardware identifier, e.g., x86_64.

Getting information about the operating system

- Let's look at how it is in practice:
- `import os`
- `print(os.uname())`
- Result:
- posix.uname_result(sysname='Linux',
- As you can see, the *uname* function returns an object containing information about the operating system. The above code was launched on Ubuntu 16.04.6 LTS, so don't be surprised if you get a different result, because it depends on your operating system.

Getting information about the operating system

- Unfortunately, the *uname* function only works on some Unix systems. If you use Windows, you can use the *uname* function in the *platform* module, which returns a similar result.
- The *os* module allows you to quickly distinguish the operating system using the *name* attribute, which supports one of the following names:
  - **posix** — you'll get this name if you use Unix;
  - **nt** — you'll get this name if you use Windows;
  - **java** — you'll get this name if your code is written in Jython.

Getting information about the operating system

- For Ubuntu 16.04.6 LTS, the *name* attribute returns the name *posix*:
- `import os`
- `print(os.name)`
- Result:
- posix
- **NOTE:** On Unix systems, there's a command called *uname* that returns the same information (if you run it with the -a option) as the *uname* function.

Creating directories in Python

- The *os* module provides a function called *mkdir*, which, like the *mkdir* command in Unix and Windows, allows you to create a directory. The *mkdir* function requires a path that can be relative or absolute. Let's recall what both paths look like in practice:
  - **my_first_directory** — this is a relative path which will create the *my_first_directory* directory in the current working directory;
  - **./my_first_directory** — this is a relative path that explicitly points to the current working directory. It has the same effect as the path above;
  - **../my_first_directory** — this is a relative path that will create the *my_first_directory* directory in the parent directory of the current working directory;
  - **/python/my_first_directory** — this is the absolute path that will create the *my_first_directory* directory, which in turn is in the *python* directory in the root directory.
-

Creating directories in Python

- Look at the code

```
import os
os.mkdir("my_first_directory")
print(os.listdir())
```

- It shows an example of how to create the *my_first_directory* directory using a relative path. This is the simplest variant of the relative path, which consists of passing only the directory name.

Creating directories in Python

- The *mkdir* function creates a directory in the specified path. Note that running the program twice will raise a *FileExistsError*.
- This means that we cannot create a directory if it already exists. In addition to the path argument, the *mkdir* function can optionally take the *mode* argument, which specifies directory permissions. However, on some systems, the *mode* argument is ignored.
- To change the directory permissions, we recommend the *chmod* function, which works similarly to the *chmod* command on Unix systems. You can find more information about it in the documentation.
-

Creating directories in Python

- In the above example, another function provided by the *os* module named *listdir* is used. The *listdir* function returns a list containing the names of the files and directories that are in the path passed as an argument.
- If no argument is passed to it, the current working directory will be used (as in the example above). It's important that the result of the *listdir* function omits the entries '.' and '..', which are displayed, e.g., when using the *ls -a* command on Unix systems.
- **NOTE:** In both Windows and Unix, there's a command called *mkdir*, which requires a directory path. The equivalent of the above code that creates the *my_first_directory* directory is the *mkdir my_first_directory* command.
-

Recursive directory creation

- The mkdir function is very useful, but what if you need to create another directory in the directory you've just created. Of course, you can go to the created directory and create another directory inside it, but fortunately the *os* module provides a function called makedirs, which makes this task easier.
- The *makedirs* function enables recursive directory creation, which means that all directories in the path will be created. Let's look at the code in the editor and see how it is in practice.

Recursive directory creation

- import os
- os.makedirs("my_first_directory/my_second_directory")
- os.chdir("my_first_directory")
- print(os.listdir())

- The code should produce the following result:
- ['my_second_directory']
- The code creates two directories. The first of them is created in the current working directory, while the second in the *my_first_directory* directory.

Recursive directory creation

- You don't have to go to the *my_first_directory* directory to create the *my_second_directory* directory, because the *makedirs* function does this for you. In the example above, we go to the *my_first_directory* directory to show that the *makedirs* command creates the *my_second_directory* subdirectory.
- To move between directories, you can use a function called *chdir*, which changes the current working directory to the specified path. As an argument, it takes any relative or absolute path. In our example, we pass the first directory name to it.

Recursive directory creation

- **NOTE:** The equivalent of the *makedirs* function on Unix systems is the *mkdir* command with the *-p* flag, while in Windows, simply the *mkdir* command with the path:
- Unix-like systems:

  mkdir -p my_first_directory/my_second_directory
- Windows:

  mkdir my_first_directory/my_second_directory
-

Where am I now?

- import os
- os.makedirs("my_first_directory/my_second_directory")
- os.chdir("my_first_directory")
- print(os.getcwd())
- os.chdir("my_second_directory")
- print(os.getcwd())

Module 4:
MISCELLANEOUS
Part 3: The os module

Deleting directories in Python

- The *os* module also allows you to delete directories. It gives you the option of deleting a single directory or a directory with its subdirectories. To delete a single directory, you can use a function called rmdir, which takes the path as its argument. Look at the code in the editor.
- The above example is really simple. First, the *my_first_directory* directory is created, and then it's removed using the *rmdir* function. The *listdir* function is used as proof that the directory has been removed successfully. In this case, it returns an empty list. When deleting a directory, make sure it exists and is empty, otherwise an exception will be raised.
- To remove a directory and its subdirectories, you can use the removedirs function, which requires you to specify a path containing all directories that should be removed:

Deleting directories in Python

```
import os
os.makedirs("my_first_directory/my_second_directory")
os.removedirs("my_first_directory/my_second_directory")
print(os.listdir())
```

- As with the *rmdir* function, if one of the directories doesn't exist or isn't empty, an exception will be raised.
- **NOTE:** In both Windows and Unix, there's a command called *rmdir*, which, just like the *rmdir* function, removes directories. What's more, both systems have commands to delete a directory and its contents. In Unix, this is the *rm* command with the *-r* flag.

The system() function

- All functions presented in this part of the course can be replaced by a function called *system*, which executes a command passed to it as a string.
- The system function is available in both Windows and Unix. Depending on the system, it returns a different result.
- In Windows, it returns the value returned by the shell after running the command given, while in Unix, it returns the exit status of the process.
-

The system() function

- Let's look at the code
- import os

- returned_value = os.system("mkdir my_first_directory")
- print(returned_value)

- Result:
- 0
- The above example will work in both Windows and Unix. In our case, we receive exit status 0, which indicates success on Unix systems.
- This means that the *my_first_directory* directory has been created. As part of the exercise, try to list the contents of the directory where you created the *my_first_directory* directory.
-