FUNCTIONS

1. What is a Function?

A function is a group of statements that perform a specific task.

2. Why function?

- Functions help us to break or divide our program into smaller and modular chunks.
- As our program grows larger and larger, functions make it more organized and manageable.
- It avoids repetition and makes the code reusable. Thus, it minimizes redundancy.
- In short we can say that a function is a block of organized, reusable code that is used to perform a single, related action.

3. Rules for creating a function:

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax :

```
def function_name(parameters):
        """docstring"""
        Statement(s)
        return [expression]
```

Example :

.

```
def sayhello():
        """
        This prints Hello
        """
        print("Hello")


Output: Hello
```

4. Calling a Function:

To call a function in python we just have to simply name it, and pass arguments, if any. The syntax for calling a function is given as shown below:

```
Function_name( argument 1, argument 2,....., argument n)
```

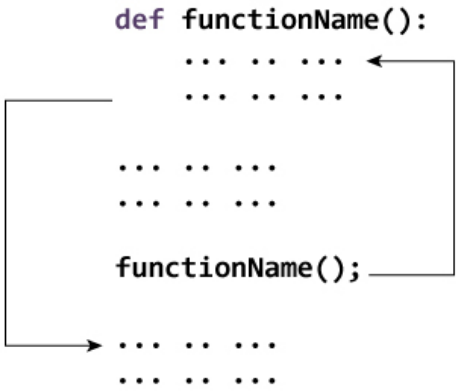
It can be understood better using an *example:*

```
def my_function( ):
        print("Just trying to understand how Functions work in Python")

my_function( )                    #calling a function    # function call
```

Output: Just trying to understand how Functions work in Python

5. How Function Works in Python?



- Basically the main program is also a function which is commonly referred as main function.
- The called function is the subprogram which contains the actual logic to be executed.
- It is basically an independent program which works in relation to the main program.
- But it is not executed at it's own unless called from the main program.
- The function is called from the main program using the calling function.
- When the calling function is executed, the control of the program passes from the calling function to the called function, the called function is executed completely and then the control of the program returns back to the calling function.

6. Difference between parameters and arguments:

- A parameter is a variable defined by a method that receives a value when the method is called.
- An argument is a actual value that is passed to a method when it is invoked

Example:

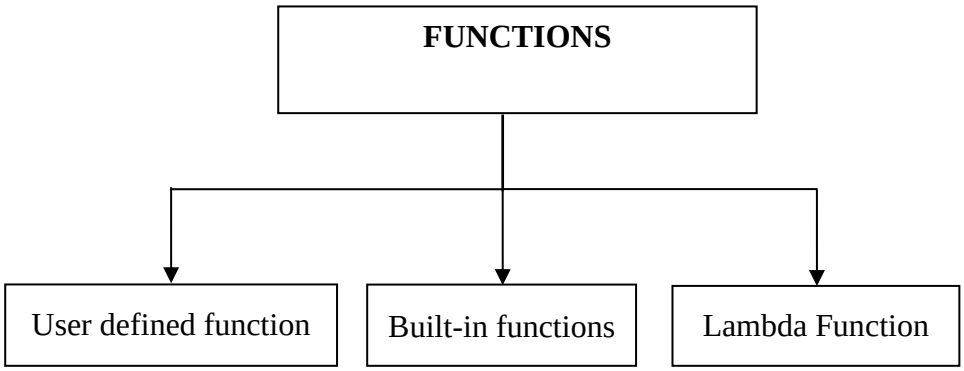
```
def add(x, y):                  # add function is created passing parameters x & y
        sum = x + y
        return sum

add(2,3)                        # function called passing arguments 2 &3


Output:  5
```

- Here, in this example add( ) has two parameters x and y. if the function is called as add(2,3) then 2 & 3 are the arguments.

7. Types of Function:



a) **User defined functions:** Any function defined by a user themselves.

Example: To find the area of rectangle using function *with arguments.*

```
def  rect( l,b ):                                   # Function Definition
        area = ( l*b)
        print( "The area of rectangle is:  " , area)

rect( 5,4 )                                         # Function Call


Output: The area of rectangle is: 20
```

Here, the function 'rect' has two parameters passed i.e. l & b (length and breadth) and then when we call the function then we have to give two arguments to get the area of the rectangle calculated.

Note: If we call any function with improper number of arguments with respect to it's numbers of parameters given while defining a function then it will give error.

