PYTHON PROGRAMMING LANGUAGE

Python Became the Best Programming Language & fastest programming language. Python is used in Machine Learning, Data Science, Big Data, Web Development, Scripting, generative ai, prompt enginering, llm model, agentic ai, cloud also we are integration python. we will learn pyton from start to end || basic to expert. if you are not done programm then that is totally fine. I will explain from starting from scratch. python software - pycharm || vs code || jupyter || spyder

PYTHON INTERPRETTER

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

PYTHON INTERPRETER --> What is Python interpreter? A python interpreter is a computer program that converts each high-level program statement into machine code. An interpreter translates the command that you write out into code that the computer can understand

PYTHON INTERPRETER EXAMPLE --> You write your Python code in a text file with a name like hello.py . How does that code Run? There is program installed on your computer named "python3" or "python", and its job is looking at and running your Python code. This type of program is called an "interpreter".

IDE (INTEGRATED DEVELOPMENT ENVIRONMENT) =>

- using IDE - one can write code, run the code, debug the code
- IDE takes care of interpreting the Python code, running python scripts, building executables, and debugging the applications.
- An IDE enables programmers to combine the different aspects of writing a computer program.
- if you wnated to be python developer only then you need to install (IDE -- PYCHARM)

PYTHON INTERPRETER & COMPILER

Both compilers and interpreters are used to convert a program written in a high-level language into machine code understood by computers. Interpreter -->

- Translates program one statement at a time
- Interpreter run every line item
- Execut the single, partial line of code
- Easy for programming

Compiler -->

- Scans the entire program and translates it as a whole into machine code.
- No execution if an error occurs
- you can not fix the bug (debug) line by line

Is Python an interpreter or compiler? Python is an interpreted language, which means the source code of a Python program is converted into bytecode that is then executed by the Python virtual machine. Python is different from major compiled languages, such as C and C + +, as Python code is not required to be built and linked like code for these languages.

How to create python environment variable 1- cmd - python (if it not works) 2- find the location where the python is installed -- > C:\Users\kdata\AppData\Local\Programs\Python\Python311\Scripts 3- system -- env - environment variable screen will pop up 4- select on system variable - click on path - create New 5- C:\Users\kdata\AppData\Local\Programs\Python\Python311 6- env - sys variable - path - new - C:\Users\kdata\AppData\Local\Programs\Python\Python311\Scripts 7- cmd - type python -version 8- successfully python install in cmd

ANACONDA

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

In [ ]:
```
1 + 1 # ADDITION
```

In [ ]:
```
2-1
```

In [ ]:
```
3*4
```

In [ ]:
```
8 / 4 # Division
```

In [ ]:
```
8 / 5 #float division
```

In [ ]:
```
8/4 ## float division
```

In [ ]:
```
8 // 4 #integer division
```

In [ ]:
```
8 + 9 - 7
```

In [ ]:
```
8 + 8 - #syntax error:
```

In [ ]:
```
5 + 5 * 5
```

In [ ]:
```
(5 + 5) * 5 # BODMAS (Bracket || Oders || Divide || Multiply || Add || Substact)
```

In [ ]:
```
2 * 2 * 2 * 2 * 2 # exponentaion
```

In [ ]:
```
2 ** 5
```

In [ ]:
```
15 / 3
```

In [ ]:
```
10 // 3
```

In [ ]:
```
14 % 2 # Modulus
```

In [ ]:
```
15 %% 2
```

In [ ]:
```
a,b,c,d,e = 15, 7.8, 'nit', 8+9j, True

print(a)
print(b)
print(c)
print(d)
print(e)
```

In [ ]:
```
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
```

In [ ]:
```
type(c)
```

- So far we code with numbers(integer)
- Lets work with string

In [ ]:
```
'Naresh IT'
```

python inbuild function - print & you need to pass the parameter in print()

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

In [ ]:
```
print('naresh it')
```

In [ ]:
```
"max it technology"
```

In [ ]:
```
s1 = 'naresh it technology'
s1
```

In [ ]:
```
a = 2
b = 3

a + b
```

In [ ]:
```
c = a + b
c
```

In [ ]:
```
a = 3
b = 'hi'
type(b)
```

In [ ]:
```
a + b
```

In [ ]:
```
print('naresh it's 'Technology')
```

In [ ]:
```
print('naresh it\'s"Technology"') #\ has some special meaning to ignore the erro
```

