Built In Functions :

**Abs()** : The abs() function returns the absolute value of the specified number.

Syntax: abs(n)

Eg. X = abs(-3)

Print(x)

o/p: 3

Here n whether it is positive or negeative number the absolute value is always positive only.

**All() :** The all() function returns True if all items in an iterable are true, otherwise it returns False. If the iterable object is empty, the all() function also returns True.

Syntax: all(iterable)

Eg: list = [0,1,1] #elements in the list  
 x = all(list)

Print(list)

o/p : false

**Ascii() :** The  ascii() function returns a readable version of any object (Strings, Tuples, Lists, etc). The ascii() function will replace any non-ascii characters with escape characters i.e., å will be replaced with \xe5.

Syntax: ascii(object)

Eg . x = ascii("My name is Ståle")

print(x)

o/p : ‘My name is St\e5le’

**Bin():** The bin() function returns the binary version of a specified integer. The  result will always start with the prefix 0b.

Syntax: bin(n)

Eg. x = bin(36)

Print(x)

o/p : 0b100100

**bool() :** The bool() function returns the boolean value of a specified object. The object will always return True, unless:

The object is empty, like [], (), {}, The object is False, The object is 0, The object is None.

Syntax : bool(object) # obj = string,list,number

Eg, x = bool(9)

Print(x)

o/p: True

**bytearray():** The bytearray() function returns a bytearray object.

It can convert objects into bytearray objects, or create empty bytearray object of the specified size.

Syntax : bytearray(x, encoding, error)

Parameter in syntax :

X = A source to use when creating the bytearray object. If it is an integer an empty byte array of the object of the specified size will be created. If it is a string make sure to specify the encoding of the source.

Encoding = The encoding of the string.

Error specifies when the encoding fails

Eg., x = bytearray(4)

print(x)

o/p : bytearray(b’\x00\x00\x00\x00’)

**bytes():** The bytes() function returns a bytes object.

It can convert objects into bytes objects, or create empty bytes object of the specified size.

The difference between bytes() and bytearray() is that bytes() returns an object that cannot be modified, and bytearray() returns an object that can be modified.

Syntax: bytes(x, encoding, error)

Eg : x = bytes(4)

print(x)

o/p : b’\x00\x00\x00\x00’

**callable() :** The callable() function returns True if the specified object is callable, otherwise it returns False.

Syntax : callable(object)

Eg., def x():

A=5

Print(callable(x))

o/p : True

eg., x = 5 # a normal variable is not callable

print(callable(x))

o/p: False

**chr() :**  The chr() function returns the character that represents the specified unicode.

Syntax: chr(number)

Eg., # get the character that represents theUnicode of 97

x = chr(97)

print(x)

o/p: a

**Compile() :** The compile() function returns the specified source as a code object, ready to be executed.

Syntax : compile(source, filename, mode, flag, dont\_inherit, optimize)

Parametres:

Source = Required, The source to compile can be a string, a byte object.

Filename = file name came from source or else a name specified by user

Mode = eval, for single exp, exec for block of statements, single if source is single interactive element.

Flags = default 0

Don’t inherit = default False

Optimize = defines the optimization level of compiler , def = -1

Eg., x = compile('print(55)\nprint(88)', 'test', 'exec')

exec(x)

o/p : 55

88

Eg., #compile text as code and execute

x = compile('print(55)', 'test', 'eval')

exec(x)

o/p: 55

**delattr() :** The delattr() function will delete the specified attribute from the specified object.

Syntax : delattr(object, attribute)

Attribute = The name of the attribute you want to remove

Eg., delete the “age” property from the “person” object

class Person:

name = "John"

age = 36

country = "Norway"

delattr(Person, 'age')

# The Person object will no longer contain an "age" property

**Dict() :** The dict() function creates a dictionary. A dictionary is a collection which is unordered, changeable and indexed.

Syntax : dict(keyword arguments)

Eg., create a dictionary

x = dict(name = "John", age = 36, country = "Norway")

print(x)

o/p : { name = "John", age = 36, country = "Norway"}

dir() : The dir() function returns all properties and methods of the specified object, without the values. This function will return all the properties and methods, even built-in properties which are default for all object.

Syntax : dir(object)

Eg., class Person:

name = "John"

age = 36

country = "Norway"

print(dir(person))

**divmod() :** The divmod() function returns a tuple containing the quotient  and the remainder when argument1 (dividend) is divided by argument2 (divisor).

Syntax : divmod(dividend, divisor)

Eg., display the quotient and the remainder of 5 didvided by 2

x = divmod(5, 2)

print(x)

o/p: (2,1)

**enumerator:** The enumerate() function takes a collection (e.g. a tuple) and returns it as an enumerate object. The enumerate() function adds a counter as the key of the enumerate object.

Syntax : enumerate(iterable, start)

# start – a number, defining the start number of the enumarate object. Default=0.

Eg., convert a tuple into a enumerate object

x = ('apple', 'banana', 'cherry')

y = enumerate(x)

print(list(y))

o/p: [(0,’apple’), (1,’banana’), (2,’cherry’)]

**eval() :** The eval() function evaluates the specified expression, if the expression is a legal Python statement, it will be executed.

Syntax : eval(expression, globals, locals)

Evaluate the expression ‘print 55’

Eg., x = 'print(55)'

eval(x)

o/p : 55

exec() : The exec() function executes the specified Python code. The exec() function accepts large blocks of code, unlike the eval() function which only accepts a single expression

syntax : exec(object, globals, locals)

eg., x = 'name = "John"\nprint(name)'

exec(x)

o/p : John

filter() : The filter() function returns an iterator were the items are filtered through a function to test if the item is accepted or not.

Syntax : filter(function, iterable)

Eg.,

# Filter the array, and return a new array with only the values equal to or above 18:

ages = [5, 12, 17, 18, 24, 32]

def myFunc(x):

if x< 18:

return False

else:

return True

adults = filter(myfunc,ages)

for x in adults:

print(x)

**float() :** The float() function converts the specified value into a floating point number.

Syntax : float(value)

Eg., convert a string into floating point

x = float("3.500")

print(x)

o/p : 3.5

**format():** The format() function formats a specified value into a specified format.

Syntax : format(value, format)

Eg., convert a value into percentage value

x = format(0.5, '%')

print(x)

o/p : 50.0000%

**frozenset() :** The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable).

Syntax: frozenset(iterable)

Eg., freeze the list and make it as a immutable

mylist = ['apple', 'banana', 'cherry']

x = frozenset(mylist)

print(x)

o/p : frozenset({‘apple’, ‘cherrry’, ‘banana’})

**getattr() :** The getattr() function returns the value of the specified attribute from the specified object.

Syntax : getattr(object, attribute, default)

Eg., # Get the value of the age property of the person object

class Person:

name = "John"

age = 36

country = "Norway"

x = getattr(Person, 'age')

print(x)

o/p: 36

**globals() :** The globals() function returns the global symbol table as a dictionary. A symbol table contains necessary information about the current program

syntax : globals()

**hasattr():** The hasattr() function returns True if the specified object has the specified attribute, otherwise False.

Syntax: hasattr(object, attribute)

Eg., #check the age property

class Person:

name = "John"

age = 36

country = "Norway"

x = hasattr(Person, 'age')

print(x)

o/p: True

**hex():** The hex() function converts the specified number into a hexadecimal value. The returned string always starts with the prefix 0x.

syntax : hex(number)

eg., convert 255 into hexadecimal value

x = hex(255)

print(x)

o/p: 0xff

**id():**  The id() function returns a unique id for the specified object. All objects in Python has its own unique id. The id is assigned to the object when it is created. The id is the object's memory address, and will be different for each time you run the program. (except for some object that has a constant unique id, like integers from -5 to 256)

syntax : id(object)

eg., return a unique id of a tuple object

x = ('apple', 'banana', 'cherry')

y = id(x)

print(y)

o/p : 83540835

**input() :** The input() function allows user input.

Syntax: input(prompt)

Eg., print("Enter your name:")

x = input()

print("Hello, " + x)

Enter your name : sai

o/p : sai

**isinstance():** The isinstance() function returns True if the specified object is of the specified type, otherwise False. If the type parameter is a tuple, this function will return True if the object is one of the types in the tuple.

**Syntax :** isinstance(object, type)

Eg., check if the number 5 is an integer

x = isinstance(5, int)

print(x)

o/p : True

eg., check if “hello” is one of the types described..

x = isinstance("Hello", (str, float, int, str, list, dict,tuple))

print(x)

o/p: True

**issubclass():** The issubclass() function returns True if the specified object is a subclass of the specified object, otherwise False.

**Syntax :** issubclass(object, subclass)

Eg., check if the class myObj is a subclass of myAge

class myAge:

age = 36

class myObj(myAge):

name = "John"

age = myAge

x = issubclass(myObj, myAge)

print(x)

o/p: True

**iter():** The iter() function returns an iterator object.

Syntax : iter(object, sentinel)

Eg., create an iterate object, and print the terms

x = iter(["apple", "banana", "cherry"])

print(next(x))

print(next(x))

print(next(x))

o/p: apple

banana

cherry

**len() :** The len() function returns the number of items in an object. When the object is a string, the len() function returns the number of characters in the string.

