1. Python Output

In [6]:
```
# Python is a case sensitive language.
print('Hello World')
print(7)
print(2.5)
print(True)
print('Hello', 1, 4.5, True)
```

```
Hello World
7
2.5
True
Hello 1 4.5 True
World,2,False
```

In [7]:
```
# Separating the output in commas
print('World', 2, False, sep=',')
```

```
World,2,False
```

In [10]:
```
# Removing the line brek after each line
print('Hello ', end='')
print('Debashis')
```

```
Hello Debashis
```

2. Data Types

In [16]:
```
# 1. Integer
print(8)
print(1e2)

# 2. Decimal/Float
print(5.6)
print(5.6e2)

# 3. Boolean
print(True)
print(False)

# 4. String
print('Hi')

# 5. Complex
print(5+2j)

# 6. List
print([1,2,3,4,5])

# 7. Tuple
```

```
print((1,2,3,4,5))

# 8. Sets
print({1,2,3,4,5})

# 9. Dictionary
print({
    'name':'Debashis',
    'gender':'Male',
    'weight':70
})

# 10. Type
print(type([1,2,3]))
```

```
8
100.0
5.6
560.0
True
False
Hi
(5+2j)
[1, 2, 3, 4, 5]
(1, 2, 3, 4, 5)
{1, 2, 3, 4, 5}
{'name': 'Debashis', 'gender': 'Male', 'weight': 70}
<class 'list'>
```

3. Variables

In [17]:
```
# Python is a dynamically typed language
# dynamically-typed languages perform type checking at runtime, while
# statically typed languages perform type checking at compile time.
name = 'Debashis'
```

In [18]:
```
# Python follows dynamic binding rules
a = 5
print(a)
a = 'Debashis'
print(a)
```

```
5
Debashis
```

In [19]:
```
# Various ways to declare a varibale
# 1.
a = 1
b = 2
print(a,b)

# 2.
c,d = 3,4
print(c,d)
```

```
# 3.
x = y = z = 5
print(x,y,z)
```

```
1 2
3 4
5 5 5
```

4. Keywords & Identifiers

In [20]:
```
# List of all python reserved keywords
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

In [22]:
```
# Identifiers
# Rules:
# 1. You can't start with a digit
# 2. You cannot use any special characters other than '_'
# 3. Identifiers cannot be keywords
name1 = 'Debashis'
_ = 'Giri'
print(name1, _)
```

```
Debashis Giri
```

5. User Input

In [23]:
```
# By default input() returns a string
num = input("Enter a number")
print(type(num))
```

```
Enter a number5
<class 'str'>
```

In [24]:
```
# WAP to take 2 nums as input and ADD the nums
num1 = int(input('Enter 1st Num'))
num2 = int(input('Enter 2nd Num'))
print(num1 + num2)
```

```
Enter 1st Num5
Enter 2nd Num4
9
```

6. Type Conversions

In [ ]:
```
# Implicit -> Automatically performed by the Python interpreter
# Explicit -> Users convert the data type of an object to required data type
```

In [26]:
```
# Implicit
print(5 + 5.6)
```

```
10.6
```

In [33]:
```
# Explicit
# 1. str()
num = 4
print(type(num))
num = str(4)
print(type(num))

# 2. int()
print(int('4'))

# 3. float()
print(float(4))
```

```
<class 'int'>
<class 'str'>
4
4.0
```

7. Literals

In [36]:
```
# 1. Binary literals
a = 0b1010

# 2. Decimal literals
b = 100

# 3. Float literals
c = 10.5

# 4. Octal literals
d = 0o310

# 5. Hexadecimal literals
e = 0x12c

# 6. Complex literals
f = 3.14j
```

```
print(a,b,c,d,e,f)
print(f.imag, f.real)
```

```
10 100 10.5 200 300 3.14j
3.14 0.0
```

In [37]:
```
# 7. String literals
str1 = 'Debashis'
str2 = "Debashis"
str3 = "D"
str4 = """
This is a
multiline string
"""
str5 = u"\U0001f600\U0001F606\U0001F923" #Unicode
str6 = r"\n is treated as normal text"

print(str1,str2,str3,str4,str5,str6)
```

```
Debashis Debashis D
This is a
multiline string
 😀😆🤣 \n is treated as normal text
```

In [38]:
```
# In python True is treated as 1 & False is treated as 0
a = True + 5
b = False + 5
print(a,b)
```

```
6 5
```

In [39]:
```
# We CANNOT declare a variable in python.
# To do so we have to initialize the variable with 'None'
k = None
print(k)
```

```
None
```

1. Operators

In [10]:
```
# 1. Arithmetic Operators
print('5+6 = ',5+6)
print('5-6 = ',5-6)
print('5*6 = ',5*6)
print('5/6 = ',5/6)
print('5//6 = ',5//6)
print('5%6 = ',5%6)
print('5**6 = ',5**6)

# 2. Relational Operators
print('4>5 = ',4>5)
print('4<5 = ',4<5)
print('4>=4 = ',4>=4)
print('4<=4 = ',4<=4)
print('4==4 = ',4==4)
print('4!=4 = ',4!=4)

# 3. Logical Operators
print('1 and 0 = ', 1 and 0)
print('1 or 0 = ', 1 and 0)
print('not 1 = ', not 1)

# 4. Bitwise Operators
print('2&3 = ', 2&3)
print('2|3 = ', 2|3)
print('2^3 = ', 2^3)
print('~3 = ', ~3)
print('4>>2 = ', 4>>2)
print('5<<2 = ', 5<<2)

# 5. Assignment Operators
a = 2
a += 3 # a = a + 3
print('a = ',a)

# 6. Membership Operators
print('D' in 'Delhi')
print('E' not in 'Sky')
```

```
5+6 =  11
5-6 =  -1
5*6 =  30
5/6 =  0.8333333333333334
5//6 =  0
5%6 =  5
5**6 =  15625
4>5 =  False
4<5 =  True
4>=4 =  True
4<=4 =  True
4==4 =  True
4!=4 =  False
1 and 0 =  0
1 or 0 =  0
not 1 =  False
2&3 =  2
2|3 =  3
2^3 =  1
~3 =  -4
4>>2 =  1
5<<2 =  20
a =  5
True
True
```

2. If-else

In [12]:
```
num = int(input('Enter number'))
if num > 0:
    print('Positive Number')
elif num < 0:
    print('Negative Number')
else:
    print('Zero')
```

```
Enter number0
Zero
```

3. Modules

In [17]:
```
# 1. math module
import math
print('Square root of 196 = ', math.sqrt(196))

# 2. keyword module
import keyword
print('keywords', keyword.kwlist)

# 3. random module
import random
print('A random number = ', random.randint(1,50))
```

```
# 4. datetime
import datetime
print('Current date & time = ', datetime.datetime.now())

# 5. help('modules')
```

```
Square root of 196 =  14.0
keywords ['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'brea
k', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
A random number =  4
Current date & time =  2022-12-08 19:19:16.400426
```

4. Loops

In [19]:
```
# 1. While loop
num = int(input('Enter number: '))
i = 1
while i < 11:
    print(num, '*', i, '=',num*i)
    i+=1
```

```
Enter number: 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

In [20]:
```
# 1.1. While loop with else
x = 1
while x < 3:
    print(x)
    x += 1
else:
    print('Limit crossed')
```

```
1
2
Limit crossed
```

In [23]:
```
# 3. For loop
# for loop in list
for i in [1,2,3]:
    print(i)
```

```
1
2
3
```

In [24]:
```python
# 3.1 for loop in range()
for i in range(1,6,2):
    print(i)
```

```
1
3
5
```

1. Nested loops

In [2]:
```
for i in range(1,3):
    for j in range(1,3):
        print(i,j)
```

```
1 1
1 2
2 1
2 2
```

2. Loop Control Statement

In [3]:
```
# 1. Break
for i in range(1,10):
    if(i==5):
        break
    print(i)
```

```
1
2
3
4
```

In [4]:
```
# 2. Continue
for i in range(1,5):
    if i == 3:
        continue
    print(i)
```

```
1
2
4
```

In [5]:
```
# 3. Pass
for i in range(1,10):
    pass
```

3. Strings

In [ ]:
```
# Strings are sequence of characters
# Strings are immutable
```

In [6]:
```
# 1. Creating strings
s = 'Hello'
s = "World"
s = '''Multiline1'''
s = """Multiline2"""
```

```
s = str('Hello world')
print(s)
```

```
Hello world
```

In [8]:
```
# 2. Accessing strings
# 2.1. Positive Indexing
s = "Hello World"
print(s[1])

# 2.2. Negative Indexing
print(s[-1])

# 2.3 Slicing
print(s[1:3])
print(s[6:0:-1])
print(s[::-1])
print(s[-1:-6:-1])
```

```
e
d
el
W olle
dlroW olleH
dlroW
```

In [9]:
```
# 3. Editing and deleting in strings
s = 'Hello World'
# s[0] = M => Editing is not possible in strings
del s
print(s)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 5
      3 # s[0] = M => Operation cannot be performed
      4 del s
----> 5 print(s)

NameError: name 's' is not defined
```

In [11]:
```
# 4. Operation on strings
# 4.1 Arithmetic Operators
print('Hello ' + 'Debashis')
print('Hello ' * 5)

# 4.2 Relational Operators
print('delhi' != 'delhi')
print('mumbai' > 'pune') # Checks lexicographically

# 4.3 Logical Operators
print('Hello' and 'world')
print('hello' or 'world')
print('' and 'world')
print('' or 'world')
print(not 'hello')
```

```
# 4.4 Membership Operators
print('D' in 'Delhi')

# 4.5 Looping in strings
for i in 'Delhi':
    print(i)
```

```
Hello Debashis
Hello Hello Hello Hello Hello
False
False
world
hello

world
False
True
D
e
l
h
i
```