In [ ]:
```
print('naresh it',  'Technology')
```

In [ ]:
```
print("naresh it',  'Technology")
```

In [ ]:
```
# print the nit 2 times
'nit' + ' nit'
```

In [ ]:
```
'nit'  ' nit'
```

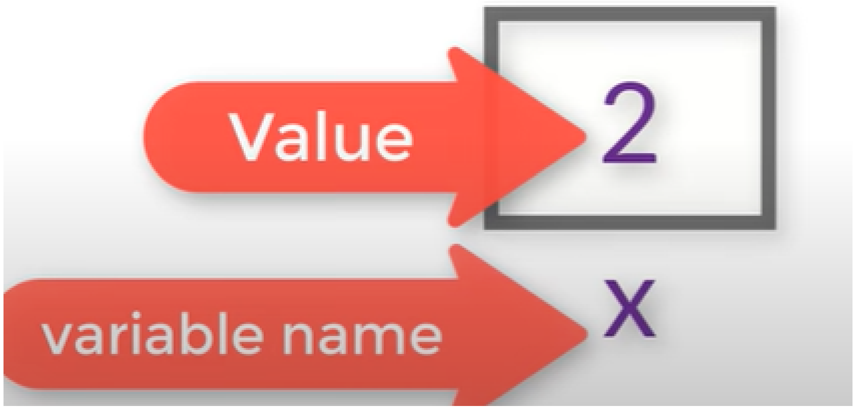
In [ ]:
```
#5 time print
5 * 'nit'
```

In [ ]:
```
5 *' nit ' # soace between words
```

In [ ]:
```
print('c:\nit') #\n -- new line
```

In [ ]:
```
print(r'c:\nit') #raw string
```

variable || identifier || object



In [ ]:
```
2
```

In [ ]:
```
x = 2 #x is variable/identifier/objec, 2 is the value
x
```

In [ ]:
```
x + 3
```

In [ ]:
```
y = 3
y
```

In [ ]:
```
x + y
```

In [ ]:
```
x = 9
x
```

In [ ]:
```
x + y
```

In [ ]:
```
x + 10
```

In [ ]:
```
y
```

In [ ]:
```
_ + y # _ understand the previous result of the
```

In [ ]:
```
y
```

In [ ]:
```
_ + y
```

In [ ]:
```
_ + y
```

In [ ]:
```
_ + y
```

In [ ]:
```
# string variable
name = 'mit'
```

In [ ]:
```
name
```

In [ ]:
```
name + 'technology'
```

In [ ]:
```
name + ' technology'
```

In [ ]:
```
name 'technology'
```

In [ ]:
```
name
```

In [ ]:
```
len(name)
```

In [ ]:
```
name[0] #python index begins with 0
```

In [ ]:
```
name[5]
```

In [ ]:
```
name[7]
```

In [ ]:
```
name[-1]
```

In [ ]:
```
name[-2]
```

In [ ]:
```
name[-6]
```

slicing

In [ ]:
```
name
```

In [ ]:
```
name[0:1] #to print 2 character
```

In [ ]:
```
name[0:2]
```

In [ ]:
```
name[1:4]
```

In [ ]:
```
name
```

In [ ]:
```
name[1:]
```

In [ ]:
```
name[:4]
```

In [ ]:
```
name[3:9]
```

In [ ]:
```
name1 = 'fine'
name1
```

In [ ]:
```
name1[0:1]
```

In [ ]:
```
name1[0:1] = 'd' # i want to change 1st character of naresh (n) - t
```

In [ ]:
```
name1
```

In [ ]:
```
name1[0] = 'd' #strings in python are immutable
```

In [ ]:
```
name1
```

In [ ]:
```
name1[1:]
```

In [ ]:
```
'd' + name1[1:] #i want to change fine to dine
```

In [ ]:
```
len(name1) #python inbuild function
```

List

In [ ]:
```
# LIST LIST LIST
nums = [10,20,30]
nums
```

In [ ]:
```
nums[0]
```

In [ ]:
```
nums[-1]
```

In [ ]:
```
nums[1:]
```

In [ ]:
```
nums[:1]
```

In [ ]:
```
num1 = ['hi', 'hallo']
```

In [ ]:
```
num1
```

In [ ]:
```
num2 = ['hi', 8.9, 34] # we can assign multiple variable
num2
```

In [ ]:
```
# can we have 2 list together
num3 = [nums, num1]
```

In [ ]:
```
num3
```

In [ ]:
```
num4 = [nums, num1, num2]
```

In [ ]:
```
num4
```

In [ ]:
```
nums
```

In [ ]:
```
nums.append(45)
```

In [ ]:
```
nums
```

In [ ]:
```
nums.remove(45)
```

In [ ]:
```
nums
```

In [ ]:
```
nums.pop(1)
```

In [ ]:
```
nums
```

In [ ]:
```
nums.pop() #if you dont assign the index element then it will consider by defaul
```

