Function

Function a named block of python statements. It has the following parts:

- keyword **def** defines a function;
- function has a name: it should be meanful;
- function may have input parameters, e.g. start,stop, stride
- function has a body, which implements functionality of the function
- function can return data as a result

In [ ]:

```
```

In [35]:

```
# function demo
def hello_function():
    print("Hello from a function")

# call a function
hello_function()
```

```
Hello from a function
```

Function input parameters

Information can be passed to functions as parameter.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

The order of the input parameters in function definition is important

In [2]:

```
# a greeting function example.
# given a name as input, the function print out "Hi, ***"

def greeting(name):
    print('Hello, ', name)

greeting('Frank')
a = 'Dianhao'
greeting(a)
```

```
Hello,  Frank
Hello,  Dianhao
```

In [12]:
```
# this function creates a list of integer numbers
def creat_num_list(start, stop, stride):
    l = []
    for i in range(start, stop, stride):
        l.append(i)
    return l, 'finished'

l, name = creat_num_list(0, 8, 2)
print(type(l))
print(name)
```

```
<class 'list'>
finished
```

Default values for input parameters

If we call the function without parameter, it uses the default value:

In [38]:
```
# this function creates a list of integer numbers
def creat_num_list(start, stop=0, stride=1):
    return list(range(start, stop, stride))

print(creat_num_list(0))
```

```
[0, 1, 2, 3, 4, 5]
```

In [39]:
```
# this function creates a list of integer numbers
def creat_num_list(start=2, stop=6, stride=1):
    return list(range(start, stop, stride))

print(creat_num_list())
```

```
[2, 3, 4, 5]
```

Keyword parameters

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

In [14]:
```
# this function creates a list of integer numbers
def creat_num_list(start, stop, stride):
    l = []
    for i in range(start, stop, stride):
        l.append(i)
    return l


print(creat_num_list(stop=10, stride=1, start=0))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [19]:
```
def get_square_cubic(num):
    x = num*num
    y= x*num
    return x, y

x = int(input('input a number'))
s, c = get_square_cubic(x)
print(s, c)
print(x)
```

```
input a number10
100 1000
10
```

Arbitrary parameters

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

In [1]:
```
def my_function(*kids):
    print(len(kids))
    print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
3
The youngest child is Linus
```

Function parameters with the Caller

When a caller passed number or string as parameters to a function, any modification to the parameter value will not affect the caller.

In [20]:
```
def my_function(num):
    num = num + 1

x = 10
my_function(x)
print(x)


num = 10
my_function(num)
print(num)
```

```
10
10
```

In [ ]:
```
When passing a list (or dictionary, set) to a function, any change to the ite
```

In [22]:
```
def update_list(names):
    for i in range(len(names)):
        names[i] = names[i].upper()

students = ['JOhn', 'Mary']
print('before function call', students)
update_list(students)
print(students)
```

```
before function call ['JOhn', 'Mary']
['JOHN', 'MARY']
```

Variable Scope

Global variables are the one that are defined and declared outside a function

- global variables can be read from inside a function

Local variables: defined inside a function

- if a local variable has the same name as a global variable, by default, local variable will be used inside the function.

In [29]:
```
def greeting():
    print(s)
    #global s
    #s="larry"
    print('inside function',s)

s = "Hello"
print('before function', s)
greeting()
print('outside function', s)
```

```
before function Hello
Hello
inside function Hello
outside function Hello
```

In [43]:
```
def greeting():
    s = "Hello"
    print(s)

greeting()
```

```
Hello
```

In [44]:
```
def greeting():
    s = "Hello"
    print(s)

s = "Hi"
greeting()
print('global variable s is: ', s)
```

```
Hello
global variable s is:  Hi
```

To change the value of a global variable inside a function, use 'global' to declare the variable.

In [31]:
```
def greeting():
    global s
    s = "Hello"
    print(s)

s = "Hi"
greeting()
print('global variable s is: ', s)
```

```
Hello
global variable s is:  Hello
```

In [49]:
```
def greeting(s):
    #s = "Hello"
    print('variable s is: ', s)

s = "Hi"
greeting(s)
print('global variable s is: ', s)
```

```
variable s is  Hi
global variable s is:  Hi
```

Function can return data

use keyword 'return'

In [32]:
```
# a function that take a number as input, return its square
def square(x):
    return x*x

a = 5
b = square(5)
print(a, b)
```

```
5 25
```

Return multiple data values

In [34]:
```
# a function that take a number as input, return its square and cubic
def s_and_c(x):
    return x*x, x*x*x

a = 5
b, c = s_and_c(5)
print(a, b, c)
```

```
5 25 125
```

Type *Markdown* and LaTeX: α^2