Files & Directories in python

Files

> Types of files

> Handling file

> with statement

> pickle & unpickle in python

> seek() & tell()

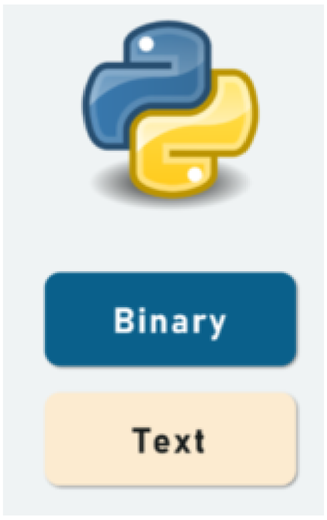
> isfile()

Directories

> Working with directories

As files are non-volatile in nature, the data will be stored permanently in a secondary device like Hard Disk and using python we will handle these files in our applications.

Types of file in python



1. Text file

- Text file store the data in the form of characters.
- Text file are used to store characters or strings.
- Usually we can use text files to store character data

- eg: python.txt

2. Binary file

- Binary file store entire data in the form of bytes.
- Binary file can be used to store text, image, audio and video.
- Usually we can use binary files to store binary data like images,video files, audio files etc

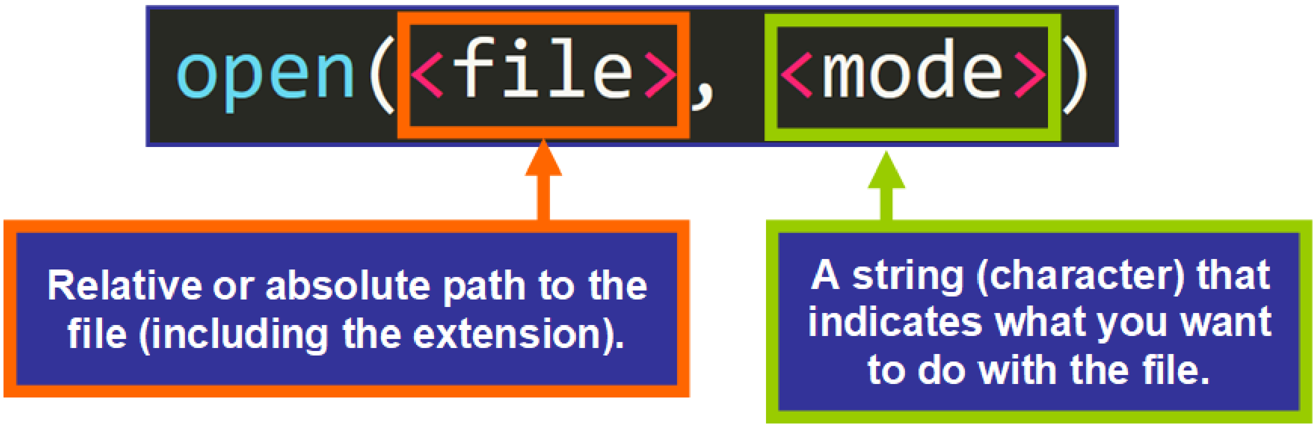
Handling files in python

In Python, a file operation takes place in the following order:

1. Open a file
2. Perform operation (read/write)
3. Close a file

Opening a file

A built-in function that opens a file and allows your program to use it and work with it.


open(<file>, <mode>)
Relative or absolute path to the file (including the extension).
A string (character) that indicates what you want to do with the file.

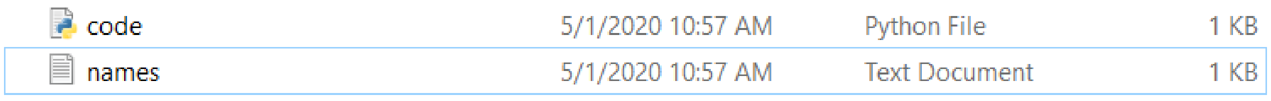
First Parameter:

We usually use a relative path, which indicates where the file is located relative to the location of the script (Python file) that is calling the open() function.

For example, the path in this function call:

open("names.txt") # The relative path is "names.txt"

Only contains the name of the file. This can be used when the file that you are trying to open is in the same directory or folder as the Python script, like this:



code	5/1/2020 10:57 AM	Python File	1 KB
names	5/1/2020 10:57 AM	Text Document	1 KB

Second Parameter: Mode The second parameter of the open() function is the mode, a string with one character. That single character basically tells Python what you are planning to do with the file in your program.

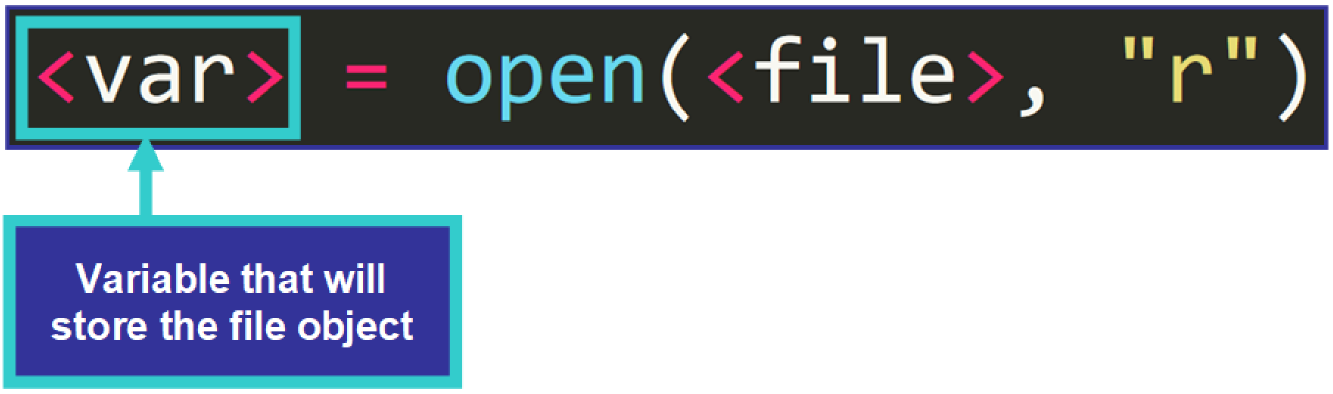
Modes to open the file are as follows:

Character	Function
r	Open file for reading only. Starts reading from beginning of file. This default mode.
rb	Open a file for reading only in binary format. Starts reading from beginning of file.
r+	Open file for reading and writing. File pointer placed at beginning of the file.
w	Open file for writing only. File pointer placed at beginning of the file. Overwrites existing file and creates a new one if it does not exists.
wb	Same as **w** but opens in binary mode.
w+	Same as **w** but also alows to read from file.
wb+	Same as **wb** but also alows to read from file.
a	Open a file for appending. Starts writing at the end of file. Creates a new file if file does not exist.
ab	Same as **a** but in binary format. Creates a new file if file does not exist.
a+	Same a **a** but also open for reading.
ab+	Same a **ab** but also open for reading.