In [ ]:
```
nums
```

In [ ]:
```
num1
```

In [ ]:
```
num1.insert(2,'nit') #insert the value as per index values i.e 2nd index we are
```

In [ ]:
```
num1
```

In [ ]:
```
num1.insert(0, 1)
```

In [ ]:
```
num1
```

In [ ]:
```
#if you want to delate multiple value
num2
```

In [ ]:
```
del num2[2:]
```

In [ ]:
```
num2
```

In [ ]:
```
# if you need to add multiple values
num2.extend([29,15,20])
```

In [ ]:
```
num2
```

In [ ]:
```
num3
```

In [ ]:
```
num3.extend(['a', 5, 6.7])
```

In [ ]:
```
num3
```

In [ ]:
```
nums
```

In [ ]:
```
min(nums) #inbuild function
```

In [ ]:
```
max(nums) #inbuild function
```

In [ ]:
```
num1
```

In [ ]:
```
min(num1)
```

In [ ]:
```
sum(nums) #inbuild function
```

In [ ]:
```
nums.sort() #sort method
```

In [ ]:
```
nums
```

Tuple

In [ ]:
```
# TUPLE TUPLE TUPLE
tup = (15,25, 35)
tup
```

In [ ]:
```
tup[0]
```

In [ ]:
```
tup[0] = 10
```

as we are unable to change any value or parameter in tuple so iteration very faster in tuple compare to list

SET

In [ ]:
```
# SET SET SET
S = {}
```

In [ ]:
```
s1 = {21,6,34,58,5}
```

In [ ]:
```
s1
```

In [ ]:
```
s3= {50,35,53,'nit', 53}
```

In [ ]:
```
s3
```

In [ ]:
```
s1[1] #as we dont have proper sequencing thats why indexing not subscriptable
```

DICTIONARY

In [ ]:
```
# DICTIONARY DICTIONARY DICTIONARY
data = {1:'apple', 2:'banana',4:'orange'}
data
```

In [ ]:
```
data[4]
```

In [ ]:
```
data[3]
```

In [ ]:
```
data.get(2)
```

In [ ]:
```
data.get(3)
```

In [ ]:
```
print(data.get(3))
```

In [ ]:
```
data.get(1,'Not Fount')
```

In [ ]:
```
data.get(3,'Not Found')
```

In [ ]:
```
data[5] = 'five'
```

In [ ]:
```
data
```

In [ ]:
```
del data [5]
```

In [ ]:
```
data
```

In [ ]:
```
#list in the dictionary
prog = {'python':['vscode', 'pycharm'], 'machine learning' : 'sklearn', 'datasci
```

In [ ]:
```
prog
```

In [ ]:
```
prog['python']
```

In [ ]:
```
prog['machine learning']
```

In [ ]:
```
prog['datascience']
```

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()
```

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

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

introduce to ID()

```
In [ ]: # variable address
        num = 5
        id(num)
```

```
In [ ]: name = 'nit'
        id(name) #Address will be different for both
```

```
In [ ]: a = 10
        id(a)
```

```
In [ ]: b = a #thats why python is more memory efficient
```

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