Syntax : len(object)

Eg., mylist = ["apple", "orange", "cherry"]

x = len(mylist)

print(x)

o/p: 3

**list() :** The list() function creates a list object. A list object is a collection which is ordered and changeable.

Syntax : list(iterable)

Eg., create a list

x = list(('apple', 'banana', 'cherry'))

print(x)

o/p: [‘apple’, ‘banana’, ‘cherry’]

**locals():**  The locals() function returns the local symbol table as a dictionary. The locals() function returns the local symbol table as a dictionary.

Syntax : locals()

**Map():** The map() function executes a specified function for each item in an iterable. The item is sent to the function as a parameter.

Syntax : map(function, iterables)

Eg., Calculate the length of each word in the tuple

def myfunc(a):

return len(a)

x = map(myfunc, ('apple', 'banana', 'cherry'))

print(list(x))

**max():** The max() function returns the item with the highest value, or the item with the highest value in an iterable. If the values are strings, an alphabetically comparison is done.

Syntax : max(n1, n2, n3, ...)

(or)

max(iterable)

eg., return the largest value

x = max(5, 10)

print(x)

o/p:10

**min() :** The min() function returns the item with the lowest value, or the item with the lowest value in an iterable. If the values are strings, an alphabetically comparison is done.

Syntax : min(n1, n2, n3, ...)

(Or)

min(iterable)

eg., Return the item in a tuple with the lowest value

a = (1, 5, 3, 9)

x = min(a)

print(x)

o/p: 1

**next():** The next() function returns the next item in an iterator. You can add a default return value, to return if the iterable has reached to its end.

Syntax : next(iterable, default)

Eg., create an iterator and print the items onebyone

mylist = iter(["siva", "sai", "prashant"])

x = next(mylist)

print(x)

x = next(mylist)

print(x)

x = next(mylist)

print(x)

o/p; siva

sai

Prashant

**Object():**  The object() function returns an empty object. You cannot add new properties or methods to this object. This object is the base for all classes, it holds the built-in properties and methods which are default for all classes.

Syntax : object()

**Oct():** The oct() function converts an integer into an octal string. Octal strings in Python are prefixed with 0o.

Syntax : oct(int)

Eg., convert 12 into octal

x = oct(12)

print(x)

o/p: 0o14

**open():** The open() function opens a file, and returns it as a file object.

Syntax : open(file, mode)

Eg., open a file and print the content

f = open("demofile.txt", "r")

print(f.read())

o/p: Hello! Welcome to demofile.txt

**ord() :** The ord() function returns the number representing the unicode code of a specified character.

Syntax: ord(character)

Eg., Return the integer that represents the character “h”

x = ord("h")

print(x)

o/p: 104

**pow():** The pow() function returns the value of x to the power of y (xy). If a third parameter is present, it returns x to the power of y, modulus z.

syntax : pow(x, y, z)

eg., Return the value of 4 to the power of 3

x = pow(4, 3)

print(x)

o/p: 64

**print():**  The print() function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax: print(object(s), sep=separator, end=end, file=file, flush=flush)

Eg., print("Hello World")

o/p: Hello World

**range():** The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax: range(start, stop, step)

Eg., x = range(6)

for n in x:

print(n)

o/p: 0

1

2

3

4

5

**Reversed():** The reversed() function returns a reversed iterator object

Syntax: reversed(sequence)

Eg., Reverse the sequence

alph = ["a", "b", "c", "d"]

ralph = reversed(alph)

for x in ralph:

print(x)

o/p: d

c

b

a

**round():** The round() function returns a floating point number that is a rounded version of the specified number, with the specified number of decimals. The default number of decimals is 0, meaning that the function will return the nearest integer.

Syntax: round(number, digits)

Eg., round off to two decimals

x = round(5.76543, 2)

print(x)

o/p: 5.77

**set() :**  The set() function creates a set object. The items in a set list are unordered, so it will appear in random order.

Syntax : set(iterable)

Eg., Create a set

x = set(("apple", "banana", "cherry"))

print(x)

o/p: {‘banana’, ‘cherry’, ‘apple’}

**setattr():**  The setattr() function sets the value of the specified attribute of the specified object.

Syntax: setattr(object, attribute, value)

Eg., change the value of age of the person object

class Person:  
 name = "John"  
  age = 36  
  country = "Norway"

setattr(Person, 'age', 40)

x = getattr(Person, 'age')

print(x)

o/p: 40

**slice():** The slice() function returns a slice object. A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.

Syntax: slice(start, end, step)

Eg., create a tuple and slice a object.

a = ("a", "b", "c", "d", "e", "f", "g", "h")

x = slice(2)

print(a[x])

o/p: (‘a’, ‘b’)

**sorted():** The sorted() function returns a sorted list of the specified iterable object. You can specify ascending or descending order. Strings are sorted alphabetically, and numbers are sorted numerically.

