Subject Artificial Intelligence:

This notebook will teach you about the functions parameters and arguments in the Python Programming Language. By the end of this lab, you'll know the basic concepts about function, arguments and how to use functions, arguments.

The Following Example defines the greet() Function:

Example: User-defined Function

```
def greet():
    """This function displays 'Hello World!'"""
    print('Hello World!')
```

Above, we have defined the greet() function. The first statement is a docstring that mentions what this function does. The second like is a print method that displays the specified string to the console. Note that it does not have the return statement.

To call a defined function, just use its name as a statement anywhere in the code. For example, the above function can be called using parenthesis, greet().

Example: Calling User-defined Function
Copy

```
greet()
```

Output

```
Hello World!
```

By default, all the functions return None if the return statement does not exist.

Example: Calling User-defined Function

```
val = greet()
print(val)
```

Output

```
None
```

The help() function displays the docstring, as shown below.

Example: Calling User-defined Function

```
>>> help(greet)
Help on function greet in module __main__:

    greet()
```

This function displays 'Hello World!'

Function Parameters:

It is possible to define a function to receive one or more parameters (also called arguments) and use them for processing inside the function block. Parameters/arguments may be given suitable formal names. The greet() function is now defined to receive a string parameter called name. Inside the function, the print() statement is modified to display the greeting message addressed to the received parameter.

Example: Parameterized Function

```
def greet(name):
    print ('Hello ', name)

greet('Steve') # calling function with argument
greet(123)
```

Output
Hello Steve
Hello 123

The names of the arguments used in the definition of the function are called formal arguments/parameters. Objects actually used while calling the function are called actual arguments/parameters.

The function parameters can have an annotation to specify the type of the parameter using parameter:type syntax. For example, the following annotates the parameter type string.

Example: Parameterized Function

```
def greet(name:str):
    print ('Hello ', name)

greet('Steve') # calling function with string argument
greet(123) # raise an error for int argument
```

Multiple Parameters:

A function can have multiple parameters. The following function takes three arguments.

Example: Parameterized Function

```
def greet(name1, name2, name3):
   print ('Hello ', name1, ' , ', name2 , ', and ', name3)

greet('Steve', 'Bill', 'Yash') # calling function with string argument
```

Output
Hello Steve, Bill, and Yash

Unknown Number of Arguments:

A function in Python can have an unknown number of arguments by putting * before the parameter if you don't know the number of arguments the user is going to pass.

Example: Parameterized Function

```
def greet(*names):
   print ('Hello ', names[0], ', ', names[1], ', ', names[3])

greet('Steve', 'Bill', 'Yash')
```

Output
Hello Steve, Bill, and Yash

The following function works with any number of arguments.

Example: Parameterized Function

```
def greet(*names):
        i=0
        print('Hello ', end='')
        while len(names) > i:
                print(names[i], end=', ')
                i+=1

greet('Steve', 'Bill', 'Yash')
greet('Steve', 'Bill', 'Yash', 'Kapil', 'John', 'Amir')
```

Output
Hello Steve, Bill, Yash,
Hello Steve, Bill, Yash, Kapil, John, Amir

Function with Keyword Arguments:

In order to call a function with arguments, the same number of actual arguments must be provided. However, a function can be called by passing parameter values using the parameter names in any order. For example, the following passes values using the parameter names.

```
def greet(firstname, lastname):
   print ('Hello', firstname, lastname)

greet(lastname='Jobs', firstname='Steve') # passing parameters in any order using keyword argument
```

Output
Hello Steve Jobs

Keyword Argument **kwarg:

The function can have a single parameter prefixed with **. This type of parameter initialized to a new ordered mapping receiving any excess keyword arguments, defaulting to a new empty mapping of the same type.

Example: Parameterized Function

```
def greet(**person):
        print('Hello ', person['firstname'],  person['lastname'])

greet(firstname='Steve', lastname='Jobs')
greet(lastname='Jobs', firstname='Steve')
greet(firstname='Bill', lastname='Gates', age=55)
greet(firstname='Bill') # raises KeyError
```

Output
Hello Steve Jobs
Hello Steve Jobs
Hello Bill Gates

When using the ** parameter, the order of arguments does not matter. However, the name of the arguments must be the same. Access the value of keyword arguments using paramter_name['keyword_argument'].

If the function access the keyword argument but the calling code does not pass that keyword argument, then it will raise the KeyError exception, as shown below.

Example: Parameterized Function

```
def greet(**person):
        print('Hello ', person['firstname'],  person['lastname'])

greet(firstname='Bill') # raises KeyError, must provide 'lastname' argument
```

Output

```
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    greet(firstname='Bill')
```

```
  File "<pyshell#19>", line 2, in greet
    print('Hello ', person['firstname'],  person['lastname'])
KeyError: 'lastname'
```

Parameter with Default Value:

While defining a function, its parameters may be assigned default values. This default value gets substituted if an appropriate actual argument is passed when the function is called. However, if the actual argument is not provided, the default value will be used inside the function.

The following greet() function is defined with the name parameter having the default value 'Guest'. It will be replaced only if some actual argument is passed.

Example: Parameter with Default Value

```
def greet(name = 'Guest'):
    print ('Hello', name)

greet()
greet('Steve')
```

Output
Hello Guest
Hello Steve

Function with Return Value:

Most of the time, we need the result of the function to be used in further processes. Hence, when a function returns, it should also return a value.

A user-defined function can also be made to return a value to the calling environment by putting an expression in front of the return statement. In this case, the returned value has to be assigned to some variable.

Example: Return Value

```
def sum(a, b):
    return a + b
```

The above function can be called and provided the value, as shown below.

Example: Parameter with Default Value

```
total=sum(10, 20)
print(total)
total=sum(5, sum(10, 20))
print(total)
```

Output
30
35

Reference
https://www.tutorialsteacher.com/

Questions

function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less?

show how to use a default parameter value with an example?

To let a function return a value, use the `return` statement in code?