Unit - 3

Python Interaction with SQLite

Module

Python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module. We may have a runnable code inside the python module.

Modules in Python provides us the flexibility to organize the code in a logical way.

To use the functionality of one module into another, we must have to import the specific module.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example

Save this code in a file named mymodule.py

```
def  display(name):

print("my name, " + name)
```

Here, we need to include this module into our main module to call the method display () defined in the module named file.

Loading the module

Python provides two types of statements as defined below.

1. The import statement
2. The from-import statement

The import statement

The import statement is used to import all the functionality of one module into another. Here, we must notice that we can use the functionality of any python source file by importing that file as the module into another python source file.

We can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times, it has been imported into our file.

Syntax:

import module1,module2, module3 .... module n

Hence, if we need to call the function display () defined in the file mymodule.py, we have to import that file as a module into our module as shown in the example below.

Example:

```
import file;
name = input("Enter the name: ")
file.display(name)
```

Output:

```
Enter the name: mayank
my name , mayank
```

The from-import statement

Python provides the flexibility to import only the specific attributes of a module. This can be done by using from - import statement.

Syntax :

```
from< module-name> import <name 1>, <name 2>..,<name n>
```

Example:

```
from file import display;
name = input("Enter the name: ")
file.display(name)
```

Output:

```
Enter the name: mayank
my name , mayank
```

We can also import all the attributes from a module by using *.

Consider the following syntax.

```
from<module> import *
```

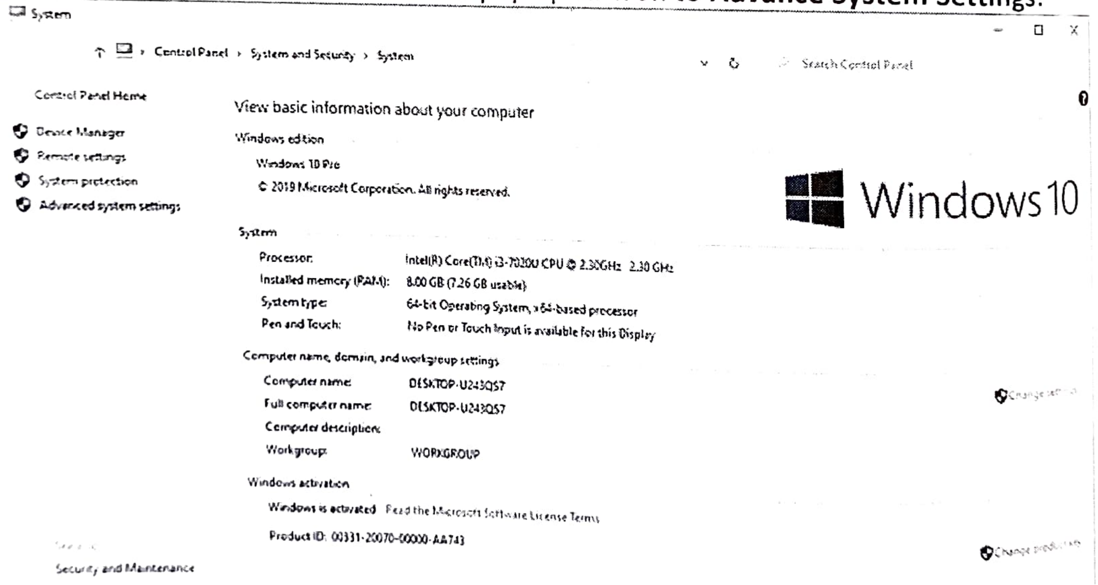
PYTHONPATH

PYTHONPATH is an environment variable which you can set to add additional directories where python will look for modules and packages. For most installations, you should not set these variables since they are not needed for Python to run. Python knows where to find its standard library.

The only reason to set PYTHONPATH is to maintain directories of custom Python libraries that you do not want to install in the global default location (i.e., the site-packages directory).

