Semana10

November 21, 2018

#
Programming Foundations @ LEIC/LETI
##
Week 10
#
Introduction To Files In Python

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Why are files an important concept? It is the way we can persist data (in other words, keep context from one execution of a program to another).

Properties of files: - They are independent of the program - During the execution of a program, a file can be in one of the following *states* - Reading state - Writing state

#
Syntax

```
file_object = open(file_name [, access_mode][, buffering])
```

Where: - **file_name** : The file_name argument is a string value that contains the name of the file that you want to access. - **access_mode**: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. This is optional and the default file access mode is read `r`. - **buffering**: If the buffering value is set to `0`, no buffering takes place. If the buffering value is `1`, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than `1`, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default behavior.

#
Access modes

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

Modes	Description
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

Modes	Description
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

Modes	Description
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

Modes	Description
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Modes	Description
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

Modes	Description
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

Modes	Description
ab	ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

Modes	Description
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Modes	Description
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

#
The file object attributes

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.

```
In [2]: fo = open("foo.txt", "w")
        print("Name of the file: ", fo.name)
        print("Closed or not : ", fo.closed)
        print("Opening mode : ", fo.mode)



        fo
```

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  w
```

```
Out[2]: <_io.TextIOWrapper name='foo.txt' mode='w' encoding='UTF-8'>
```

#

The close Method

The close method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close method to close a file.

```
fileObject.close()
```

```
In [3]: # Open a file
        fo = open("foo.txt", "wb")

        # Close opend file
        fo.flush()
        fo.close()

        print("Closed or not : ", fo.closed)

        fo
```

```
Closed or not :  True
```

```
Out[3]: <_io.BufferedWriter name='foo.txt'>
```

#
Reading and Writing Files

The file object provides a set of access methods to make our lives easier. We would see how to use read and write methods to read and write files.

#
The write Method

The write method writes any string to an open file. The write method does not add a newline character \n to the end of the string.

```
fileObject.write(string);
```

```
In [7]: # Open a file
        fo = open("foo.txt", "r+")

        #wirte to the file
        fo.write( "AAA");

        # Close opend file
        fo.flush()
        fo.close()
```

```
In [10]: # Open a file
         fo = open("foo1.txt", "wb")

         #write to the file
         fo.write( b'Python is a great language.\nYeah its great!!\n' );

         # Close opend file
         fo.close()


         print(list(str.encode("Python is a great language.\nYeah its great!!\n")))
```

```
[80, 121, 116, 104, 111, 110, 32, 105, 115, 32, 97, 32, 103, 114, 101, 97, 116, 32, 108, 97, 110
```

#
The read Method

The read method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

```
In [17]: # Open a file
         fo = open("foo.txt", "r")
```

```
        s = fo.read(10);
        print("Read String is : ", s)

        s = fo.read(10);
        print(s)

        s = fo.readline();
        print(s)

        s = fo.readlines()
        print(s)

        # Close opend file
        fo.flush()
        fo.close()
```

```
Read String is :  Python is_
a great la
nguage.

['Yeah its great!!\n', 'XPTO111']
```

#
File Positions

The `tell` method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The `seek(offset[, from])` method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.

If from is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

```
In [21]: # Open a file
         fo = open("foo.txt", "r+")
         str = fo.read(10);

         print("Read String is : ", str)

         # Check current position
         position = fo.tell();
         print("Current file position : ", position)

         # Reposition pointer at the beginning once again
         position = fo.seek(fo.tell()-2,0);
         str = fo.read(10);
         print("Again read String is : ", str)

         # Close opend file
         fo.close()


         import os
         print(os.getcwd())
```

```
Read String is :  Python is_
Current file position :  10
Again read String is :  s_a great
/Users/ruimaranhao/Desktop/IST/fp17-tagus/notebooks
```

#
Other operations

Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files. To use this module you need to import it first and then you can call any related functions.

```
import os
# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )

# Delete file test2.txt
os.remove("text2.txt")
```

All files are contained within various directories, and Python has no problem handling these too. The os module has several methods that help you `create`, `remove`, and `change` directories. Go learn this yourself!

```
In [27]: f = open('foo.txt', 'r')

         for line in f:
             print(line, end='')

         f.flush()
         f.close()
```

```
Python is_a great language.
Yeah its great!!
XPTO111
```

```
In [33]: #Advanced topics

         #lst = []
         #with open('foo.txt') as f:
         #    for x in f:
```

```
         #        for c in x:
         #            lst.append(c)
         #
         #print(lst)

         #read a file line by line
         #with open('foo.txt') as f:
         #    lines = f.readlines()
         #    print(lines)

         #list comprehension
         #lines = [line for line in open('foo.txt')]
         #print(lines)

         #print(list(open('foo.txt')))

         #lstc = [x for line in open('foo.txt') for x in line]
         #print(lstc)
```

```
['Python is_a great language.\n', 'Yeah its great!!\n', 'XPTO111']
```

1 Extra: Binary files

Up to now, we have discussed text files. For binary files, check this out:
https://www.devdungeon.com/content/working-binary-data-python

```
In [34]: l = [0,1,2]

         def add(x):
             x = list(x)
             print(x)
             x[0] = 2
             print(x)


         print(l)
         add(l)
         print(l)
```

```
[0, 1, 2]
[0, 1, 2]
[2, 1, 2]
[2, 1, 2]
```

```
In [ ]:
```