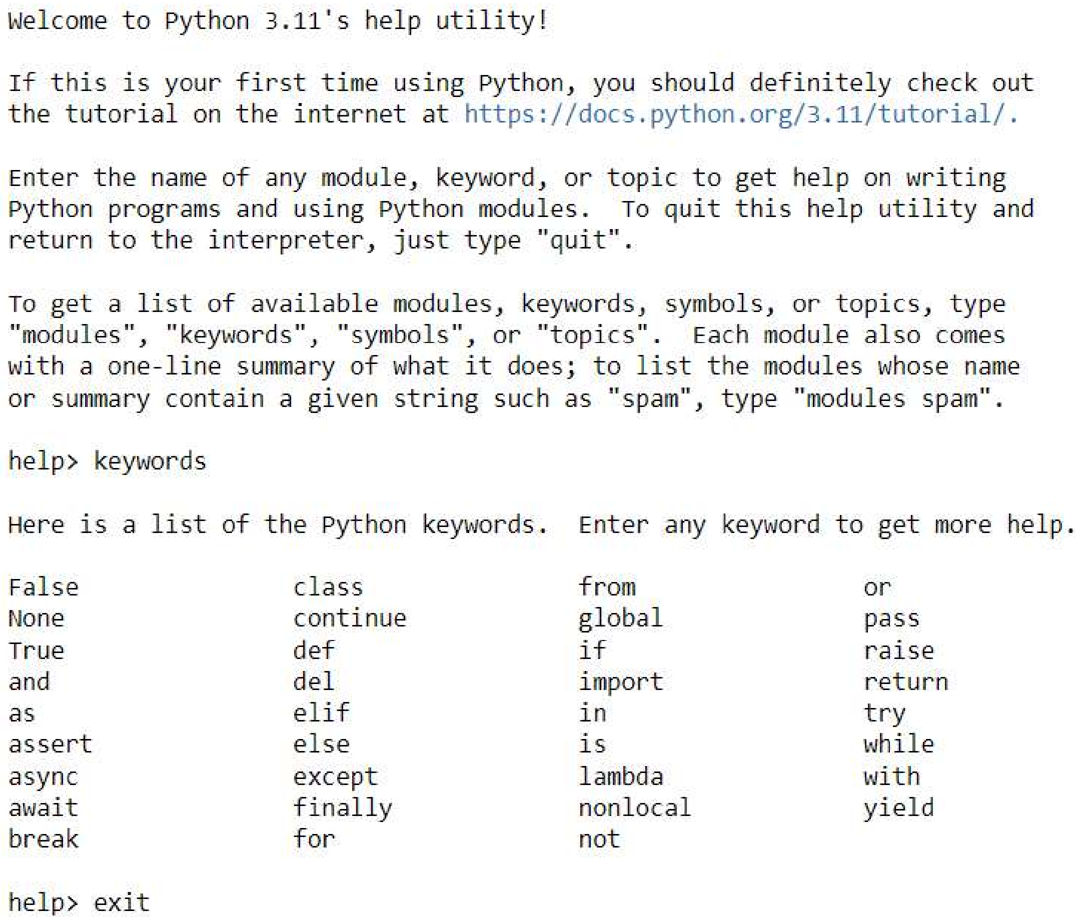
How to create environment variable

STEPS TO SET UP EXECUTE PYTHON IN SYSTEM CMD (TO CREATE ENVIRONMENT VARIABLE) Open cmd # python (You will get error when you execute 1st time) search with environment variable - system variable: (C:\Users\kdata\AppData\Local\Microsoft\WindowsApps) restart the cmd & type python in cmd it will work now

to find help

STEPS TO FIND HELP OPTION --> 1- help() | 2- topics | 3- search as per requirments | 4- quit if you want help on any command then help(list) || help(tuple)

```
In [ ]: help()
```



```
Welcome to Python 3.11's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.11/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

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

help> exit
```

In [6]:

```
help(list)
```

```
 |      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.
 |
```

In [7]:

```
2 + 3
```

Out[7]:

```
5
```

In [8]:

```
help(tuple)
```

```
Help on class tuple in module builtins:

class tuple(object)
 |  tuple(iterable=(), /)
 |
 |  Built-in immutable sequence.
 |
 |  If no argument is given, the constructor returns an empty tuple.
 |  If iterable is specified the tuple is initialized from iterable's items.
 |
 |  If the argument is a tuple, the return value is the same object.
 |
 |  Built-in subclasses:
 |      asyncgen_hooks
 |      UnraisableHookArgs
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __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).
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |  
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |      
 |      Raises ValueError if the value is not present.
 |  
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |  
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585
 |  
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
```

introduce to ID()

In [9]:

```
# variable address
num = 5
id(num)
```

Out[9]:

```
140711552390056
```

In [10]:

```
name = 'nit'
id(name) #Address will be different for both
```

Out[10]:

```
1584926327920
```

In [11]:

```
a = 10
id(a)
```

Out[11]:

```
140711552390216
```

In [12]:

```
b = a #thats why python is more memory efficient
```

In [13]:

```
id(b)
```

Out[13]:

```
140711552390216
```

In [14]:

```
id(10)
```

Out[14]:

```
140711552390216
```

In [15]:

```
k = 10
id(k)
```

Out[15]:

```
140711552390216
```

In [16]:

```
a = 20  # as we change the value of a then address will change
id(a)
```

Out[16]:

```
140711552390536
```

```
In [17]: id(b)
```

```
Out[17]: 140711552390216
```

what ever the variale we assigned the memory and we not assigned anywhere then we can use as garbage collection.|| VARIABLE - we can change the values || CONSTANT - we cannot change the value -can we make VARIABLE as a CONSTANT (note - in python you cannot make variable as constant)

```
In [19]: PI = 3.14  #in math this is alway constant but python we can chang
         PI
```

```
Out[19]: 3.14
```

```
In [20]: PI = 3.15
         PI
```