Setting PYTHONPATH on a **windows** machine follow the below steps:

Step 1: Open your **This PC (or My Computer)** and write click and click on **properties.**
Step 2: After the properties window pop up click on to **Advance System Settings:**



View basic information about your computer

Windows edition

Windows 10 Pro

© 2019 Microsoft Corporation. All rights reserved.

System

Processor: Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHz
Installed memory (RAM): 8.00 GB (7.26 GB usable)
System type: 64-bit Operating System, x64-based processor
Pen and Touch: No Pen or Touch Input is available for this Display

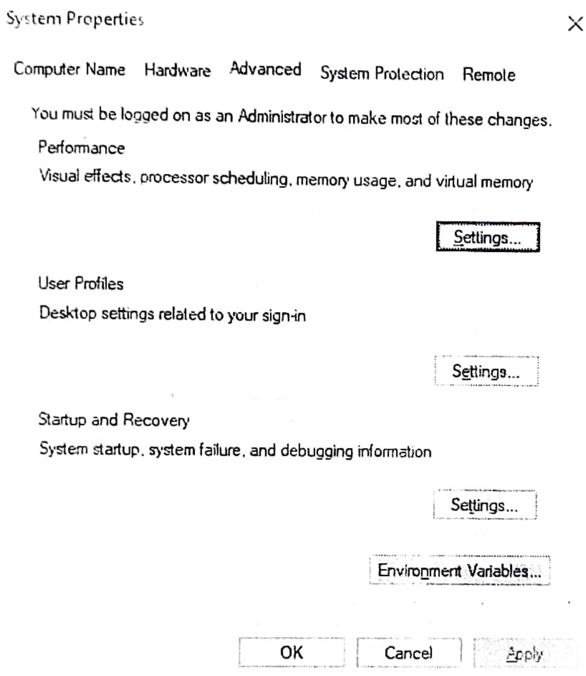
Computer name, domain, and workgroup settings

Computer name: DESKTOP-U243QS7
Full computer name: DESKTOP-U243QS7
Computer description:
Workgroup: WORKGROUP

Windows activation

Windows is activated Read the Microsoft Software License Terms
Product ID: 00331-20070-00000-AA743

Step 3: Now click on the environment variable button in the new popped up window as shown below:



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

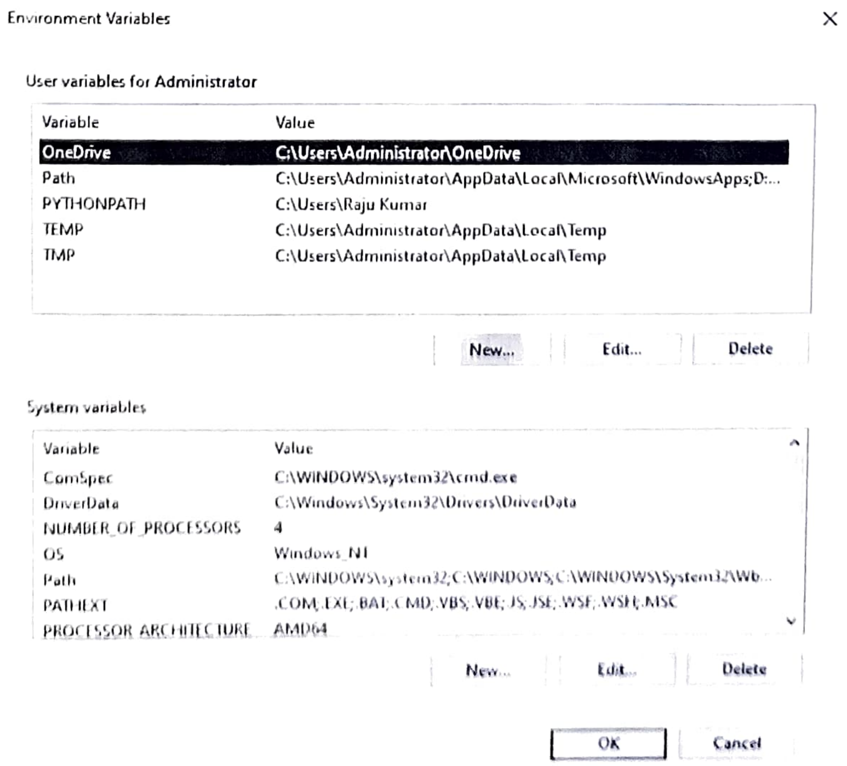
User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

