PYTHON

UNIT 3

Contents

- Functions
- Built in functions
- Mathematical Functions
- Date time Functions
- Random Numbers
- Writing User defined Functions
- Composition of Functions
- Parameter and Arguments
- Default Parameters
- Function Calls
- Return Statements
- Using Global variables
- Recursion

Functions-Introduction

What is a function in Python?
A function in python is a group of statements within a program that performs a specific task.
Usually functions input data, process it, and "return" a result. Once a function is written, it can be used repeatedly.

- Functions are self contained programs that perform some particular tasks.

- Once the function is created by the programmer for a specific task, this function can be called anytime to perform that task.

• Types of Functions in Python

There are mainly two types of functions in Python
•**Built-in library function:** These are Standard functions in Python that are available to use.
•**User-defined function:** We can create our own functions based on our requirements.

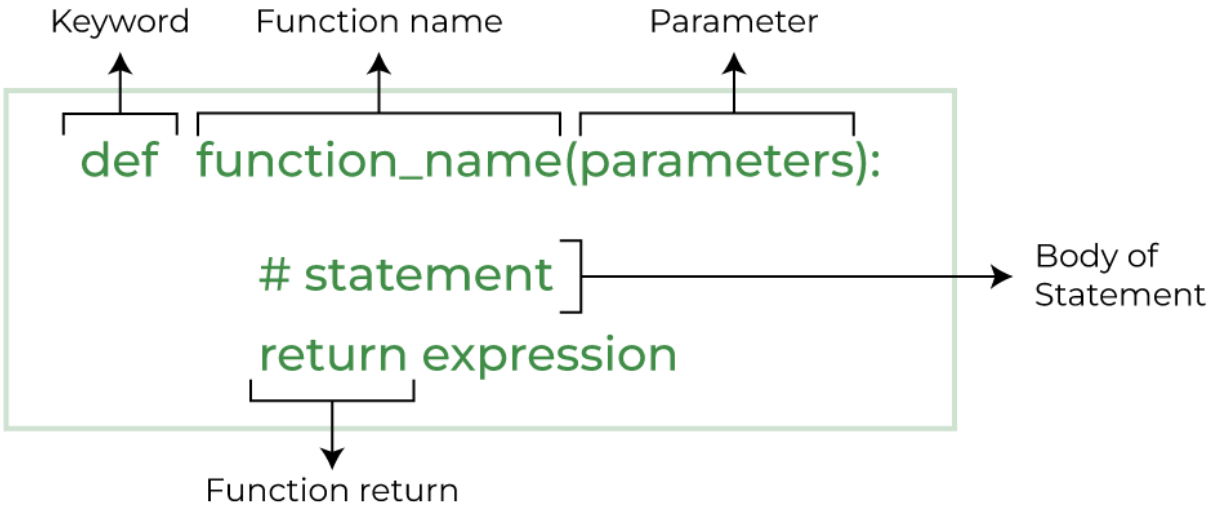
Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.
Some **Benefits of Using Functions**
•Increase Code Readability
•Increase Code Reusability

Advantages of Python Functions

•Once defined, Python functions can be called multiple times and from any location in a program.
•Our Python program can be broken up into numerous, easy-to-follow functions if it is significant.
•The ability to return as many outputs as we want using a variety of arguments is one of Python's most significant achievements.
•However, Python programs have always incurred overhead when calling functions.

SYNTAX:

```
#  An example Python Function
def function_name( parameters ):
   # code block
```



•function_name is the function's name, which we can use to distinguish it from other functions. We will utilize this name to call the capability later in the program. Name functions in Python must adhere to the same guidelines as naming variables.

•Using parameters, we provide the defined function with arguments. Notwithstanding, they are discretionary.

•A colon (:) marks the function header's end.

•We can utilize a documentation string called docstring in the short structure to make sense of the reason for the function.

•Several valid Python statements make up the function's body. The entire code block's indentation depth-typically four spaces-must be the same.

•A return expression can get a value from a defined function.

. Python Built in Functions

Built-in functions in Python are pre-defined functions provided by the Python language that can be used to perform common tasks.
To use a built-in function, simply call it with the appropriate arguments, like this: ***function_name(argument1, argument2).***
Python provides a lot of built-in functions that ease the writing of code.