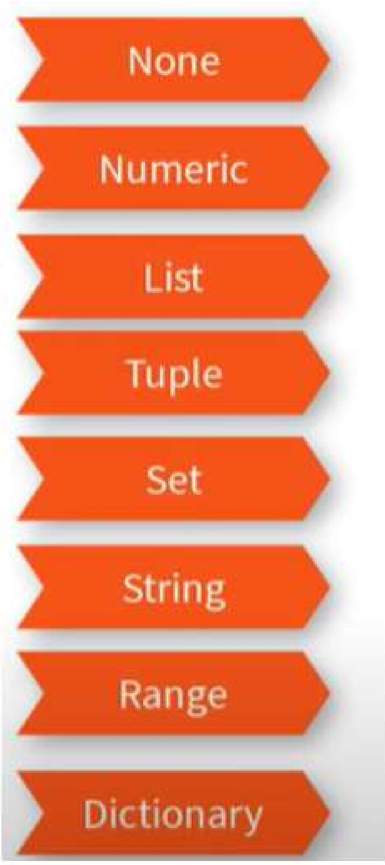
```
Out[20]: 3.15
```

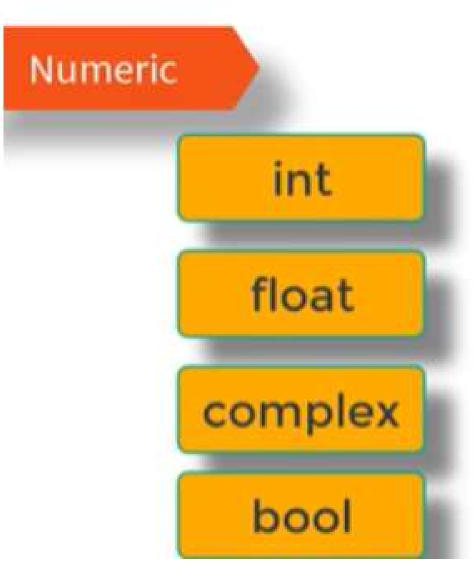
```
In [21]: type(PI)
```

```
Out[21]: float
```

DATA TYPES & DATA STRUCTURES-->

1- NUMERIC || 2-LIST || 3-TUPLE || 4-SET || 5-STRING || 6-RANGE || 7-DICTIONARY





1- NUMERIC :- INT || FLOAT || COMPLEX || BOOL

In [22]:
```
w = 2.5
type(w)
```
Out[22]:
```
float
```
In [23]:
```
(a)
```
Out[23]:
```
20
```
In [24]:
```
w2 = 2 + 3j #so hear j is represent as root of -1
type(w2)
```
Out[24]:
```
complex
```
In [25]:
```
#convert flot to integer
a = 5.6
b = int(a)
```
In [26]:
```
b
```
Out[26]:
```
5
```
In [27]:
```
type(b)
```
Out[27]:
```
int
```
In [28]:
```
type(a)
```
Out[28]:
```
float
```
In [29]:
```
k = float(b)
```
In [30]:
```
k
```
Out[30]:
```
5.0
```

In [31]:

```
print(a)
print(b)
print(k)
```

```
5.6
5
5.0
```

In [32]:

```
k1 = complex(b,k)
```

In [33]:

```
print(k1)
type(k1)
```

```
(5+5j)
```

Out[33]:

```
complex
```

In [34]:

```
b < k
```

Out[34]:

```
False
```

In [35]:

```
condition = b<k
condition
```

Out[35]:

```
False
```

In [36]:

```
type(condition)
```

Out[36]:

```
bool
```

In [37]:

```
int(True)
```

Out[37]:

```
1
```

In [38]:

```
int(False)
```

Out[38]:

```
0
```

In [39]:

```
l = [1,2,3,4]
print(l)
type(l)
```

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

Out[39]:

```
list
```

In [40]:

```
s = {1,2,3,4}
s
```

Out[40]:

```
{1, 2, 3, 4}
```

In [41]:

```
type(s)
```

Out[41]:

```
set
```

In [42]:

```
t = (10,20,30)
t
```

Out[42]:

```
(10, 20, 30)
```

```
In [43]: type(t)
```

```
Out[43]: tuple
```

```
In [44]: str = 'nit' #we dont have character in python
         type(str)
```

```
Out[44]: str
```

```
In [45]: st = 'n'
         type(st)
```

```
Out[45]: str
```

range()

```
In [47]: r = range(0,10)
         r
```

```
Out[47]: range(0, 10)
```

```
In [48]: type(r)
```

```
Out[48]: range
```

```
In [49]: # if you want to print the range
         list(range(0,10))
```

```
Out[49]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [50]: r1 = list(r)
         r1
```

```
Out[50]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [51]: #if you want to print even number
         even_number = list(range(2,10,2))
         even_number
```

```
Out[51]: [2, 4, 6, 8]
```

```
In [52]: d= {1:'one', 2:'two', 3:'three'}
         d
```

```
Out[52]: {1: 'one', 2: 'two', 3: 'three'}
```

```
In [53]: type(d)
```

```
Out[53]: dict
```

```
In [54]: # print the keys
         d.keys()
```

```
Out[54]: dict_keys([1, 2, 3])
```

```
In [55]: # how to get particular value
         d[2]
```

```
Out[55]: 'two'
```

In [56]:

```
# other way to get value as
d.get(2)
```

Out[56]:

```
'two'
```

operator

1- ARITHMETIC OPERATOR (+ , -, *, /, %, %%, **, ^) 2- ASSIGNMEN OPERATOR (=) 3- RELATIONAL OPERATOR 4- LOGICAL OPERATOR 5- UNARY OPERATOR

