File Handling in Python

File Handling in Python

- The file handling plays an important role when the data needs to be stored permanently into the file. File is the **smallest unit** to store the data permanently.
- Why do you need to store the data - Persistence.
- A file is a named location on disk to store related information. We can access the stored information (non-volatile) after the program termination.
- File Handling is easier and shorter in Python.

File Handling in Python

- In Python, a file may be in the text or binary format.
- Each line of a file is ended with the special character called newline character.
- In Python, we represent the newline character as a \n
- A file operation can be done in the following order:
  - Open a file
  - Read or write - Performing operation
  - Close the file

- The key function for working with files in Python is the open() function.

- The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

- In addition you can specify if the file should be handled as binary or text mode
- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

- To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

- Note: Make sure the file exists, or else you will get an error.

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

- f = open("demofile.txt", "r")
- print(f.read())

If the file is located in a different location, you will have to specify the file path, like this

- f = open("D:\\myfiles\welcome.txt", "r")
  print(f.read())

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

- Return the 5 first characters of the file:
- f = open("demofile.txt", "r")
- print(f.read(5))

Read Lines

- You can return one line by using the readline() method:
- Read one line of the file:
- f = open("demofile.txt", "r")
- print(f.readline())

By calling readline() two times, you can read the two first lines:

Example

Read two lines of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

Loop through the file line by line:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Close Files

- It is a good practice to always close the file when you are done with it.
- Example
- Close the file when you are finish with it:
-
```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

Remove the file "demofile.txt":

```
import os

os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

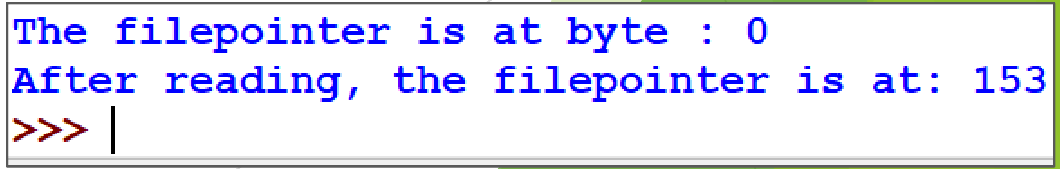
File Pointer Operations

- Python provides the **tell()** method which is used to print the byte number at which the file pointer currently exists. Consider the following example.

```
# open the file file2.txt in read mode
fileptr = open("file2.txt","r")

#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())

#reading the content of the file
content = fileptr.read();
"""
after the read operation file pointer modifies. tell() returns the location of th
e fileptr.
"""
print("After reading, the filepointer is at:",fileptr.tell())
```



```
The filepointer is at byte : 0
After reading, the filepointer is at: 153
>>> 
```

Modifying File Pointer position

In real-world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.

seek() method which enables us to modify the file pointer position externally.

Modifying File Pointer position

Consider the following example:

```
# open the file file2.txt in read mode
fileptr = open("file2.txt","r")

#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())

#changing the file pointer location to 10.
fileptr.seek(10);

#tell() returns the location of the fileptr.
print("After reading, the filepointer is at:",fileptr.tell())
```