```
In [ ]: id(10)
```

```
In [ ]: k = 10
        id(k)
```

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

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

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)

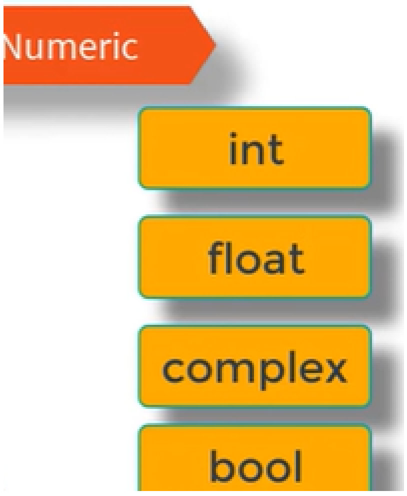
```
In [ ]: PI = 3.14  #in math this is alway constant but python we can chang
        PI
```

In [ ]:
```
PI = 3.18
PI
```

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

DATA TYPES & DATA STRUCTURES-->

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

1- NUMERIC :- 

INT || FLOAT || COMPLEX || BOOL

None
Numeric
List
Tuple
Set
String
Range
Dictionary

In [ ]:
```
w = 2.5
type(w)
```

In [ ]:
```
a
```

In [ ]:
```
(a)
```

In [ ]:
```
w2 = 2 + 3j #so hear j is represent as root of -1
type(w2)
```

In [ ]:
```
#convert flot to integer
a = 5.6
b = int(a)
```

In [ ]:
```
b
```

In [ ]:
```
type(b)
```

In [ ]:
```
type(a)
```

In [ ]:
```
k = float(b)
```

In [ ]:
```
k
```

In [ ]:
```
print(a)
print(b)
print(k)
```

In [ ]:
```
k1 = complex(b,k)
```

In [ ]:
```
print(k1)
type(k1)
```

In [ ]:
```
b < k
```

In [ ]:
```
condition = b<k
condition
```

In [ ]:
```
type(condition)
```

In [ ]:
```
int(True)
```

In [ ]:
```
int(False)
```

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

In [ ]:
```
s = {1,2,3,4}
s
```

In [ ]:
```
type(s)
```

In [ ]:
```
s1 = {1,2,3,4,4,3,11} #duplicates are not allowed
s1
```

In [ ]:
```
t = (10,20,30)
t
```

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

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

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

range()

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

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

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

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

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

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

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

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

```
In [ ]: d.values()
```

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

```
In [ ]: # other way to get value as
        d.get(2)
```

operator's in python

Arithmetic Operators
Assignment Operators
Relational Operators
Logical Operators
Unary Operator

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

Arithmetic operator

```
In [ ]: x1, y1 = 10, 5
```

```
In [ ]: x1  + y1
```

```
In [ ]: x1 - y1
```

```
In [ ]: x1 * y1
```

```
In [ ]: x1 / y1
```

```
In [ ]: x1 // y1
```

```
In [ ]: x1 % y1
```

```
In [ ]: x1 ** y1
```

```
In [ ]: 2 ** 3
```

Assignment operator

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

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

In [ ]:
```
x
```

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

In [ ]:
```
x
```

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

In [ ]:
```
x
```

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

In [ ]:
```
x
```

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

In [ ]:
```
x
```

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

In [ ]:
```
x
```

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

In [ ]:
```
a
```

In [ ]:
```
b
```

unary operator

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

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

In [ ]:
```
m
```

In [ ]:
```
n
```

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

Relational operator

we are using this operator for comparing

In [ ]:
```
a = 5
b = 7
```

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

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

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

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

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

In [ ]:
```
a = 10
```

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

In [ ]:
```
# hear if i change b = 6
b = 10
```

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

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

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

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

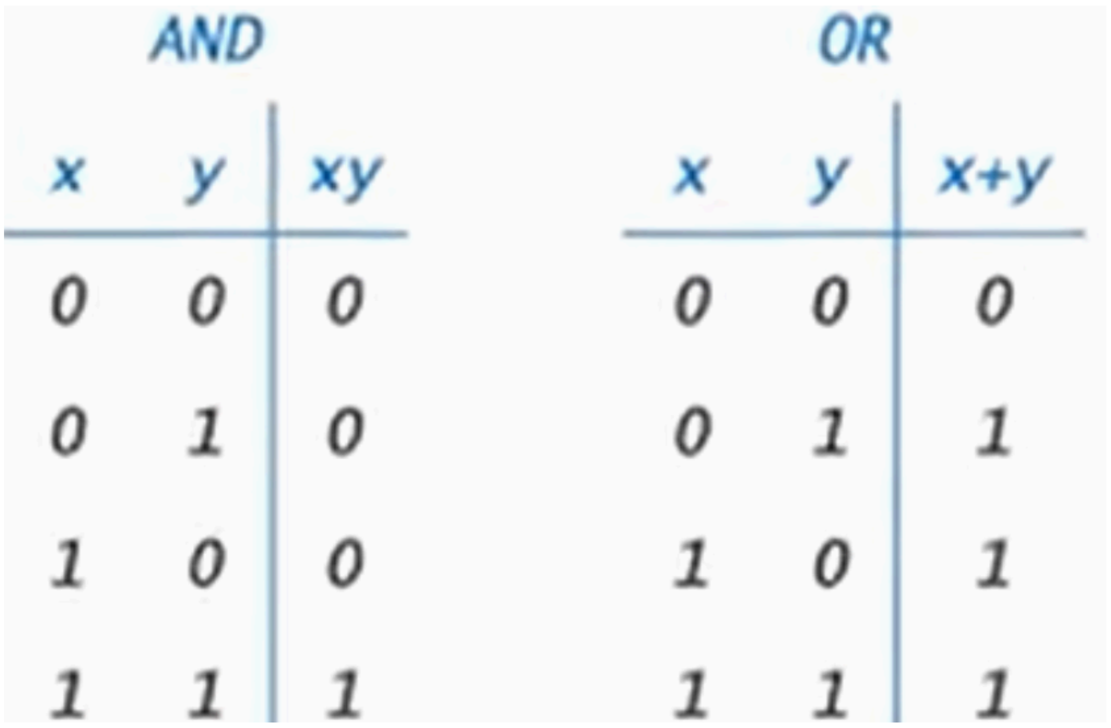
In [ ]:
```
a>b
```

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

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

LOGICAL OPERATOR

AND, OR, NOT



AND			OR		
x	y	xy	x	y	x+y
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

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

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

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

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

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

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

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

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

