**Reading and Writing Files**

So far, we have learned how to write programs and communicate our intentions to the Central Processing Unit using conditional execution, functions, and iterations. We have learned how to create and use data structures in the Main Memory. The CPU and memory are where our software works and runs. It is where all of the ``thinking'' happens.

But if you recall from our hardware architecture discussions, once the power is turned off, anything stored in either the CPU or main memory is erased. So up to now, our programs have just been transient fun exercises to learn Python.

we start to work with Secondary Memory (or files). Secondary memory is not erased even when the power is turned off. Or in the case of a USB flash drive, the data can we write from our programs can be removed from the system and transported to another system.

We will primarily focus on reading and writing text files such as those we create in a text editor. Later we will see how to work with database files which are binary files, specifically designed to be read and written through database software

**Obtaining References to Files**

Reading from, and writing to, text files in Python is relatively straightforward. The built in open() function creates a file object for you that you can use to read and/ or write data from and/ or to a file. The function requires as a minimum the name of the file you want to work with. Optionally you can specify the access mode (e.g. read, write, append etc.). If you do not specify a mode then the file is open in read-only mode. You can also specify whether you want the interactions with the file to be buffered which can improve performance by grouping data reads together

The syntax for the open() function is

---> file\_object = open(file\_name, access\_mode, buffering)

Where

• file\_name indicates the file to be accessed.

• access\_mode The access\_mode determines the mode in which the file is to be opened, i.e. read, write, append, etc. A complete list of possible values is given below in the table. This is an optional parameter 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.

The access\_mode values are given in the following table.

​

​

Mode           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

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

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

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

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

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

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

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

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

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 itself has several useful attributes such as

• file.closed returns True if the file has been closed (can no longer be accessed because the close() method has been called on it).

• file.mode returns the access mode with which the file was opened.

• file.name The name of the file.

The file.close() method is used to close the file once you have finished with it. This will flush any unwritten information to the file (this may occur because of buffering) and will close the reference from the file object to the actual underlying operating system file. This is important to do as leaving a reference to a file open can cause problems in larger applications as typically there are only a certain number of file references possible at one time and over a long period of time these may all be used up resulting in future errors being thrown as files can no longer be opened

filehand **=** open("myfile.txt","w+")

print("The name of the file is ",filehand.name)

print("The mode of opening is",filehand.mode)

print("Whether the file is closed",filehand.closed) *#before closing*

​

filehand.close() *#closing the file*

print("Whether the file is closed",filehand.closed) *#after closing*

​

​

​

The name of the file is myfile.txt

The mode of opening is w+

Whether the file is closed False

Whether the file is closed True

**Reading Files**

Of course, having set up a file object we want to be able to either access the contents of the file or write data to that file (or do both). Reading data from a text file is supported by the read(), readline() and readlines() methods:

• The read() method This method will return the entire contents of the file as a single string.

• The readline() method reads the next line of text from a file. It returns all the text on one line up to and including the newline character. It can be used to read a file a line at a time.

• The readlines() method returns a list of all the lines in a file, where each item of the list represents a single line.

Note that once you have read some text from a file using one of the above operations then that line is not read again. Thus using readlines() would result in a further readlines() returning an empty list whatever the contents of the file

**%%**writefile myfile.txt

"This is the first file that is been created"

"Writing the contents to the file"

Overwriting myfile.txt

filehand **=** open("myfile.txt","r+")

lines **=** filehand.readlines()

print(lines) *#prints as list*

**for** line **in** lines:

print(line,end**=**'') *#line by line will be printed with newline character by default , to avoid it can use end value*

filehand.close()

​

*# Notice that within the for loop we have indicated to the print function that*

*# we want the end character to be ' ' rather than a newline; this is because the line*

*# string already possesses the newline character read from the file.*

['"This is the first file that is been created"\n', '"Writing the contents to the file"\n']

"This is the first file that is been created"

"Writing the contents to the file"

**File Contents Iteration**

As suggested by the previous example; it is very common to want to process the contents of a file one line at a time. In fact Python makes this extremely easy by making the file object support iteration. File iteration accesses each line in the file and makes that line available to the for loop. We can therefore write:

filehand **=** open('myfile.txt', 'r')