Perform operation

1. Reading a file

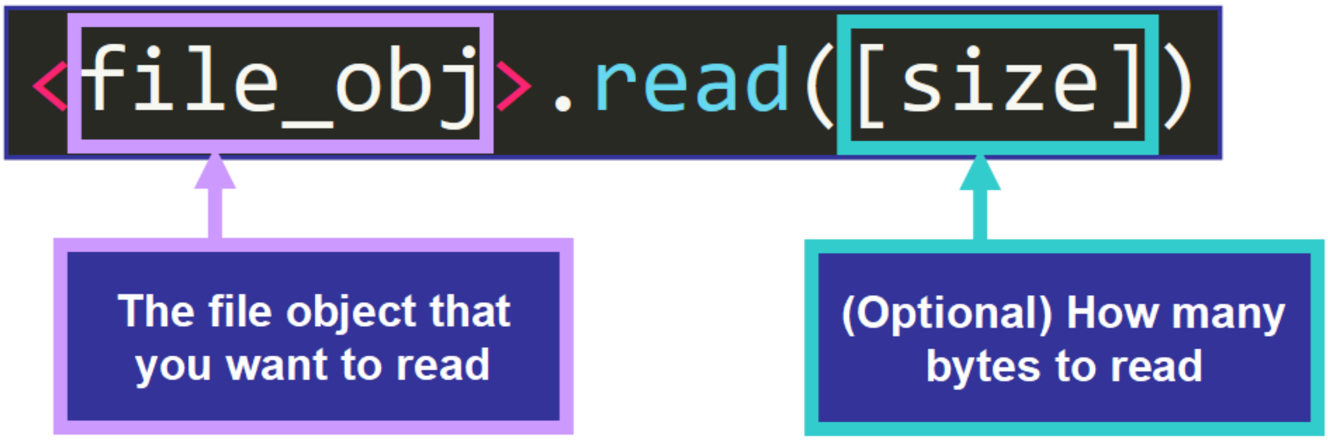
Basic syntax of opening a file is as follows:


```
<var> = open(<file>, "r")
```

open() is going to return file object.

File object is an object that lets us work and interact with existing files in our Python program.

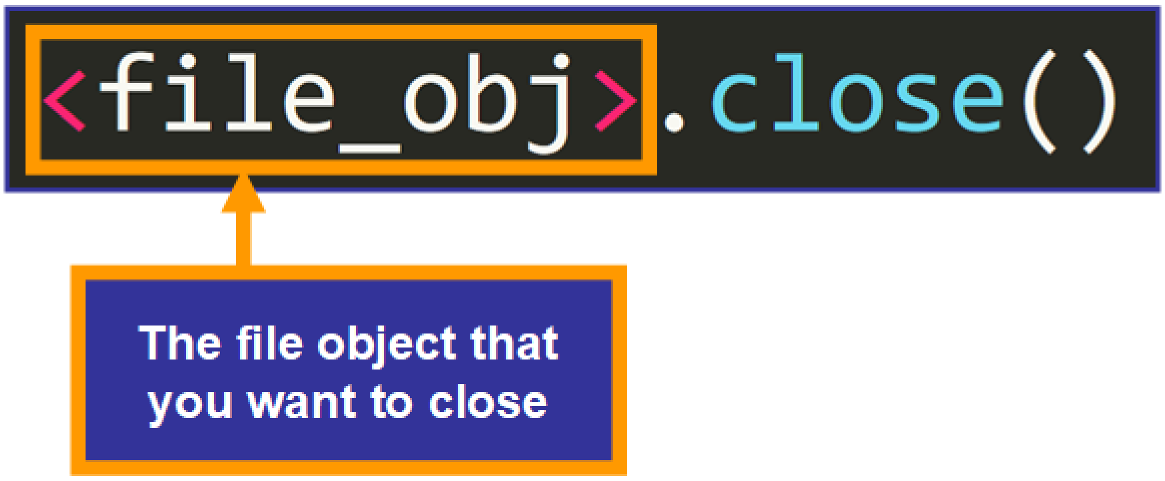
read() It returns the entire content of the file as a string.



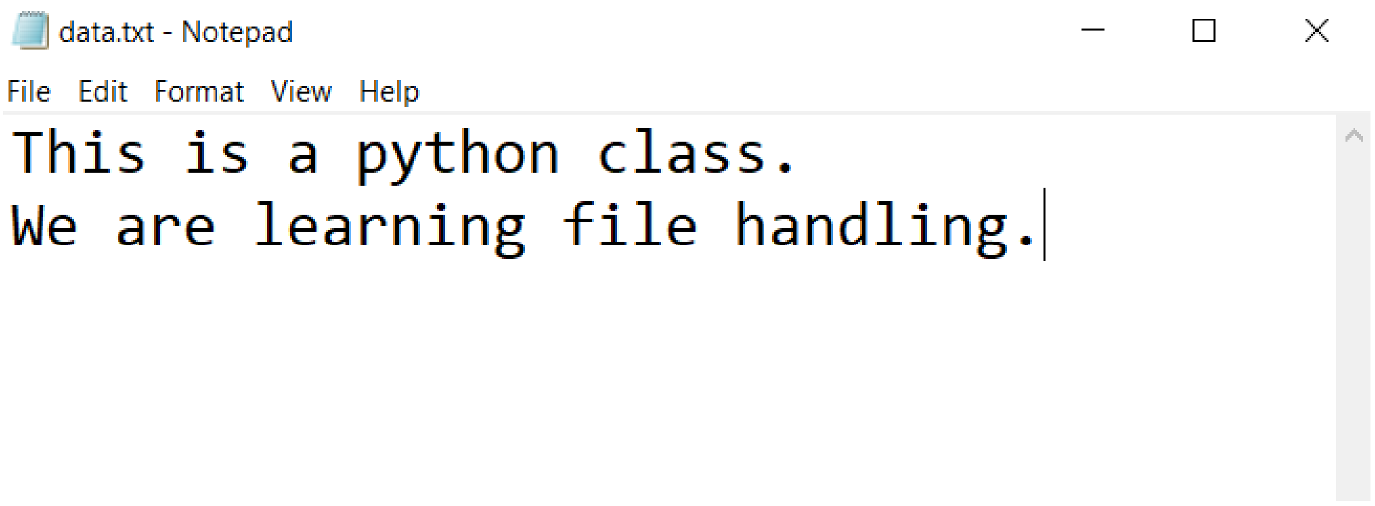
<file_obj>.read([size])
The file object that you want to read
(Optional) How many bytes to read

close()

You need to close a file after the task has been completed to free the resources associated to the file. To do this, you need to call the close() method, like this:



Let's consider data.txt file:



```
This is a python class.
We are learning file handling.
```

```
f=open("data.txt","r")
print(f.read())
print(f.mode)   #attribute of file object
print(f.name)   #attribute of file object
f.close()
```

```
This is a python class.
We are learning file handling.
r
data.txt
```

2. Writing a file

We need write() to modify the file.

