Python - Functions

Function A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- 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 code block within every function starts with a colon (:) and is indented.

Syntax

Creating a Function

In Python a function is defined using the def keyword:

Example:

```
def my_function():
  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example:

```
def my_function():
  print("Hello from a function")
my_function()
```

Output:

```
Hello from a function
```

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 following example has a function with one parameter (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example:

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
```

```
my_function("Tobias")
my_function("Linus")
```

Output:

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Return Values

To let a function return a value, use the return statement:

Example:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output

```
15
25
45
```

Global and local variables

Global Variables

In Python, a variable declared outside of the function or in global scope is known as global variable. This means, global variable can be accessed inside or outside of the function.

Let's see an example on how a global variable is created in Python.

Example 1: Create a Global Variable

```
x = "global"

def var():
    print("x inside :", x)

var()
print("x outside:", x)
```

Output:

```
X inside:globalx
Outside:global
```

Local Variables

Example : Create a Local Variable

Normally, we declare a variable inside the function to create a local variable.

```
def var():
    y = "local"
    print(y)
var()
```

Output:

Local

Example : Using Global and Local variables in same code

```
x = "global"
def var():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)
    var()
```

Output:

global global

local

Example : Global variable and Local variable with same name

```
x = 5
def foo():
    x = 10
    print("local x:", x)
foo()
print("global x:", x)
```

Output:

Local x:10

Global x:5

Python Recursion Function

What is recursion in Python?

Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

Python Recursive Function

We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

Following is an example of recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1*2*3*4*5*6 = 720.

Example of recursive function

```
# An example of a recursive function to

# find the factorial of a number

def calc_factorial(x):

    """This is a recursive function

    to find the factorial of an integer"""

  if x == 1:

        return 1

    else:

        return (x * calc_factorial(x-1))

num = 4

print("The factorial of", num, "is", calc_factorial(num))
```

Output:

The factorial of 4 is 24

In the above example, `calc_factorial()` is a recursive functions as it calls itself.

```
calc_factorial(4)           # 1st call with 4
4 * calc_factorial(3)       # 2nd call with 3
4 * 3 * calc_factorial(2)   # 3rd call with 2
4 * 3 * 2 * calc_factorial(1)  # 4th call with 1
4 * 3 * 2 * 1               # return from 4th call as number=1
4 * 3 * 2                   # return from 3rd call
4 * 6                       # return from 2nd call
24                          # return from 1st call
```

Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

```
# Python program to find the sum of natural numbers up to n using recursive function

def recur_sum(n):
   if n <= 1:
       return n
   else:
       return n + recur_sum(n-1)
num = int(input("Enter a number: "))
if num < 0:
   print("Enter a positive number")
else:
   print("The sum is",recur_sum(num))
```

output:

```
Enter a number: 5
The sum is 15
```

Python program to display the Fibonacci sequence up to n-th term using recursive functions

```
def recur_fibo(n):
   if n <= 1:
      return n
   else:
      return(recur_fibo(n-1) + recur_fibo(n-2)
nterms=int(input("How many terms? "))
            if nterms <= 0:
            print("Plese enter a positive integer")
            else:
            print("Fibonacci sequence:")
            for i in range(nterms):
            print(recur_fibo(i))
```

Output:

0

1

1

2

3

5

8

13

21

34

Python Function Arguments

Python Default Arguments

Python Keyword Arguments

Python Arbitrary Arguments

Python Default Arguments

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

Example:

```
def greet(name, msg = "Good morning!"):
print("Hello",name + ', ' + msg)
greet("Kate")
greet("Bruce","How do you do?")
```

Output:

Hello Kate, Good morning!

Hello Bruce, How do you do?

Python Keyword Arguments

When we call a function with some values, these values get assigned to the arguments according to their position.

Example:

```
# Define plus() function
def plus(a,b):
  return a + b
  # Call plus() function with parameters
plus(2,3)
# Call plus() function with keyword arguments
plus(a=1, b=2)
```

Output:

3

Python 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 arbitrary number of arguments.

In the function definition we use an asterisk (*) before the parameter name to denote this kind of argument. Here is an example.

```
def greet(*names):
   """This function greets all
   the person in the names tuple."""
# names is a tuple with arguments
   for name in names:
    print("Hello",name)
greet("Monica","Luke","Steve","John")
```

Output:

Hello Monica

Hello Luke

Hello Steve

Hello John

Python Built In Functions

Lambda, Map, Filter, Reduce

Prepared by
Ramesh Yajjala
Assistant Professor(c)
Dept of CSE, IIIT-Srikakulam, RGUKT-AP