In [ ]:
```
#Agenda of Today:
                        1. Functions in Python
                        2. Strings in Python
                        3. Problem Solving on them.

            Covered Topics: (Day 1 to Day 4)
                1 . Python Basics
                2.  Data types
                3.  Variables
                4.  Keywords
                5.  Operators
                5.  Conditonal Statements
                6.  Iterations
```

In [ ]:
```
#Functions in Python:
           # what is function?
     A function is a  group of related statements or instructions that perform a spcific task
        or
        Block of code
    #why we use functions?
    1. Code Reusablity
    2. Do Complex Tasks
    3. Reduce Time Consuming
    4. Easy to Debug

#Types of Functions:
  1. Built-in functions (Developed by Developers)
  2. User -Defined Functions (Developed by Users or Programmers)
  3. Recursion Functions. (A function itself called)
```

In [1]:
```
#Built-Functions:
#1. print():
print("Hello every one This is Surya From Apssdc")
```

```
Hello every one This is Surya From Apssdc
```

In [5]:
```
import keyword
print(keyword.kwlist,end=" ")
print("\n")
print(len(keyword.kwlist))
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'de
l', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonl
ocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

35
```

In [8]:
```
#TO Check python version in jupiter notebook
import sys
print(sys.version)
```

```
3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
```

In [10]:
```
from platform import python_version
print(python_version())
```

```
3.7.3
```

In [11]:
```
#len() : to find the length of any sequnce
s = "apssdc"
len(s)
```

Out[11]:
```
6
```

In [18]:
```
#abs(): Its returns the absolute value of the given number.
print(abs(-6))
print(abs(100))
print(abs(-5000.55))
```

```
6
100
5000.55
```

In [25]:
```
#all(): Its returns True when all elements in the given iterable are true.
# syntax: all(iterable)
li = [1,2,4,5]
print(all(li))
print(all((0,1,0)))
print((all((1,1,1))))
print(all((0,0,0)))
print(all(()))
```

```
True
False
True
False
True
```

In [31]:
```
#ascii(): it returns a string containing printabble representattion of an object
s1 = " Python is super"
print(s1)
print(ascii(s1))
```

```
 Python is super
' Python is super'
```

In [35]:
```
#ord(): it returns the ascii vales of given characters
print(ord("A"))
print(ord("a"))
print(ord("Z"))
print(ord("z"))
```

```
65
97
90
122
```

In [40]:
```
#chr()
print(chr(97))
print(chr(69))
print(chr(111))
print(chr(100))
```

```
a
E
o
d
```

In [44]:
```
#bin(): its method converts and returns the binary equivalent of a given integer
print(bin(2))
print(bin(10))
print(bin(555533))
```

```
0b10
0b1010
0b10000111101000001101
```

In [48]:
```
#bool():
#syntax: bool[value]
print(bool([1]))
print(bool((0)))
print(bool([5.5]))
print(bool(("Easy string")))
```

```
True
False
True
True
```

In [50]:
```
#dir(): To view the all directories in a given sequnce:
print(dir(list),end=" ")
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__for
mat__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init_
_', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce
__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__
str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

In [54]:
```
#enumerate() function: its add counter to an iterable and returns it
s = "apssdc"
for ch in s:
    print(ch,end="")
#syntax:
enumerate(iterable,start=0)
```

```
apssdc
```

In [60]:
```
li = [1,2,4,5,6,7,8,9]
print(li)
print(list(enumerate(li)))
```

```
[1, 2, 4, 5, 6, 7, 8, 9]
[(0, 1), (1, 2), (2, 4), (3, 5), (4, 6), (5, 7), (6, 8), (7, 9)]
```

In [67]:
```
#eval(): method parses the expression and returns the values
x = 10
print(eval('x+1+x**2'))
print(eval('x^2+x**3+4*x'))
```

```
111
1048
```

In [ ]:
```
#filter():
#syntax:
#filter(function, iterable)
alpha = ["a","b","e","i","s","u","w"]
```

In [1]:
```
#help():
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
```

```
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
|  copy(self, /)
|      Return a shallow copy of the list.
```

```
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  extend(self, iterable, /)
|      Extend list by appending elements from the iterable.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  insert(self, index, object, /)
|      Insert object before index.
|
|  pop(self, index=-1, /)
|      Remove and return item at index (default last).
|
|      Raises IndexError if list is empty or index is out of range.
|
|  remove(self, value, /)
|      Remove first occurrence of value.
|
|      Raises ValueError if the value is not present.
|
|  reverse(self, /)
|      Reverse *IN PLACE*.
|
|  sort(self, /, *, key=None, reverse=False)
|      Stable sort *IN PLACE*.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