**for** line **in** filehand:

print(line, end**=**'')

filehand.close()

​

"This is the first file that is been created"

"Writing the contents to the file"

*# It is also possible to use the list comprehension to provide a very concise way to*

*# load and process lines in a file into a list. It is similar to the effect of readlines()*

*# but we are now able to pre-process the data before creating the list:*

filehand **=** open('myfile.txt', 'r')

lines **=** [line.upper() **for** line **in** filehand]

filehand.close()

print(lines)

['"THIS IS THE FIRST FILE THAT IS BEEN CREATED"\n', '"WRITING THE CONTENTS TO THE FILE"\n']

**Writing Data to Files**

Writing a string to a file is supported by the write() method. Of course, the file object we create must have an access mode that allows writing (such as 'w'). Note that the write method does not add a newline character (represented as '\n') to the end of the string—you must do this manually

filehand **=** open("yourfile.txt","w+")

filehand.write("Heloo from python\n")

filehand.write("Working with files is easy\n")

filehand.write("its easy\n")

filehand.close()

filehand **=** open("yourfile.txt","r")

print(filehand.readlines())

filehand.close()

['Heloo from python\n', 'Working with files is easy\n', 'its easy\n']

*#we can perform string operation on the data reteived*

filehand **=** open("yourfile.txt","r")

**for** line **in** filehand:

line **=** line.lstrip() *#can also use rstrip or lstrip*

**if** line.startswith("Working"): *# we can perform any of the string operations*

print(line)

Working with files is easy

*# we can even ask the user to enter the filename*

fname **=** input("Enter the filename to open")

print(fname)

**try**:

filehand **=** open(fname,"r")

print(filehand.readlines())

filehand.close()

**except**:

print("File can not be opened")

exit()

Enter the filename to openyourfile.txt

yourfile.txt

['Heloo from python\n', 'Working with files is easy\n', 'its easy\n']

**Using Files and with Statements**

Like several other types where it is important to shut down resources; the file object class implements the Context Manager Protocol and thus can be used with the with statement. It is therefore common to write code that will open a file using the with as structure thus ensuring that the file will be closed when the block of code is finished with,

​

**with** open("yourfile.txt","r") **as** filehand:

lines **=** filehand.readlines()

**for** line **in** lines:

print(line)

​

​

​

*#need not close the file exclusively as the context manager will close the file once it comes out of the block*

Heloo from python

Working with files is easy

its easy

**The Fileinput Module**

In some situations, you may need to read the input from several files in one go. You could do this by opening each file independently and then reading the contents and appending that contents to a list etc. However, this is a common enough requirement that the fileinput module provides a function fileinput.input() that can take a list of files and treat all the files as a single input significantly simplifying this process,

**import** fileinput

**with** fileinput.input(files **=**("myfile.txt","yourfile.txt")) **as** filehand:

**for** line **in** filehand:

print(line)

"This is the first file that is been created"

"Writing the contents to the file"

Heloo from python

Working with files is easy

its easy

Features provided by the fileinput module include

• Return the name of the file currently being read.

• Return the integer “file descriptor” for the current file.

• Return the cumulative line number of the line that has just been read.

• Return the line number in the current file. Before the first line has been read this returns 0.

• A boolean function that indicates if the current line just read is the first line of its file

**import** fileinput

**with** fileinput.input(files **=** ("myfile.txt","yourfile.txt")) **as** filehand:

line **=** filehand.readline()

print("filename: " ,filehand.filename())

print("isfirstline:",filehand.isfirstline())

print("lineno:",filehand.lineno())

print("filelineno:",filehand.filelineno())

**for** lines **in** filehand: *#should be within the block of with statement else the file will be closed*

print(lines)

filename: myfile.txt

isfirstline: True

lineno: 1

filelineno: 1

"Writing the contents to the file"

Heloo from python

Working with files is easy

its easy

**Random Access Files**

All the examples presented so far suggest that files are accessed sequentially, with the first line read before the second and so on. Although this is (probably) the most common approach it is not the only approach supported by Python; it is also possible to use a random-access approach to the contents within a file. To understand the idea of random file access it is useful to understand that we can maintain a pointer into a file to indicate where we are in that file in terms of reading or writing data. Before anything is read from a file the pointer is before the beginning of the file and reading the first line of text would for example, advance the point to the start of the second line in the file etc When randomly accessing the contents of a file the programmer manually moves the pointer to the location required and reads or writes text relative to that pointer. This means that they can move around in the file reading and writing data. The random-access aspect of a file is provided by the seek method of the file object:

• file.seek (offset, whence) this method determines where the next read or write operation (depending on the mode used in the open() call) takes place.