```
In [ ]: x
```

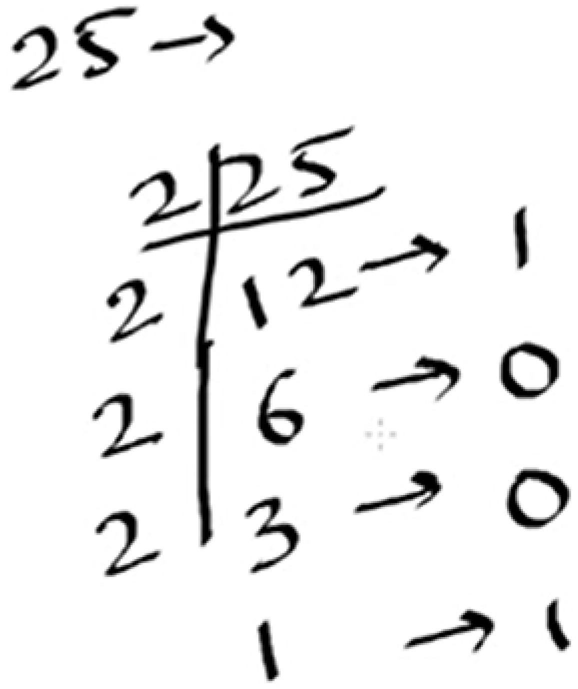
```
In [ ]: not x
```

Number system coverstion (bit-binary digit)

binary : base (0-1) --> please divide 15/2 & count in reverse order octal : base (0-7) hexadecimal : base (0-9 & then a-f) when you check ipaddress you will these format --> cmd - ipconfig

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