In [20]:
```
#id(): its returns the identify(unique intger) of an object
li = [2,4,6,7]
print(id([7]))
print(id(6))
print(id(2))
print(max(li))
print(min(li))
print(sum(li))
print(pow(2,5))
s1 = "apssdc"
print(list(reversed(s1)))
li = [2,3,5,6,7,8]
print(list(reversed(li)))
print(type(s1))
print(type(li))
```

```
2869373935496
140717777327072
140717777326944
7
2
19
32
['c', 'd', 's', 's', 'p', 'a']
[8, 7, 6, 5, 3, 2]
<class 'str'>
<class 'list'>
```

In [ ]:
```
#User - Defined functions:
#       for declaring the functions in python we must use def keyword:
def function_name(parameters):   #(formal parameters) : its indicates indention
    """this is demo function"""     #doc string
    block of code
    return                          #its returns the value from function
function_name(parameters)   #function calling  (actual parameters)

#Note:
we dont use the keywords as function names
```

In [25]:
```
#example:
def displayname(name):     #name - formal parameter
    """this function to display names given by user"""
    print("Hello," + name + " Good Evening to All")
#displayname("surya")     #surya - actual parameter  - static value
displayname(input("Enter your Name"))     #- Dynamice parameter passing at runtime
```

```
Enter your Namepython
Hello,python Good Evening to All
```

In [31]:
```
#function attribues:
print(displayname.__doc__)
print(displayname.__name__)
print(dir(displayname),end=" ")
print("\n")
print(dir(abs),end= " ")
```

```
this function to display names given by user
displayname
['__annotations__', '__call__', '__class__', '__closure__', '__code__', '__defaults__', '__delattr__', '__dic
t__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__globals__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__', '__module__', '_
_name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__s
izeof__', '__str__', '__subclasshook__']

['__call__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattrib
ute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__name__',
'__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__self__', '__setattr__', '_
_sizeof__', '__str__', '__subclasshook__', '__text_signature__']
```

In [37]:
```
#ex: function with out parameters:
def hello():
    return "Hello Students"
    #print("Hi...Buddiees")
hello()
```

Out[37]: 'Hello Students'

In [46]:
```
#Function with Parameters:
def add_ofv(x,y):
    return int(x+y)                          #type casting
x =int(input("enter x value"))
y =float(input("enter y value"))
add_ofv(x,y)
```

```
enter x value50
enter y value50.6
```

Out[46]:
```
100
```

In [51]:
```
#function with multiple parameters:
def add_of3(a,b,c):
    a = b+c
    b = c+a
    c = a+b
    return (a,b,c)
add_of3(50,70,100)
```

Out[51]:
```
(170, 270, 440)
```

In [ ]:
```
#return statement:
#syntax:
  return [expression_list] - returns a value
  return           - returns NONE object
```

In [53]:
```
#exa:
def AB():
    print("Hello I am from AB()")
    return 5
def CD():
    return 100 * AB()
print(AB())
print(CD())
```

```
Hello I am from AB()
5
Hello I am from AB()
500
```

In [2]:
```
#Recursion:(a function itself called is called as recursive function )
def factorial(x):
    """This function finds the factorial of a given number"""
    if x == 1:
        return 1
    else:
        return (x *factorial(x-1))      #recursive function call

num = int(input("enter value of num"))
print("the factorial of",num,"is",factorial(num))
```

```
enter value of num10
the factorial of 10 is 3628800
```

In [ ]:
```
#types of arguments:
1. default arguments (these values are fixed at function defintion)
2. keyword arguments ( these values are  fixed at fuction calling)
3. arbitary arguments ( one argument name can handle multiple values of arguments)
```

In [9]:
```
#default:
def display(name=" students ",msg="please Come to Office"):
    print("Hello"+name+","+msg)
display()
```

```
Hello students ,please Come to Office
```

In [11]:
```
#keyword:
def display(name,msg):
    print("hello"+name+","+msg)

display(msg="please complete tasks",name=" Mr.student")
```

```
hello Mr.student,please complete tasks
```

In [12]:
```
#arbitary: (it can be done by * astrick symbol)
def indianteam(*names):
    return names
indianteam("dhoni","Kohli","Rohit","pandey","dhavan")
```

Out[12]:
```
('dhoni', 'Kohli', 'Rohit', 'pandey', 'dhavan')
```

In [ ]: