Object Resuability / Memory Resuability

- int: -5 to 256
- bool: True and False
- string: A-Z, a-z, 0-9, _ and special character (till the time the string is not complex)
- For float, complex or anyother derived datatype, a new object will be created and assigned to a new memory space

In [1]:

```python
a = 10
b = 10
print(id(a), id(b))
```

```
140733156463688 140733156463688
```

In [2]:

```python
a = 257
b = 257
print(id(a), id(b))
```

```
1939983840976 1939983839216
```

In [3]:

```python
a = -6
b = -6
print(id(a), id(b))
```

```
1939983839120 1939983840144
```

In [4]:

```python
a = True
b = True
print(id(a), id(b))
```

```
140733154994720 140733154994720
```

In [5]:

```python
a = 'mayank'
b = 'mayank'
print(id(a), id(b))
```

```
1939984953136 1939984953136
```

In [6]:

```python
a = '@'
b = '@'
print(id(a), id(b))
```

```
140733156507592 140733156507592
```

In [8]:

```python
a = 'mayank@' #'m'+'a'+'y'+'a'+'n'+'k'+'@'
b = 'mayank@'
print(id(a), id(b))
```

```
1939984980528 1939984979184
```

In [9]:

```python
for i in a:
    print(i, id(i))
```

```
m 140733156510112
a 140733156509440
y 140733156510784
a 140733156509440
n 140733156510168
k 140733156510000
@ 140733156507592
```

In [11]:

```python
a = 'Mayank_ghai'
b = 'Mayank_ghai'
print(id(a), id(b))
```

```
1939984946288 1939984946288
```

Identity Operator

- It is used to check whether two values are having the same memory location (or address) or not
- It compares the memory location of the value , they dont compare the values directly
- It will give the answer in boolean values
    - is
    - is not

In [12]:

```python
a = 10
b = 10
print(a is b)
```

```
True
```

In [13]:

```python
print(a is not b)
```

```
False
```

In [14]:

```python
a = 10.0
b = 10.0
print(a is b)
```

```
False
```

In [15]:

```
a = [1]
b = [1]
print(type(a), type(b))
```

```
<class 'list'> <class 'list'>
```

In [17]:

```
print(id(a), id(b))
```

```
1939984579584 1939985037184
```

In [16]:

```
print(a is b)
```

```
False
```

Bitwise Operator

- Bit ---> smallest unit of memory (0,1)
- Bytes ---> 8 bits
- Binary --- is made up of bytes

bin() function is used to get the binary equivalent of a number

In [24]:

```
bin(28)
```

Out[24]:

```
'0b11100'
```

In [25]:

```
0b11100
```

Out[25]:

```
28
```

In [19]:

```
bin(127)
```

Out[19]:

```
'0b1111111'
```

oct() function is used to get the octal equavalent of a number

In [20]:

```
oct(127)
```

Out[20]:

```
'0o177'
```

hex() is used to get the hexadecimal equivalent of a number

In [21]:

```
hex(127)
```

Out[21]:

```
'0x7f'
```

In [ ]:

```
00011100 ---8 bit format ---- 1Byte
0000000000011100 - 16 bit format ----- 2 Bytes
00000000000000000000000000011100 -----> 4 bytes
```

Steps for the Bitwise Operator

- Convert the nuumber into bits
- Perform the operation bit by bit
- Convert the number back to the number

There are 6 Bitwise Operator

- Bitwise AND operator (&)
- Bitwise OR operator (|)
- Bitwise complement Operator (~)
- Bitwise XOR operator(^)
- Bitwise left shift operator (<<)
- Bitwise Right shift operator (>>)

Bitwise AND operator(&)

1 ---> True 0 ---> False

In [ ]:

```
0  and 0 ----> 0
0  and 1 ----> 0
1  and 0 ---> 0
1 and  1 --->1
```