In [34]:

```
# 5. String Functions
# 5.1 Common functions
print('Length: ', len('Hello'))
print('Max: ', max('Hello'))
print('Min: ', min('Hello'))
print('Sorted: ', sorted('hello', reverse=True))

# 5.2 Capitalize/Title/Upper/Lower/Swapcase
s = 'hello world'
print('Capitalize: ', s.capitalize())
print('Title: ', s.title())
print('Upper: ', s.upper())
print('Lower: ', s.lower())
print('Swapcase: ', s.swapcase())

# 5.3 Count/Find/Index
s = 'Hello Debashis'
print(s.count('s'))
print(s.find('e'))
print(s.index('e'))
# find() returns -1 if value not found whereas
# index() throws error

# 5.4 endswith/startswith
s = 'Hello world'
print(s.endswith('ld'))
print(s.startswith('j'))

# 5.5 format
fname = 'Debashis'
mname = 'Debadutta'
lname = 'Giri'
print('My name is {} {} {}'.format(fname,mname,lname))
```

```
# 5.6 isalnum/isalpha/isdigit/isidentifier
print('Debashis34'.isalnum())
print('Debashis'.isalpha())
print('123'.isdigit())
print('print'.isidentifier())

# 5.7 split/join
print('My name is Debashis'.split())
print(' '.join(['Hello', 'Debashis']))

# 5.8 Replace
print('Hi my name is Gudu'.replace('Gudu','Debashis'))

# 5.9 Strip
print('Debashis      '.strip())
```

```
Length:  5
Max:  o
Min:  H
Sorted:  ['o', 'l', 'l', 'h', 'e']
Capitalize:  Hello world
Title:  Hello World
Upper:  HELLO WORLD
Lower:  hello world
Swapcase:  HELLO WORLD
2
1
1
True
False
My name is Debashis Debadutta Giri
True
True
True
True
['My', 'name', 'is', 'Debashis']
Hello Debashis
Hi my name is Debashis
Debashis
```

1. Lists

In [ ]:
```
# Q1. What are lists?
# List is a data type where you can store multiple items under 1 name.
# More technically, lists act like dynamic arrays which means
# you can add more items on the fly.

# Q2. Lists vs Arrays
# Arrays are of FIXED size whereas lists are DYNAMIC
# Array are HOMOGENEOUS whereas lists are HETEROGENEOUS
# Arrays are FASTER than lists
# Arrays takes LESS memory than lists

# Q3. How elements are stored in lists?
# List contains the memory address of individual items.
# Memory is not contiguous for list items

# Q4. Characterstics of a list
# 1. Ordered
# 2. Mutable
# 3. Heterogeneous
# 4. Can have duplicates
# 5. Are dynamic
# 6. Can be nested
# 7. Items can be accessed
# 8. Can contain any kind of objects in python

# Q5. Disadvantages of python list
# 1. Slow
# 2. Risky usage
# 3. Eats up more memory
```

In [1]:
```
li = [1,2,3]
print(id(li))
print(id(1))
print(id(li[0]))
```

```
1357606196992
140726500909864
140726500909864
```

In [3]:
```
# 1.1 Creating a list
l1 = []
l2 = [1,2,3,4,5]
l3 = [1,2,[3,4,5]]
l4 = [[1,2],[3,4]]
l5 = [1, True, 5.6, 'Hello']
l6 = list('Hello')
print(l1, l2, l3, l4, l5, l6, sep='\n')
```

```
[]
[1, 2, 3, 4, 5]
[1, 2, [3, 4, 5]]
[[1, 2], [3, 4]]
[1, True, 5.6, 'Hello']
['H', 'e', 'l', 'l', 'o']
```

In [5]:
```
# 1.2 Accessing items
li = [[1,2],[3,4]]
print(li[0][1])

# Slicing
li2 = [1,2,3,4,5,6]
print(li2[::-1])
```

```
2
[6, 5, 4, 3, 2, 1]
```

In [15]:
```
# 1.3 Adding items
# Append - append add at the end
li1 = [1,2,3,4]
li1.append([5,6])
print(li)

# Extend - it adds everything as list item
li2 = [1,2,3,4]
li2.extend([5,6])
print(li2)

# Insert - insert at a specific index value
li3 = [1,2,3,4]
li3.insert(1, 5)
print(li3)
```

```
[1, 2, 3, 4, True, [5, 6]]
[1, 2, 3, 4, 5, 6]
[1, 5, 2, 3, 4]
```

In [16]:
```
# 1.4 Editing items
li = [1,2,3,4]
li[-1] = 5
print(li)
li[1:3] = [200,300]
print(li)
```

```
[1, 2, 3, 5]
[1, 200, 300, 5]
```

In [20]:
```
# 1.5 Deleting items
# Del
li1 = [1,2,3,4]
del li[2:]
print(li)

# Remove
li2 = [1,2,3,4]
li2.remove(3)
```

```
print(li2)

# Pop
li3 = [1,2,3,4]
li3.pop()
print(li3)

# Clear
li4 = [1,2,3,4]
li4.clear()
print(li4)
```

```
[1, 2]
[1, 2, 4]
[1, 2, 3]
[]
```

In [26]:

```
# 1.6 Operations
# Arithmetic Operators
li1 = [1,2,3,4]
li2 = [5,6,7,8]
print(li + li2)
print(li1 * 2)

# Membership Operator
li3 = [1,2,3,4]
print(2 in li3)
print(3 not in li3)

# Looping
li4 = [1,2,3,4]
for i in li4:
    print(i, end=" ")
```

```
[1, 2, 5, 6, 7, 8]
[1, 2, 3, 4, 1, 2, 3, 4]
True
False
1 2 3 4
```

