Functions

Objectives:

- Basic Functions
- More on Function Parameters
- Local and Global Scope
- Docstring

1. Why Function

Function is a group of related statements that perform a specific task.

Benefits of Function

- Functions help break program into smaller and modular chunks.
- It supports code reusability and reduces repeative code.
- It make large program more organized and manageable.

What functions have used so far?

We have used a few functions when we learnt about Pyhton basic data types. Can you name any?

- How do you check data type of an object?
- How do you convert a value from one data type to another?
- How to you print out value(s) in console?

In [6]:
```
1 a = '123'
2 print(type(a))
3 b = int(a)
4 print(b)
```

```
<class 'str'>
123
```

2. Basic Functions

Simplest Function that does nothing

Let's define a simplest function `do_nothing()` , which does nothing.

- Keyword `pass` is used to indicate nothing to be done in the function body.

In [3]:
```
def do_nothing():
    pass
```

Check the type of `do_nothing` .

In [2]:
```
type(do_nothing)
```

Out[2]:
```
function
```

Call the function `do_nothing()` .

In [4]:
```
do_nothing()
```

Exercise:

Let's define a basic function which simply prints "Hell World".

In [7]:
```
def hello_world():
    print("Hello World")

hello_world()
```

```
Hello World
```

Input Parameters

Function can takes in zero or more input parameters.

- Input parameters are considered as local variables in a function.

Exercise:

Enhance the `hello()` function with a `who` input parameter.

In [14]:
```
def hello(who):
    print("Hello " + who)

hello("Singapore")
```

```
Hello Singapore
```

Return Statement

Function can exit with a `return` statement. The return statement can return a value at the same time.

Exercise:

Define a function `add(a, b)` which takes in parameters `a` & `b`, and returns sum of them.

In [16]:
```
def add(a, b):
    return a + b

c = add(3, 5)
print(c)
```

```
8
```

Implicit Return Statement

If a function has no `return` statement in a function, or its `return` statement doesn't followed by any object, the function returns a `None`.

Exercise:

What is the data type of return value from `hello(who)` function?

In [18]:
```
s = hello('World')
print(s)
```

```
Hello World
None
```

Return Multiple Values

A function may return **multiple** values. When multiple values are returned, they are packed into a tuple.

In [22]:
```
def mult_result():
    return 2,3,5,7

result = mult_result()
print(type(result))
```

```
<class 'tuple'>
```

Exercise:

Define a function `simple_math(a, b)` which returns sum, difference, multiplication and division values of `a` and `b`.

In [23]:
```
def simple_calc(a, b):
    return a + b, a - b, a * b, a / b

result = simple_calc(20,10)
print(result)
```

```
(30, 10, 200, 2.0)
```

3. Function Arguments

Let's define a function `simple_add()` which takes in 3 values and returns sum of them

In [25]:
```
def simple_add(a, b, c):
    return a + b + c
```

Call with Positional Arguments

All arguments, `a` and `b` and `c` are **required arguments**. You need to pass in all required values before you can call `simple_add` function.

In [26]:
```
simple_add(1, 2, 3)
```

Out[26]: 6

Call with Keyword Arguments

Instead of pass arguments in order, you can passs arguments identified by their name, i.e. keyword arguments.

- With keyword argument, order of arguments is not required.
- It can make your code easier to read.

In [27]:
```
simple_add(10, c = 30, b = 20)
```

Out[27]: 60

Default Arguments

Default arguments have arguments with default values. When there is no value is passed, that argument will use its default value.

- **Note:** All required arguments must be before default arguments.

Exercise:

Modify the `simple_add()` function to give default value 10 and 20 to `b` and `c` respectively.

In [28]:
```
def simple_add(a, b = 10, c = 20):
    s = a + b + c
    return s

simple_add(5, c=15)
```

Out[28]: 30

4. Variable Scope - Global vs Local

The scope of a variable determines the portion of the program where you can access a particular variable. There are two basic variable scopes in Python:

- **Global variables:** variables defined **outside** a function body
- **Local variables:** variables defined **inside** a function body

Global and Local variables are in different scopes

- Local variables can be accessed only inside the function in which they are declared
- Global variables can be accessed throughout the program body

Accessing Global Variables

Global variable `x` can be accessed both inside and outside of a function.

In [36]:
```
x = 3;  # Global variable

def show():
    print('Inside function:', x)

show()
print('Outside Function:', x);
```

```
Inside function: 3
Outside Function: 3
```

Accessing Local Variable

The `y` variable is a local variable. It does not exist outside the function.

In [38]:
```
# Clean up any `val` variable from previous example
if 'y' in globals():
    del y


def show():
    y = 3
    print("In function:", y)

show()
print("Outside function:", y)
```

```
In function: 3

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-38-ec0ef725fde8> in <module>
      9
     10 show()
---> 11 print("Outside function:", y)

NameError: name 'y' is not defined
```

Variable Created in Different Scope

Whenever a variable is assigned, it will be automatically created if it does not exist in current scope.

In [39]:
```
x = 2
print(id(x))

def foo():
    x = 3
    print(id(x))

foo()
print(x)
```

```
140727757059904
140727757059936
2
```

Question: Why there is an error in following code?

In [ ]:
```
x = 2

def foo():
    x = x * 2
    print(x)

foo()
```

Keyword `global`

Question: But what if I would like to modify a global variable in the function?

- To access global a variable in a function, you can use the `global` keyword.

In [40]:
```
x = 2

def foo():
    global x
    x = 3

foo()
print(x)
```

```
3
```

5. Docstring

The first string after the function header is called **documentation string**, which is commonly called **docstring**.

It is commonly added to functions and modules to serve as documentation for your function.

In [29]:
```
def hello(who):
    '''Prints hello message
    Arguments:
        - who: name in string
    '''
    print('Hello ', who)
```

How to access Docstring?

Docstring of a function or class can be accessed using **help()** function or `__doc__` attribute of the function or class.

In [31]:
```
1 help(hello)
```

```
Help on function hello in module __main__:

hello(who)
    Prints hello message
    Arguments:
        - who: name in string
```