Function Name	Description
Python abs() Function	Return the absolute value of a number
Python aiter() Function	It takes an asynchronous iterable as an argument and returns an asynchronous iterator for that iterable
Python all() Function	Return true if all the elements of a given iterable(List, Dictionary, Tuple, set, etc) are True else it returns False
Python any() Function	Returns true if any of the elements of a given iterable(List, Dictionary, Tuple, set, etc) are True else it returns False
Python anext() Function	used for getting the next item from an asynchronous iterator

Python ascii() Function	Returns a string containing a printable representation of an object
Python bin() Function	Convert integer to a binary string
Python bool() Function	Return or convert a value to a Boolean value i.e., True or False
Python bytearray() Function	Returns a byte array object which is an array of given bytes
Python bytes() Function	Converts an object to an immutable byte-represented object of a given size and data
Python callable() Function	Returns True if the object passed appears to be callable

Python chr() Function	Returns a string representing a character whose Unicode code point is an integer
Python classmethod() Function	Returns a class method for a given function
Python compile() Function	Returns a Python code object
Python complex() Function	Creates Complex Number
Python delattr() Function	Delete the named attribute from the object
Python dict() Function	Creates a Python Dictionary
Python dir() Function	Returns a list of the attributes and methods of any object
Python divmod() Function	Takes two numbers and returns a pair of numbers consisting of their quotient and remainder

Python enumerate() Function	Adds a counter to an iterable and returns it in a form of enumerating object
Python eval() Function	Parses the expression passed to it and runs Python expression(code) within the program
Python exec() Function	Used for the dynamic execution of the program
Python filter() Function	Filters the given sequence with the help of a function that tests each element in the sequence to be true or not
Python float() Function	Return a floating-point number from a number or a string
Python format() Function	Formats a specified value
Python getattr() Function	Access the attribute value of an object
Python hasattr() Function	Check if an object has the given named attribute and return true if present

Python hash() Function	Encode the data into an unrecognizable value
Python help() Function	Display the documentation of modules, functions, classes, keywords, etc
Python hex() Function	Convert an integer number into its corresponding hexadecimal form
Python id() Function	Return the identity of an object
Python input() Function	Take input from the user as a string
Python int() Function	Converts a number in a given base to decimal
Python isinstance() Function	Checks if the objects belong to a certain class or not

Python issubclass() Function	Check if a class is a subclass of another class or not
Python iter() Function	Convert an iterable to an iterator
Python len() Function	Returns the length of the object
Python list() Function	Creates a list in Python
Python locals() Function	Returns the dictionary of the current local symbol table
Python map() Function	Returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable
Python max() Function	Returns the largest item in an iterable or the largest of two or more arguments

Python min() Function	Returns the smallest item in an iterable or the smallest of two or more arguments
Python next() Function	Receives the next item from the iterator
Python object() Function	Returns a new object
Python oct() Function	returns an octal representation of an integer in a string format.
Python open() Function	Open a file and return its object
Python pow() Function	Compute the power of a number
Python print() Function	Print output to the console
Python property() Function	Create a property of a class
Python range() Function	Generate a sequence of numbers
Python repr() Function	Return the printable version of the object

Python reversed() Function	Returns an iterator that accesses the given sequence in the reverse order
Python round() Function	Rounds off to the given number of digits and returns the floating-point number
Python set() Function	Convert any of the iterable to a sequence of iterable elements with distinct elements
Python setattr() Function	Assign the object attribute its value
Python slice() Function	Returns a slice object
Python sorted() Function	Returns a list with the elements in a sorted manner, without modifying the original sequence
Python str() Function	Returns the string version of the object
Python sum() Function	Sums up the numbers in the list
Python super() Function	Returns a temporary object of the superclass
Python tuple() Function	Creates a tuple in Python
Python type() Function	Returns the type of the object
Python vars() Function	Returns the __dict__ attribute for a module, class, instance, or any other object
Python zip() Function	Maps the similar index of multiple containers
Python __import__() Function	Imports the module during runtime

Mathematical Functions

- Python provide a math module that contains most of the familiar and important mathematical functions.
- A module is a file that contains some predefined python codes.
- Before using a module in a python we have to import it.
- >>>import math

Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
ldexp(x, i)	Returns x * (2**i)

modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
expm1(x)	Returns e**x - 1
log(x[, b])	Returns the logarithm of x to the base b (defaults to e)
log1p(x)	Returns the natural logarithm of 1+x
log2(x)	Returns the base-2 logarithm of x
log10(x)	Returns the base-10 logarithm of x
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
acos(x)	Returns the arc cosine of x
asin(x)	Returns the arc sine of x
atan(x)	Returns the arc tangent of x
atan2(y, x)	Returns atan(y / x)