In the above the offset parameter indicates the position of the read/ write pointer within the file. The move can also be forwards or backwards (represented by a negative offset). The optional whence parameter indicates where the offset is relative to. The values used for whence are

0 indicates that the offset is relative to start of file (the default).

• 1 means that the offset is relative to the current pointer position.

• 2 indicates the offset is relative to end of file.

Thus, we can move the pointer to a position relative to the start of the file, to the end of the file, or to the current position. For example, in the following sample code we create a new text file and write a set of characters into that file. At this point the pointer is positioned after the ‘z’ in the file. However, we then use seek() to move the point to the 10th character in the file and now write ‘Hello’, next we reposition the pointer to the 6th character in the file and write out ‘BOO’. We then close the file. Finally, we read all the lines from the file using a with as statement and the open() function and from this

**with** open("myfile.txt","w+") **as** filehand:

filehand.write("abcdefghijklmnopqrstuvwxyz\n")

filehand.seek(10,0)

filehand.write("BOO")

filehand.write("HELLO")

**with** open("myfile.txt","r") **as** filehand:

line **=** filehand.readlines()

print(line)

['abcdefghijBOOHELLOstuvwxyz\n']

*# copying content from temp.txt to temp2.txt,*

*# Bad for large files*

​

**with** open("myfile.txt","r") **as** temp1:

**with** open("yourfile.txt","w") **as** temp2: *#opened in write mode , so the contents of yourfile is erased and myfile contents is written*

temp2.writelines(temp1.readlines())

**with** open("yourfile.txt","r") **as** filehand:

line **=** filehand.readlines()

print(line)

['"This is the first file that is been created"\n', '"Writing the contents to the file"\n']

*#can use even write method to write the file contents*

**with** open("myfile.txt","r") **as** temp1:

**with** open("yourfile.txt", "w") **as** temp2:

**for** line **in** temp1.readlines():

print(line,end **=**" ")

temp2.write(line)

abHELLOhijBOOnopqrstuvwxyz

*#we can provide multiple opening of files in one line itself*

**with** open("myfile.txt","r") **as** temp1 , open("yourfile.txt","w") **as** temp2:

**for** line **in** temp1.readlines():

print(line,end **=**" ")

temp2.write(line)

"This is the first file that is been created"

"Writing the contents to the file"

*# If we have to copy only few top lines from file, than something similar to below example can be used.*

*# In it we are using optional parameter size to tell how much data needs to be copied.*

*# if we use function to readlines then the sixe provided will read that many lines , for read function will read that many chars*

​

**with** open("myfile.txt","r") **as** temp1 ,open("yourfile.txt","w") **as** temp2:

**for** line **in** temp1.read(8):

print(line)

temp2.write(line)

"

T

h

i

s

i

s

**with** open("myfile.txt","r") **as** temp1 ,open("yourfile.txt","w") **as** temp2:

**for** s **in** range(8,13):

**for** line **in** temp1.read(s):

print("\*"**\***s,s)

print(line)

temp2.write(line)