There are two ways of doing it which are append and write.

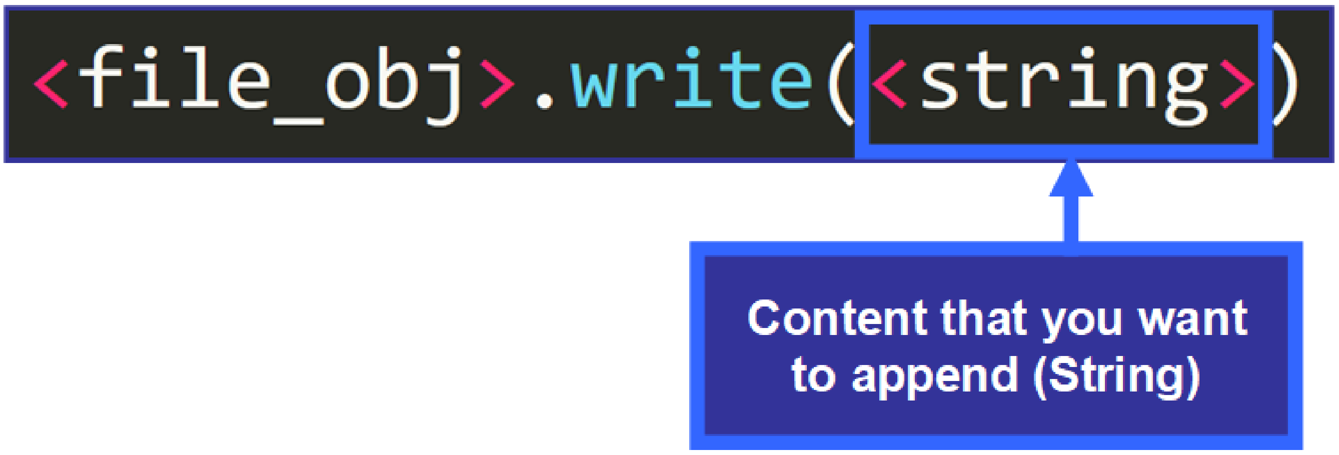
Append

"Appending" means adding something to the end of another thing. The "a" mode allows you to open a file to append some content to it.

Suppose we have "data.txt" file, and we want to append something on it.

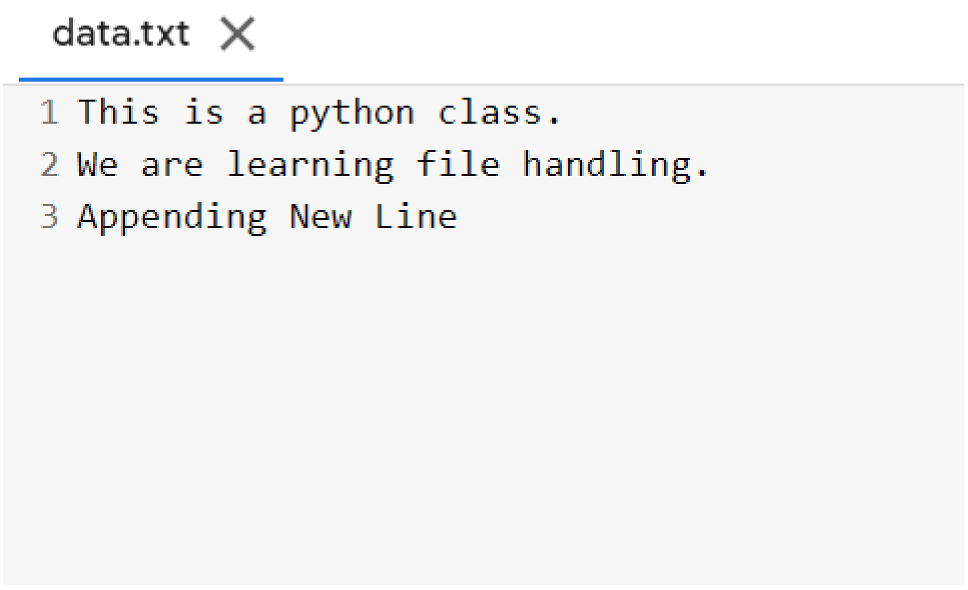
We can open the file using the "a" mode (append) and then, call the write() method, passing the content that we want to append as argument.

Syntax of write():


<file_obj>.write(<string>)
Content that you want to append (String)

```
f = open("data.txt", "a")
f.write("\nAppending New Line")
f.close()
```

After appending:


data.txt

```
This is a python class.
We are learning file handling.
Appending New Line
```

```
#checking whether new line appended
f=open("data.txt","r")
print(f.read())
f.close()
```

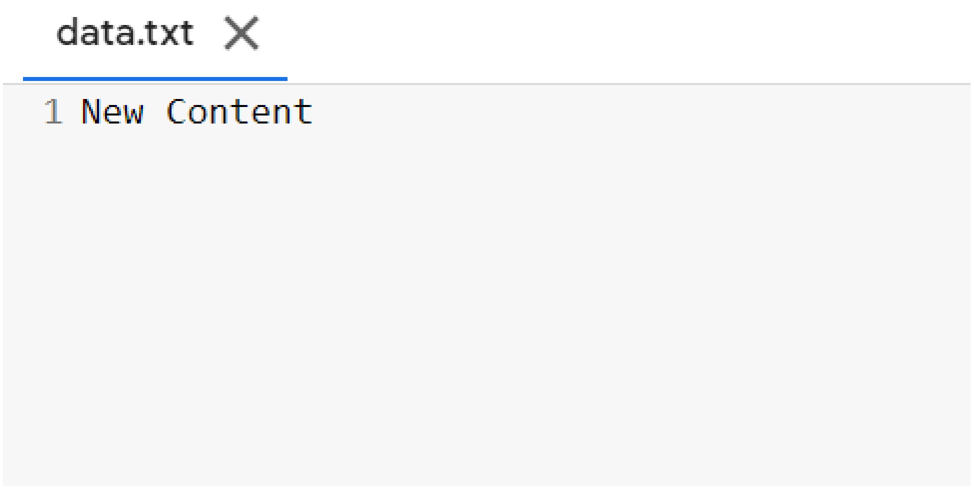
```
This is a python class.
We are learning file handling.
Appending New Line
```

Write

Sometimes, you may want to delete the content of a file and replace it entirely with new content. You can do this with the write() method if you open the file with the "w" mode.

```
f = open("data.txt", "w")
f.write("New Content")
f.close()
```

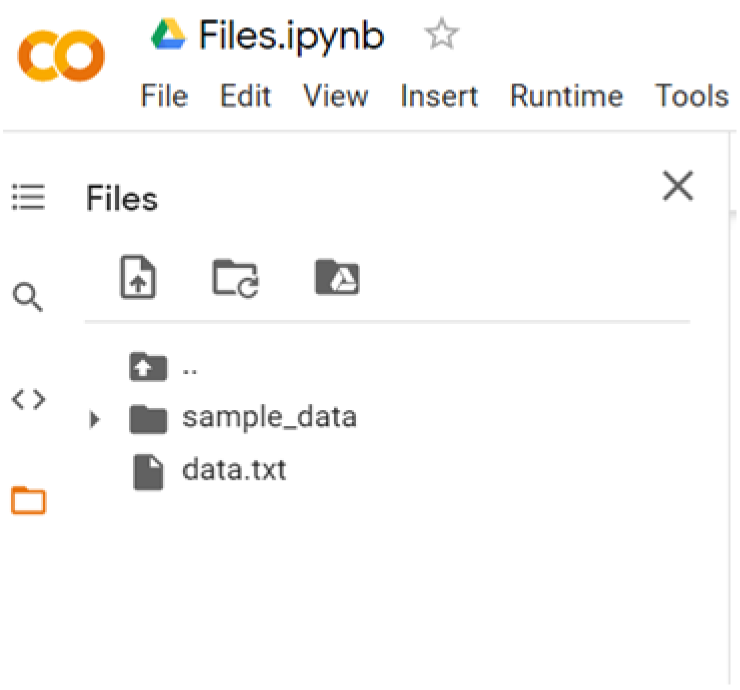
After Writing:


data.txt

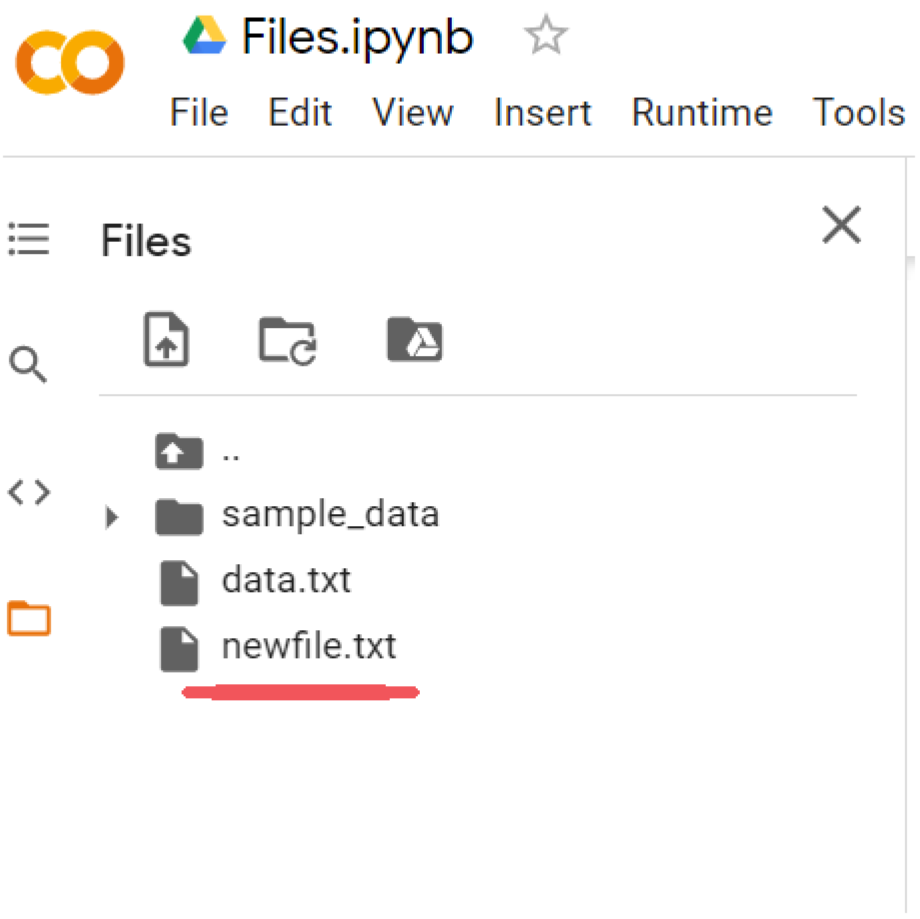
```
New Content
```

```
f = open("newfile.txt", "w")
f.write("New file created")
f.close()
```

Files before above code cell executed:



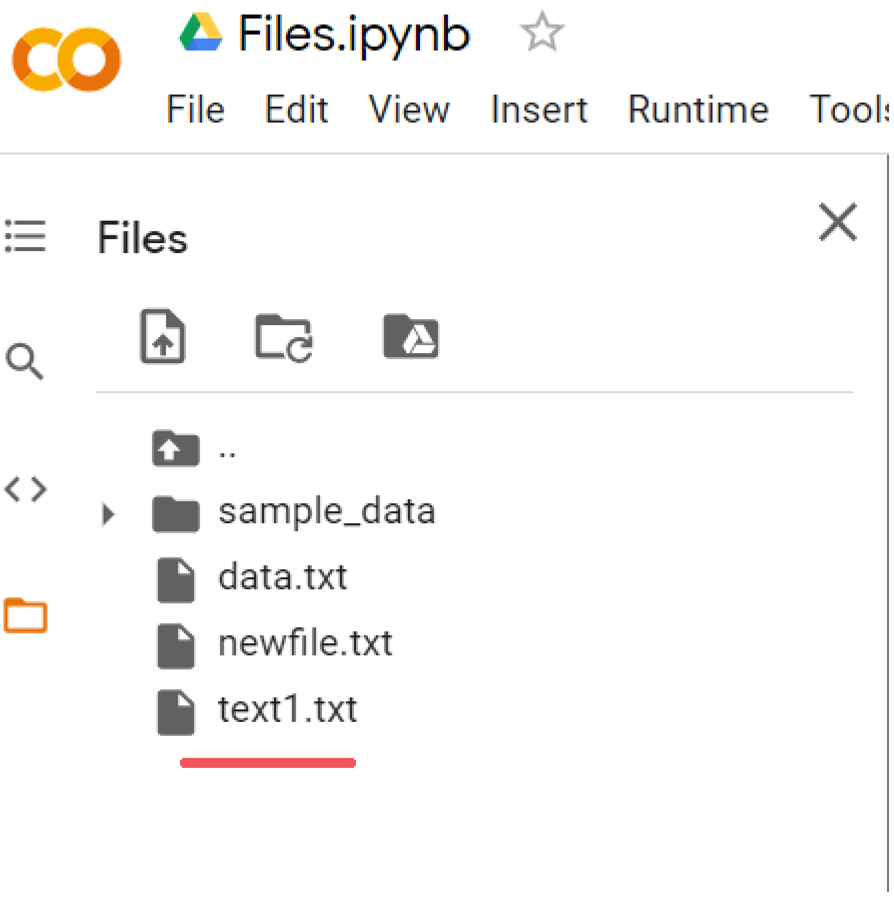
Files after above code cell executed:



Write a program to store group of strings into a text file.

```
f=open("text1.txt","w")
print("Enter text (@ indiactes end of strings)")
while 1:
  str=input()
  if str=='@':
    break
  f.write(str+"\n")
f.close()
```

```
Enter text (@ indiactes end of strings)
This is line1
This is line2
This is line3
@
```



With statement

- It is used to open a file.
- There is no need to close a file explicitly

Syntax:

```
with open("filename","openmode") as fileobject:
```

```
with open("text1.txt","r") as f:
  for line in f:
    print(line)
```

```
This is line1

This is line2

This is line3
```

Pickle & unpickle in python

Pickle in python

It is a process where a Python object hierarchy is converted into a byte stream, so that it can be stored into a file.

This is done using dump() of pickle module.

Also called as serialization.

Syntax:

```
dump(object, file_object)
```

Example: Python program to pickle employee class objects

```
class Employee:
  def __init__(self,id,name,salary):
    self.id = id
    self.name = name
    self.salary = salary
  def display(self):
    print(self.id," ",self.name," ",self.salary)


import pickle

f = open('Employee.dat','wb')
n = int(input("No. of Employee"))

for i in range(n):
  id = int(input("Enter id "))
  name = input("Enter name ")
  salary = float(input("enter salary "))

  e = Employee(id,name,salary)
  pickle.dump(e,f)
f.close()
```

```
No. of Employee3
Enter id 1
Enter name abc
enter salary 10000
Enter id 2
Enter name bcd
enter salary 20000
Enter id 3
Enter name def
enter salary 30000
```

Unpickle in python

It is the inverse of Pickling process where a byte stream is converted into an object hierarchy.

This is done using load() of pickle method.

Also called as deserialization.

Syntax:

```
load(file_object)
```

```
f=open("Employee.dat","rb")
print("Employee details")
while 1:
  try:
    e=pickle.load(f)
    e.display()
  except EOFError:
    print("End of file reached")
    break
f.close()
```

```
Employee details
1   abc   10000.0
2   bcd   20000.0
3   def   30000.0
End of file reached
```

Following program pickle a dictionary object into a binary file.

```
import pickle
f = open("text2.txt","wb")
dct = {"name":"Ravi", "age":23, "Gender":"M","marks":75}
pickle.dump(dct,f)
f.close()
```

To unpickle or deserialize data from binary file back to dictionary, run following program.

```
import pickle
f = open("text2.txt","rb")
d = pickle.load(f)
print (d)
f.close()
```

```
    {'name': 'Ravi', 'age': 23, 'Gender': 'M', 'marks': 75}
```

Seek() & tell()

seek()

It is used to change the position of the file handle to a given specific position.

Syntax:

```
 f.seek(offset, from_where)
```

Offset: Number of postions to move forward

from_where takes three arguments:

0: sets the reference point at the beginning of the file

1: sets the reference point at the current file position

2: sets the reference point at the end of the file

Note:

Default value of from_where is 0.

```
f=open("data.txt","r")
f.seek(3,0)
print(f.read())
```

```
     Content
```

tell()

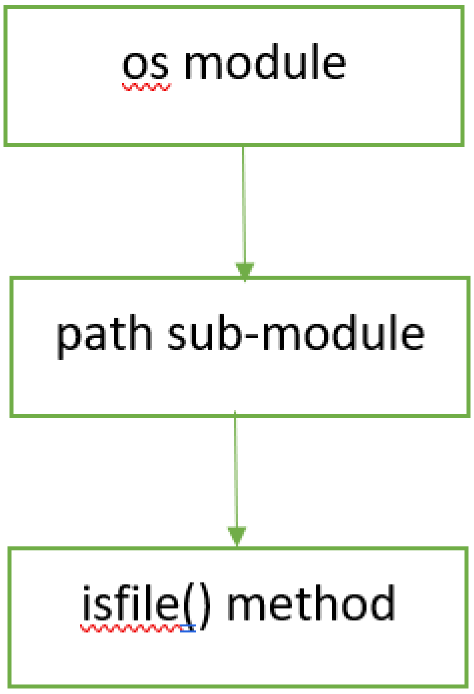
tell() method returns the current file position in a file stream.

```
print(f.tell())
f.seek(5,0)
print(f.tell())
```

```
    11
    5
```

isfile()

It returns true if file exists else false.



```
import os, sys
fname=input("Enter filename: ")
if os.path.isfile(fname):
  f=open(fname,"r")
else:
  print(fname+" does not exist")
  sys.exit()
print("The file contents are: ")
print(f.read())
f.close()
```

Directories

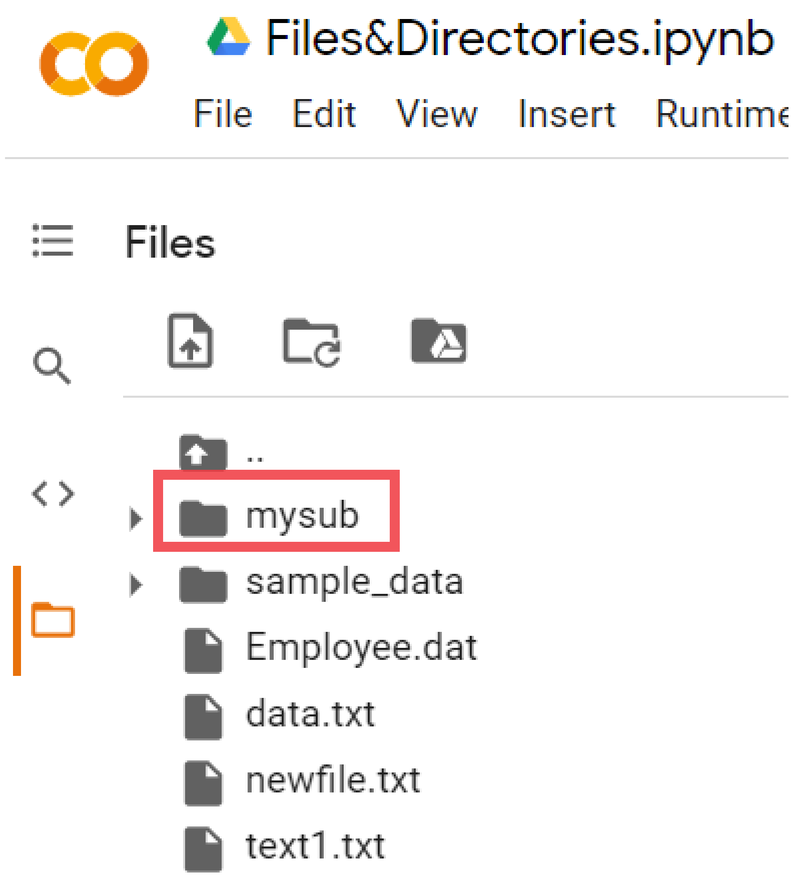
os module is useful to perform simple operations on directories.

```
import os
current=os.getcwd()   #get current working directory
print(current)
```

```
     /content
```

