

Unit-3: Python Interaction with SQLite**Que) What is Module?**

Ans: A module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include Runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.

Que) How to create module?

Ans:

Step-1: To create a module just save the code you want in a file with the file First.py

First.py

Step-2: Write the following python code into First.py file.

```
def display(name):  
    print ("My name is :" + name)  
    display("Pooja")
```

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

Step-3: Execute the python code

Example: create addition.py

```
def add(a,b):  
    return(a+b)  
def sub(a,b):  
    return(a-b)  
print(add(10,5))  
print(sub(5,10))
```

Loading the Module:

Python provides two types of statements as defined below.

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

1. The import Statement.

We can use any Python source file as a module by executing an import statement in some other Python source file.

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches for importing a module.

Syntax: import module1[, module2[,... moduleN]

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

Example: mymodule.py

```
def display(name):
    print("My name is : " + name)
display("Pooja")
```

Now, to import mymodule.py in to importmodule.py

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

Output:

```
Enter the name: Pooja
My name is: Pooja
```

2. The From...import Statement:

Python's *from* statement lets you import specific attributes from a module into the current namespace.

Syntax:

```
from<module_name>import<name 1>,<name2>,,,,,,<name n>
```

Example:

```
fromimport.py
from mymodule import display
Name= input("Enter Name :")
display(name)
```

Output:

```
Enter Name : Raj
My name is : Raj
```

Example:

```
from math import sqrt, factorial
print(sqrt(16))
print(factorial(6))
```

The from...import * Statement

It is also possible to import all names from a module into the current namespace by using the following import statement

Syntax:from <module>import *

Example:

```
from math import sqrt, factorial
print(sqrt(16))
print(factorial(6))
```

Que) What is PYTHONPATH?

Ans: PYTHONPATH is an environment variable that is used to specify the location of Python libraries. It is typically used by developers to ensure that their code can find the required Python libraries when it is run.

Setting PYTHONPATH on windows machine follow the below steps:

1. Open your This PC right click and click on properties.

2. After the properties window pop up click on the Advance System Settings.
3. Now click on the environment variable button.
4. Now in the new environment variable dialog box click on New.
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.

Que) Explain Concepts of Namespace and Scope:**Ans:**

- 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.

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.

• 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.

• 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 programming environment.

Scope of Namespace:

- The namespace has 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.

Example:

```
def outer_function():
    global a
    a=20
def inner_function():
    global a
    a=30
    print('a =' ,a)
inner_function()
    print('a =' ,a)
    a=10
    print('a =' ,a)
outer_function()
    print('a =' ,a)
```

Output:

```
a =10
a =30
a =30
a =30
```

Explanation:

- **Line 1 (a = 10):** You declare a global variable **a** and set it to **10** right before calling `outer_function()`.
- **Line 2 (a = 30):** Inside `outer_function`, **a** is set to **20**. But then `inner_function()` runs, uses global **a**, and overwrites that same global variable to 30. The first print happens here.
- **Line 3 (a = 30):** Control returns to `outer_function`. Since `inner_function` permanently changed the global **a** to 30, printing **a** here outputs 30.
- **Line 4 (a = 30):** `outer_function` finishes. The final print statement looks at the global variable **a**, which remains 30.

Packages in Python:

- We usually organize our files in different folders and subfolders based on some criteria, so that they can be managed easily and efficiently. For example, we keep all our games in a Games folder and we can even subcategorize according to the genre of the game or something like that. The same analogy is followed by the Python Packages.
- Python Packages are a way to organize and structure your Python code into reusable components. Think of it like a folder that contains related Python files (modules) that work together to provide certain functionality.
- Packages help keep your code organized, make it easier to manage and maintain, and allow you to share your code with others. They're like a toolbox where you can store and organize your tools (functions and classes) for easy access and reuse in different projects.

How to Create Package in Python?

- **Create a Directory:** Start by creating a directory (folder) for your package. This directory will serve as the root of your package structure.
- **Add Modules:** Within the package directory, you can add Python files (modules) containing your code. Each module should represent a distinct functionality or component of your package.
- **Init File:** Include an `__init__.py` file in the package directory. This file can be empty or can contain an initialization code for your package. It signals to Python that the directory should be treated as a package.
- **Sub packages:** You can create sub-packages within your package by adding additional directories containing modules, along with their own `__init__.py` files.
- **Importing:** To use modules from your package, import them into your Python scripts using dot notation.

Example:

- First Create Directory Named **mypackage**.
- Inside `mypackage`, create two Python files: **module1.py** and **module2.py**.
- Create an **__init__.py** file inside `mypackage` (it can be empty).
- Add some code to the modules.
- **# module1.py**

```
def greet(name):
    #print("Hello, {name}!")
    print(f"Hello, {name}!")
```

- `# module2.py`
`def add(a, b):`
 `return a + b`
- `final.py`
`from mypackage import module1, module2`
`# Using functions from module1`
 `module1.greet("Pooja")`
`# Using functions from module2`
 `result = module2.add(3, 5)`
`print("The result of addition is:", result)`

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 2.5x onwards.
- To use sqlite3 module, you must first create a connection object that represents that database and then optionally you can create a cursor object, which will help you in executing all the SQL statements.
- Importing SQLite3 :
`import sqlite3`

Connecting to Database

- Connecting to the SQLite Database can be established using the **connect()** method, passing the name of the database to be accessed as a parameter. If that database does not exist, then it'll be created.
- `import sqlite3`
`conn=sqlite3.connect('stud.db')`
`print("opened database successfully")`

Execute()

- This routine executes an SQL statement.

Example: `cursor.execute("insert into student values(1,'raj')")`

Fetchone()

- It 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()` .

Fetchall()

- It fetches all rows of query result, returning a list. Get `resultset(all rows)` from the cursor object using a `cursor.fetchall()`.

Example:

Step-1 Create a table 'STUDENT' within the database "demo.db" . (createtable.py)

```
import sqlite3
conn=sqlite3.connect('demo.db')
print("opened 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()
```

Step-2 Insert records in STUDENT table("insert.py")

```
import sqlite3
conn=sqlite3.connect('demo.db')
print("Opened successfully")
conn.execute("INSERT INTO STUDENT VALUES(1,'RAJ',20,'UDHNA')")
conn.execute("INSERT INTO STUDENT VALUES(2,'RAJU',18,'SACHIN')")
conn.execute("INSERT INTO STUDENT VALUES(3,'JAY',21,'ADAJAN')")
conn.execute("INSERT INTO STUDENT VALUES(4,'AJAY',25,'CITYLIGHT')")
conn.commit()
print("records inserted successfully")
conn.close()
```

Step-3 Fetch and display your records from the table STUDENT using select statement.(select.py)

```
import sqlite3
conn =sqlite3.connect('demo.db')
data=conn.execute("select * from student")
For row in data:
print("ID = ",row[0])
print("NAME = ",row[1])
print("AGE = ",row[2])
print("ADDRESS = ",row[3])
conn.close()
```

Step-4 Delete record in STUDENT table("delete.py")

```
import sqlite3
conn=sqlite3.connect('demo.db')
id=int(input("Enter ID to delete:"))
conn.execute("DELETE FROM STUDENT WHERE ID=",id)
conn.commit()
print("Successfully deleted.")
conn.close()
```

Step- 5 Update student record in student table(update.py)

```
import sqlite3
conn=sqlite3.connect('demo.db')
id=input("Enter ID :")
age=input("Enter Age :")
sql="UPDATE STUDENT SET AGE = ? WHERE ID = ? "
conn.execute(sql,(age,id))
items =conn.execute("SELECT * from STUDENT")
for item in items:
    print(item)
conn.commit()
conn.close()
```