In [30]:

```
# 1.7 List functions
# len/min/max/sorted
li1 = [2,1,5,7,0]
print('Len: ', len(li1))
print('Min: ', min(li1))
print('Max: ', max(li1))
print('Sorted: ', sorted(li1,reverse=True))

# Count
li2 = [1,2,3,4,5]
print('Count: ', li.count(li2))

# Index
li3 = [1,2,3,4,5]
print('Index: ', li3.index(4))

# Reverse
```

```
li4 = [2,1,5,7,0]
li4.reverse()
print(li4)

# Sort
li5 = [2,1,5,7,0]
li5.sort()
print(li5)

# Copy
li6 = [2,1,5,7,0]
li7 = li6.copy()
print(li7)
print(id(li6), id(li7))
```

```
Len:  5
Min:  0
Max:  7
Sorted:  [7, 5, 2, 1, 0]
Count:  0
Index:  3
[0, 7, 5, 1, 2]
[0, 1, 2, 5, 7]
[2, 1, 5, 7, 0]
1357622013312 1357622012992
[0, 1, 2, 5, 7] [2, 1, 5, 7, 0]
```

In [41]:

```
# 1.8 List Comprehension
# List Comprehension provides a concise way of creating lists.
# Advantages of List Comprehension
# 1. More time-efficient and space-efficient than loops.
# 2. Require fewer lines of code.
# 3. Transforms iterative statement into a formula.

# Traditional method
li1 = []
for i in range(1,4):
    li1.append(i)
print(li1)

# List comprehension method
li2 = [i for i in range(1,4)]
print(li2)

# Q. Scalar Multiplication on a vector
vector = [2,3,4]
scalar = 3
print([scalar * i for i in vector])

# Q. Add squares
li3 = [1,2,3,4]
print([i**2 for i in li3])

# Q. Print all numbers divisible by 5 in the range of 1 to 50
print([i for i in range(1,51) if i%5 == 0])
```

```
# Q. Find languages which start with letter p
languages = ['java','python','php','c','javascript']
print([language for language in languages if language.startswith('p')])

# Q. Print a (3,3) matrix using list comprehension
print([[i*j for i in range(1,4)] for j in range(1,4)])

# Q. cartesian products -> List comprehension on 2 lists together
li4 = [1,2,3,4]
li5 = [5,6,7,8]
print([i*j for i in li4 for j in li5])
```

```
[1, 2, 3]
[1, 2, 3]
[6, 9, 12]
[1, 4, 9, 16]
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
['python', 'php']
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
[5, 6, 7, 8, 10, 12, 14, 16, 15, 18, 21, 24, 20, 24, 28, 32]
```

In [44]:
```
# 1.9 Traversing
# Itemwise
li1 = [1,2,3,4]
for i in li1:
    print(i,end=" ")

print()
# Indexwise
li2 = [1,2,3,4]
for j in range(0, len(li2)):
    print(li2[j], end=" ")
```

```
1 2 3 4
1 2 3 4
```

In [49]:
```
# 1.10 Zip
li1 = [1,2,3,4]
li2 = [-1,-2,-3,-4]
print(list(zip(li1,li2)))
print([i+j for i,j in zip(li1,li2)])
```

```
[(1, -1), (2, -2), (3, -3), (4, -4)]
[0, 0, 0, 0]
```

1. Tuples

In [ ]:
```
# A tuple in Python is similar to a list.
# Tuples are immutable

# Characteristics
# 1. Ordered
# 2. Immutable
# 3. Allows duplicates

# Difference between Lists and Tuples
# 1. Syntax
# 2. Mutablity - List is mutable whereas tuples are not
# 3. Speed - Tuples are FASTER than lists
# 4. Memory - Tuples takes LESS memory than lists
# 5. Built in functionality
# 6. Error Prone - Tuples are LESS prone to errors
# 7. Usability
```

In [2]:
```
# 1.1 Creating a tuple
# Empty
t1 = ()

# Tuple with only 1 element
t2 = ('Hello',)

# Homogeneous
t3 = (1,2,3,4)

# Heterogeneous
t4 = (1, True, 5.6)

# Nested tuple
t5 = (1,2,(3,4))

# Using type conversion
t6 = (tuple('Hello'))

print(t1,t2,t3,t4,t5,t6, sep="\n")
```

```
()
('Hello',)
(1, 2, 3, 4)
(1, True, 5.6)
(1, 2, (3, 4))
('H', 'e', 'l', 'l', 'o')
```

In [3]:
```
# 1.2 Accessing Items
# Indexing
t1 = (1,2,3,4)
print(t1[2])
print(t1[-1])
```

```
# Slicing
t2 = (1,2,3,4)
print(t2[1:3])
```

```
3
4
(2, 3)
```

In [ ]:
```
# 1.3 Adding items & Deleting Items
# Tuples are not mutable so adding items is not possible
```

In [5]:
```
# 1.4 Operations
# Arithmetic Operators
t1 = (1,2,3,4)
t2 = (5,6,7,8)
print(t1 + t2)
print(t1 * 2)

# Membership Operators
t3 = (1,2,3,4)
print(2 in t3)

# Looping
t4 = (1,2,3,4)
for i in t4:
    print(i, end=" ")
```

```
(1, 2, 3, 4, 5, 6, 7, 8)
(1, 2, 3, 4, 1, 2, 3, 4)
True
1 2 3 4
```

In [9]:
```
# 1.5 Tuple functions
# len/sum/min/max/sorted
t1 = (1,2,3,4)
print('Len: ', len(t1))
print('Sum: ', sum(t1))
print('Min: ', min(t1))
print('Max: ', max(t1))
print('Sorted: ', sorted(t1, reverse=True))

# Count
t2 = (1,2,3,4,5)
print('Count: ', t2.count(2))

# Index
t3 = (1,2,3,4,5)
print('Index: ', t3.index(3))
```

```
Len:  4
Sum:  10
Min:  1
Max:  4
Sorted:  [4, 3, 2, 1]
Count:  1
Index:  2
```

In [11]:
```
# 1.6 Copy tuple
t1 = (1,2,3)
t2 = t1
t1 = t1 + (4,)
print(t1,t2)
```

```
(1, 2, 3, 4) (1, 2, 3)
```

In [15]:
```
# 1.7 Tuple Unpacking
t1,t2,t3 = (1,2,3)
print(t1,t2,t3)

t4, t5, *others = (1,2,3,4,5)
print(t4,t5,others)
```

```
1 2 3
1 2 [3, 4, 5]
```

In [16]:
```
# Q. Swap 2 numbers
t1 = 1
t2 = 2
t1,t2 = t2,t1
print(t1,t2)
```

```
2 1
```

In [17]:
```
# 1.8 Zipping Tuples
t1 = (1,2,3,4)
t2 = (5,6,7,8)
print(tuple(zip(t1,t2)))
```

```
((1, 5), (2, 6), (3, 7), (4, 8))
```

2. Sets

In [ ]:
```
# A set is an unordered collection of items.
# Every set element is unique (no duplicates)
# Set is immutable (cannot be changed).
# However, a set itself is mutable.
# We can add or remove items from it.
# Sets can also be used to perform mathematical set operations.
# Ex: Union, intersection, symmetric difference
# Characterstics
# 1. Unordered
# 2. Mutable
# 3. No Duplicates
# 4. Can't contain mutable data types
```

In [3]:
```
# 2.1. Creating Sets
# Empty Set
s1 = set()

# 1d & 2d Set
s2 = {1,2,3}
s3 = {1,2,(3,4)}
```

```
# Homogeneous
s4 = {1,2,3,4}
s5 = {1,2,'Hello',4.5,(1,2,3)}

# Using type conversion
s6 = set([1,2,3])

# Duplicates NOT allowed
s7 = {1,2,2,3,1}

print(s1,s2,s3,s4,s5,s6,s7,sep="\n")
```

```
set()
{1, 2, 3}
{1, 2, (3, 4)}
{1, 2, 3, 4}
{1, 2, 4.5, 'Hello', (1, 2, 3)}
{1, 2, 3}
{1, 2, 3}
```

In [5]:
```
# 2.2. Accessing items
# Accessing is not posible becoz sets are unordered
```

In [4]:
```
# 2.3. Editing items
# Editing is not posible becoz set items are immutable
```

In [8]:
```
# 2.4. Adding items
# Add
s1 = {1,2,3,4,5}
s1.add(5)
s1.add((6,7))
print(s1)

# Update
s2 = {1,2,3,4}
s2.update([5,6,7])
print(s2)
```

```
{1, 2, 3, 4, 5, (6, 7)}
{1, 2, 3, 4, 5, 6, 7}
```

In [19]:
```
# 2.3. Deleting items
# Del
s1 = {1,2,3,4}
# del s1[0]
# 'set' object doesn't support item deletion

# Discard
s2 = {1,2,3,4}
s2.discard(4)
print('Discard: ', s2)

# Remove
s3 = {1,2,3,4}
s3.remove(3)
```

```
print('Remove: ', s3)
# Discard don't throws an error if item not found

# Pop
s4 = {1,2,3,4}
s4.pop()
print('Pop: ', s4)

# Clear
s5 = {1,2,3,4}
s5.clear()
print('Clear: ', s5)
```

```
Discard:  {1, 2, 3}
Remove:  {1, 2, 4}
Pop:  {2, 3, 4}
Clear:  set()
```

In [22]:
```
# 2.4. Set Opeartions
s1 = {1,2,3,4,5}
s2 = {5,6,7,8,9}

# Union
print('Union:', s1 | s2)

# Intersection
print('Intersection:', s1 & s2)

# Difference
print('Difference:', s1 - s2)

# Symmetric Difference
print('Symmetric Diff:', s1 ^ s2)

# Membership test
print('Membership test:', 1 not in s2)

# Looping
for i in s1:
    print(i, end=" ")
```

```
Union: {1, 2, 3, 4, 5, 6, 7, 8, 9}
Intersection: {5}
Difference: {1, 2, 3, 4}
Symmetric Diff: {1, 2, 3, 4, 6, 7, 8, 9}
Membership test: True
1 2 3 4 5
```

In [13]:
```
# 2.5 Set Functions
# len/sum/min/max/sorted
s1 = {3,1,4,5,2,7}
print('Length:', len(s1))
print('Sum:', sum(s1))
print('Min:', min(s1))
print('Max:', max(s1))
print('Sorted:', sorted(s1, reverse=True))
```

```
# Union/Update
# Union returns a new set whereas update modifies the existing set
s2 = {1,2,3,4,5}
s3 = {4,5,6,7,8}
print('Union:', s2.union(s3))
s2.update(s3)
print('Update:', s2)

# Intersection/Intersection_update
s4 = {1,2,3,4,5}
s5 = {4,5,6,7,8}
print('Intersection:', s4.intersection(s5))
s4.intersection_update(s5)
print('Intersection_update:', s4)

# Difference/Difference_update
s6 = {1,2,3,4,5}
s7 = {4,5,6,7,8}
print('Difference:', s6.difference(s7))
s6.difference_update(s7)
print('Difference_update:', s6)

# Symmetric_difference/Symmetric_difference_update
s8 = {1,2,3,4,5}
s9 = {4,5,6,7,8}
print('Symmetric_diff:', s8.symmetric_difference(s9))
s8.symmetric_difference_update(s9)
print('Symmetric_diff_update:', s8)

# isdisjoint/issubset/issuperset
s10 = {1,2,3,4}
s11 = {7,8,5,6}
print('isDisjoint:', s10.isdisjoint(s11))
print('isSuperset:', s10.issuperset(s11))

# Copy
s12 = {1,2,3}
s13 = s12.copy()
print('s12:', s12)
print('s13:', s13)
```

```
Length: 6
Sum: 22
Min: 1
Max: 7
Sorted: [7, 5, 4, 3, 2, 1]
Union: {1, 2, 3, 4, 5, 6, 7, 8}
Update: {1, 2, 3, 4, 5, 6, 7, 8}
Intersection: {4, 5}
Intersection_update: {4, 5}
Difference: {1, 2, 3}
Difference_update: {1, 2, 3}
Symmetric_diff: {1, 2, 3, 6, 7, 8}
Symmetric_diff_update: {1, 2, 3, 6, 7, 8}
isDisjoint: True
isSuperset: False
s12: {1, 2, 3}
s13: {1, 2, 3}
```

In [16]:
```
# 2.6 Frozenset
# Frozen set is just an immutable version of a Python set object.
# All read functions works with frozenset
# All write function don't work
# Used in 2D sets
fs1 = frozenset([1,2,3])
fs2 = frozenset([3,4,5])
fs3 = frozenset([1,2,frozenset([3,4])])
print('Union:', fs1 | fs2)
print('Nested frozen set:', fs3)
```

```
Union: frozenset({1, 2, 3, 4, 5})
Nested frozen set: frozenset({frozenset({3, 4}), 1, 2})
```

In [17]:
```
# 2.7 Set Comprehension
print({i**2 for i in range(1,11) if i>5})
```

```
{64, 36, 100, 49, 81}
```

2. Dictionary

In [ ]:
```
# Dictionary in Python is a collection of keys values,
# used to store data values like a map, which,
# unlike other data types which hold only a single value as an element.
# In some languages it is known as map or assosiative arrays.
# Characterstics:
# 1. Mutable
# 2. Indexing has no meaning
# 3. Keys can't be duplicated
# 4. Keys can't be mutable items
```

In [29]:
```
# 2.1 Create Dictionary
# Empty dictionary
d1 = {}

# 1D dictionary
d2 = {
```

```
    'name':'Debashis',
    'gender':'male',
    (1,2,3):1
}

# 2D dictionary
d3 = {
    'name':'Debashis',
    'subjects':{
        'dsa':50,
        'maths':56
    }
}

# Using sequence and dict function
d4 = dict([('name','Debashis'),('age',23)])

# Duplicate keys
# later one will be considered
d5 = {
    'name':'Debashis',
    'name':'Rahul'
}

# Immutable items as keys
d6 = {
    'name':'Debashis',
    (1,2,3):2
}

print(d1,d2,d3,d4,d5,d6,sep='\n')
```

```
{}
{'name': 'Debashis', 'gender': 'male', (1, 2, 3): 1}
{'name': 'Debashis', 'subjects': {'dsa': 50, 'maths': 56}}
{'name': 'Debashis', 'age': 23}
{'name': 'Rahul'}
{'name': 'Debashis', (1, 2, 3): 2}
```

In [34]:

```
# 2.2 Accessing items
d1 = {
    'name':'Debashis',
    'age':23
}
# using []
print('[]:', d1['name'])

# using get()
print('get:', d1.get('name'))
```

```
[]: Debashis
get: Debashis
```

In [47]:

```
# 2.3 Adding key-value pair
d1 = {
    'name':'Debashis',
    'age':23,
```

```
    'subjects':{
        'dsa':21
    }
}
d1['subjects']['math'] = 24
d1['gender'] = 'male'
print(d1)
```

```
{'name': 'Debashis', 'age': 23, 'subjects': {'dsa': 21, 'math': 24}, 'gender': 'ma
le'}
```

In [43]:
```
# 2.4 Removing key-value pair
d1 = {
    'name': 'nitish',
    'age': 32,
    'gender': 'male',
    'weight': 72
}
# Pop
d1.pop('weight')
print(d1)

# Popitem
d1.popitem()
print(d1)

# Del
del d1['age']
print(d1)

# Clear
d1.clear()
print(d1)
```

```
{'name': 'nitish', 'age': 32, 'gender': 'male'}
{'name': 'nitish', 'age': 32}
{'name': 'nitish'}
{}
```

In [48]:
```
# 2.5 Editing key-value pair
d1 = {
    'name': 'nitish',
    'age': 32,
    'gender': 'male',
    'weight': 72
}
d1['age'] = 23
print(d1)
```

```
{'name': 'nitish', 'age': 23, 'gender': 'male', 'weight': 72}
```

In [51]:
```
# 2.6 Dictionary Opeartions
d1 = {
    'name': 'nitish',
    'age': 32,
    'gender': 'male',
    'weight': 72
```

```
}
# Membership
print('name' in d1)

# Iteration
for i in d1:
    print(i, d1[i])
```

```
True
name nitish
age 32
gender male
weight 72
```

In [54]:

```
# 2.7 Dictionary Functions
d1 = {
    'name': 'nitish',
    'age': 32,
    'gender': 'male',
    'weight': 72
}
# len/sorted
print('Length:', len(d1))
print('Sorted:', sorted(d1,reverse=True))

# items/keys/values
print('Items:', d1.items())
print('Keys:', d1.keys())
print('Values:', d1.values())

# Update
d2 = {'color':'rice-brown'}
d1.update(d2)
print('Update:', d1)
```

```
Length: 4
Sorted: ['weight', 'name', 'gender', 'age']
Items: dict_items([('name', 'nitish'), ('age', 32), ('gender', 'male'), ('weight',
72)])
Keys: dict_keys(['name', 'age', 'gender', 'weight'])
Values: dict_values(['nitish', 32, 'male', 72])
Update: {'name': 'nitish', 'age': 32, 'gender': 'male', 'weight': 72, 'color': 'ri
ce-brown'}
```

In [62]:

```
# 2.8 Dictionary Comprehension
# Q. Print 1st 10 nums and their squares
print({i:i**2 for i in range(1,5)})

# Q. Multiply distances by 0.62
distances = {'delhi':1000,'mumbai':2000,'bangalore':3000}
print({i:j*0.62 for (i,j) in distances.items()})

# Using zip()
days = ["Sunday", "Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
temp_C = [30.5,32.6,31.8,33.4,29.8,30.2,29.9]

print({i:j for (i,j) in zip(days,temp_C)})
```

```
# Using if condition
products = {'phone':10,'laptop':0,'charger':32,'tablet':0}
print({key:value for (key,value) in products.items() if value>0})

# Nested Comprehension
# Q. Print tables of number from 2 to 3
print({i:{j:i*j for j in range(1,11)} for i in range(2,4)})
```

```
{1: 1, 2: 4, 3: 9, 4: 16}
{'delhi': 620.0, 'mumbai': 1240.0, 'bangalore': 1860.0}
{'Sunday': 30.5, 'Monday': 32.6, 'Tuesday': 31.8, 'Wednesday': 33.4, 'Thursday': 2
9.8, 'Friday': 30.2, 'Saturday': 29.9}
{'phone': 10, 'charger': 32}
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18, 10: 20}, 3: {1: 3,
2: 6, 3: 9, 4: 12, 5: 15, 6: 18, 7: 21, 8: 24, 9: 27, 10: 30}}
```

1. Functions

In [ ]:
```
# Function is a block of code which give o/p when we provide some i/p
# There are two types of functions:
# 1. Buit-in functions
# 2. User defined functions
# Two principle use in function:
# 1. Abstraction
# 2. Decomposition
# Function by default return none without return statement.
# Benefits of using function:
# 1. Code Modularity
# 2. Code Readability
# 3. Code Reusability
```

In [7]:
```
# 1.1 Function syntax
# Function defination
def is_even(i):
    '''
    Optional docstring which tells us about the function.
    This function return true if the given num is even.
    i/p - any valid integer
    o/p - true/false

    '''
    return i % 2 == 0

# Function call
print(is_even(3))

# To access the documentation
print(is_even.__doc__)
```

```
False

    Optional docstring which tells us about the function.
    This function return true if the given num is even.
    i/p - any valid integer
    o/p - true/false
```

In [3]:
```
# 1.2 Types of Arguements
# 1. Default Arguement
def power(a = 1, b = 1):
    return a ** b
print('Default:', power(2))

# 2. Positional Arguement
def power(a, b):
    return a ** b
print('Positional:', power(2, 3))
```

```
# 3. Keyword Arguement
def power(a, b):
    return a ** b
print('Keyword:', power(b = 3, a = 2))
```

```
Default: 2
Positional: 8
Keyword: 8
```

In [15]:
```
# 1.3 *args & **kwargs
# These are special keywords that are use to pass the varibale length of arguments
# Order of arguments matter (normal -> *args -> **args)
# 'args' & 'kwargs' are only a convention, we can use any name.

# *args: allows us to pass a varibale num of non-keyword arguments to a function.
def multiply(*args):
    print('Value in args:', args)
    product = 1
    for i in args:
        product *= i
    return product
print('Multiply:', multiply(2, 3, 6))

# **kwargs: allows us to pass a varibale num of keyword arguments to a function.
# keyword arguments mean that they contain a key-value pair
def display(**kwargs):
    for (key,value) in kwargs.items():
        print(key, '->', value)
display(India = 'Delhi', Nepal = 'Kathmandu')
```

```
Value in args: (2, 3, 6)
Multiply: 36
India -> Delhi
Nepal -> Kathmandu
```

In [18]:
```
# 1.4 Global scope vs Local Scope
# Ex-1:
def f(y):
    x = 1
    x += 1
    print(x)  # 2
x = 5
f(x)
print(x)      # 5

# Ex-2:
def  h(y):
    global x
    x += 1
x = 5
h(x)
print(x)      # 6
```

```
2
5
6
```

In [20]:
```
# 1.5 Nested Functions
# Ex-1:
def f():
    def g():
        print('Inside function g')
    g()
    print('Inside function f')
f()

# Ex-2:
def g(x):
    def h(x):
        x = x+1
        print("in h(x): x = ", x)
    x = x + 1
    print('in g(x): x = ', x)
    h(x)
    return x

x = 3
z = g(x)
print('in main program scope: x = ', x)
print('in main program scope: z = ', z)
```

```
Inside function g
Inside function f
in g(x): x =  4
in h(x): x =  5
in main program scope: x =  3
in main program scope: z =  4
```

In [31]:
```
# 1.6 Functions are first class citizens
# Function is immutable
def square(num):
    return num ** 2
# Type and id
print('Square:', square(4))
print('Type of square:', type(square))
print('Address of square:', id(square))

# Reassign
x = square
print('x:', x(4))
print('Type of x:', type(x))
print('Address of x:', id(x))

# Deleting a function
# del square

# Storing
li = [1,2,3,4,square]
print('List:', li)
print('Calling function from list:', li[-1](3))
```

```
Square: 16
Type of square: <class 'function'>
Address of square: 2811079278976
x: 16
Type of x: <class 'function'>
Address of x: 2811079278976
List: [1, 2, 3, 4, <function square at 0x0000028E8161C180>]
Calling function from list: 9
```

In [36]:
```python
# Returning a function
def f():
    def x(a, b):
        return a + b
    return x
val = f()(3, 4)
print(val)
```

```
7
```

In [37]:
```python
# Function as argument
def fun_a():
    print('Inside fun_a')
def fun_b(z):
    print('Inside fun_b')
    return z()
print(fun_b(fun_a))
```

```
Inside fun_b
Inside fun_a
None
```

In [42]:
```python
# 1.7 Lambda Function
# A lambda function is a small anonymous function
# It can take any number of arguments, but can only have one expression.
# Difference between lambda vs normal function
# 1. No name
# 2. lambda has no return value, infact returns a function
# 3. lambda is written in 1 line
# 4. Not reusable
# Q. Why we use lambda function?
# Ans: They are used with HOF (Higher Order Function)

# Ex-1:
a = lambda x : x**2
print(a(4))

# Ex-2:
sum = lambda x,y : x+y
print(sum(5,4))

# Ex-3:
check = lambda s : 'a' in s
print(check('Hello'))

# Ex-4:
even_odd = lambda x : 'even' if x%2 == 0 else 'odd'
print(even_odd(6))
```

```
16
9
False
even
```

In [45]:
```python
# 1.8 Higher Order Function
# A function that returns a function.
# Or a function that takes function as input

# Ex-1:
def transform(fun, li):
    output = []
    for i in li:
        output.append(fun(i))
    print(output)
li = [1,2,3,4,5]
transform(lambda x : x**2, li)
```

```
[1, 4, 9, 16, 25]
```

In [52]:
```python
# 1.9 Map function
# Q. Square the items of a list
print(list(map(lambda x : x**2, [1,2,3,4,5])))

# Q. odd/even labelling of list items
li = [1,2,3,4,5]
print(list(map(lambda x : 'even' if x%2 == 0 else 'odd',li)))

# Q. fetch names from a list of dict
users = [
    {
        'name':'Debashis',
        'age':23,
        'gender':'male'
    },
    {
        'name':'Smruti',
        'age':22,
        'gender':'male'
    },
    {
        'name':'Odishi',
        'age':21,
        'gender':'female'
    }
]
print(list(map(lambda users : users['name'],users)))
```

```
[1, 4, 9, 16, 25]
['odd', 'even', 'odd', 'even', 'odd']
['Debashis', 'Smruti', 'Odishi']
```

In [54]:
```python
# 1.10 Filter function
# Ex-1:
li1 = [3,4,5,6,7]
print(list(filter(lambda x : x>5, li1)))
```

```
# Ex-2:
li2 = ['apple', 'guave', 'cherry']
print(list(filter(lambda x : x.startswith('a'), li2)))
```

```
[6, 7]
['apple']
```

In [59]:
```
# 1.11 Reduce function
import functools
# Ex-1:
print('Sum of all:', functools.reduce(lambda x,y : x+y, [1,2,3,4,5]))

# Ex-2:
print('Minimum of all:', functools.reduce(lambda x,y : x if x < y else y, [23,11,45
```

```
Sum of all: 15
Minimum of all: 1
```