Arithmetic operator

In [57]:

```
x1, y1 = 10, 5
```

In [58]:

```
#x1 ^ y1
```

In [59]:

```
x1  + y1
```

Out[59]:

```
15
```

In [60]:

```
x1 - y1
```

Out[60]:

```
5
```

In [61]:

```
x1 * y1
```

Out[61]:

```
50
```

In [62]:

```
x1 / y1
```

Out[62]:

```
2.0
```

In [63]:

```
x1 // y1
```

Out[63]:

```
2
```

In [64]:

```
x1 % y1
```

Out[64]:

```
0
```

In [65]:

```
x1 ** y1
```

Out[65]:

```
100000
```

In [66]:

```
x2 =3
y2 = 3
x2 ** y2
```

Out[66]:

```
27
```

Assignment operator

```
In [67]: x = 2
```

```
In [68]: x = x + 2 # if you want to increment by 2
```

```
In [69]: x += 2
         x
```

```
Out[69]: 6
```

```
In [70]: x += 2
         x
```

```
Out[70]: 8
```

```
In [71]: x *= 2
```

```
In [72]: x
```

```
Out[72]: 16
```

```
In [73]: x -= 2
```

```
In [74]: x
```

```
Out[74]: 14
```

```
In [75]: x /= 2
```

```
In [76]: x
```

```
Out[76]: 7.0
```

```
In [77]: a, b = 5,6 # you can assigned variable in one line as well
```

```
In [78]: a
```

```
Out[78]: 5
```

```
In [79]: b
```

```
Out[79]: 6
```

unary operator

unary means 1 || binary means 2 Here we are applying unary minus operator(-) on the operand n; the value of m becomes -7, which indicates it as a negative value.

```
In [80]: n = 7 #negattion
         n
```

```
Out[80]: 7
```

```
In [81]: m = -(n)
         m
```

```
Out[81]: -7
```

```
In [82]: n
```

```
Out[82]: 7
```

```
In [83]: -n
```

```
Out[83]: -7
```

Relational operator

we are using this operator for comparing

```
In [84]: a = 5
         b = 6
```

```
In [85]: a<b
```

```
Out[85]: True
```

```
In [86]: a>b
```

```
Out[86]: False
```

```
In [87]: # a = b # we cannot use = operatro that means it is assigning
```

```
In [88]: a == b
```

```
Out[88]: False
```

```
In [89]: a != b
```

```
Out[89]: True
```

```
In [90]: # hear if i change b = 6
         b = 5
```

```
In [91]: a == b
```

```
Out[91]: True
```

```
In [92]: b
```

```
Out[92]: 5
```

```
In [93]: a >= b
```

```
Out[93]: True
```

```
In [94]: a <= b
```

```
Out[94]: True
```

```
In [95]: a < b
```

```
Out[95]: False
```

```
In [96]: a>b
```

```
Out[96]: False
```

```
In [97]: b = 7
```

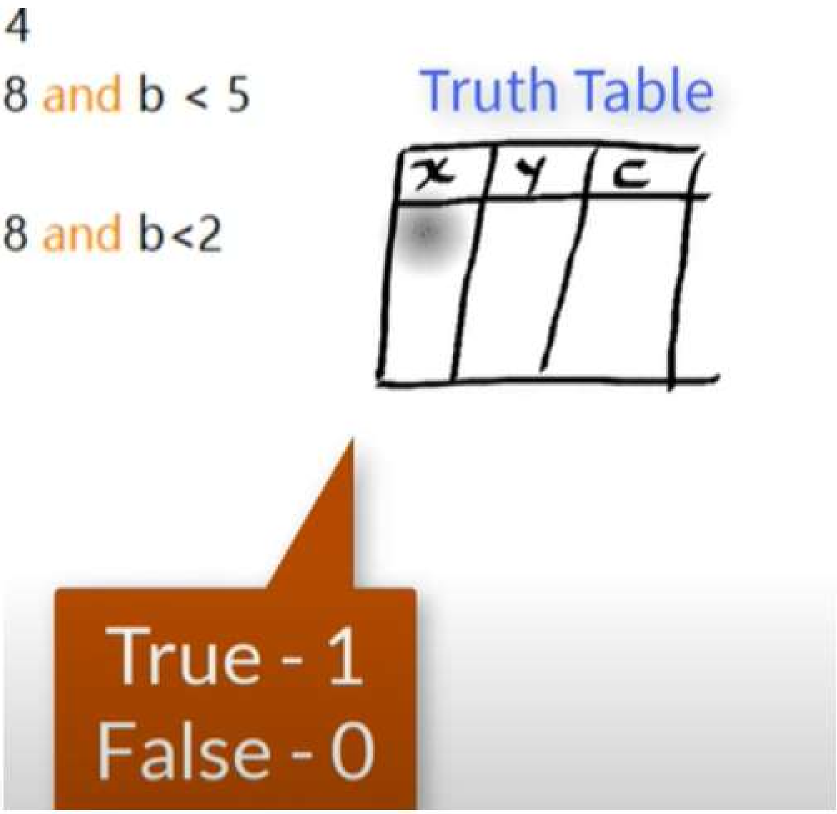
```
In [98]: a != b
```

```
Out[98]: True
```

LOGICAL OPERATOR

logical operator you need to understand about true & false table 3 importand part of logical operator is --> AND, OR, NOT

lets understand the truth table:- in truth table you can represent (true-1 & false means- 0)