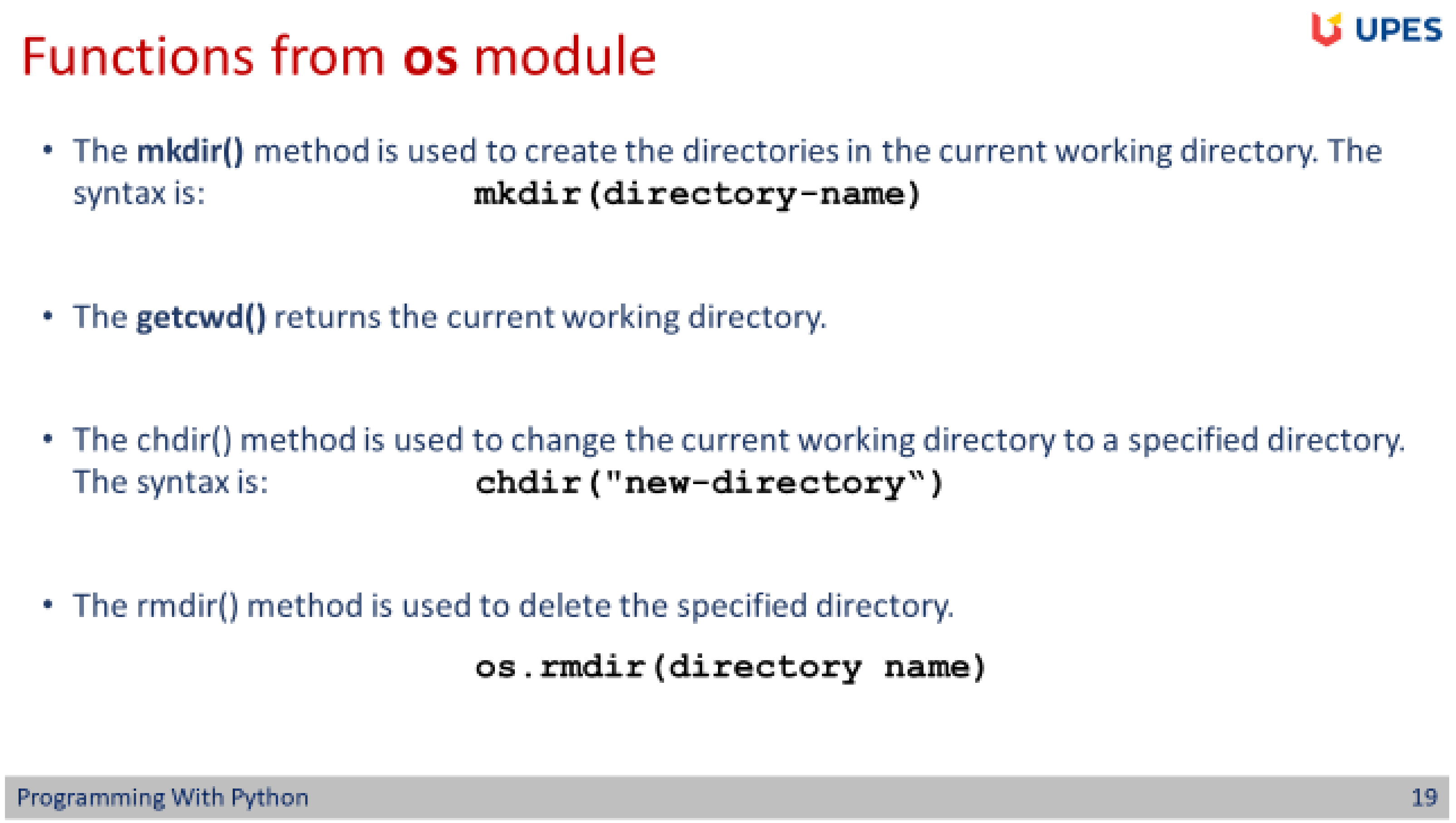
File Methods

SN	Method	Description
1	file.close()	It closes the opened file. The file once closed, it can't be read or write anymore.
2	File.fush()	It flushes the internal buffer.
3	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4	File.isatty()	It returns true if the file is connected to a TTY device, otherwise returns false.
5	File.next()	It returns the next line from the file.
6	File.read([size])	It reads the file for the specified size.
7	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.

File Methods

8	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.
9	File.seek(offset[,from)	It modifies the position of the file pointer to a specified offset with the specified reference.
10	File.tell()	It returns the current position of the file pointer within the file.
11	File.truncate([size])	It truncates the file to the optional specified size.
12	File.write(str)	It writes the specified string to a file
13	File.writelines(seq)	It writes a sequence of the strings to a file.

Questions



Functions from **os** module

UPES

- The **mkdir()** method is used to create the directories in the current working directory. The syntax is:

```
mkdir(directory-name)
```

- The **getcwd()** returns the current working directory.

- The chdir() method is used to change the current working directory to a specified directory. The syntax is:

```
chdir("new-directory")
```

- The rmdir() method is used to delete the specified directory.

```
os.rmdir(directory name)
```

Programming With Python 19