**UNIT-2 FUNCTIONS**

**PART-II**

**Functions and Modules:** Introduction, Defining and Calling a Void Function, Designing a Program to Use Functions, Local Variables, Passing Arguments to Functions, Global Variables and Global Constants, Value-Returning Functions-Generating Random Numbers, The math Module, Storing Functions in Modules.

**Introduction**:

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.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

**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 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 functionname( parameters ):

 "function\_docstring"

 function\_suite

 return [expression]

**Example**

The following function takes a string as input parameter and prints it on standard screen.

def printme( str ):

 "This prints a passed string into this function"

 print str

 return

**Calling a Function**

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function

**Example:**

# Function definition is here

def printme( str ):

 "This prints a passed string into this function"

 print str

 return;

# Now you can call printme function

printme("I'm first call to user defined function!")

printme("Again second call to the same function")

**output:**

I'm first call to user defined function!

Again second call to the same function

**Pass by reference vs value**

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

**For example −**

# Function definition is here

def changeme( mylist ):

 "This changes a passed list into this function"

 mylist.append([1,2,3,4]);

 print "Values inside the function: ", mylist

 return

# Now you can call changeme function

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result –

**output**

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

**Example2:**

# Function definition is here

def changeme( mylist ):

 "This changes a passed list into this function"

 mylist = [1,2,3,4]; # This would assig new reference in mylist

 print "Values inside the function: ", mylist

 return

# Now you can call changeme function

mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist

The parameter *mylist* is local to the function changeme. Changing mylist within the function does not affect *mylist*. The function accomplishes nothing and finally this would produce the following result −

Values inside the function: [1, 2, 3, 4]

Values outside the function: [10, 20, 30]

**Function Arguments**

You can call a function by using the following types of formal arguments −

* Required arguments
* Keyword arguments
* Default arguments
* Variable-length arguments

**Required arguments**

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

To call the function printme(), you definitely need to pass one argument, otherwise it gives a syntax error as follows −

[Live](http://tpcg.io/KIgYav)

# Function definition is here

def printme( str ):

 "This prints a passed string into this function"

 print str

 return;

# Now you can call printme function

printme()

When the above code is executed, it produces the following result –

Traceback (most recent call last):

 File "test.py", line 11, in <module>

 printme();

TypeError: printme() takes exactly 1 argument (0 given)

**Keyword arguments**

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the printme() function in the following ways −

# Function definition is here

def printme( str ):

 "This prints a passed string into this function"

 print str

 return;

# Now you can call printme function

printme( str = "My string")

When the above code is executed, it produces the following result –

My string

**Example2:**

# Function definition is here

def printinfo( name, age ):

 "This prints a passed info into this function"

 print "Name: ", name

 print "Age ", age

 return;

# Now you can call printinfo function

printinfo( age=50, name="miki" )

**Output:**

Name: miki

Age 50

**Default arguments**

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed −

# Function definition is here

def printinfo( name, age = 35 ):

 "This prints a passed info into this function"

 print "Name: ", name

 print "Age ", age

 return;

# Now you can call printinfo function

printinfo( age=50, name="miki" )

printinfo( name="miki" )

**Output:**

Name: miki

Age 50

Name: miki

Age 35

**Variable-length arguments**

You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

Syntax for a function with non-keyword variable arguments is this −

def functionname([formal\_args,] \*var\_args\_tuple ):

 "function\_docstring"

 function\_suite

 return [expression]

An asterisk (\*) is placed before the variable name that holds the values of all non keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Following is a simple example –

# Function definition is here

def printinfo( arg1, \*vartuple ):

 "This prints a variable passed arguments"

 print "Output is: "

 print arg1

 for var in vartuple:

 print var

 return;

# Now you can call printinfo function

printinfo( 10 )

printinfo( 70, 60, 50 )

When the above code is executed, it produces the following result

Output is:

10

Output is:

70

60

50

**The Anonymous Functions**

These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.

Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.

An anonymous function cannot be a direct call to print because lambda requires an expression

Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

Syntax

The syntax of lambda functions contains only a single statement, which is as follows −

lambda [arg1 [,arg2,.....argn]]:expression

Following is the example to show how lambda form of function works –

# Function definition is here

sum = lambda arg1, arg2: arg1 + arg2;

# Now you can call sum as a function

print "Value of total : ", sum( 10, 20 )

print "Value of total : ", sum( 20, 20 )

Value of total : 30

Value of total : 40

**Value-Returning Functions**

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.

You can return a value from a function as follows –

# Function definition is here

def sum( arg1, arg2 ):

 # Add both the parameters and return them."

 total = arg1 + arg2

 print "Inside the function : ", total

 return total;

# Now you can call sum function

total = sum( 10, 20 );

print "Outside the function : ", total

When the above code is executed, it produces the following result −

Inside the function : 30

Outside the function : 30

**Scope of Variables**

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

* Global variables
* Local variables

**Global vs. Local variables**

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example −

total = 0; # This is global variable.

# Function definition is here

def sum( arg1, arg2 ):

 # Add both the parameters and return them."

 total = arg1 + arg2; # Here total is local variable.

 print "Inside the function local total : ", total

 return total;

# Now you can call sum function

sum( 10, 20 );

print "Outside the function global total : ", total

When the above code is executed, it produces the following result −

Inside the function local total : 30

Outside the function global total : 0

Generating Random Numbers

Python defines a set of functions that are used to generate or manipulate random numbers through the**random module.** Functions in the random module rely on a pseudo-random number generator function **random()**, which generates a random float number between 0.0 and 1.0. These particular type of functions is used in a lot of games, lotteries, or any application requiring a random number generation

Random Number Operations

**1.**[**choice()**](https://www.geeksforgeeks.org/python-numbers-choice-function/):- choice() is an inbuilt function in the Python programming language that returns a random item from a list, tuple, or string.

**Example:**

# Python3 program to demonstrate the use of

# choice() method

# import random

import random

# prints a random value from the list

list1 = [1, 2, 3, 4, 5, 6]

print(random.choice(list1))

# prints a random item from the string

string = "striver"

print(random.choice(string))

**Output:**

5

t

**2.**[**randrange(beg, end, step)**](https://www.geeksforgeeks.org/randrange-in-python/)**:-** The random module offers a function that can generate random numbers from a specified range and also allowing rooms for steps to be included, called randrange().

**Example:**

# Python code to demonstrate the working of

# choice() and randrange()

# importing "random" for random operations

import random

# using choice() to generate a random number from a

# given list of numbers.

print("A random number from list is : ", end="")

print(random.choice([1, 4, 8, 10, 3]))

# using randrange() to generate in range from 20

# to 50. The last parameter 3 is step size to skip

# three numbers when selecting.

print("A random number from range is : ", end="")

print(random.randrange(20, 50, 3))

**Output:**

A random number from list is : 4

A random number from range is : 41

**3. random():-** This method is used to generate a float random number less than 1 and greater or equal to 0.

**4.**[**seed()**](https://www.geeksforgeeks.org/random-seed-in-python/)**:-** Seed function is used to save the state of a random function so that it can generate some random numbers on multiple executions of the code on the same machine or on different machines (for a specific seed value). The seed value is the previous value number generated by the generator. For the first time when there is no previous value, it uses current system time.

**5.**[**shuffle()**](https://www.geeksforgeeks.org/random-shuffle-function-in-python/)**:-** It is used to shuffle a sequence (list). Shuffling means changing the position of the elements of the sequence. Here, the shuffling operation is in place.

**Example:**

|  |
| --- |
| # import the random moduleimport random # declare a listsample\_list = ['A', 'B', 'C', 'D', 'E'] print("Original list : ")print(sample\_list) # first shufflerandom.shuffle(sample\_list)print("\nAfter the first shuffle : ")print(sample\_list) # second shufflerandom.shuffle(sample\_list)print("\nAfter the second shuffle : ")print(sample\_list) |

**Output:**

Original list :

['A', 'B', 'C', 'D', 'E']

After the first shuffle :

['A', 'B', 'E', 'C', 'D']

After the second shuffle :

['C', 'E', 'B', 'D', 'A']

**6.**[**uniform(a, b)**](https://www.geeksforgeeks.org/python-number-uniform-method/)**:-** This function is used to generate a floating point random number between the numbers mentioned in its arguments. It takes two arguments, lower limit(included in generation) and upper limit(not included in generation).