4

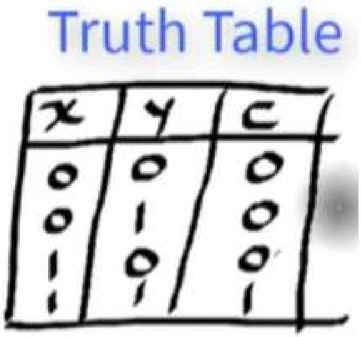
8 and b < 5

8 and b<2

Truth Table

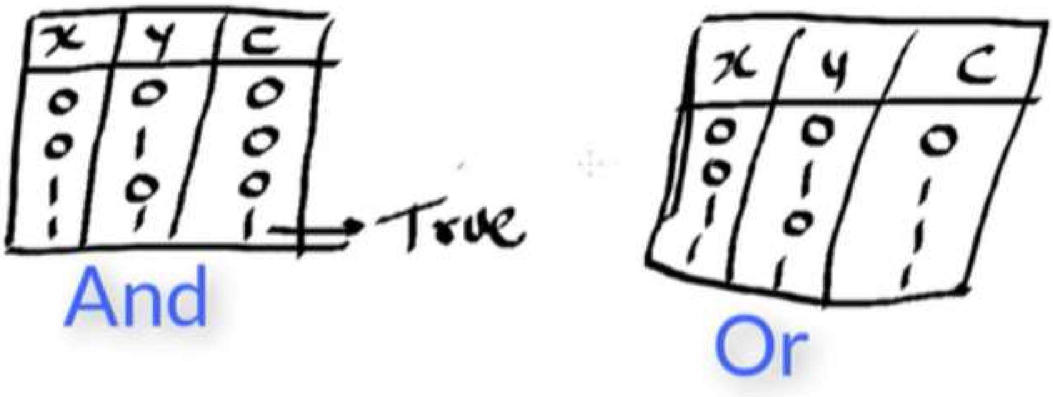
x	y	c

True - 1
False - 0



Truth Table

x	y	c
0	0	0
0	1	0
1	0	0
1	1	1



x	y	c
0	0	0
0	1	0
1	0	0
1	1	1

→ True

And

x	y	c
0	0	0
0	1	1
1	0	1
1	1	1

Or

```
In [99]: a = 5
         b = 4
```

```
In [100]: a < 8 and b < 5 #refer to the truth table

Out[100]: True
```

```
In [101]: a < 8 and b < 2

Out[101]: False
```

```
In [102]: a < 8 or b < 2

Out[102]: True
```

```
In [103]: a>8 or b<2

Out[103]: False
```

```
In [104]: x = False
          x

Out[104]: False
```

```
In [105]: not x  # you can reverse the operation

Out[105]: True
```

```
In [106]: x = not x
          x

Out[106]: True
```

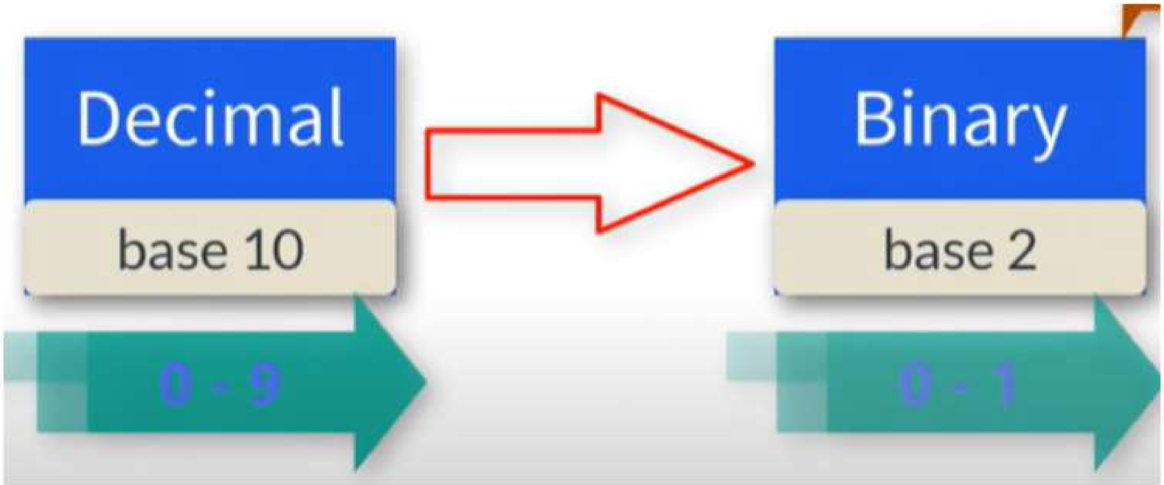
```
In [107]: x

Out[107]: True
```

```
In [108]: not x

Out[108]: False
```

Number system coverstion (bit-binary digit)

In the programing we are using binary system, octal system, decimal system & hexadecimal system but where do we use this in cmd - you can check your ip address & lets understand how to convert from one system to other system when you check ipaddress you will these format --> cmd - ipconfig

binary : base (0-1) --> please divide 15/2 & count in reverse order ocatl : base (0-7) hexadecimal :base (0-9 & then a-f)



Decimal

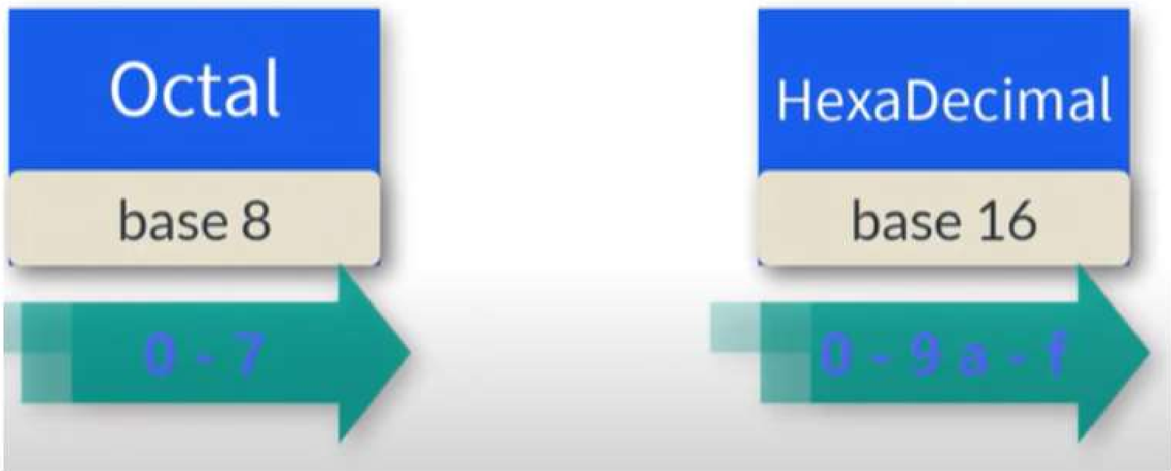
base 10

0 - 9

Binary

base 2

0 - 1



Octal

base 8

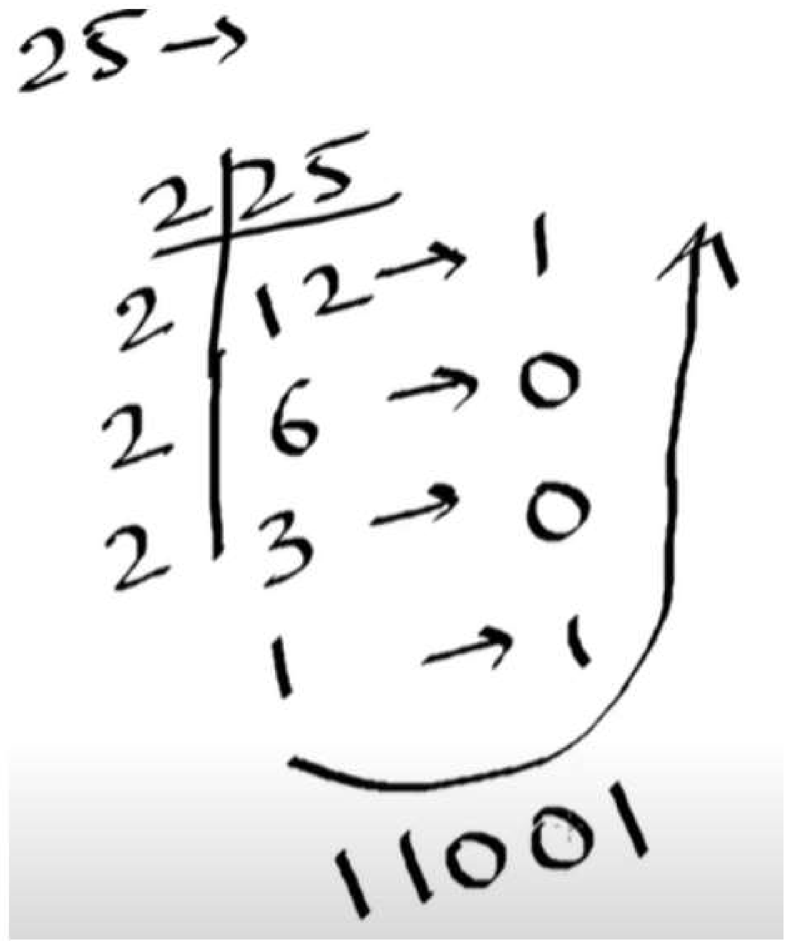
0 - 7

HexaDecimal

base 16

0 - 9 a - f

```
In [109]: bin(25)

Out[109]: '0b11001'
```



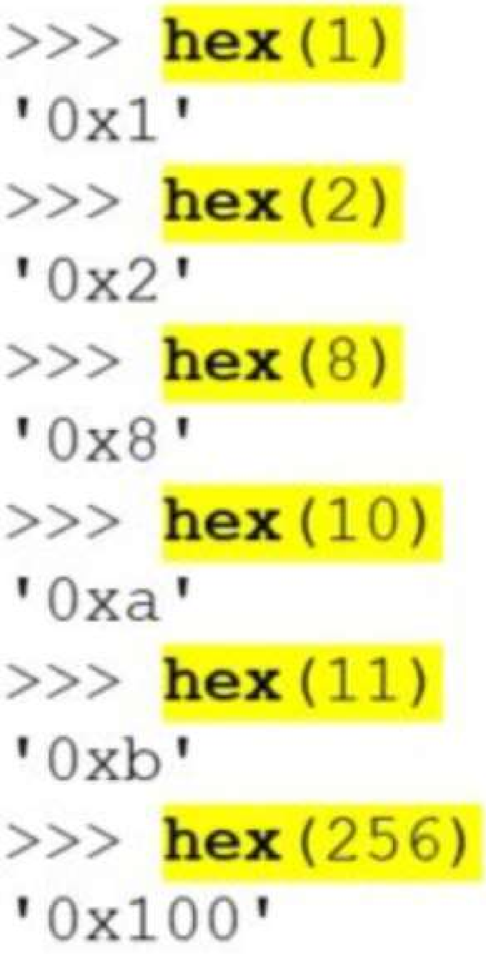
```
In [110]: 0b11001
Out[110]: 25

In [111]: int(0b11001)
Out[111]: 25

In [112]: bin(7)
Out[112]: '0b111'

In [113]: oct(25)
Out[113]: '0o31'

In [114]: 0o31
Out[114]: 25

In [115]: int(0o31)
Out[115]: 25

In [116]: hex(25)
Out[116]: '0x19'
```

```
In [117]: 0x19
Out[117]: 25
```

```
In [118]: hex(16)
Out[118]: '0x10'
```

```
In [119]: 0xa
Out[119]: 10
```

```
In [120]: 0xb
Out[120]: 11
```



```
>>> hex(1)
'0x1'
>>> hex(2)
'0x2'
>>> hex(8)
'0x8'
>>> hex(10)
'0xa'
>>> hex(11)
'0xb'
>>> hex(256)
'0x100'
```

```
In [121]: hex(25)
Out[121]: '0x19'
```

```
In [122]: 0x19
Out[122]: 25
```

```
In [123]: 0x15
Out[123]: 21
```

swap 2-variable in python

(a,b = 5,6) After swap we should get ==> (a, b = 6,5)

