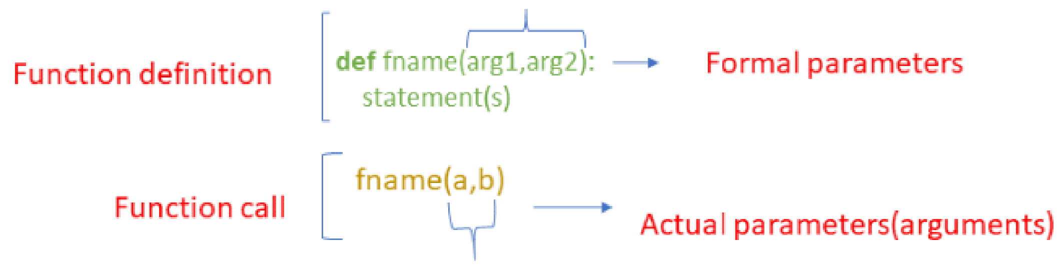
Define a Function in Python

Shouke Wei, Ph.D. Professor

Email: shouke.wei@gmail.com

1. Python Function Struture



- Create a function
- Call a function
- Function Parameters/Arguments
- Gobal variables and local variables

1.1 Create a function

- A function is a block of code, which can be called to run when it is needed
- `def` is used to creat a function

In [8]:
```
def welcome():
  print("Hello evevery! Welcome to my Python tutorial!")
```

1.2 Call a function

- use the function name followed by parenthesis:

In [9]:
```
welcome()
```

```
Hello evevery! Welcome to my Python tutorial!
```

2. Arguments/Parameters

- The terms parameter and argument can be used for the same thing
  - information that are passed into a function, but
  - A parameter: the variable listed inside the parentheses in the function definition
  - An argument: the value that is sent to the function when it is called

2.1 One argument

In [1]:
```
def welcome(Name):
    print(f'Hello {Name}, Welcome to my Python tutorial!')
```

In [2]:
```
name = 'Jack'
welcome(name)
```

```
Hello Jack, Welcome to my Python tutorial!
```

2.2 Two arguments

In [1]:
```
def welcome(fName,lName):
    print('Hello {} {}, Welcome to my Python tutorial!'.format(fName,lName))
```

In [17]:
```
welcome('Jack','Smith')
```

```
Hello Mr. Jack Smith, Welcome to my Python tutorial!
```

2.3 More argments

- Function with 3 arguments

In [23]:
```
def sum_caculator(x,y,z):
    print("sum:",x+y+z)

sum_caculator(8,22,38)
```

```
sum: 68
```

It works very well when you pass three arguments to the function, but how about we pass four or more arguments?

In [24]:
```
sum_caculator(8,22,38,30)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-c52e52d9dbc7> in <module>
----> 1 sum_caculator(8,22,38,30)

TypeError: sum_caculator() takes 3 positional arguments but 4 were given
```

2.4 Arbitrary Arguments

- `*args` or `**kwargs` : we are unsure about the number of arguments to pass in the functions
    - `*args` (Non Keyword Arguments)
    - `**kwargs` (Keyword Arguments)
- make the function flexible

args example

In [4]:
```
#  Using *args to pass the variable length arguments to the function
def sum_caculator(*args):
    sum = 0

    for n in args:
        sum+=n

    print("Sum:",sum)
```

In [7]:
```
sum_caculator(4,6)
sum_caculator(4,6,8,10)
```
```
Sum: 10
Sum: 28
```

In [9]:
```
def sum_caculator(*num):
    sum = 0

    for n in num:
        sum += n

    print("Sum:",sum)
```

In [10]:
```
sum_caculator(4,6)
sum_caculator(4,6,8,10)
```
```
Sum: 10
Sum: 28
```

***kwargs example*

In [7]:
```
def info(**kwargs):
    print("Data type of argument:",type(kwargs))

    for key, value in kwargs.items():
        print(f"{key} is {value}.")
```

In [8]:
```
info(Firstname="Sita", Lastname="Sharma", Age=22, Phone=1234567890)
```
```
Data type of argument: <class 'dict'>
Firstname is Sita.
Lastname is Sharma.
Age is 22.
Phone is 1234567890.
```

In [9]:
```
info(Firstname="John", Lastname="Wood", Email="johnwood@nomail.com", Country="USA", Age=25, Phone=98
```
```
Data type of argument: <class 'dict'>
Firstname is John.
Lastname is Wood.
Email is johnwood@nomail.com.
Country is USA.
Age is 25.
Phone is 9876543210.
```

2.5 Default Parameter Value

- If we call the function without argument, it uses the default value

In [35]:
```
def greeting(name = 'there'):
  print(f'Hello {name}!')
```

In [37]:
```
greeting("Susan")
```
```
Hello Susan!
```

In [38]:
```
greeting()
```
```
Hello there!
```

2.6 List Argument

- We can send any data types of argument to a function (string, number, list, dictionary etc.)

In [49]:
```
def mystudent(students):

    for name in students:
        print(f"{name} is my student.")
```

In [50]:
```
studentlist = ["Jack", "Tom", "Ophelia"]
```

In [51]:
```
mystudent(studentlist)
```

```
Jack is my student.
Tom is my student.
Ophelia is my student.
```

3. Return Values

- The `return` statement make a function return a value

In [52]:
```
def sum_caculator(x,y,z):
    sum = x+y+z
    return sum
```

In [53]:
```
sum_caculator(8,22,38)
```

Out[53]:
```
68
```

4. Global and local variables

4.1 Global variables

- Variables that are created outside a function
- Global variables can be used both inside and outside of functions

In [60]:
```
name = "Jack"

def hello():
  print(f"Hello, {name}!")
```

In [61]:
```
hello()
```

```
Hello, Jack!
```

In [62]:
```
print(name)
```

```
Jack
```

4.2 Local variables

- Variables that are created inside a function
- They can be used only inside the function

In [14]:
```
def subtractor():
  x = 10
  y = 5
  print(x-y)
```

In [15]:
```
subtractor()
```

```
5
```

```
In [16]: print(x)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17960/1353120783.py in <module>
----> 1 print(x)

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