Python for Class XII _- Part 12

Topics:

Using Python libraries: create and import Python libraries.

Introduction

Python **modules** are `.py` files that consist of Python code. Any Python file can be referenced as a module.

Some modules are available through the Python Standard Library and are therefore installed with your Python installation. Others can be installed with Python's package manager `pip`. Additionally, you can create your own Python modules since modules are comprised of Python `.py` files.

This tutorial will guide you through writing Python modules for use within other programming files.

Writing and Importing Modules

Writing a module is just like writing any other Python file. Modules can contain definitions of functions, classes, and variables that can then be utilized in other Python programs.

From our Python 3 local programming environment or server-based programming environment, let's start by creating a file `hello.py` that we'll later import into another file.

To begin, we'll create a function that prints `Hello, World!`:

```
# module: hello.py
# Define a function
def display():
    print("Hello, World!")
```

If we run the program on the command line with `python hello.py` nothing will happen since we have not told the program to do anything.

Let's create a second file **in the same directory** called `main_prog.py` so that we can import the module we just created, and then call the function. This file needs to be in the same directory so that Python knows where to find the module since it's not a built-in module.

```
# main_prog.py
# import hello module
import hello

# Call the function
hello.display()
```

Because we are importing a module, we need to call the function by referencing the module name in dot notation.

We could instead import the module as `from hello import display` and call the function directly as `display ()`. You can learn more about this method by reading how to using `from` … `import` when importing modules.

Now, we can run the program on the command line:

```
python main_prog.py
```

When we do, we'll receive the following output:

```
Hello, World!
```

To see how we can use variables in a module, let's add a variable definition in our `hello.py` file:

```
# module: hello.py
# Define a function
def display():
    print("Hello, World!")
# define a variable
shark = "Sammy"
```

Next, we'll call the variable in a `print()` function within our `main_prog.py` file:

```
# main_prog.py
# import hello module
import hello

# Call the function
hello.display()

# Print variable
print(hello.shark)
```

Once we run the program again, we'll receive the following output:

Hello, World!
Sammy

It is important to keep in mind that though modules are often definitions, they can also implement code. To see how this works, let's rewrite our `hello.py` file so that it implements the `display()` function:

```
# module: hello.py
# Define a function
def display():
    print("Hello, World!")
# Call the function within module
display()
```

We have also deleted the other definitions in the file.

Now, in our `main_prog.py` file, we'll delete every line except for the import statement:

```
# main_prog.py
# import hello module
import hello
```

When we run `main_program.py` we'll receive the following output:

```
Hello, World!
```

This is because the `hello` module implemented the `world()` function which is then passed to `main_prog.py` and executes when `main_prog.py` runs.

Conclusion:

A module is a Python program file composed of definitions or code that you can leverage in other Python program files.