In [125]:

```
a = 5
b = 6
```

In [126]:

```
a = b
b = a
```

In [127]:

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

```
6
6
```

In [128]:

```
# in above scenario we lost the value 5
a1 = 7
b1 = 8
```

In [129]:

```
temp = a1
a1 = b1
b1 = temp
```

In [130]:

```
print(a1)
print(b1)
```

```
8
7
```

in the above code we are using third variable in interview they might ask can we swap better way without using 3rd variable

In [131]:

```
a2 = 5
b2 = 6
```

In [132]:

```
#swap variable formulas without using 3rd formula
a2 = a2 + b2 # 5+6 = 11
b2 = a2 - b2 # 11-6 = 5
a2 = a2 - b2 # 11-5 = 6
```

In [133]:

```
print(a2)
print(b2)
```

```
6
5
```

In [134]:

```
print(0b101) # 101 is 3 bit
print(0b110) # 110 also 3bit
```

```
5
6
```

In [135]:

```
print(0b110)
print(0b101)
```

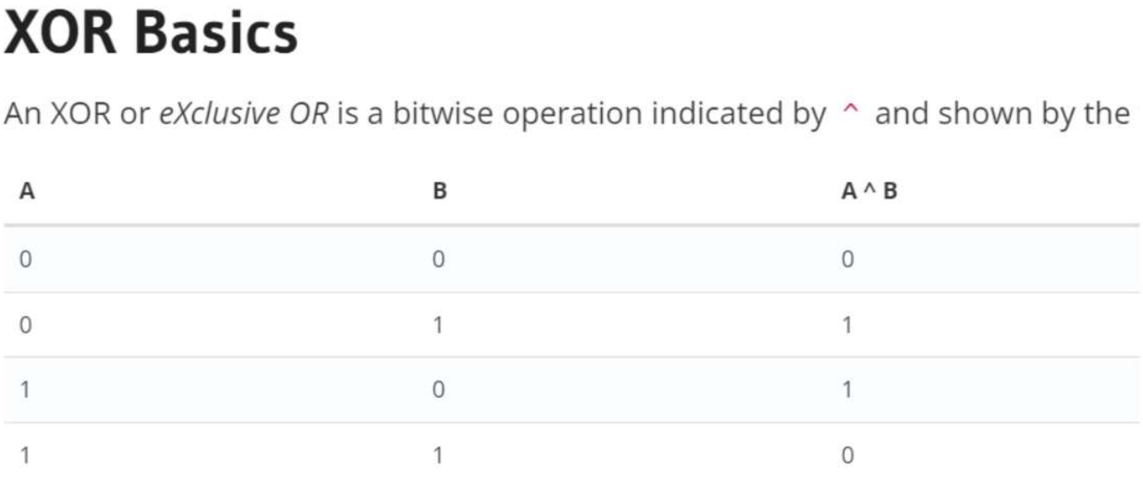
```
6
5
```

In [136]:

```
#but when we use a2 + b2 then we get 11 that means we will get 4 bit which is 1 bit extra
print(bin(11))
print(0b1011)
```

```
0b1011
11
```

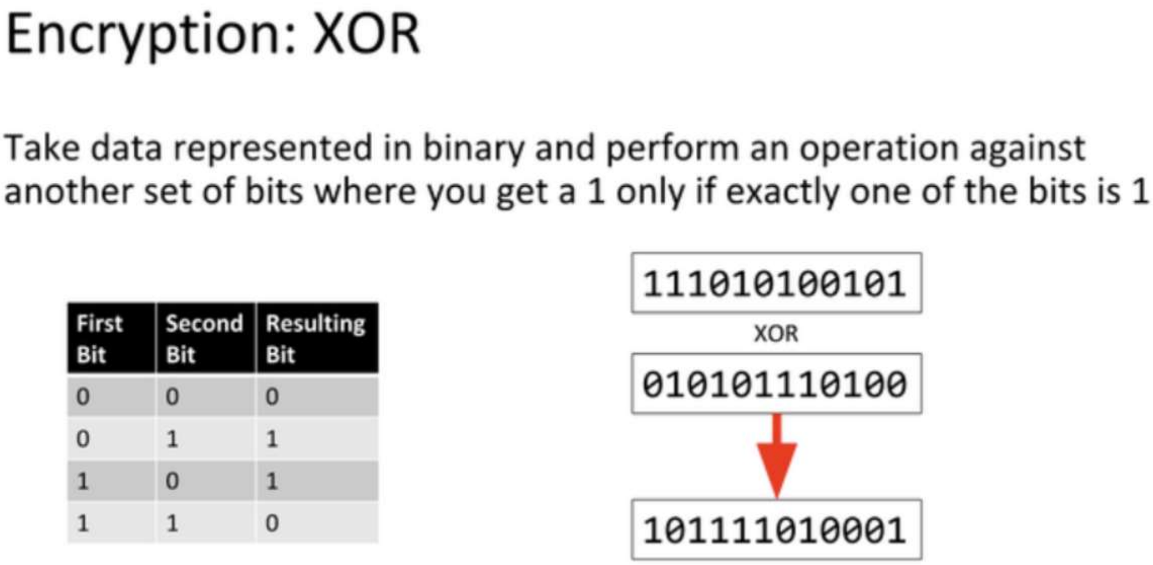
-there is other way to work using swap variable also which is XOR because it will not waste extra bit



XOR Basics

An XOR or *eXclusive OR* is a bitwise operation indicated by ^ and shown by the

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0



Encryption: XOR

Take data represented in binary and perform an operation against another set of bits where you get a 1 only if exactly one of the bits is 1