```
In [ ]: 0b11001
```



```
In [ ]: int(0b11001)
```

```
In [ ]: bin(35)
```

```
In [ ]: int(0b100011)
```

```
In [ ]: bin(20)
```

```
In [ ]: int(0b10100)
```

```
In [ ]: 0b1111
```

```
In [ ]: oct(15)
```

```
In [ ]: 0o17
```

```
In [ ]: hex(9)
```

```
In [ ]: 0xf
```

```
In [ ]: hex(10)
```

```
In [ ]: 0xa
```

```
In [ ]: hex(25)
```

```
In [ ]: 0x19
```

In [ ]:
```
0x15
```

swap variable in python

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

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

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

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

In [ ]:
```
print(a)
print(b)
```

In [ ]:
```
# in above scenario we lost the value 5
a1 = 7
b1 = 8
```

In [ ]:
```
temp = a1
a1 = b1
b1 = temp
```

In [ ]:
```
print(a1)
print(b1)
```

In [ ]:
```
a2 = 5
b2 = 6
```

In [ ]:
```
#swap variable formulas
a2 = a2 + b2
b2 = a2 - b2
a2 = a2 - b2
```

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

In [ ]:
```
print(0b101) # 101 is 3 bit
print(0b110) # 110 also 3bit
```

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

In [ ]:
```
#there is other way to work using swap variable also which is XOR because it wil
a2 = a2 ^ b2
b2 = a2 ^ b2
a2 = a2 ^ b2
```

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