Example:

# Python code to demonstrate the working of

# shuffle() and uniform()

# importing "random" for random operations

import random

# Initializing list

li = [1, 4, 5, 10, 2]

# Printing list before shuffling

print("The list before shuffling is : ", end="")

for i in range(0, len(li)):

    print(li[i], end=" ")

print("\r")

# using shuffle() to shuffle the list

random.shuffle(li)

# Printing list after shuffling

print("The list after shuffling is : ", end="")

for i in range(0, len(li)):

    print(li[i], end=" ")

print("\r")

# using uniform() to generate random floating number in range

# prints number between 5 and 10

print("The random floating point number between 5 and 10 is : ", end="")

print(random.uniform(5, 10))

**Output:**

*The list before shuffling is : 1 4 5 10 2*

*The list after shuffling is : 2 1 4 5 10*

*The random floating point number between 5 and 10 is : 5.183697823553464*

### 7.  randint()

This function generates an integer between the specified limits. It takes two arguments x and y and produces integer i such that x <= i <= y.

>>> import random

>>> random.randint(3, 6)

Output:

5

### 8. sample()

If you want more than one random element from a sequence, you can use sample(). It takes two arguments population and k, where population is a sequence and k is an integer. Then it returns a list of k random elements from the sequence population.

>>> import random

>>> seq = (12, 33, 67, 55, 78, 90, 34, 67, 88)

>>> random.sample(seq, 5)

**Output:**

[33, 90, 78, 88, 12]

# Python Modules

A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.

### ****Example: create a simple module****

# A simple module, calc.py

def add(x, y):

    return (x+y)

def subtract(x, y):

    return (x-y)

# Import Module in Python –  Import statement

We can import the functions, classes defined in a module to another module using the [**import statement**](https://www.geeksforgeeks.org/import-module-python/) in some other Python source file.

**Syntax:**

**import module\_name**

When the interpreter encounters an import statement, it imports the module if the module is present in the searc

h path. A search path is a list of directories that the interpreter searches for importing a module. For example, to import the module calc.py, we need to put the following command at the top of the script.

**Note:**This does not import the functions or classes directly instead imports the module only. To access the functions inside the module the dot(.) operator is used.

### ****Example: Importing modules in Python****

|  |
| --- |
| # importing  module calc.pyimport calc print(calc.add(10, 2)) |

**Output:**

12

## ****The from importStatement****

Python’s *from*statement lets you import specific attributes from a module without importing the module as a whole.

### Example: Importing specific attributes from the module

# importing sqrt() and factorial from the module math

from math import sqrt, factorial

# if we simply do "import math", then math.sqrt(16) and math.factorial() are #required.

print(sqrt(16))

print(factorial(6))

**Output:**

4.0

720

## ****Import all Names – From import \*  Statement****

The \* symbol used with the from import statement is used to import all the names from a module to a current namespace.

**Syntax:**

**from module\_name import \***

The use of \* has its advantages and disadvantages. If you know exactly what you will be needing from the module, it is not recommended to use \*, else do so.

### Example: Importing all names

# importing sqrt() and factorial from the

# module math

from math import \*

# if we simply do "import math", then

# math.sqrt(16) and math.factorial()

# are required.

print(sqrt(16))

print(factorial(6))

**Output**

4.0

720

## Locating Modules

Whenever a module is imported in Python the interpreter looks for several locations. First, it will check for the built-in module, if not found then it looks for a list of directories defined in the [sys.path](https://www.geeksforgeeks.org/sys-path-in-python/). Python interpreter searches for the module in the following manner –

* First, it searches for the module in the current directory.
* If the module isn’t found in the current directory, Python then searches each directory in the shell variable [PYTHONPATH](https://www.geeksforgeeks.org/pythonpath-environment-variable-in-python/). The PYTHONPATH is an environment variable, consisting of a list of directories.
* If that also fails python checks the installation-dependent list of directories configured at the time Python is installed.

### ****Example: Directories List for Modules****

# importing sys module

import sys

# importing sys.path

print(sys.path)

**Output:**

*[‘/home/nikhil/Desktop/gfg’, ‘/usr/lib/python38.zip’, ‘/usr/lib/python3.8’, ‘/usr/lib/python3.8/lib-dynload’, ”, ‘/home/nikhil/.local/lib/python3.8/site-packages’, ‘/usr/local/lib/python3.8/dist-packages’, ‘/usr/lib/python3/dist-packages’, ‘/usr/local/lib/python3.8/dist-packages/IPython/extensions’, ‘/home/nikhil/.ipython’]*

## Importing and renaming module

We can rename the module while importing it using the as keyword.