\*\*\*\*\*\*\*\* 8

a

\*\*\*\*\*\*\*\* 8

b

\*\*\*\*\*\*\*\* 8

H

\*\*\*\*\*\*\*\* 8

E

\*\*\*\*\*\*\*\* 8

L

\*\*\*\*\*\*\*\* 8

L

\*\*\*\*\*\*\*\* 8

O

\*\*\*\*\*\*\*\* 8

h

\*\*\*\*\*\*\*\*\* 9

i

\*\*\*\*\*\*\*\*\* 9

j

\*\*\*\*\*\*\*\*\* 9

B

\*\*\*\*\*\*\*\*\* 9

O

\*\*\*\*\*\*\*\*\* 9

O

\*\*\*\*\*\*\*\*\* 9

n

\*\*\*\*\*\*\*\*\* 9

o

\*\*\*\*\*\*\*\*\* 9

p

\*\*\*\*\*\*\*\*\* 9

q

\*\*\*\*\*\*\*\*\*\* 10

r

\*\*\*\*\*\*\*\*\*\* 10

s

\*\*\*\*\*\*\*\*\*\* 10

t

\*\*\*\*\*\*\*\*\*\* 10

u

\*\*\*\*\*\*\*\*\*\* 10

v

\*\*\*\*\*\*\*\*\*\* 10

w

\*\*\*\*\*\*\*\*\*\* 10

x

\*\*\*\*\*\*\*\*\*\* 10

y

\*\*\*\*\*\*\*\*\*\* 10

z

\*\*\*\*\*\*\*\*\*\* 10

*#we should not use readlines for larger file size*

**with** open("myfile.txt","r") **as** temp ,open("yourfile.txt","w") **as** filehand:

**for** line **in** temp:

print(line,end **=**"")

filehand.writelines(line)

abHELLOhijBOOnopqrstuvwxyz

**File Systems**

Both Unix like systems and Windows operating systems are hierarchical structures comprising directories and files. The os module has several functions that can help with creating, removing and altering directories

• mkdir() This function is used to create a directory, it takes the name of the directory to create as a parameter. If the directory already exists FileExistsError is raised.

• chdir() This function can be used to change the current working directory. This is the directory that the application will read from/ write to by default.

• getcwd() This function returns a string representing the name of the current working directory.

• rmdir() This function is used to remove/ delete a directory. It takes the name of the directory to delete as a parameter.

• listdir() This function returns a list containing the names of the entries in the directory specified as a parameter to the function (if no name is given the current directory is used).

**import** os

​

print("Get the current working directory",os.getcwd())

print("List the contents of the directory",os.listdir())

print("Create new directory",os.mkdir("MyDir")) *#while creating the directory give the name of the directory to be created*

print("List the updated contents with the new directory created",os.listdir()) *#it will display MyDir folder also*

print("Now check the path of the directory ",os.getcwd())

​

Get the current working directory C:\Users\asha.t\AppData\Local\Programs\Python\Python37-32\Python - Training\11 - FileSystem

List the contents of the directory ['.ipynb\_checkpoints', '08-Files.ipynb', 'binary\_pickle', 'binary\_pickle.dat', 'emp.dat', 'Emp.py', 'Example.csv', 'File1.txt', 'File2.txt', 'File3.txt', 'Files System.ipynb', 'myfile.txt', 'newtest.zip', 'samplenew.txt', 'test', 'test.txt', 'test.zip', 'yourfile.txt']

Create new directory None

List the updated contents with the new directory created ['.ipynb\_checkpoints', '08-Files.ipynb', 'binary\_pickle', 'binary\_pickle.dat', 'emp.dat', 'Emp.py', 'Example.csv', 'File1.txt', 'File2.txt', 'File3.txt', 'Files System.ipynb', 'MyDir', 'myfile.txt', 'newtest.zip', 'samplenew.txt', 'test', 'test.txt', 'test.zip', 'yourfile.txt']

Now check the path of the directory C:\Users\asha.t\AppData\Local\Programs\Python\Python37-32\Python - Training\11 - FileSystem

print("Change to the newly created directory",os.chdir("MyDir"))

print("get the path of the newly created directory",os.getcwd())

Change to the newly created directory None

get the path of the newly created directory C:\Users\asha.t\AppData\Local\Programs\Python\Python37-32\Python - Training\11 - FileSystem\MyDir

print("Change back to the previous directory",os.chdir('C:\\Users\\asha.t\\AppData\\Local\\Programs\\Python\\Python37-32\\Python - Training\\11 - FileSystem'))

print("Now check the path",os.getcwd())

print(os.listdir())

Change back to the previous directory None

Now check the path C:\Users\asha.t\AppData\Local\Programs\Python\Python37-32\Python - Training\11 - FileSystem