cos(x)	Returns the cosine of x
hypot(x, y)	Returns the Euclidean norm, sqrt(x*x + y*y)
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
degrees(x)	Converts angle x from radians to degrees
radians(x)	Converts angle x from degrees to radians
acosh(x)	Returns the inverse hyperbolic cosine of x
asinh(x)	Returns the inverse hyperbolic sine of x
atanh(x)	Returns the inverse hyperbolic tangent of x
cosh(x)	Returns the hyperbolic cosine of x
sinh(x)	Returns the hyperbolic cosine of x
tanh(x)	Returns the hyperbolic tangent of x
erf(x)	Returns the error function at x
erfc(x)	Returns the complementary error function at x
gamma(x)	Returns the Gamma function at x
lgamma(x)	Returns the natural logarithm of the absolute value of the Gamma function at x
pi	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...)
e	mathematical constant e (2.71828...)

DateTime Functions

Python Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times, and time intervals. Date and DateTime are an object in Python, so when you manipulate them, you are actually manipulating objects and not strings or timestamps.
The DateTime module is categorized into 6 main classes –
•**date** – An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month, and day.
•**time** – An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.
•**datetime** – Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.
•**timedelta** – A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.
•**tzinfo** – It provides time zone information objects.
•**timezone** – A class that implements the tzinfo abstract base class as a fixed offset from the UTC (New in version 3.2).

Python Date Class

The date class is used to instantiate date objects in Python. When an object of this class is instantiated, it represents a date in the format YYYY-MM-DD. The constructor of this class needs three mandatory arguments year, month, and date.

Python Date class Syntax:

```
class datetime.date(year, month, day)
```

The arguments must be in the following range –

- MINYEAR <= year <= MAXYEAR
- 1 <= month <= 12
- 1 <= day <= number of days in the given month and year

Note – If the argument is not an integer it will raise a TypeError and if it is outside the range a ValueError will be raised.

Eg:

```
# Python program to demonstrate date class
# import the date class
from datetime import date
my_date = date(1996, 12, 11)
print("Date passed as argument is", my_date)
```

Output:

```
Date passed as argument is 1996-12-11
```

Eg:

```
# Python program to print current date
from datetime import date
# calling the today  function of date class
today = date.today()
print("Today's date is", today)
```

Output

```
Today's date is 2023-10-9
```

Eg:

```
from datetime import datetime
# Getting Datetime from timestamp
date_time = datetime.fromtimestamp(1887639468)
print("Datetime from timestamp:", date_time)
```

Output

```
Datetime from timestamp: 2029-10-25 16:17:48
```

List of Date Class Methods

Function Name	Description
ctime()	Return a string representing the date
fromisocalendar()	Returns a date corresponding to the ISO calendar
fromisoformat()	Returns a date object from the string representation of the date
fromordinal()	Returns a date object from the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1
fromtimestamp()	Returns a date object from the POSIX timestamp
isocalendar()	Returns a tuple year, week, and weekday
isoformat()	Returns the string representation of the date
isoweekday()	Returns the day of the week as an integer where Monday is 1 and Sunday is 7
replace()	Changes the value of the date object with the given parameter
strftime()	Returns a string representation of the date with the given format
timetuple()	Returns an object of type time.struct_time
today()	Returns the current local date
toordinal()	Return the proleptic Gregorian ordinal of the date, where January 1 of year 1 has ordinal 1
weekday()	Returns the day of the week as integer where Monday is 0 and Sunday is 6

Python Time class

The time class creates the time object which represents local time, independent of any day.

Constructor Syntax:

class datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None, *, fold=0)

All the arguments are optional. tzinfo can be None otherwise all the attributes must be integer in the following range –

- 0 <= hour < 24
- 0 <= minute < 60
- 0 <= second < 60
- 0 <= microsecond < 1000000
- fold in [0, 1]

Eg:

```
# Python program to demonstrate time class
from datetime import time
# calling the constructor
my_time = time(13, 24, 56)
print("Entered time", my_time)
# calling constructor with 1  argument
my_time = time(minute=12)
print("\nTime with one argument", my_time)
# Calling constructor with  0 argument
my_time = time()
print("\nTime without argument", my_time)
```

Output:

```
Entered time 13:24:56
Time with one argument 00:12:00
Time without argument 00:00:00
```

Eg:

```
from datetime import time
Time = time(11, 34, 56)
print("hour =", Time.hour)
print("minute =", Time.minute)
print("second =", Time.second)
print("microsecond =", Time.microsecond)
```

Output:

```
hour = 11 minute = 34 second = 56 microsecond = 0
```

Eg:

```
from datetime import time
# Creating Time object
Time = time(12,24,36,1212)
# Converting Time object to string
Str = Time.isoformat()
print("String Representation:", Str)
print(type(Str))
```