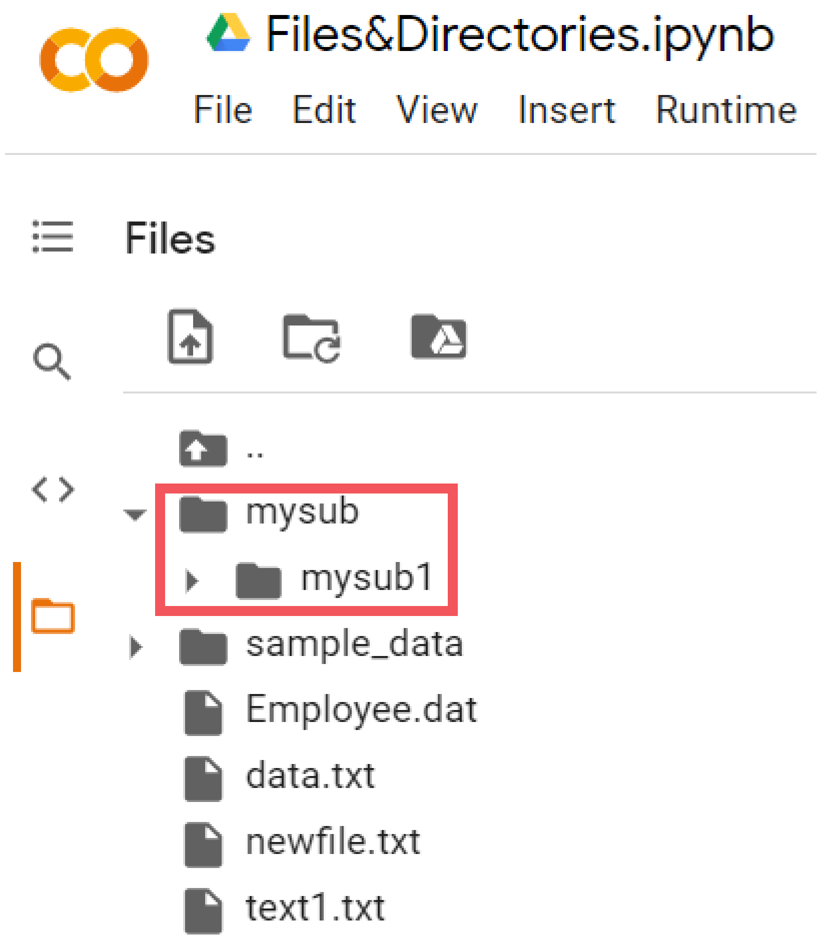
```
os.mkdir("mysub")   #create sub-directory
```

Output:



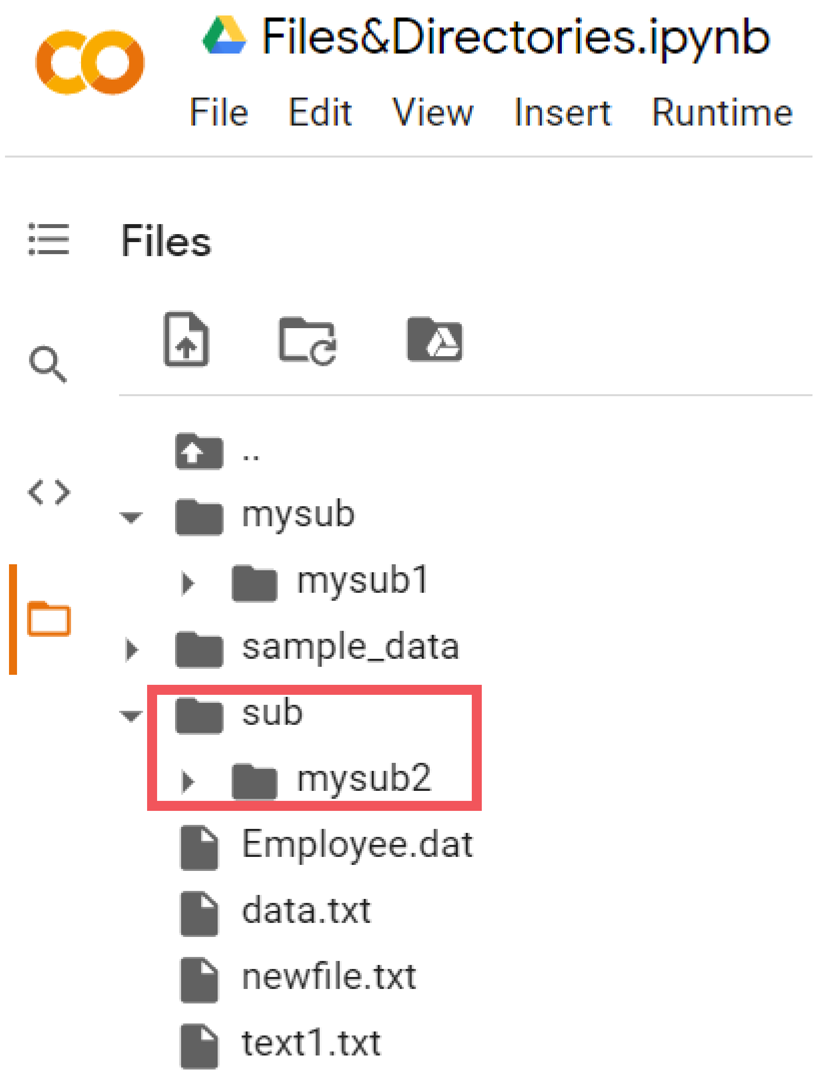
```
os.mkdir("mysub/mysub1")    #create sub-sub directory only if sub-directory exist
```

Output:



```
os.makedirs("sub/mysub2")     #if sub does not exist it will first create sub
```

Output:



```
..
mysub
    mysub1
sample_data
sub
    mysub2
Employee.dat
data.txt
newfile.txt
text1.txt
```

```
os.chdir("mysub/mysub1")
print(os.getcwd())
```

```
/content/mysub/mysub1
```

```
os.chdir("..")    #go back to parent directory
print(os.getcwd())
```

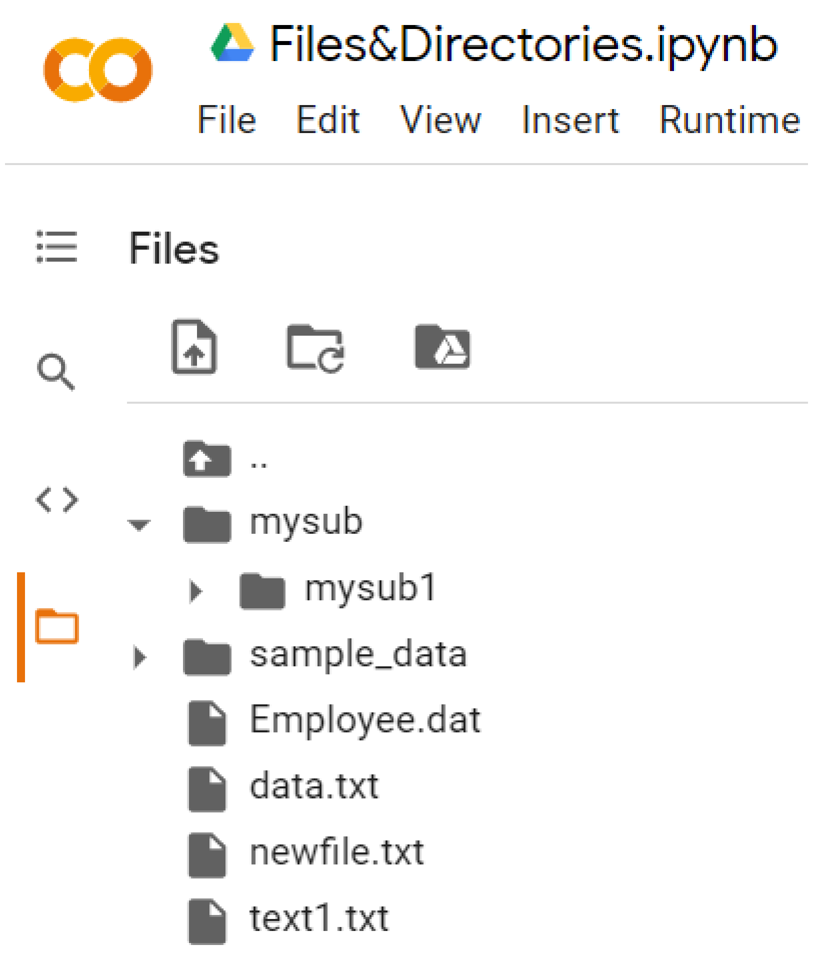
```
/content/mysub
```

```
os.chdir("..")    #go back to parent directory
print(os.getcwd())
```

```
/content
```

```
os.removedirs("sub/mysub2")   #remove mysub2 then sub
```

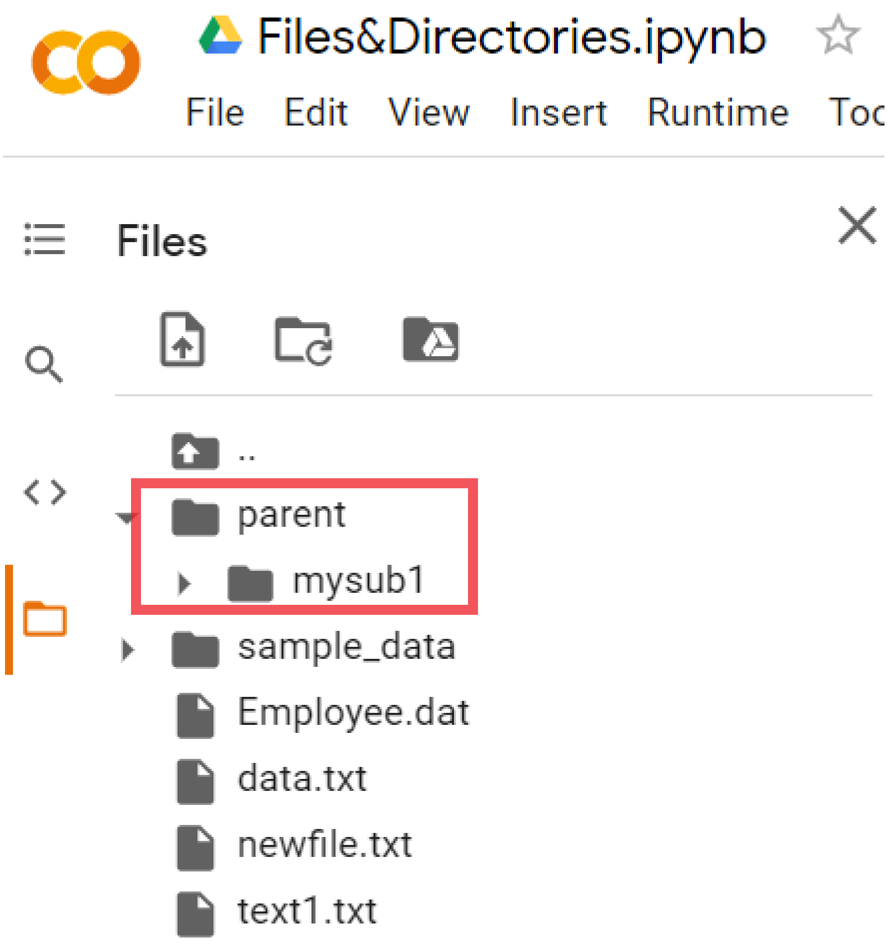
Output:



```
..
mysub
    mysub1
sample_data
Employee.dat
data.txt
newfile.txt
text1.txt
```

```
os.rename("mysub","parent")   #rename directory
```

Output:



walk()