['.ipynb\_checkpoints', '08-Files.ipynb', 'binary\_pickle', 'binary\_pickle.dat', 'emp.dat', 'Emp.py', 'Example.csv', 'File1.txt', 'File2.txt', 'File3.txt', 'Files System.ipynb', 'MyDir', 'myfile.txt', 'newtest.zip', 'samplenew.txt', 'test', 'test.txt', 'test.zip', 'yourfile.txt']

print("Remove the newly created directory",os.rmdir("MyDir"))

print("List the contents now",os.listdir())

Remove the newly created directory None

List the contents now ['.ipynb\_checkpoints', '08-Files.ipynb', 'binary\_pickle', 'binary\_pickle.dat', 'emp.dat', 'Emp.py', 'Example.csv', 'File1.txt', 'File2.txt', 'File3.txt', 'Files System.ipynb', 'myfile.txt', 'newtest.zip', 'samplenew.txt', 'test', 'test.txt', 'test.zip', 'yourfile.txt']

*# os.rmdir("MyDir")*

print("To create directory",os.mkdir("MyDir"))

print("To create sub directory",os.mkdir("MyDir/MySubDir"))

print("List the updated contents",os.listdir())

print("Chge to MyDir",os.chdir("MyDir"))

print("Now List",os.listdir())

To create directory None

To create sub directory None

List the updated contents ['.ipynb\_checkpoints', '08-Files.ipynb', 'binary\_pickle', 'binary\_pickle.dat', 'emp.dat', 'Emp.py', 'Example.csv', 'File1.txt', 'File2.txt', 'File3.txt', 'Files System.ipynb', 'MyDir', 'myfile.txt', 'newtest.zip', 'samplenew.txt', 'test', 'test.txt', 'test.zip', 'yourfile.txt']

Chge to MyDir None

Now List ['MySubDir']

os.chdir('..')

print("To remove directories",os.removedirs("MyDir/MySubDir"))

To remove directories None

*#os.walk(path, topdown=True, onerror=none,followlikns=False) - will return iterator object whose contents can be*

*#displayed using loop, it contains directory apth,directory names and filenames*

*#to display contents of the current directory*

​

*# When we call os.walk and give it a starting directory, it will ``walk'' through all of the directories and sub-directories*

*# recursively.*

*# The string ``.'' indicates to start in the current directory and walk downward.*

*# As it encounters each directory, we get three values in a tuple in the body of the for loop.*

*# The first value is the current directory name, the second value is the list of sub-directories in the current directory,*

*# and the third value is a list of files in the current directory.*

​

​

**for** dirpath,dirnames,filenames **in** os.walk('.'): *#from current directory*

print("Dirpath is :",dirpath)

print("Dirnames are :",dirnames)

print("All the files ",filenames)

Dirpath is : .

Dirnames are : ['.ipynb\_checkpoints', 'test']

All the files ['08-Files.ipynb', 'binary\_pickle', 'binary\_pickle.dat', 'emp.dat', 'Emp.py', 'Example.csv', 'File1.txt', 'File2.txt', 'File3.txt', 'Files System.ipynb', 'myfile.txt', 'sample.txt', 'test.txt', 'test.zip', 'yourfile.txt']

Dirpath is : .\.ipynb\_checkpoints

Dirnames are : []

All the files ['08-Files-checkpoint.ipynb', 'Files System-checkpoint.ipynb']

Dirpath is : .\test

Dirnames are : []

All the files []

Renaming Files

A file can be renamed using the os.rename() function. This function takes two arguments, the current filename and the new filename. It is part of the Python os module which provides methods that can be used to perform a range of file-processing operations (such as renaming a file).