Output

```
String Representation: 12:24:36.001212 <class 'str'>
```

List of Time class Methods

Function Name	Description
dst()	Returns tzinfo.dst() is tzinfo is not None
fromisoformat()	Returns a time object from the string representation of the time
isoformat()	Returns the string representation of time from the time object
replace()	Changes the value of the time object with the given parameter
strftime()	Returns a string representation of the time with the given format
tzname()	Returns tzinfo.tzname() is tzinfo is not None
utcoffset()	Returns tzinfo.utcffsets() is tzinfo is not None

Python Datetime class

The DateTime class contains information on both date and time. Like a date object, datetime assumes the current Gregorian calendar extended in both directions; like a time object, datetime assumes there are exactly 3600*24 seconds in every day.

Constructor Syntax:

class datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None, *, fold=0)

The year, month, and day arguments are mandatory. tzinfo can be None, rest all the attributes must be an integer in the following range –

- MINYEAR <= year <= MAXYEAR
- 1 <= month <= 12
- 1 <= day <= number of days in the given month and year
- 0 <= hour < 24
- 0 <= minute < 60
- 0 <= second < 60
- 0 <= microsecond < 1000000
- fold in [0, 1]

Note – Passing an argument other than integer will raise a TypeError and passing arguments outside the range will raise ValueError.

Eg:

```
# Python program to demonstrate datetime object
from datetime import datetime
# Initializing constructor
a = datetime(1999, 12, 12)
print(a)
# Initializing constructor with time parameters as well
a = datetime(1999, 12, 12, 12, 12, 12, 342380)
print(a)
```

Output:

```
1999-12-12 00:00:00
1999-12-12 12:12:12.342380
```

Eg:

```
from datetime import datetime
# Calling now() function
today = datetime.now()
print("Current date and time is", today)
```

Output:

```
Current date and time is 2019-10-25 11:12:11.289834
```

List of Datetime Class Methods

All the methods of date class and time class comes under the datetime class

combine(),ctime(),date(),fromisoformat(),fromordinal(),fromtimestamp(),isocalendar(),isoformat(),isoweekday(),
now(),replace(),strftime(),strptime(),time(),timetuple(),timetz(),today(),toordinal(),tzname(),utcfromtimestamp()
,offset(),utcnow(),weekday()...

Python Timedelta Class

Python timedelta class is used for calculating differences in dates and also can be used for date manipulations in Python. It is one of the easiest ways to perform date manipulations.

Constructor syntax:

class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

Returns : Date

Eg:

```
from datetime import datetime,timedelta
# Using current time
ini_time_for_now = datetime.now()
# printing initial_date
print("initial_date", str(ini_time_for_now))
# Calculating future dates for two years
future_date_after_2yrs = ini_time_for_now + timedelta(days=730)
future_date_after_2days = ini_time_for_now + timedelta(days=2)
# printing calculated future_dates
print('future_date_after_2yrs:', str(future_date_after_2yrs))
print('future_date_after_2days:', str(future_date_after_2days))
```

Output

```
initial_date 2019-10-25 12:01:01.227848
future_date_after_2yrs: 2021-10-24 12:01:01.227848
future_date_after_2days: 2019-10-27 12:01:01.227848
```

Eg:

```
# Timedelta function demonstration
from datetime import datetime, timedelta
# Using current time
ini_time_for_now = datetime.now()
# printing initial_date
print("initial_date", str(ini_time_for_now))
# Some another datetime
new_final_time = ini_time_for_now + \ timedelta(days=2)
# printing new final_date
print("new_final_time", str(new_final_time))
# printing calculated past_dates
print('Time difference:', str(new_final_time -ini_time_for_now))
```

Output:

```
initial_date 2019-10-25 12:02:32.799814
new_final_time 2019-10-27 12:02:32.799814
Time difference: 2 days, 0:00:00
```

Operations supported by Timedelta Class

Operator	Description
Addition (+)	Adds and returns two timedelta objects
Subtraction (-)	Subtracts and returns two timedelta objects
Multiplication (*)	Multiplies timedelta object with float or int
Division (/)	Divides the timedelta object with float or int
Floor division (//)	Divides the timedelta object with float or int and return the int of floor value of the output
Modulo (%)	Divides two timedelta object and returns the remainder
+(timedelta)	Returns the same timedelta object
-(timedelta)	Returns the resultant of -1*timedelta
abs(timedelta)	Returns the +(timedelta) if timedelta.days > 1=0 else returns -(timedelta)
str(timedelta)	Returns a string in the form (+/-) day[s], HH:MM:SS.UUUUUU
repr(timedelta)	Returns the string representation in the form of the constructor call

Python DateTime.tzinfo()

The datetime.now() function contains no information regarding time zones. It only makes use of the current system time. Tzinfo is an abstract base class in Python. It cannot be directly instantiated. A concrete subclass must derive from this abstract class and implement the methods offered by it.

