# 02 Function, Object and File

- Functions, Generators, Co-routines
- Objects and classes, Exceptions and modules
- File Input and Output

# Functions

- use the *def* statement to create a function

```
def remainder(a,b):
    q = a // b      # // is truncating division.
    r =  a - q*b
    return r
```

- To invoke a function, simply use the name of the function followed by its enclosed in parentheses,

```
result = remainder(37,15)
```

# Functions

- To assign a default value to a function parameter, use assignment:

```
def connect(hostname, port ,timeout=300):
```

# Generators

- A function can generate an entire sequence of results if it uses the *yield* statement.
- next() : produces a sequence of results through successive calls

# Co-routines

- Functions operate on a single set of input arguments.
- A function can also be written to operate as a task that processes a sequence of inputs sent to it.
- This type of function is known as a coroutine and is created by using the yield statement.

# Co-routines

## send()

- A coroutine is suspended until a value is sent to it

## close()

- This continues until the coroutine function returns or close

# Objects and classes

- All values used in a program are objects.

- An object consists of internal data and method that perform various kinds of operations.

```
>>>items = [37, 42]   # Create a list object
>>>items.append(73) #Call append() method
```

# Objects and classes

- dir() : lists the methods available on an object and is a useful tool for interactive experimentation.
- Special methods that always begin and end with a double underscore. Eg. __ init__()

# Exceptions

- If an error occurs in program,an exception is raised and a traceback message appears:

```
Traceback (most recent call last):
File "foo.py", line 12, in <module>
IOError: [Errno 2] No such file or directory:   'file.txt‘
```

- The traceback message indicates the type of error that occurred, along with its location.

# Modules

- Python allows you to put definitions in a file and use them as a module that can be imported into other programs and scipts.

```
# file : div.py
 def divide(a,b):
     q = a/b  # If a and b are integers, q is an integer
     r = a - q*b
     return (q,r)
```

# Modules

- To use your module in other programs, you can use the import statement:

```
import div
a, b = div.divide (2305, 29)
```

- To load all of a module's contents into the current namespace, you can also use the following:

```
from div import *
```

# Modules

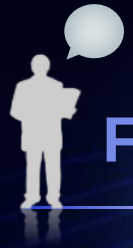
- If you want to import a module using a different name, supply the import statement with an optional as qualifier, as follows:

```
import div as foo
a,b = foo.divide(2305,29)
```

# Modules

- To import specific definitions into the current namespace, use the from statement:

```
from div import divide
a,b = divide(2305,29)
```



# File Input and Output

**open( )**

- returns a new file object

**readline()**

- reads a single line of input, including the terminating newline

# File Input and Output

**>>**

- **print the output to a file**