First Bit	Second Bit	Resulting Bit
0	0	0
0	1	1
1	0	1
1	1	0

111010100101

XOR

010101110100

101111010001

```
In [137]: print(a2)
          print(b2)

          6
          5
```

```
In [138]: #there is other way to work using swap variable also which is XOR because it will not waste extra bi
          a2 = a2 ^ b2
          b2 = a2 ^ b2
          a2 = a2 ^ b2
```

```
In [139]: print(a2)
          print(b2)

          5
          6
```

```
In [140]: a2, b2

Out[140]: (5, 6)
```

```
In [141]: a2 ,b2 = b2, a2 # how it work is b2 6 a2 is 5 first it goes into stack & then it reverse the 2 vlaue
```

```
In [142]: print(a2)
          print(b2)

          6
          5
```

BITWISE OPERATOR

WE HAVE 6 OPERATORS COMPLEMENT (~) || AND (&) || OR (|) || XOR (^) || LEFT SHIFT (<<) || RIGHT SHIFT (>>)

```
In [143]: print(bin(12))
          print(bin(13))

          0b1100
          0b1101
```

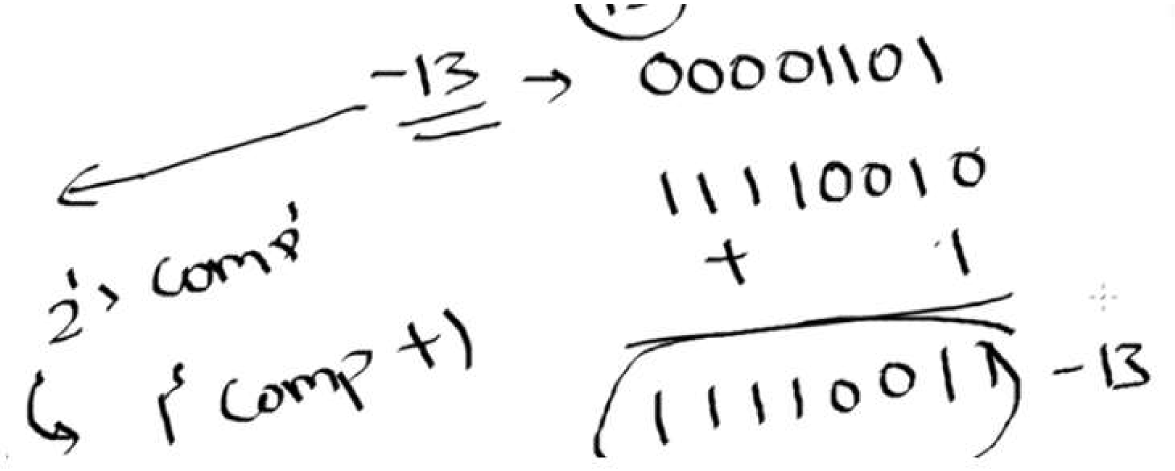
```
In [144]: 0b1101

Out[144]: 13
```

```
In [145]: 0b1100

Out[145]: 12
```

complement --> you will get this key below esc character

12 ==> 1100 ||

first thing we need to understand what is mean by complement. complement means it will do reverse of the binary format i.e. - ~0 it will give you 1 ~1 it will give 0 12 binary format is 00001100 (complement of ~00001100 reverse the number - 11110011 which is (-13) in the virtual memory we cant store -ve number & the only way to store the -ve value by using complimentory but the question is why we got -13 to understand this concept (we have concept of 2's complement 2's complement mean (1's complement + 1) in the system we can store +Ve number but how to store -ve number lets understand binary form of 13 - 00001101 + 1



In [146]:
```
# COMPLEMENT (~) (TILDE  OR TILD)
~12 # why we get -13 . first we understand what is complment means (reversr of binary format)
```

Out[146]:
```
-13
```

In [147]:
```
~46
```

Out[147]:
```
-47
```

In [148]:
```
~54
```

Out[148]:
```
-55
```

In [149]:
```
~-6
```

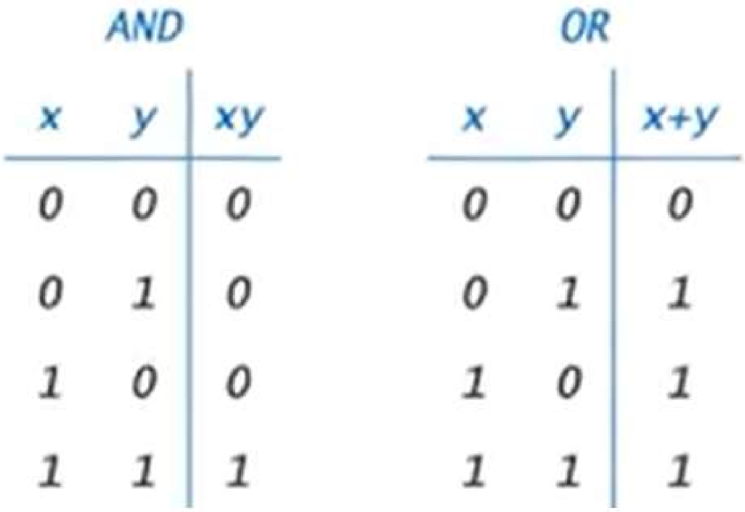
Out[149]:
```
5
```

In [150]:
```
~-1
```

Out[150]:
```
0
```

bit wise and operator

AND - LOGICAL OPERATOR ||| & - BITWISE AND OPERATOR (we know that 1 & 1 is 1) 12 - 00001100 13 - 00001101 when we are add both then then outut we will get as 12



AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

12 00001100

13 00001101

00001100 → 12

```
In [152]: 12 & 13
Out[152]: 12
```

```
In [153]: 12 | 13
Out[153]: 13
```

```
In [154]: 1 & 0
Out[154]: 0
```

```
In [155]: 1 | 0
Out[155]: 1
```

12 00001100

13 & 00001101

00001100 → 12

00001100

| 00001101

00001101

In [156]:
```
35 & 40 #please do the homework conververt 35,40 to binary format
```
Out[156]:
```
32
```
In [157]:
```
35 | 40
```
Out[157]:
```
43
```
In [158]:
```
# in XOR if the both number are different then we will get 1 or else we will get 0

12 ^ 13
```
Out[158]:
```
1
```
In [159]:
```
25^30
```
Out[159]:
```
7
```
In [160]:
```
bin(10)
```
Out[160]:
```
'0b1010'
```
In [161]:
```
# BIT WISE LEFT SHIFT OPERATOR
# in left shift what we need to to we need shift in left hand side & need to shift 2 bits
#bit wise left operator bydefault you will take 2 zeros ( )
#10 binary operator is 1010 | also i can say 1010
10<<2
```
Out[161]:
```
40
```
In [162]:
```
bin(20)
```
Out[162]:
```
'0b10100'
```
In [163]:
```
0b101000000
```
Out[163]:
```
320
```
In [164]:
```
20<<4 #can we do this
```
Out[164]:
```
320
```
In [165]:
```
bin(10)
```
Out[165]:
```
'0b1010'
```
In [166]:
```
10>>2
```
Out[166]:
```
2
```
In [167]:
```
10>>3
```
Out[167]:
```
1
```
In [168]:
```
bin(20)
```
Out[168]:
```
'0b10100'
```

In [169]:

```
20>>4
```

Out[169]:

```
1
```

import math function

In [170]:

```
x = sqrt(25) #sqrt is inbuild function
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[170], line 1
----> 1 x = sqrt(25)

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

In [171]:

```
import math # math is module
```

In [172]:

```
x = math.sqrt(25)
x
```

Out[172]:

```
5.0
```

In [173]:

```
x1 = math.sqrt(15)
x1
```

Out[173]:

```
3.872983346207417
```

In [174]:

```
print(math.floor(3.87)) #floor - minimum or least value
```

```
3
```

In [175]:

```
print(math.ceil(3.87)) #ceil - maximum or highest value
```

```
4
```

In [176]:

```
print(math.pow(3,2))
```

```
9.0
```

In [177]:

```
print(math.pi) #these are constant
```

```
3.141592653589793
```

In [178]:

```
print(math.e) # e - epsilon values
```

```
2.718281828459045
```

In [179]:

```
m.sqrt(25)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[179], line 1
----> 1 m.sqrt(25)

AttributeError: 'int' object has no attribute 'sqrt'
```

In [180]:

```
import math as m # we need to use concept aliseing, instead of math we are using as m
m.sqrt(10) #if you are lazy to type then you can use m or else you can use math
```

Out[180]:

```
3.1622776601683795
```

In [181]:

```
from math import sqrt,pow # math has many function if you want to import specific function then use
print(pow(2,3))
print(math.sqrt(10))
```

```
8.0
3.1622776601683795
```

In [182]:

```
round(pow(2,3))
```

Out[182]:

```
8
```

In [183]:

```
help(math)
```

```
    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    cbrt(x, /)
        Return the cube root of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    comb(n, k, /)
        Number of ways to choose k items from n items without repetition and without order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
```

user input function in python || command line input

how to get input from user

In [184]:

```
x = input()
y = input()
z = x + y
print(z) # console is waiting for user to enter input
# also if you work in idle
```

```
1
2
12
```

In [187]:

```
type(x)
```

Out[187]:

```
str
```

In [186]:

```
x1 = input('Enter the 1st number') #whenevery you works in input function it always give you string
y1 = input('Enter the 2nd number') # it wont understand as arithmetic operator
z1 = x1 + y1
print(z1)
```

```
Enter the 1st number2
Enter the 2nd number4
24
```

In [188]:

```
print(type(x1))
print(type(y1))
```

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

In [189]:

```
x1 = input('Enter the 1st number') #whenevery you works in input function it always give you string
a1 = int(x1)
y1 = input('Enter the 2nd number') # it wont understand as arithmetic operator
b1 = int(y1)
z1 = a1 + b1
print(z1)
```

```
Enter the 1st number32
Enter the 2nd number54
86
```

for the above code notice we are using many lines because fo that wasting some memory spaces as well

In [191]:

```
x2 = int(input('Enter the 1st number'))
y2 = int(input('Enter the 2nd number'))
z2 = x2 + y2
z2
```

```
Enter the 1st number88
Enter the 2nd number66
```

Out[191]:

```
154
```

lets take input from the user in char format, but we dont have char format in python

In [192]:

```
ch = input('enter a char')
print(ch)
#print(type(ch))
```

```
enter a charluffy
luffy
```

In [193]:

```
print(ch[0])
```

```
l
```

In [194]:

```
print(ch[0:2])
```

```
lu
```

```
In [195]: print(ch[1])
```

```
u
```

```
In [196]: print(ch[-1])
```

```
y
```

```
In [197]: ch = input('enter a char')[0]
          print(ch)
```

```
enter a charzoro
z
```

```
In [198]: ch = input('enter a char')[1]
          print(ch)
```

```
enter a charnami
a
```

```
In [199]: ch = input('enter a char')[1:3]
          print(ch)
```

```
enter a charsanji
an
```

```
In [200]: ch = input('enter a char')
          print(ch) # if you enter as 2 + 6 -1 we get output as 2 + 6-1 only cuz 2+6-1 as expression
```

```
enter a chargodusap
godusap
```

EVAL function using input

```
In [201]: result = eval(input('enter an expr'))
          print(result)
```

```
enter an expr123
123
```

```
In [ ]:
```