In [31]:

```
25 & 72
```

Out[31]:

```
8
```

In [28]:

```
bin(25)
```

Out[28]:

```
'0b11001'
```

In [29]:
```
bin(72)
```

Out[29]:
```
'0b1001000'
```

In [ ]:
```
0b0011001
0b1001000
0b0001000
```

In [30]:
```
0b0001000
```

Out[30]:
```
8
```

In [35]:
```
53 & 61
```

Out[35]:
```
53
```

In [32]:
```
bin(53)
```

Out[32]:
```
'0b110101'
```

In [33]:
```
bin(61)
```

Out[33]:
```
'0b111101'
```

In [ ]:
```
0b110101
0b111101
0b110101
```

In [34]:
```
0b110101
```

Out[34]:
```
53
```

Bitwise Or Operator (|)

In [ ]:
```
0 or 0 --->0
1 or 0 ---->1
0 or 1 ---->1
1 or 1 ----> 1
```

In [39]:
```
45|78
```

Out[39]:
```
111
```

```
In [36]: bin(45)
```

```
Out[36]: '0b101101'
```

```
In [37]: bin(78)
```

```
Out[37]: '0b1001110'
```

```
In [ ]: 0b0101101
        0b1001110
        0b1101111
```

```
In [38]: 0b1101111
```

```
Out[38]: 111
```

```
In [43]: 98|23
```

```
Out[43]: 119
```

```
In [40]: bin(98)
```

```
Out[40]: '0b1100010'
```

```
In [41]: bin(23)
```

```
Out[41]: '0b10111'
```

```
In [ ]: 0b1100010
        0b0010111
        0b1110111
```

```
In [42]: 0b1110111
```

```
Out[42]: 119
```

Bitwise Complementary / Bitwise Not operator (~)

- It is going to take one operand or value
- we will going to take 2's complement of a numeber
- Process of taking 2's complement of a number is
    - take the 1's complement of the givem number i.e we will convert number to binary and convert 0 to 1 and 1 to 0
    - add 1 in the binary format to the 1's complement of the given number
    - add a negative sign at the beginning of it

      Formula:

~num = -(num+1)

In [46]:
```
~5
```
Out[46]:
```
-6
```

In [49]:
```
bin(5)
```
Out[49]:
```
'0b101'
```

Binary Addition

In [ ]:
```
1 + 0 --->1
0 + 1 ---> 1
0 + 0 ---> 0
1 + 1 ----> 0 and carry on 1
```

In [50]:
```
ob101
ob001
0b110
```
Out[50]:
```
2
```

In [51]:
```
0b011
```
Out[51]:
```
10
```

In [47]:
```
~78 #-(78+1)
```
Out[47]:
```
-79
```

In [52]:
```
bin(78)
```
Out[52]:
```
'0b1001110'
```

In [ ]:
```
0b1001110 -----> 0b0110001
```

In [53]:
```
bin(1)
```
Out[53]:
```
'0b1'
```

In [ ]:

```
1 + 0 --->1
0 + 1 ---> 1
0 + 0 ---> 0
1 + 1 ----> 0 and carry on 1
```

In [ ]:

```
0b1001110 --- 0110001
0b0000001 ---->
0b1001111
```

In [63]:

```
0b1001111
```

Out[63]:

```
79
```

In [58]:

```
0b0110001
```

Out[58]:

```
49
```

In [64]:

```
b= 'mus#kan'
for i in b:
    print(i,id(i))

```

```
m 140733156510112
u 140733156510560
s 140733156510448
# 140733156505968
k 140733156510000
a 140733156509440
n 140733156510168
```

In [65]:

```
bin(78)
```

Out[65]:

```
'0b1001110'
```

In [ ]:

```
0b0110001
```

In [66]:

```
bin(79)
```

Out[66]:

```
'0b1001111'
```

In [ ]:

```
0b0110000
0b0000001
0b0110001
```

In [ ]:

```
ob1001110
```