Deleting Files

A file can be deleted using the os.remove() method. This method deletes the file specified by the filename passed to it. Again, it is part of the os module

**import** os

os.rename("sample.txt","samplenew.txt") *#originalfile , newfilename*

​

​

os.remove("yourfile.txt")

**Working with Paths**

The pathlib module provides a set of classes representing filesystem paths; that is paths through the hierarchy of directories and files within an operating systems file structure. It was introduced in Python 3.4. The core class in this module is the Path class. A Path object is useful because it provides operations that allow you to manipulate and manage the path to a file or directory. The Path class also replicates some of the operations available from the os module (such as mkdir, rename and rmdir) which means that it is not necessary to work directly with the os module. A path object is created using the Path constructor function; this function actually returns a specific type of Path depending on the type of operating system being used such as a WindowsPath or a PosixPath (for Unix style systems). The Path() constructor takes the path to create for example ‘D:/mydir’ (on Windows) or ‘/Users/user1/mydir’ on a Mac or ‘/var/temp’ on Linux etc. You can then use several different methods on the Path object to obtain information about the path such as:

• exists() returns True of False depending on whether the path points to an existing file or directory.

• is\_dir() returns True if the path points to a directory. False if it references a file. False is also returned if the path does not exist.

• is\_file() returns True of the path points to a file, it returns False if the path does not exist or the path references a directory.

• absolute() A Path object is considered absolute if it has both a root and (if appropriate) a drive.

• is\_absolute() returns a Boolean value indicating whether the Path is absolute or not.

*# Note that ‘..’ is a short hand for the parent directory of the current directory and*

*# ‘.’ is short hand for the current directory.*

**from** pathlib **import** Path

p **=** Path(".")

print("The path is",p)

print("Whether path exiists",p.exists())

print("Whether it is pointing to dir",p.is\_dir())

print("Whether it is pointing to file",p.is\_file()) *#it points to directory also*

print("What is the absolute path",p.absolute())

print("whether the path is absolute or not",p.is\_absolute())

​

The path is .

Whether path exiists True

Whether it is pointing to dir True

Whether it is pointing to file False

What is the absolute path C:\Users\asha.t\AppData\Local\Programs\Python\Python37-32\Python - Training\11 - FileSystem

whether the path is absolute or not False

There are also several methods on the Path class that can be used to create and remove directories and files such as:

• mkdir() is used to create a directory path if it does not exist. If the path already exists, then a FileExistsError is raised.

• rmdir() remove this directory; the directory must be empty otherwise an error will be raised

• rename(target) rename this file or directory to the given target.

• unlink() removes the file referenced by the path object.

• joinpath(\*other) appends elements to the path object e.g. path.joinpath(‘/ temp’).

• with\_name(new\_name) return a new path object with the name changed.

• The ‘/’ operator can also be used to create new path objects from existing paths for example path/ ‘test’/ ‘output’ which would append the directories test and out to the path object.

Two Path class methods can be used to obtain path objects representing key directories such as the current working directory (the directory the program is logically in at that point) and the home directory of the user running the program:

• Path.cwd() return a new path object representing the current directory.

• Path.home() return a new path object representing the user’s home directory

p **=** Path.cwd()

print("Setup new directory")

newdir **=** p **/**'test'

print("Check if new directory exists are not",newdir.exists()) *#it will be false as new directory is not yet created*

newdir.mkdir()

​

Setup new directory

Check if new directory exists are not False

print("Check if the new directory exists are not",newdir.exists())

Check if the new directory exists are not True

Modern operating systems store files in hierarchical structures called file systems. Several features related to file systems are implemented in the module os.path, such as:

os.path.basename(): returns the final component of a path

os.path.dirname(): returns a path without the final component.

os.path.exists(): returns True if the path exists or False otherwise.

os.path.getsize(): returns the size of the file in bytes.

os.path.join(): joins the files, directory

glob is another module related to the file system:

**import** os.path

​

a **=** r"/C/asha/"

b **=** r"conf/myprogram.conf"

​

c**=** os.path.join(a,b)

print(c)

/C/asha/conf/myprogram.conf

**import** os.path

​

list\_files **=** [r"/C/asha/",

r"/C/conf/"]

filename **=**"myconf.txt"

**for** conf **in** list\_files:

print(os.path.join(conf,filename))

/C/asha/myconf.txt

/C/conf/myconf.txt

**import** os.path

​

print(os.path.join(r"/C/asha/",r"conf/","myconf.txt"))

/C/asha/conf/myconf.txt

*#Similar to join even split is there*

**import** os.path

​

conf\_files **=** [r"/home/asha/apps/ls/warzone/myprogram.conf",

r"C:\Users\God\mayaappvhelper\conf\user.conf"