List of Python DateTime.tzinfo() :

Function Name	Description
dst()	Returns tzinfo.dst() is tzinfo is not None
fromutc()	The purpose of this function is to adjust the date time data, returning an equivalent DateTime in self's local time.
tzname()	Returns tzinfo.tzname() is tzinfo is not None
utcoffset()	Returns tzinfo.utcffsets() is tzinfo is not None

Example

The tzinfo class instance can be provided to the DateTime and time object constructors. It is used in scenarios such as converting local time to UTC or accounting for daylight savings time.

```
import datetime as dt
from dateutil import tz
tz_string = dt.datetime.now(dt.timezone.utc).astimezone().tzname()
print("datetime.now() :", tz_string)
NYC = tz.gettz('Europe / Berlin')
dt1 = dt.datetime(2022, 5, 21, 12, 0)
dt2 = dt.datetime(2022, 12, 21, 12, 0, tzinfo=NYC)
print("Naive Object :", dt1.tzname())
print("Aware Object :", dt2.tzname())
```

Output:

```
datetime.now() : IST
Naive Object : None
Aware Object : None
```

Python DateTime timezone

Timezones in DateTime can be used in the case where one might want to display time according to the timezone of a specific region. This can be done using the pytz module of Python. This module serves the date-time conversion functionalities and helps users serving international client bases.

```
from datetime import datetime
from pytz import timezone
format = "%Y-%m-%d %H:%M:%S %Z%z"
# Current time in UTC
now_utc = datetime.now(timezone('UTC'))
print(now_utc.strftime(format))
timezones = ['Asia/Kolkata', 'Europe/Kiev', 'America/New_York']
for tzone in timezones:
        # Convert to Asia/Kolkata time zone
        now_asia = now_utc.astimezone(timezone(tzone))
        print(now_asia.strftime(format))
```

Output

```
2021-08-19 18:27:28 UTC+0000
2021-08-19 23:57:28 IST+0530
2021-08-19 21:27:28 EEST+0300
2021-08-19 14:27:28 EDT-0400
```

Random Number

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.

Example:

```
import random
num = random.random()
print(num)
```

Will produce a random number between 0 and 1.

Different Ways to Generate a Random Number in Python:

There are a number of ways to generate a random numbers in Python using the functions of the Python random module. Let us see a few different ways.

Generating a Random number using choice()

Python random.choice() is an inbuilt function in the Python programming language that returns a random item from a list, tuple, or string.

```
# 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
```

Generating a Random Number using randrange()

The random module offers a function that can generate Python random numbers from a specified range and also allows room for steps to be included, called 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
```

Generating a Random number using seed()

Python random.seed() function is used to save the state of a random function so that it can generate some random numbers in Python 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 the current system time.

Eg:

```
# importing "random" for random operations
import random
# using random() to generate a random number between 0 and 1
print("A random number between 0 and 1 is : ", end="")
print(random.random())
# using seed() to seed a random number
random.seed(5)
# printing mapped random number
print("The mapped random number with 5 is : ", end="")
print(random.random())
# using seed() to seed different random number
random.seed(7)
# printing mapped random number
print("The mapped random number with 7 is : ", end="")
print(random.random())
```

```
# using seed() to seed to 5 again
random.seed(5)
# printing mapped random number
print("The mapped random number with 5 is : ", end="")
print(random.random())
# using seed() to seed to 7 again
random.seed(7)
# printing mapped random number
print("The mapped random number with 7 is : ", end="")
print(random.random())
```

Output:

```
A random number between 0 and 1 is : 0.510721762520941
The mapped random number with 5 is : 0.6229016948897019
The mapped random number with 7 is : 0.32383276483316237
The mapped random number with 5 is : 0.6229016948897019
The mapped random number with 7 is : 0.32383276483316237
```

Generating a Random number using shuffle()

The shuffle() function 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.

```
# import the random module
import random
# declare a list
sample_list = ['A', 'B', 'C', 'D', 'E']
print("Original list : ")
print(sample_list)
# first shuffle
random.shuffle(sample_list)
print("\nAfter the first shuffle : ")
print(sample_list)
# second shuffle
random.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']
```

Generating a Random number using uniform()

The uniform() function is used to generate a floating point Python 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).

```
# 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
```