Syntax: sorted(iterable, key=key, reverse=reverse)

Eg., sort a tuple

a = ("b", "g", "a", "d", "f", "c", "h", "e")

x = sorted(a)

print(x)

o/p: [‘a’, ‘b’, ‘c’, ‘d’, ‘e’ , ‘f’, ‘g’, ‘h’]

**str():** The str() function converts the specified value into a string.

Syntax: str(object, encoding=encoding, errors=errors)

Eg., Convert a string into integer

x = int("12")

print(x)

o/p: 12

**sum():** The sum() function returns a number, the sum of all items in an iterable.

Syntax: sum(iterable, start)

Eg., add all elements in a tuple and return the result

a = (1, 2, 3, 4, 5)

x = sum(a)

print(x)

o/p: 15

**super():** The super() function is used to give access to methods and properties of a parent or sibling class. The super() function returns an object that represents the parent class.

Syntax: super()

Eg., create a class that inherit all the methods and the properties from another class

class Parent:

def \_\_init\_\_(self, txt):

self.message = txt

def printmessage(self):

print(self.message)

class Child(Parent):

def \_\_init\_\_(self, txt):

super().\_\_init\_\_(txt)

x = Child("Hello, and welcome!")

x.printmessage()

o/p: Hello, and welcome!

**Tuple():** The tuple() function creates a tuple object.

Syntax: tuple(iterable)

Eg., create a tuple

x = tuple(("ant", "bat", "cat"))

print(x)

(‘ant’, ‘bat’, ‘cat’)

**Type():**  The type() function returns the type of the specified object

Syntax: type(object, bases, dict)

Eg., Return the types of objects

a = ('apple', 'banana', 'cherry')

b = "Hello World"

c = 33

x = type(a)

y = type(b)

z = type(c)

print(x)

print(y)

print(z)

o/p: <class ‘tuple’>

<class ‘str’>

<class ‘int’>

**Vars():** The vars() function returns the \_\_dic\_\_ attribute of an object. The \_\_dict\_\_ attribute is a dictionary containing the object's changeable attributes.

Syntax: vars(object)

**Zip():**  The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.

Syntax: zip(iterator1, iterator2, iterator3 ...)

Eg., join two tuples together

a = ("John", "Charles", "Mike")

b = ("Jenny", "Christy", "Monica")

x = zip(a, b)

print(tuple(x))

o/p: ((‘John’, ‘Jenny’), (‘Charles’, ‘christy’), (‘mike’, ‘monica’))

1. What is list comprehension and lambda function?
2. List Comprehension:

It is to generate a list form existing list

Ex:

lst1=[0,1,2,3,4,5,6,7,8,9]

list\_comp = [x \* 2 for x in lst1]

print(list\_comp)

Output: [0,2,4,6,8,10,12,14,16,18]

Lambda Function:

It is name less function all so.once control will go to the next line memory allocated to the perticular function removed.

Ex:

lst1=[0,1,2,3,4,5,6,7,8,9]

lambda\_list=list(map(lambda x:x\*2,lst1))

print(lambda\_list)

Output: [0,2,4,6,8,10,12,14,16,18]

1. difference between list comprehesion and lambda function?
2. List comprehesion is an create a list in existing list.We can creates list just like mathematical statements and in a single line only

Lambda function: In python lambda is name less function.lambda key word is used to create unnous function

1. Difference between LC and TC?
2. list comprehesion is to generate a list from existing list

Tuple comprehesion is to generate a tuple.

1. Difference between generator and decorator?
2. Generator:

Python Generators are a simple way of creating iterators.the way of implementing these iterators are called Generators

Ex:

def ten():

yield 1

yield 2

yield 3

yield 4

values = ten()

Output:print(values.\_\_next\_\_()) :1

print(values.\_\_next\_\_()) :2

Decorators:

Decorator is a design pattern in python that allows a users add new functionality to an existing object with out modifying its structure

Ex:

#decorators

def outer(func):

def inner(\*args,\*\*kwargs):

#Add functionality before calling func

print("\*" \*50)

func(\*args,\*\*kwargs)

#Add functionality after calling func

print("#"\*50)

return inner

@outer

def fun(data):

print(data)

Output:fun(data="I am in banglour")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I am in banglour

1. Difference between generator and normal function
2. 1.Generator function contains one or more Yield statements.
3. when called,it returns an object(iterator)but does not start execution immediately
4. Methods like \_\_iter\_\_() and \_\_next\_\_() are implemented automatically.so we can

Iterate through the items using next().

1. Once the functionality yield,the function is paused and the control is transferred to

The caller

1. Finally, when the function terminates, StopIteration is raised automatically on

Future calls.