Step 4: Now in the new Environment Variable dialog box click on New as shown below:



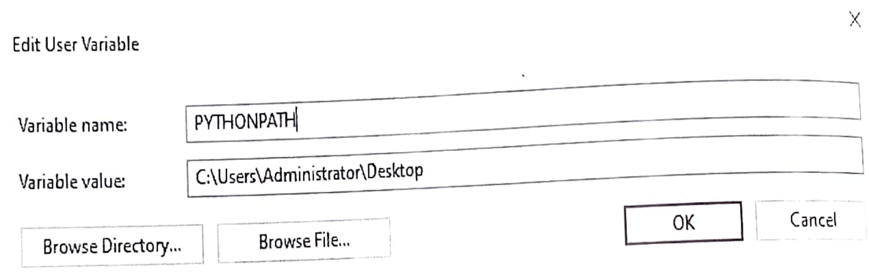
User variables for Administrator

Variable	Value
OneDrive	C:\Users\Administrator\OneDrive
Path	C:\Users\Administrator\AppData\Local\Microsoft\WindowsApps;D:...
PYTHONPATH	C:\Users\Raju Kumar
TEMP	C:\Users\Administrator\AppData\Local\Temp
TMP	C:\Users\Administrator\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wb...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

Step 5: Now in the variable dialog box add the name of the variable as PYTHONPATH and in value add the location to the module directory that you want python to check every time as shown below:


Edit User Variable

Variable name: PYTHONPATH

Variable value: C:\Users\Administrator\Desktop

Your PYTHONPATH is set.

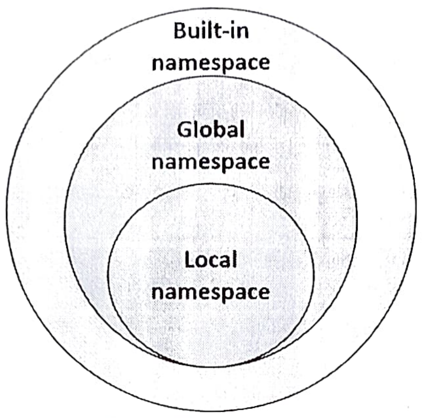
Concepts of Namespace and Scope

In python we deal with variables, functions, libraries and modules etc. There is a chance the name of the variable you are going to use is already existing as name of another variable or as the name of another function or another method. In such scenario, we need to learn about how all these names are managed by a python program. This is the concept of namespace.

Its Name (which means name, a unique identifier) + Space (which talks something related to scope). Here, a name might be of any Python method or variable and space depends upon the location from where is trying to access a variable or a method.

Following are the three categories of namespace

- **Local Namespace:** All the names of the functions and variables declared by a program are held in this namespace. This namespace exists as long as the program runs. < /p>
- **Global Namespace:** This namespace holds all the names of functions and other variables that are included in the modules being used in the python program. It encompasses all the names that are part of the Local namespace.
- **Built-in Namespace:** This is the highest level of namespace which is available with default names available as part of the python interpreter that is loaded as the programing environment. It encompasses Global Namespace which in turn encompasses the local namespace.



Type of Namespaces

Scope of Namespace

The namespace has a lifetime when it is available. That is also called the scope. Also the scope will depend on the coding region where the variable or object is located. You can see in the below program how the variables declared in an inner loop are available to the outer loop but not vice-versa. Also please note how the name of the outer function also becomes part of a global variable.

