Day 8: **Module, Packages, Random Data Generation and Exception**

Use the existing module and packages and create user defined module and packages, Generating random data.
Basics of Exception handling

Dr. Jasaswi Prasad Mohanty
Silicon Institute of Technology, Bhubaneswar

Python Modules

- A module is a file containing Python definitions and statements.
- A module can define functions, classes and variables.
- Grouping related code into a module makes the code easier to understand and use.
- Example:

```
# A simple module, Calculation.py
 def add(x, y):
    return (x+y)
def subtract(x, y):
    return (x-y)
```

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 interpreter encounters an import statement, it imports the module if the module is present in the search path.
- To import the module calculation.py, we need to put the import command at the top of the script :

```
import Calculation
print(Calculation.add(10,2))       #12
print(Calculation.subtract(10,2))  #8
```

The *from import* statement

- Python's *from* statement lets you import specific attributes from a module.
- The *from … import …* has the following syntax:

```
# importing sqrt() and factorial() from the module math
from math import sqrt, factorial
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.
- Syntax:

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

- Example:

```
import Calculation as cal
print("Sum = ",cal.add(5,10))
```

Packages

- Packages are a way of structuring many packages and modules which helps in a well-organized hierarchy of data set, making the directories and modules easy to access.
- Packages help us in storing other sub-packages and modules, so that it can be used by the user when necessary.

Creating and Exploring Packages

Silicon
...beyond teaching

- To inform Python that a particular directory is a package, we create a file named __init__.py inside it.
- We may create other modules and sub-packages within it. This __init__.py file can be left blank or can be coded with the initialization code for the package.
- Steps to create a Package:
    1. Create a directory and specify some name as a package name, preferably related to its operation.
    2. Put the classes and the required functions in it.
    3. Create an __init__.py file inside the directory, to let Python know that the directory is a package.

Creating Packages: Example

- Create a new folder named 'MyApp' under the current directory.
- Create an empty __init__.py file in the MyApp folder.
- Inside MyApp, create a subfolder with the name 'mypackage'.
- Create an empty __init__.py file in the mypackage folder.

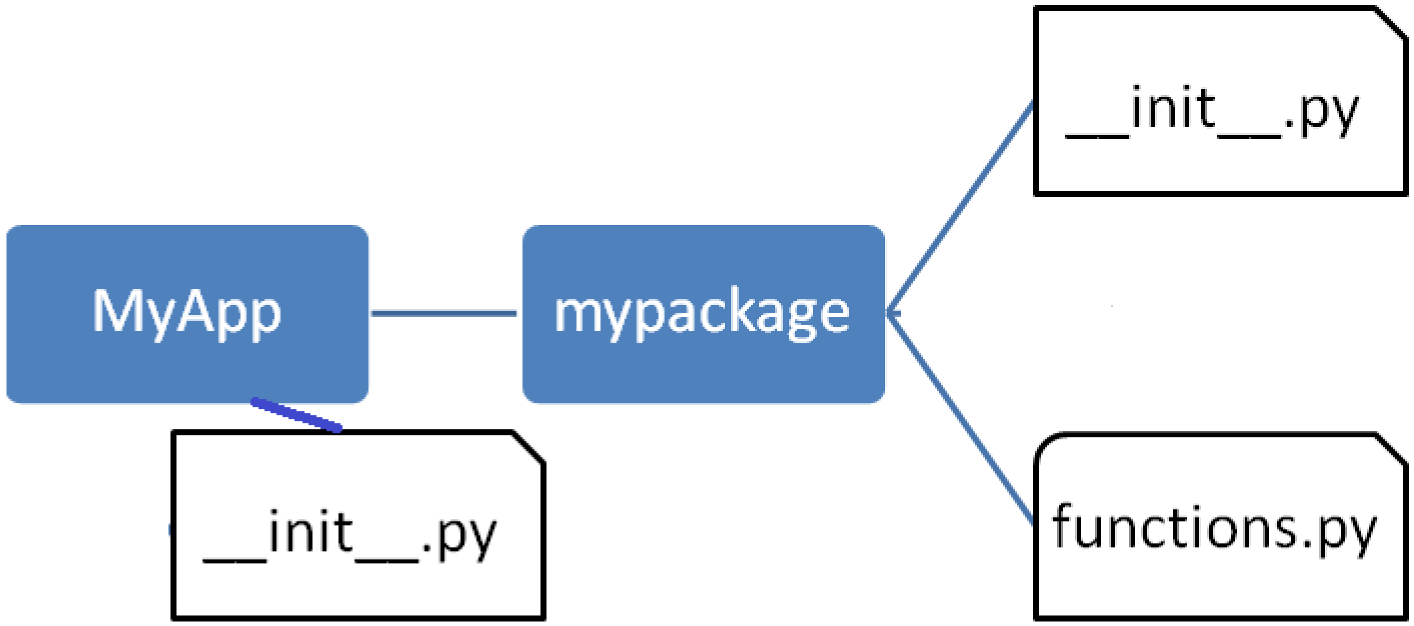
Creating Packages: Example

- Using a Python-aware editor like Spyder, create a module functions.py with following code:

```
def sum(x,y):
    return x+y
def average(x,y):
    return (x+y)/2
def power(x,y):
    return x**y
```

Creating Packages: Example-contd…

- So we have created the following structure:



Creating Packages: Example-contd…

- Now, to test the package, invoke the Python prompt from the MyApp folder by creating a file named ‘test.py’ under MyApp folder with the following code:

```
from MyApp.mypackage import functions
print(functions.sum(3,2))        # Output 5
```

- It is also possible to import specific functions from a module in the package. Change the test.py code as follows and observe:

```
from MyApp.mypackage.functions import average
print(average(3,2))              # Output 2.5
```

Creating Random Numbers using random.randint()

- The randint(start, stop) includes both start and stop numbers while generating random integer.
- It will generate a random number from the **inclusive** range.

Syntax:

```
random.randint(start,stop))
```

Example:

```
import random              #module to create random nos
print(random.randint(1,10))   # generates random
   integer from 1 to 10
print(random.randint(0,100)) # generates random
   integer from 0 to 100
```

Creating Random Numbers using random.randrange()

Syntax: random.randrange(start, stop[, step])

- This function returns a randomly selected integer from range(start, stop, step). This function takes three parameters. Out of the three parameters, start and step are the optional parameters.
  - The start argument is the starting number in a random range. i.e., lower limit. The default value of start is 0.
  - The stop argument is the last number in a random range. the stop argument is the upper limit.
  - The step is a difference between each number in the sequence. The step is optional parameters. The default value of the step is 1 if not specified.
- The randrange(start, stop, step) doesn't include the stop number while generating random integer, i.e., it is **exclusive**.

Example:

```
import random                    #module to create random nos
print(random.randrange(1,10))    # generates random integer from 1 to 9
```

Creating Random Numbers using random.randrange()

- Generating the random integer number of a specific length

```
import random                          #module to create random nos
print(random.randrange(1000,10000)) # generates random integer of length 4
```

- Generating the random integer number multiple of n

```
import random                          #module to create random nos
print(random.randrange(3, 300, 3)) # generates random integer between 3 and 300 which are multiple of 3
```

- Generating a random negative integer

```
import random                          #module to create random nos
print(random.randrange(-60, -6)) # generates random negative integer between -60 and 6
```

Generating Random Numbers from a list using random.choice()

- import random #module to create random nos

  print(random.choice([1,2,3,4])) # generates random numbers from [1,2,3,4]
- import random #module to create random nos

  print(random.choice([-1,1])) # generates random numbers -1 or 1

Generating a list of random numbers

```
import random
randomList = []
# Set a length of the list to 10
for i in range(0, 10):
    randomList.append(random.randint(0, 1000)) # any random numbers from 0 to 1000
print("Printing list of 10 random numbers")
print(randomList)
```

Generating a list of random numbers without repetition

- Using **random.sample()** we can create a list of unique random numbers.
- The random.sample() returns a sampled list of selected random numbers within a range of values.
- It never repeats the element, so that we can get a list of random numbers without duplicates

```
import random
randomList = random.sample(range(0, 1000), 10)
print(randomList)
```

Generating multidimensional array of random integers

- To create a random multidimensional array of integers within a given range, we can use the following **NumPy methods**:
  - randint()
  - np.randint(low, high, size, dtype) To get random integers array from low (inclusive) to high (exclusive).
  - np.random_integers(low, high, size) To get Random integers array of type NumPy int between low and high, inclusive.

Generating a 4 x 4 array of integers between 10 and 50 (exclusive)

```
import random
newArray = numpy.random.randint(10, 50, size=(4, 4))
print(newArray)
```

Output:

```
[[10 48 30 24]
 [13 46 30 11]
 [12 28 49 26]
 [30 18 49 35]]
```

Points to remember about randint() and randrange()

- Use randint() when you want to generate a random number from an inclusive range.
- Use randrange() when you want to generate a random number within a range by specifying step value. It produces a random number from an exclusive range.
- The randint() rand randrange() only works with integers. You cannot use float numbers.
- To generate a random float number use random.random().

  # generates float values from 0 to 1

Silicon

...beyond teaching

Thank You