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

In [ ]:
```
a2 , b2 = b2, a2
```

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

BITWISE OPERATOR

- WE HAVE 6 OPERATORS

COMPLEMENT (~) || AND (&) || OR (|) || XOR (^) || LEFT SHIFT (<<) || RIGHT SHIFT (>>)

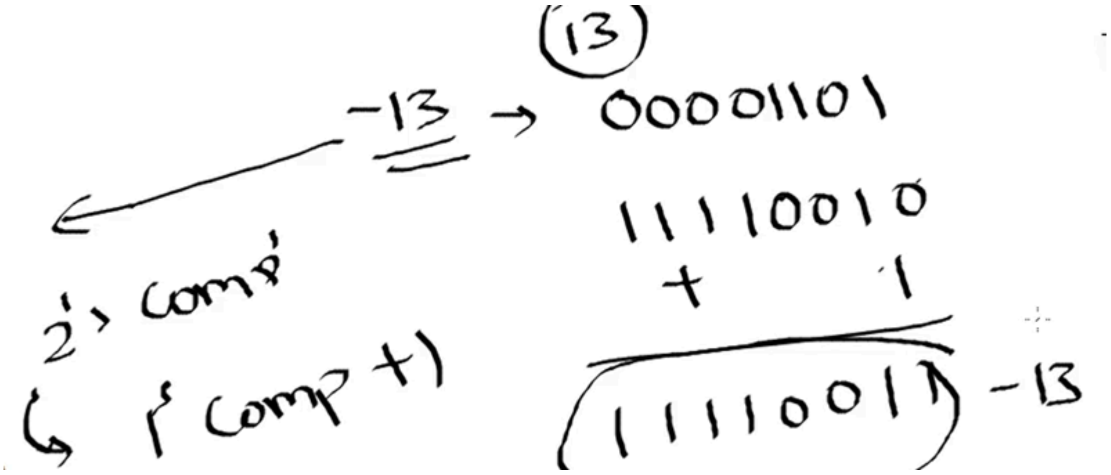
In [ ]:
```
print(bin(12))
print(bin(13))
```

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)

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

In [ ]:
```
~45
```

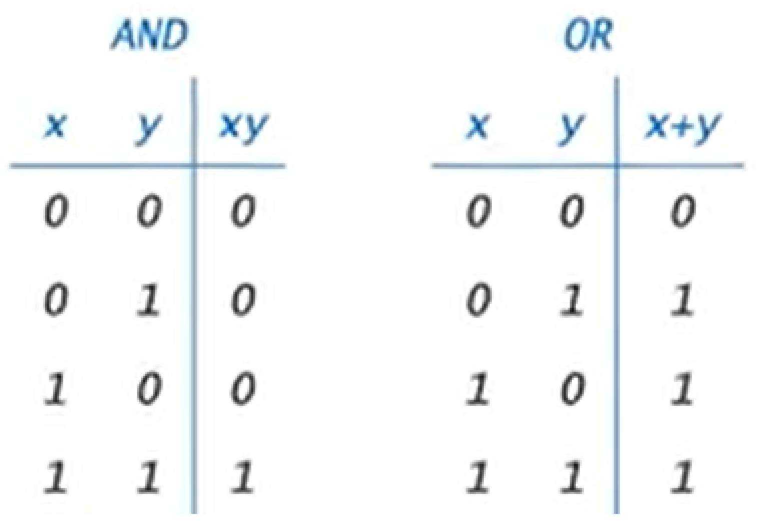
In [ ]:
```
~6
```

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

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

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

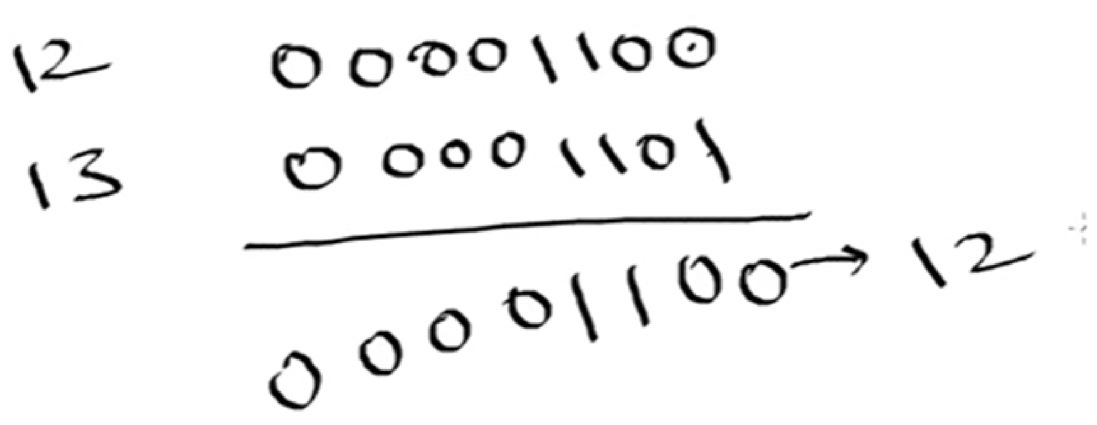


AND

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



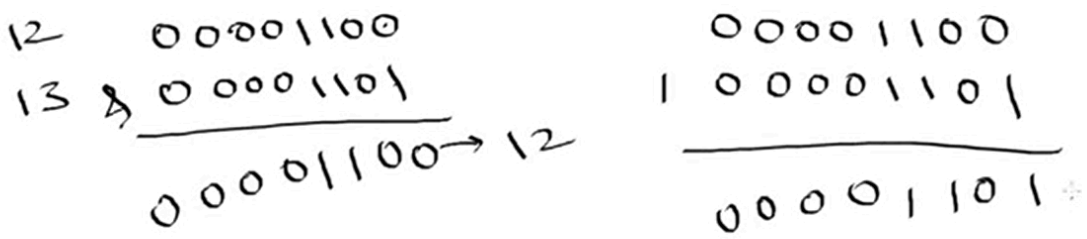
In [ ]:
```
12 & 13
```

In [ ]:
```
1 & 1
```

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

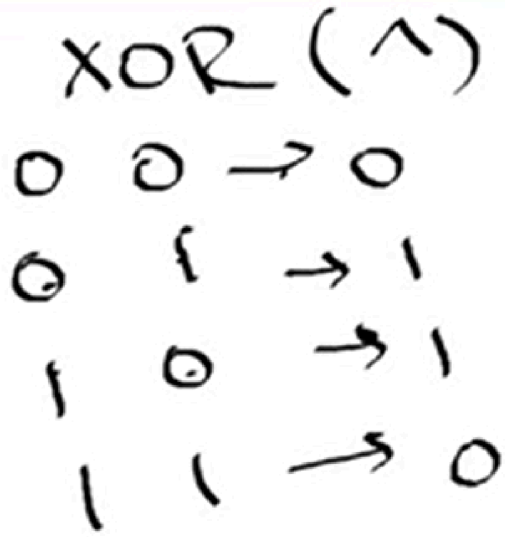
In [ ]:
```
1 & 0
```

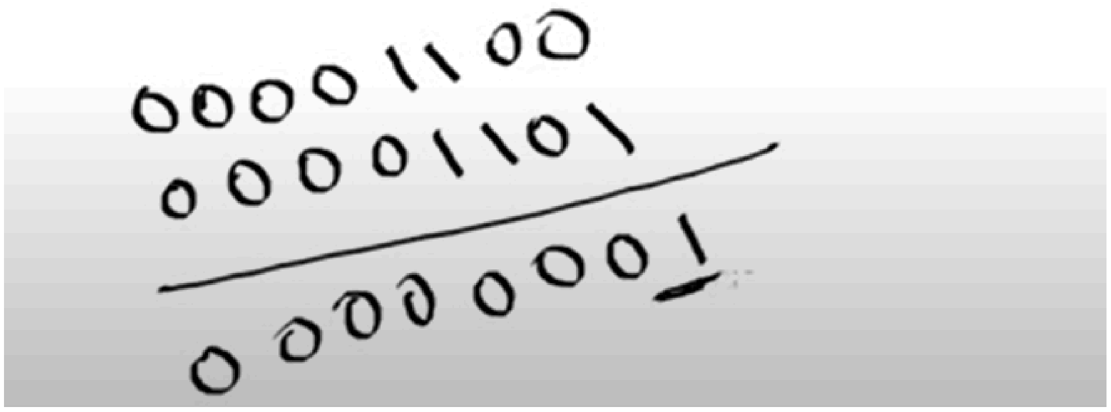
In [ ]:
```
12 | 13
```



In [ ]:
```
35 & 40 #please do the homework conververt 35,40 to binary format
```

In [ ]:
```
35 | 40
```





In [ ]:
```
# in XOR if the both number are different then we will get 1 or else we will get

