A function is a block of code which only runs when it is called. You can pass data , known as parameters , into a function. A function can return data as a result.

Creating a function:

In python a function is defined using the def keyword

Syntax:

def my\_function():

Print(“Hello santosh peddinti”)

my\_function()

Calling a function:

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

Syntax:

def my\_function():

print(“Hello from a function”)

my\_function()

Arguments:

Information can be passed into functions are arguments. Arguments are specified after the function name, inside the parentheses, you can add as many arguments as you want, just separate from with a comma.

Syntax:

def my\_function(first\_name):

print(first\_name+ “ Noted ”)

my\_function(“santosh”)

my\_function(“Aditya”)

my\_function(“Bhaskar”)

parameters or arguments:

the terms parameter and argument can be used for the same thing information that are passed into a function.

Syntax:

def my\_function(fname,lname):

print(fname+ “ “ + lname)

my\_function(“santoshkumar”,”peddinti”)

Arbitrary arguments, \*args

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

Syntax:

def my\_functions(\*items):

print(“The youngest child is “ + items[2])

my\_function(“Santosh”, “Aditya”, “bhaskar”)

keyword Arguments:

you can also send arguments with the key= value syntax this way the order of the arguments does not matter.

Syntax:

def my\_function(child1,child2,child3):

print(“The youngest child is “ + child3)

my\_function(child1=”santosh”, child2=”Aditya”, child3=”Bhaskar”)

Arbitrary keyword arguments , \*\* kwargs :

If you do not know how many keyword arguments that will be passed into you function, add two asterisk \*\* before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly

Syntax:

def my\_function(\*\*items):

print(“His last name is “ + items[“lname”])

my\_function(fname = “ santosh”, lname=”peddinti”)

Default parameter value:

The following example shows how to use a default parameter value. If we call the function without argument, it uses the default value:

def my\_function(country=”india”):

print(“ I am from “ + country)

my\_functions(“usa”)

my\_function(“ uk “)

my\_function()

my\_function(“Brazil”)

Passing a list as an argument:

You an send my data types of argument to a function (string, number, list ,dictionary et..), and it will be treated as the same data type inside the function.

E.g. if you send a list as an argument, it will still be a list when it reaches the function:

def my\_function(food):

for x in food:

print(x)

fruits= [ “apple”, “banana”, “cherry”]

my\_function(fruits)s

Return Values:

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

Syntax:

def my\_function(x):

return 5 \* x

print(my\_function(3))

print(my\_function(5))

print(my\_function(9))

Recursion:

Python also accepts function recursion, which means a defined function can call itself. Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result

In this example, tri\_recursion() is a function that we have defined to call itself (“recurse”) we use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

Syntax:

def my\_function\_recursion(data):  
 if(data > 0):  
 result=data+my\_function\_recursion(data-1)  
 print(result)  
 else:  
 result=0  
 return result  
  
print("Recursion Example results: ")  
  
my\_function\_recursion(5)