In [2]:
```
print("Hello World")
```

```
Hello World
```

In [3]:
```
a=9.567890
print(type(a))
```

```
<class 'float'>
```

In [4]:
```
# strings are of 3 types . They are single quoted,double quoted,triple quoted(for documentation purpose)
""" THis is for documentation purpose"""
b='hello there'
print(type(b))
```

```
<class 'str'>
```

In [5]:
```
flag = True, False
#list is a colection of elemets of various datatypes.
# Datatypes - Integer,Float,String,Boolean,Complx,List,Tuple,Set
lst=[] #mutable , collection of objects , ordered
tup=() # can't be modified if defined once , immutable , ordered
s={} #duplicate elements are not allowed , doesn't follow insertion order, i.e, unordered , mutable , indexing is not allowed
```

In [6]:
```
a="this is a note"
print(a[5])
```

```
i
```

In [7]:
```
a = (1,2,3,4,5)
print(type(a))
b=(1) # int coz of BODMAS Rule.
print(type(b))
c=(1,)
print(type(c))
```

```
<class 'tuple'>
<class 'int'>
<class 'tuple'>
```

In [8]:
```
s={1,1,1,2,2,3,3,4,4,5,6}
print(s)
d={'a','b','a','d'}
print(d)
```

```
{1, 2, 3, 4, 5, 6}
{'a', 'b', 'd'}
```

In [9]:
```
#Operators
#Arithmetic --> +,-,*,/(Float division),//(integer or floor division),%
#Logical    --> and(if both r true,then only o/p is True) , or , not
   # 1 and anything=anything
  # used in conditional statements and in that it gives you either True or False as an o/p
#comparision and relational <,>,<=,>=,==,!= , o/p is either True or False
#Bitwise --> and(&),or(|),XOR
#Assignment --> =
#Membership --> o/p is True/False , in,not in
#Identity   --> is
#3 type of operators : Unary,Binary,Terenary(?:)
```

In [10]:
```
#23 and 7
"""16 8 4 2 1
   1  0 1 1 1 and
   0  0 1 1 1
 = 0  0 1 1 1 which equals to 7 """
print(23 and 7)
```

```
7
```

In [11]:
```
print(bin(74))
#0b is for compilers own purpose to diff normal strings and bin strings
#bin() for converting int to bin
print(type(bin(12)))
```

```
0b1001010
<class 'str'>
```

In [12]:
```
#Terenary Operator
# Syntax -->  (condition) ? True part : False Part
```

In [13]:
```
#List methods

#lst=[]
#lst.append(only 1 arg)  to add an ele
#lst.insert(index,ele)  takes 2 args and returns None
#len(lst)
#lst.pop(takes index value)  to remove ele at that index, no index-> removes last ele
#print(lst.pop())   --> returns ele that is removed
#lst.remove(takes value,i.e, ele), 1 arg is compulsory
#print(lst.remove(5))  --> returns None
```

In [14]:
```
# a.extend(b)
#adds elements of list b in to list a
#a.count(takes val)
#a.clear()
#b=a.copy()
a=[10,20,30]
b=a.copy()
b[0]=100
print(a)
print(b)
#a.reverse()
#a.sort()  --> returns None
#a.sort(reverse=True) in decreasing order
#b=sorted(a) --> returns sorted list, but doesn't sort original list
#b=sorted(reverse=True) decreasing order
```

```
[10, 20, 30]
[100, 20, 30]
```

In [15]:
```
#Type conversions --> 2 types
#implicit , comp does internally
#explicit , we do it like int(),str(),list(),float() etc....
```

In [16]:
```
a=list('12345')
print(a)
b=map(int,a)
print(b)
c=list(map(int,a))
print(c)
# map(datatype,obj)

#0x..................Hexadecimal
#0b.........binary
```

```
['1', '2', '3', '4', '5']
<map object at 0x0000027AE3E02640>
[1, 2, 3, 4, 5]
```

In [17]:
```
a=input("Enter : ")
b=int(input("Enter : "))
print(a)
print(b)
```

```
Enter : 12
Enter : 63
12
63
```