### Example: Renaming the module

# importing sqrt() and factorial from the

# module math

import math as gfg

# if we simply do "import math", then math.sqrt(16) and math.factorial()

# are required.

print(gfg.sqrt(16))

print(gfg.factorial(6))

**Output**

4.0

720

**The dir() function**

The dir() built-in function returns a sorted list of strings containing the names defined by a module. The list contains the names of all the modules, variables, and functions that are defined in a module.

#  Import built-in module  random

import  random

print(dir(random))

**Example program**

# importing built-in module math

import math

# using square root(sqrt) function contained

# in math module

print(math.sqrt(25))

# using pi contained in math module

print(math.pi)

# 2 radians = 114.59 degreees

print(math.degrees(2))

# 60 degrees = 1.04 radians

print(math.radians(60))

# Sine of 2 radians

print(math.sin(2))

# Cosine of 0.5 radians

print(math.cos(0.5))

# Tangent of 0.23 radians

print(math.tan(0.23))

# 1 \* 2 \* 3 \* 4 = 24

print(math.factorial(4))

# importing built in module random

import random

# printing random integer between 0 and 5

print(random.randint(0, 5))

# print random floating point number between 0 and 1

print(random.random())

# random number between 0 and 100

print(random.random() \* 100)

List = [1, 4, True, 800, "python", 27, "hello"]

# using choice function in random module for choosing

# a random element from a set such as a list

print(random.choice(List))

# importing built in module datetime

import datetime

from datetime import date

import time

# Returns the number of seconds since the

# Unix Epoch, January 1st 1970

print(time.time())

# Converts a number of seconds to a date object

print(date.fromtimestamp(454554))

**Output:**

5.0

3.14159265359

114.591559026

1.0471975512

0.909297426826

0.87758256189

0.234143362351

24

3

0.401533172951

88.4917616788

True

1461425771.87

1970-01-06

 **Python math Module**

Python has a built-in module that you can use for mathematical tasks.

The math module has a set of methods and constants.

**Math Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| [math.acos()](https://www.w3schools.com/python/ref_math_acos.asp)  | Returns the arc cosine of a number |
| [math.acosh()](https://www.w3schools.com/python/ref_math_acosh.asp) | Returns the inverse hyperbolic cosine of a number |
| [math.asin()](https://www.w3schools.com/python/ref_math_asin.asp) | Returns the arc sine of a number |
| [math.asinh()](https://www.w3schools.com/python/ref_math_asinh.asp) | Returns the inverse hyperbolic sine of a number |
| [math.atan()](https://www.w3schools.com/python/ref_math_atan.asp) | Returns the arc tangent of a number in radians |
| [math.atan2()](https://www.w3schools.com/python/ref_math_atan2.asp) | Returns the arc tangent of y/x in radians |
| [math.atanh()](https://www.w3schools.com/python/ref_math_atanh.asp) | Returns the inverse hyperbolic tangent of a number |
| [math.ceil()](https://www.w3schools.com/python/ref_math_ceil.asp) | Rounds a number up to the nearest integer |
| [math.comb()](https://www.w3schools.com/python/ref_math_comb.asp) | Returns the number of ways to choose k items from n items without repetition and order |
| [math.copysign()](https://www.w3schools.com/python/ref_math_copysign.asp) | Returns a float consisting of the value of the first parameter and the sign of the second parameter |
| [math.cos()](https://www.w3schools.com/python/ref_math_cos.asp) | Returns the cosine of a number |
| [math.cosh()](https://www.w3schools.com/python/ref_math_cosh.asp) | Returns the hyperbolic cosine of a number |
| [math.degrees()](https://www.w3schools.com/python/ref_math_degrees.asp) | Converts an angle from radians to degrees |
| [math.dist()](https://www.w3schools.com/python/ref_math_dist.asp) | Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point |
| [math.erf()](https://www.w3schools.com/python/ref_math_erf.asp) | Returns the error function of a number |
| [math.erfc()](https://www.w3schools.com/python/ref_math_erfc.asp) | Returns the complementary error function of a number |
| [math.exp()](https://www.w3schools.com/python/ref_math_exp.asp) | Returns E raised to the power of x |
| [math.expm1()](https://www.w3schools.com/python/ref_math_expm1.asp) | Returns Ex - 1 |
| [math.fabs()](https://www.w3schools.com/python/ref_math_fabs.asp) | Returns the absolute value of a number |
| [math.factorial()](https://www.w3schools.com/python/ref_math_factorial.asp) | Returns the factorial of a number |
| [math.floor()](https://www.w3schools.com/python/ref_math_floor.asp) | Rounds a number down to the nearest integer |
| [math.fmod()](https://www.w3schools.com/python/ref_math_fmod.asp) | Returns the remainder of x/y |
| [math.frexp()](https://www.w3schools.com/python/ref_math_frexp.asp) | Returns the mantissa and the exponent, of a specified number |
| [math.fsum()](https://www.w3schools.com/python/ref_math_fsum.asp) | Returns the sum of all items in any iterable (tuples, arrays, lists, etc.) |
| [math.gamma()](https://www.w3schools.com/python/ref_math_gamma.asp) | Returns the gamma function at x |
| [math.gcd()](https://www.w3schools.com/python/ref_math_gcd.asp) | Returns the greatest common divisor of two integers |
| [math.hypot()](https://www.w3schools.com/python/ref_math_hypot.asp) | Returns the Euclidean norm |
| [math.isclose()](https://www.w3schools.com/python/ref_math_isclose.asp) | Checks whether two values are close to each other, or not |
| [math.isfinite()](https://www.w3schools.com/python/ref_math_isfinite.asp) | Checks whether a number is finite or not |
| [math.isinf()](https://www.w3schools.com/python/ref_math_isinf.asp) | Checks whether a number is infinite or not |
| [math.isnan()](https://www.w3schools.com/python/ref_math_isnan.asp) | Checks whether a value is NaN (not a number) or not |
| [math.isqrt()](https://www.w3schools.com/python/ref_math_isqrt.asp) | Rounds a square root number downwards to the nearest integer |
| [math.ldexp()](https://www.w3schools.com/python/ref_math_ldexp.asp) | Returns the inverse of [math.frexp()](https://www.w3schools.com/python/ref_math_frexp.asp) which is x \* (2\*\*i) of the given numbers x and i |
| [math.lgamma()](https://www.w3schools.com/python/ref_math_lgamma.asp) | Returns the log gamma value of x |
| [math.log()](https://www.w3schools.com/python/ref_math_log.asp) | Returns the natural logarithm of a number, or the logarithm of number to base |
| [math.log10()](https://www.w3schools.com/python/ref_math_log10.asp) | Returns the base-10 logarithm of x |
| [math.log1p()](https://www.w3schools.com/python/ref_math_log1p.asp) | Returns the natural logarithm of 1+x |
| [math.log2()](https://www.w3schools.com/python/ref_math_log2.asp) | Returns the base-2 logarithm of x |
| [math.perm()](https://www.w3schools.com/python/ref_math_perm.asp) | Returns the number of ways to choose k items from n items with order and without repetition |
| [math.pow()](https://www.w3schools.com/python/ref_math_pow.asp) | Returns the value of x to the power of y |
| [math.prod()](https://www.w3schools.com/python/ref_math_prod.asp) | Returns the product of all the elements in an iterable |
| [math.radians()](https://www.w3schools.com/python/ref_math_radians.asp) | Converts a degree value into radians |
| [math.remainder()](https://www.w3schools.com/python/ref_math_remainder.asp) | Returns the closest value that can make numerator completely divisible by the denominator |
| [math.sin()](https://www.w3schools.com/python/ref_math_sin.asp) | Returns the sine of a number |
| [math.sinh()](https://www.w3schools.com/python/ref_math_sinh.asp) | Returns the hyperbolic sine of a number |
| [math.sqrt()](https://www.w3schools.com/python/ref_math_sqrt.asp) | Returns the square root of a number |
| [math.tan()](https://www.w3schools.com/python/ref_math_tan.asp) | Returns the tangent of a number |
| [math.tanh()](https://www.w3schools.com/python/ref_math_tanh.asp) | Returns the hyperbolic tangent of a number |
| [math.trunc()](https://www.w3schools.com/python/ref_math_trunc.asp) | Returns the truncated integer parts of a number |

Math Constants

|  |  |
| --- | --- |
| **constant** | **Description** |
| [math.e](https://www.w3schools.com/python/ref_math_e.asp) | Returns Euler's number (2.7182...) |
| [math.inf](https://www.w3schools.com/python/ref_math_inf.asp) | Returns a floating-point positive infinity |
| [math.nan](https://www.w3schools.com/python/ref_math_nan.asp) | Returns a floating-point NaN (Not a Number) value |
| [math.pi](https://www.w3schools.com/python/ref_math_pi.asp) | Returns PI (3.1415...) |
| [math.tau](https://www.w3schools.com/python/ref_math_tau.asp) | Returns tau (6.2831...) |