Python Modules

- A 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 an executable 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.
- Grouping related code into a module makes the code easier to understand and use.

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 greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule
mymodule.greeting("Jonathan")
```

Variables in Module

Save this code in the file mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Loading the module in our python code

We need to load the module in our python code to use its functionality. Python provides two types of statements as defined below.

- The import statement
- The from-import statement

The import statement

The import statement is used to import all the functionality of one module into another.

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.

The syntax to use the import statement is given below.

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

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

```
import file;
name = input("Enter the name?")
file.displayMsg(name)
```

Output:

```
Enter the name?Amit
Hi Amit
```

The from-import statement

Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific attributes of a module. This can be done by using from? import statement.

The syntax to use the from-import statement is given below.

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

Consider the following module named as calculation which contains three functions as summation, multiplication, and divide.

calculation.py:

```
#place the code in the calculation.py
def summation(a,b):
    return a+b
def multiplication(a,b):
    return a*b;
def divide(a,b):
    return a/b;
```

Main.py:

```
from calculation import summation
#it will import only the summation() from calculation.py
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b))
#we do not need to specify the module name while accessing summation()
```

Output:

```
Enter the first number10
Enter the second number20
Sum =  30
```

The from...import statement is always better to use if we know the attributes to be imported from the module in advance. It doesn't let our code to be heavier. We can also import all the attributes from a module by using *.

Consider the following syntax.

from <module> import *

```
# importing sqrt() and factorial from the
# module math
from math import *

# if we simply do "import math", then
# math.sqrt(16) and math.factorial()
# are required.
print(sqrt(16))
print(factorial(6))
```

Renaming a module

Python provides us the flexibility to import some module with a specific name so that we can use this name to use that module in our python source file.

The syntax to rename a module is given below.

import <module-name> as <specific-name>

Example:

#the module calculation of previous example is imported in this example as cal.

```
import calculation as cal;
a = int(input("Enter a?"));
b = int(input("Enter b?"));
print("Sum = ",cal.summation(a,b))
```

Output:

```
Enter a?10
Enter b?20
Sum =  30
```

Built-in Modules: Using dir() function

The dir() function returns a sorted list of names defined in the passed module. This list contains all the sub-modules, variables and functions defined in this module.

Consider the following example.

Example:

```
import datetime
List = dir(datetime)
print(List)
```

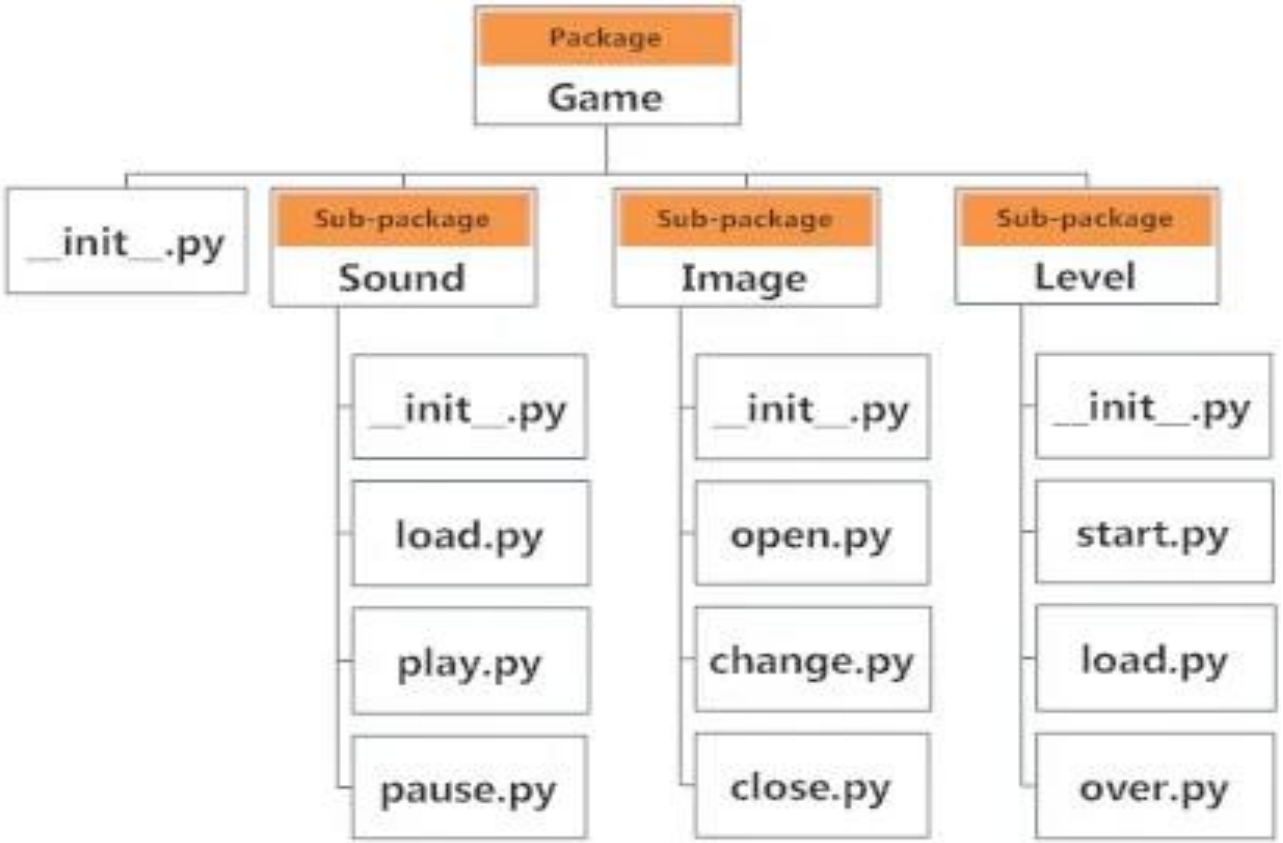
Python packages

- We don't usually store all of our files on our computer in the same location. We use a well-organized hierarchy of directories for easier access.
- Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory.
- As our application program grows larger in size with a lot of modules, we place 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 we generally place the initialization code for that package in this file.

Here is an example. Suppose we are developing a game. One possible organization of packages and modules could be as shown in the figure below.



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:

start.select_difficulty(2)

Another way of importing just the required function (or class or variable) from a module within a package would be as follows:

from Game.Level.start import select_difficulty

Now we can directly call this function.

select_difficulty(2)

Create package

Let's create a package named Employees in your home directory. Consider the following steps.

1. Create a directory with name Employees on path /**home**.

2. Create a python source file with name ITEmployees.py on the path /**home/Employees**.

ITEmployees.py

```
def getNames():
    List = [“S", “T", “R",    "M"]
    return List
```

3. Similarly, create one more python file with name BPOEmployees.py and create a function getBPONames().

```
def getNames():
    List = ["A", "B", "C", "D"]
    return List
```

__init__.py

The package folder contains a special file called __init__.py, which stores the package's content. It serves two purposes:

- The Python interpreter recognizes a folder as the package if it contains __init__.py file.
- __init__.py exposes specified resources from its modules to be imported.
- An empty __init__.py file makes all functions from the above modules available when this package is imported. Note that __init__.py is essential for the folder to be recognized by Python as a package.

4. Now, the directory Employees which we have created in the first step contains two python modules. To make this directory a package, we need to include one more file here, that is __init__.py which contains the import statements of the modules defined in this directory.

__init__.py

from Employees import ITEmployees

from Employees import BPOEmployees

5. Now, the directory Employees has become the package containing two python modules. Here we must notice that we must have to create __init__.py inside a directory to convert this directory to a package.

6. To use the modules defined inside the package Employees, we must have to import this in our python source file. Let's create a simple python source file at our home directory (/home) which uses the modules defined in this package.

Test.py

```
import Employees
print(Employees. ITEmployees.getNames())
print(Employees. BPOEmployees.getNames())
```

Output:

['S', 'T', 'R', 'M']