]

​

**for** conf **in** conf\_files:

print(os.path.split(conf))

('/home/asha/apps/ls/warzone', 'myprogram.conf')

('C:\\Users\\God\\mayaappvhelper\\conf', 'user.conf')

*#to forcibly use windows file seperrator only then ntpath is used*

**import** ntpath

db\_file **=** ntpath.join("appfolder", "config","database","db.sqlite")

print(db\_file)

​

cp **=** r"C:\Users\God\mayaappvhelper\conf\user.conf"

print(ntpath.split(cp))

appfolder\config\database\db.sqlite

('C:\\Users\\God\\mayaappvhelper\\conf', 'user.conf')

**import** os.path

​

conf\_files **=** [

("home", "asha", "app", "myprogram.conf"),

("C:", "Users", "God", "user.conf")

]

​

**for** conf **in** conf\_files:

print(os.path.join(**\***conf))

*# print(os.path.splittext(db\_file))*

home\asha\app\myprogram.conf

C:Users\God\user.conf

conf\_file **=** "appfolder\config\database\db.sqlite"

print(os.path.splitext(conf\_file))

print(os.path.splitext(conf\_file)[0] **+** ".json")

('appfolder\\config\\database\\db', '.sqlite')

appfolder\config\database\db.json

A very useful method **in** the Path object **is** the glob(pattern) method. This

method returns all elements within the path that meet the pattern specified.

For example path.glob(‘**\***.py’) will **return** all the files ending .py within

the current path.

Note that ‘**\*\*/\***.py’ would indicate the current directory **and** any sub directory.

For example, the following code will **return** all files where the file name ends **with**

‘.txt’ **for** a given path:

**import** os.path

**import** glob

**for** file **in** glob.glob('\*.txt'):

print(os.path.getsize(file)) *#getting the size of the files also along with files*

print(file)

​

0

File1.txt

0

File2.txt

0

File3.txt

28

myfile.txt

19

sample.txt

25

test.txt

57

yourfile.txt

**Temporary Files**

During the execution of many applications it may be necessary to create a temporary file that will be created at one point and deleted before the application finishes. It is of course possible to manage such temporary files yourself however, the tempfile module provides a range of facilities to simplify the creation and management of these temporary files. Within the tempfile module TemporaryFile, NamedTemporaryFile, TemporaryDirectory, and SpooledTemporaryFile are high-level file objects which provide automatic cleanup of temporary files and directories. These objects implement the Context Manager Protocol. The tempfile module also provides the lower-level function mkstemp() and mkdtemp() that can be used to create temporary files that require the developer to management them and delete them at an appropriate time. The high-level feature for the tempfile module are:

• TemporaryFile(mode=‘w+b’) Return an anonymous file-like object that can be used as a temporary storage area. On completion of the managed context (via a with statement) or destruction of the file object, the temporary file will be removed from the filesystem. Note that by default all data is written to the temporary file in binary format which is generally more efficient.

• NamedTemporaryFile(mode=‘w+b’) This function operates exactly as TemporaryFile() does, except that the file has s visible name in the file system.

• SpooledTemporaryFile(max\_size=0, mode=‘w+b’) This function operates exactly as TemporaryFile() does, except that data is spooled in memory until the file size exceeds max\_size, or until the file’s fileno () method is called, at which point the contents are written to disk and operation proceeds as with TemporaryFile().

• TemporaryDirectory(suffix=None, prefix=None, dir=None) This function creates a temporary directory. On completion of the context or destruction of the temporary directory object the newly created temporary directory and all its contents are removed from the filesystem.

The lower level functions include:

• mkstemp() Creates a temporary file that is only readable or writable by the user who created it.

• mkdtemp() Creates a temporary directory. The directory is readable, writable, and searchable only by the creating user ID.

• gettempdir() Return the name of the directory used for temporary files. This defines the default value for the default temporary directory to be used with the other functions in this module.