Example

```
prog_var = 'Hello'
defouter_func():
outer_var = 'x'
definner_func():
inner_var = 'y'
print(dir(), ' Local Variable in Inner function')

inner_func()
print(dir(), 'Local variables in outer function')

outer_func()
print(dir(), 'Global variables ')
```

Output

```
        ['inner_var'] Local Variable in Inner function
        ['inner_func', 'outer_var'] Local variables in outer function
```

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'outer_func', 'prog_var']
Global variables
```

Packages in Python

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.

Consider a file *Display.py* available in *pack1* directory. This file has following line of source code –

```
def disp():

        print "student doing bca"
```

Similar way, we have another two files having different functions with the same name as above –

- *pack1/a1.py* file having function a1()
- *pack1/b1.py* file having function b1()

Now, create one more file __init__.py in *pack1* directory –

- pack1/__init__.py

To make all of your functions available when you've imported *pack1* , you need to put explicit import statements in __init__.py as follows –

```
# Now import your pack1 Package.
        import pack1


        pack1.disp()
        pack1.a1()
        pack1.b1()
```

Output:

```
student doing bca
      this is from a1
      this is from b1
```

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

Importing sqlite3 module

SQLite3 can be integrated with Python using sqlite3 module, which was written by Gerhard Haring. It provides an SQL interface compliant with the DB-API 2.0.
You do not need to install this module separately because it is shipped by default along with Python version 2.5.x onwards.
To use sqlite3 module, you must first create a connection object that represents the database and then optionally you can create a cursor object, which will help you in executing all the SQL statements.
Importing sqlite3 :

```
import sqlite3
```

connect()

This routine opens a connection to the SQLite database file. You can use ":memory:" to open a database connection to a database that resides in RAM instead of on disk. If database is opened successfully, it returns a connection object.
When a database is accessed by multiple connections, and one of the processes modifies the database, the SQLite database is locked until that transaction is committed. The timeout parameter specifies how long the connection should wait for the lock to go away until raising an exception. The default for the timeout parameter is 5.0 (five seconds).

If the given database name does not exist then this call will create the database. You can specify filename with the required path as well if you want to create a database anywhere else except in the current directory.
Syntax:

```
sqlite3.connect(database [,timeout ,other optional arguments])
```

Example:

```
import sqlite3
conn = sqlite3.connect(student.db')
print "Opened database successfully";
```

Now connection is created with the student database.

Execute()

This routine executes an SQL statement. The SQL statement may be parameterized (i. e. placeholders instead of SQL literals). The sqlite3 module supports two kinds of placeholders: question marks and named placeholders (named style).

Syntax:

```
cursor.execute(sql [, optional parameters])
```

For example – cursor.execute("insert into student values (?, ?)", (who, age))

fetchone()

fetchone() method returns a single record or None if no more rows are available.
To fetch a single row from a result set we can use cursor.fetchone(). This method returns a single tuple.
It can return a none if no rows are available in the resultset. cursor.fetchone() increments the cursor position by one and return the next row.

Syntax:

```
cursor.fetchone()
```

Example:

```
import sqlite3

try:
connection = sqlite3.connect(student.db')
cursor = connection.cursor()
print("Connected to database")

query = "SELECT * from student"
cursor.execute(query)
print("Fetching single row")
record = cursor.fetchone()
print(record)

print("Fetching next row")
record = cursor.fetchone()
print(record)

cursor.close()

except sqlite3.Error as error:
print("Failed to read data from table", error)
finally:
if connection:
connection.close()
print("The Sqlite connection is closed")
```

Output:

```
Connected to database
Fetching single row
(1, 'amit', 'BCA',88)

Fetching next row
(2, 'rani', 'MCA',98)
The Sqlite connection is closed
```

fetchall()

fetchall() method fetches all (remaining) rows of a query result, returning a list. An empty list is returned when no rows are available. Get resultSet (all rows) from the cursor object using a cursor.fetchall().

Example:

```
import sqlite3

try:
connection = sqlite3.connect('student.db')
cursor = connection.cursor()
print("Connected to SQLite")

query = "SELECT * from student"
cursor.execute(query)
records = cursor.fetchall()
print("Total rows are:  ", len(records))
print("Printing each row")
for row in records:
print("Id: ", row[0])
print("Name: ", row[1])
print("course: ", row[2])
print("marks: ", row[3])
print("\n")

cursor.close()

except sqlite3.Error as error:
print("Failed to read data from table", error)
```