Example:

```
rect(4)                          # Function Call

Output:

Error: rect( ) missing 1 required positional argument: 'b'
```

Example: To find the area of the circle without arguments.

```
def circle( ):                   # circle function is created without arguments
    radius = 5
    pi = 3.14
    area = pi*r*r
    print(" The area of circle is: ", circle)

circle( )                        # calling function circle without arguments

Output: The area of circle is: 78.5
```

Here, in the above example we have made a function circle without arguments for calculating the area of circle where we need to give inputs required i.e. value of radius and pi.

b) **Built-in functions:** There are some set of predefined functions in python which are called as Built-in functions.

Example:

abs( ) : returns the absolute value of a number.

chr( ) : returns the character in python for an ASCII value.

callable( ) : tells us whether the object can be called.

c) Lambda Function:

- It is also known as Anonymous function. It is defined without a name.
- While normal functions are defined using the 'def' keyword in Python, anonymous functions are defined using the 'lambda' keyword.

Syntax

```
lambda arguments: expression
```

- Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. They can be used wherever function objects are required.

Example:

```
double = lambda x: x*2
print(double(5))


Output: 10
```

- In the above program, "lambda x: x*2" is the lambda function where x is the argument and x*2 is the expression that gets evaluated and returned.
- This function has no name.
- It returns a function object which is assigned to the identifier double.
- Hence it can be called as a normal function.
- The statement:

```
double = lambda x: x*2
```

is nearly same as:

```
def double(x):
        return x*2
```

8. Types of Arguments:

- There are various types of Python arguments functions.

a) Default Arguments:

- Function arguments can have default values in Python.
- We can provide a default value to an argument by using the assignment operator (=).

Example:

```
def greet(name, msg = "Good morning!"):          # Function Definition
    """
    This function greets to the person with the provided message.
    If message is not provided, it defaults to "Good morning!"
    """
    print("Hello",name + ', ' + msg)

greet("Kate")                                    # Function Call
greet("Bruce","How do you do?")                  # Function Call
```

Output: Hello Kate, Good morning!
Hello Bruce, How do you do?

b) Arbitrary Arguments:

- Sometimes, we do not know in advance the number of arguments that will be passed into a function.
- Python allows us to handle this kind of situation through function calls with an arbitrary number of arguments.
- In the function definition, we use an asterisk (*) before the parameter name to denote this kind of argument.

Example:

```
def  fruits(*fnames):
        """This function displays the fruits names."""
        #fnames is a tuple with arguments.
        For fruits in fnames:
        print(fruit)
fruits("Orange, Apple, Mango, Cherry ")
Output: Orange
        Apple
        Mango
        Cherry
```

c) **Keyword Arguments:**

We have learned that when we pass the values during function call, they are assigned to the respective arguments according to their position.

For example if a function is defined like this:

def demo(name, age): and we are calling the function like this: demo("Steve", "35") then the value "Steve" is assigned to the argument name and the value "35" is assigned to the argument age.

Such arguments are called positional arguments.

Python allows us to pass the arguments in non-positional manner using keyword arguments.

Lets take an example to understand this.

```
def  demo(name, age):
        print(name + "is" + age + "years old")
        #2 keyword arguments ( in order )
        demo( name = "Ram", age  = "25")
        #2 keyword arguments ( not in order )
        demo ( age  = "35", name = "Mohan" )
        #1 positional and 1 keyword argument
        Demo (  "Bhargav", age = "40" )
```

Output:

Ram is 25 years old

Mohan is 35 years old

Bhargav is 40 years old

9. Scope and Lifetime of Variables:

A variable isn't visible everywhere and alive every time. We study this in functions because the scope and lifetime for a variable depend on whether it is inside a function.

- **Scope:**
  - A variable's scope tells us where in the program it is visible.
  - A variable may have local or global scope.

i. **Local Scope:**
   - A variable that's declared inside a function has a local scope.
   - In other words, it is local to that function.

ii. **Global Scope:**
   - When you declare a variable outside python function, or anything else, it has global scope.
   - It means that it is visible everywhere within the program. However, we can't change its value from inside a local scope (here, inside a function).
   - To do so, we must declare it global inside the function, using the 'global' keyword.

- **Lifetime:**
  - The lifetime of a variable is the period throughout which the variable exists in the memory.
  - The lifetime of variables inside a function is as long as the function executes.
  - They are destroyed once we return from the function.
  - Hence, a function does not remember the value of a variable from its previous calls.

10. Video Link: https://youtu.be/BVfCWuca9nw

References:

- https://www.programiz.com/python-programming/function
- https://www.geeksforgeeks.org/functions-in-python/
- https://www.tutorialspoint.com/python/python_functions.htm
- https://www.w3schools.com/python/python_functions.asp

… Happy Learning ! ...