**import** tempfile

print("Get temporary directory",tempfile.gettempdir())

temp **=** tempfile.TemporaryFile('w+') *#under temp folder some temp file will be created , can use with contect manager*

print("The name of the temp file",temp.name)

print("The mode of the temp file",temp.mode)

​

print("Write some contents into temp files",temp.write("Hello")) *#returns how many characters written inside the file*

temp.seek(0)

print("Read the contents",temp.readline())

​

Get temporary directory C:\Users\asha.t\AppData\Local\Temp

The name of the temp file C:\Users\asha.t\AppData\Local\Temp\tmp37oq1vx6

The mode of the temp file w+

Write some contents into temp files 5

Read the contents Hello

*#Zipping and unzipping files*

**from** zipfile **import** **\***

f **=** ZipFile('test.zip','w',ZIP\_DEFLATED)

​

f.write('File1.txt')

f.write('File2.txt')

f.write('File3.txt')

​

f.close()

​

​

z**=** ZipFile('test.zip','r')

z.extractall('c:\\Users\\asha.t')

​

*#while creating the zipfile add the contents also*

**import** zipfile

​

text **=** "These are the contents of the zip file"

​

**with** zipfile.ZipFile('newtest.zip','w',ZIP\_DEFLATED) **as** zfile:

zfile.writestr('text1.txt',text)

zfile.writestr('text2.txt',text)

zfile.writestr('text3.txt',text)

​

​

*#To read the zipfile*

*#open the zip file to read*

**import** zipfile

​

**with** zipfile.ZipFile("newtest.zip",'r',ZIP\_DEFLATED) **as** zfile:

*#get the list of the compressed files*

file\_lst **=** zfile.namelist()

print(file\_lst)

**for** arg **in** file\_lst:

*#show the filename*

print(arg)

*#get the file infor by passing each file*

info **=** zfile.getinfo(arg)

print("Orginal file size:",info.file\_size)

print("Compressed file size:",info.compress\_size)

['text1.txt', 'text2.txt', 'text3.txt']

text1.txt

Orginal file size: 38

Compressed file size: 36

text2.txt

Orginal file size: 38

Compressed file size: 36

text3.txt

Orginal file size: 38

Compressed file size: 36

*#To read the contents from the command line use sys.argv*

**import** sys

**for** arg **in** sys.argv:

print(arg)

**for** arg **in** sys.argv[2]: *#can retrive indiviual values like file names*

print(arg, end**=**'')

c:\users\asha.t\appdata\local\programs\python\python37-32\lib\site-packages\ipykernel\_launcher.py

-f

C:\Users\asha.t\AppData\Roaming\jupyter\runtime\kernel-89cdf033-062b-4fdd-a39a-982491e4872b.json

C:\Users\asha.t\AppData\Roaming\jupyter\runtime\kernel-89cdf033-062b-4fdd-a39a-982491e4872b.json

**Pipes**

Most operating systems provide a command-line interface, also known as a shell. Shells usually provide commands to navigate the file system and launch applications. For example, in Unix, you can change directories with cd, display the contents of a directory with ls, and launch a web browser by typing (for example) firefox.

Any program that you can launch from the shell can also be launched from Python using a pipe. A pipe is an object that represents a running process.

For example, the Unix command1 ls -l normally displays the contents of the current directory (in long format). You can launch ls with os.popen:

**import** os

cmd **=** 'dir \*.txt'

*# cmd = 'ls'*

fp **=** os.popen(cmd) *#opening command line using pipe mechanism and execute shell command*

*# The argument is a string that contains a shell command.*

*# The return value is a file pointer that behaves just like an open file.*

*# You can read the output from the ls process one line at a time with readline or get the whole thing at once with read:*

**for** x **in** fp:

res **=** fp.readline()

print(res)

*#When you are done, you close the pipe like a file:*

stat **=** fp.close()

print(stat)

Volume Serial Number is 26EA-BC3B

Directory of C:\Users\asha.t\AppData\Local\Programs\Python\Python37-32\Python - Training\11 - FileSystem

04/01/2020 01:31 PM 0 File1.txt

04/01/2020 01:31 PM 0 File3.txt

04/01/2020 12:58 PM 19 samplenew.txt

08/10/2020 04:09 PM 8 yourfile.txt

0 Dir(s) 406,009,315,328 bytes free

None

​