```
finally:
if connection:
connection.close()
print("The Sqlite connection is closed")
```

Output:

```
Connected to database
Total rows are:  3

Printing each row
Id:  1
Name:  raj
course:  BCA
marks:  88

Id:  2
Name:  rani
course:  MCA
marks:  99

Id:  3
Name:  deep
course:  BCA
marks:  50

The Sqlite connection is closed
```

Create a table using python

Create a table "STUDENT" within the database "demo_db".

Create a python file "createtable.py", having the following code:

```
import sqlite3
conn = sqlite3.connect('demo_db ')
print "Opened database successfully";

conn.execute("CREATE TABLE STUDENT
        (ID  INT PRIMARY KEY    NOT NULL,
        NAME  TEXT    NOT NULL,
        AGE   INT     NOT NULL,
        ADDRESS  CHAR(50)
        );''')

    print ("Table created successfully")

    conn.close()
```

A table "STUDENT " is created in the "demo_db" database.

Insert Records

Insert some records in " STUDENT " table.

Create a python file "insert.py", having the following code:

```
import sqlite3

conn = sqlite3.connect('javatpoint.db')

print ("Opened database successfully")

conn.execute("INSERT INTO STUDENT (ID,NAME,AGE,ADDRESS) VALUES (1, 'aditya',
27, 'surat' )")

conn.commit()

print ("Records inserted successfully")

conn.close()
```

output:

```
    Opened database successfully

    Records inserted successfully
```

Select Records

To fetch and display your records from the table "STUDENT" by using SELECT statement.

Create a python file "select.py", having the following code:

```
import sqlite3

conn = sqlite3.connect('javatpoint.db')

data = conn.execute("select * from Employees")

for row in data:
   print ("ID = ", row[0]  )
   print ("NAME = ", row[1]  )
   print ("AGE = ", row[2]  )
   print ("ADDRESS = ", row[3] , "\n")

conn.close();
```

Output:

```
ID =1
NAME =aditya
AGE=27
ADDRESS =surat

ID =2
NAME =raj
AGE=32
ADDRESS =vapi
```

Delete Records

Delete some records in " STUDENT " table.

Create a python file "delete.py", having the following code:

```
import sqlite3

conn = sqlite3.connect('javatpoint.db')

print ("Opened database successfully")

id  = int(input("Enter id to delete:")
conn.execute("DELETE FROM STUDENT WHERE ID = ",id)

conn.commit()

print ("Total number of rows deleted :", conn.total_changes)

conn.close()
```

Output:

```
Opened database successfully
Enter id to delete:1
Total number of rows deleted :1
```

total_changes Returns the total number of database rows that have been modified, inserted, or deleted since the database connection was opened

commit()

commit() method commits the current transaction. If you don't call this method, anything you did since the last call to commit() is not visible from other database connections. If you wonder why you don't see the data you've written to the database, please check you didn't forget to call this method.

Example:

```
import sqlite3
con = sqlite3.connect('demo_db')
cur = con.cursor()

con.execute("""CREATE TABLE student
        (id integer, value text)""")
con.execute("INSERT INTO student VALUES (1, 'mehul')")
con.commit()
result = con.execute("SELECT * from student")
for row in result:
        print (row)
```

Output:

```
(1, 'mehul')
```

Here we create one table in demo_db database and insert one record using execute() method and then commit() method call to commits the current transaction and then data was display from database using select query using execute () method.

Exercise

Short Question

1. Define Module.
2. Explain PYTHONPATH.
3. Define Packages.
4. Difference between fetchone() and fetchall() methods.
5. Explain commit () method

Long Question

1. How to load module in code?
2. Explain Concepts of Namespace and Scope.
3. Explain Packages in Python.
4. Explain connect () and execute() methods of sqlite3 module.
5. Write a code to Insert, update, delete using execute () method in below table Student_tbl(id, name, dob, class, gender, city)