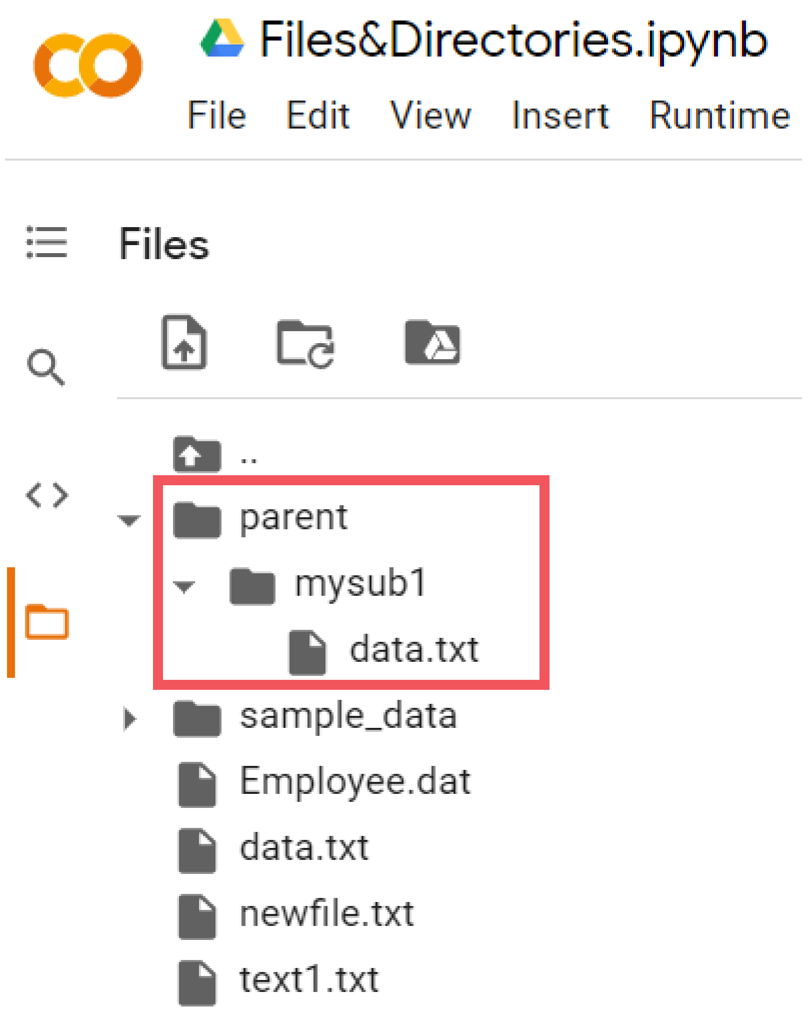
Python method walk() generates the file names in a directory tree by walking the tree either top-down or bottom-up.

Syntax:

```
walk(directory, topdown=1|0)

where, topdown=1 (by default)
```

Current directories scenario:



```
import os
for a,b,c in os.walk('parent'):
  print("Current path: ",a)
  print("Directories: ",b)
  print("Files: ",c)
```

```
    Current path:  parent
    Directories:  ['mysub1']
    Files:  []
    Current path:  parent/mysub1
    Directories:  []
    Files:  ['data.txt']
```

Packages

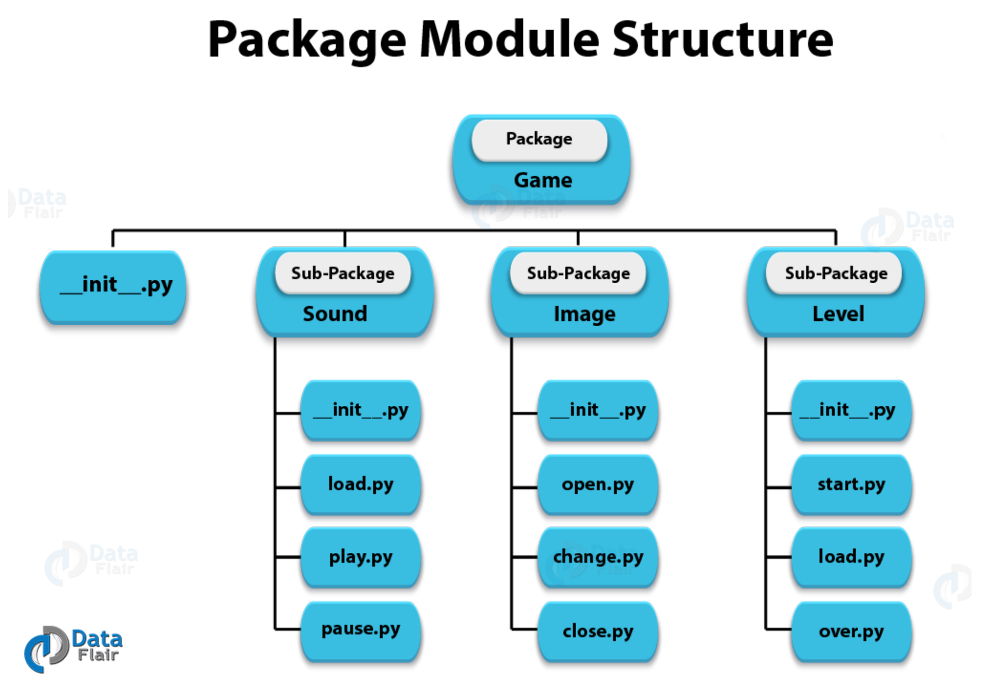
> A package is a **hierarchical file directory structure** that defines a single Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

Why? As application program grows larger in size with a lot of modules, develoer places similar modules in one package and different modules in different packages. This makes a project (program) **easy to manage** and conceptually clear.

Similarly, as a directory can contain subdirectories and files, a **Python package** can have **sub-packages and modules.**

A directory must contain a file named **_ _init _ _.py** in order for Python to consider it as a package. This file can be left empty but generally initialization code is placed for that package in this file.

Example



Importing module from a package

We can import modules from packages using the **dot (.) operator.**

For example, if we want to import the start module in the above example, it can be done as follows:

```
import Game.Level.start
```

Now, if this module contains a function named select_difficulty(), we must use the full name to reference it.

```
Game.Level.start.select_difficulty(2)
```

> If this construct seems lengthy, we can import the module without the package prefix as follows:
>
> > ```
> > from Game.Level import start
> > ```
>
> We can now call the function simply as follows:

```
import os
print(os.getcwd()) #Current directory should be content
```

```
     /content
```

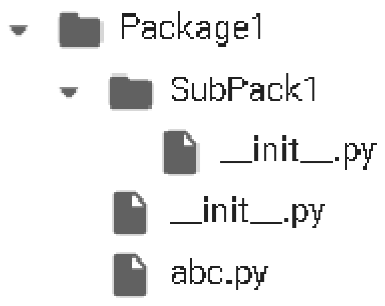
```
os.makedirs("Package1/SubPack1") #create directory strucutre
```

```
os.chdir('Package1') #Moving to Package directory to  include __init__.py
```

```
f = open("__init__.py","w")
f.close()
f = open('abc.py',"w")   #Creating abc.y module in Package1
f.close()
```

```
os.chdir('SubPack1')   #Making Sub directory SubPack1 as Sub Package
f = open("__init__.py","w")
f.close()
```

```
import Package1
import Package1.abc
import Package1.SubPack1
```



✓ 0s completed at 15:22