12 ^ 13
```

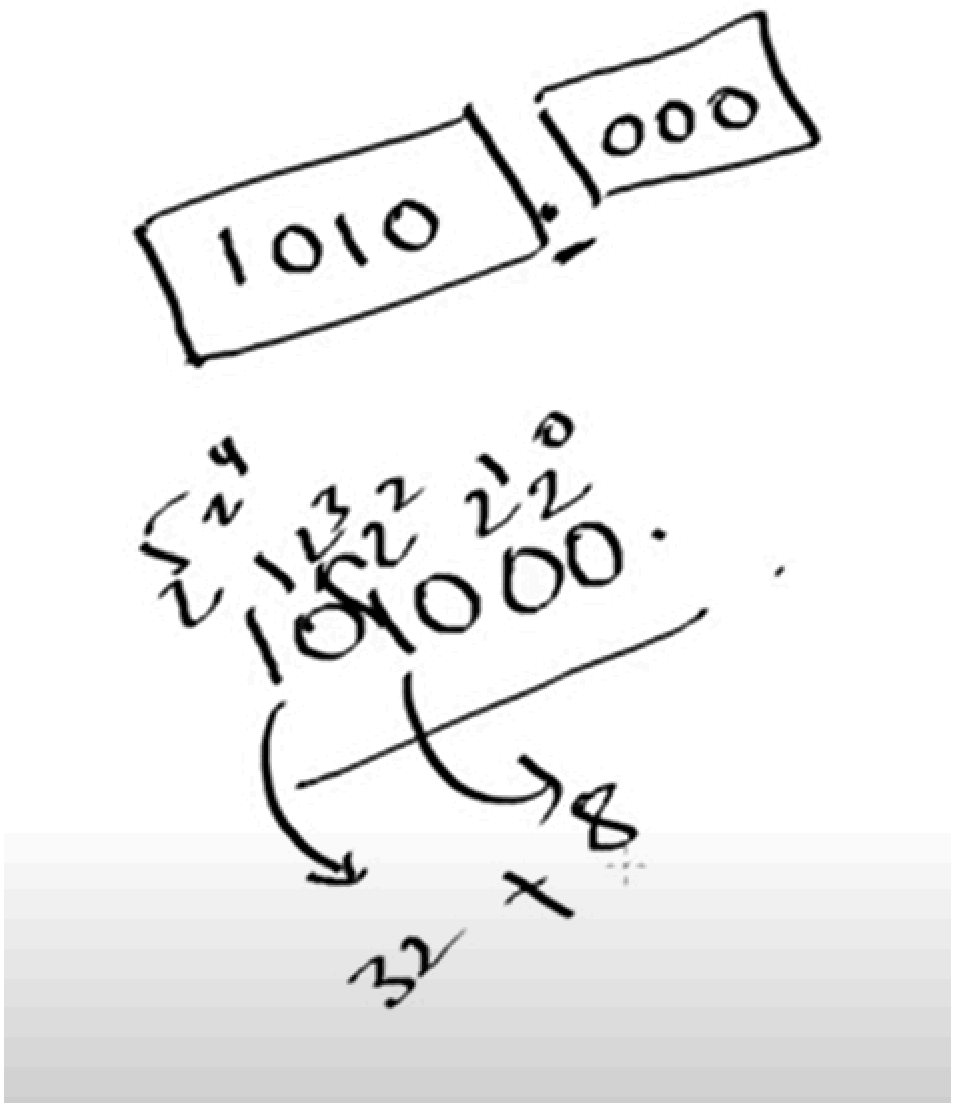
In [ ]:
```
25 ^ 30
```

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

In [ ]:
```
bin(30)
```

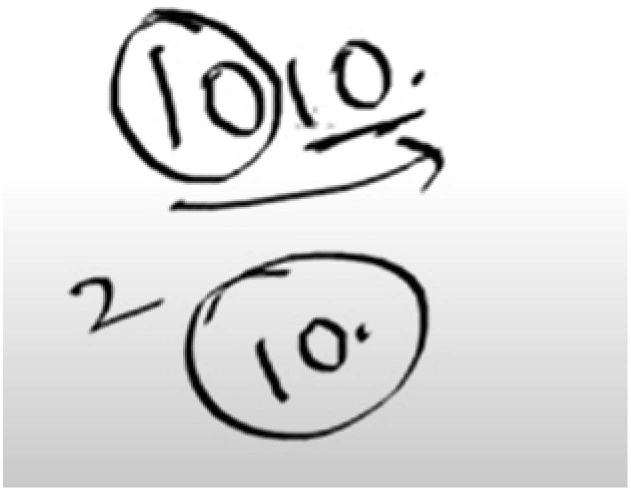
In [ ]:
```
int(0b000111)
```

In [ ]:
```
# BIT WISE LEFT OPERATOR
#bit wise left operator bydefault you will take 2 zeros ( )
#10 binary operator is 1010 | also i can say 1010
10<<2
```



In [ ]:
```
20<<4 #can we do this
```

BITWISE RIGHTSHIFT OPERATOR



In [ ]:
```
10>>2
```

In [ ]:
```
bin(20)
```

In [ ]:
```
20>>4
```

import math module

https://docs.python.org/3/library/math.html

In [ ]:
```
x = sqrt(25) #sqrt is inbuild function
```

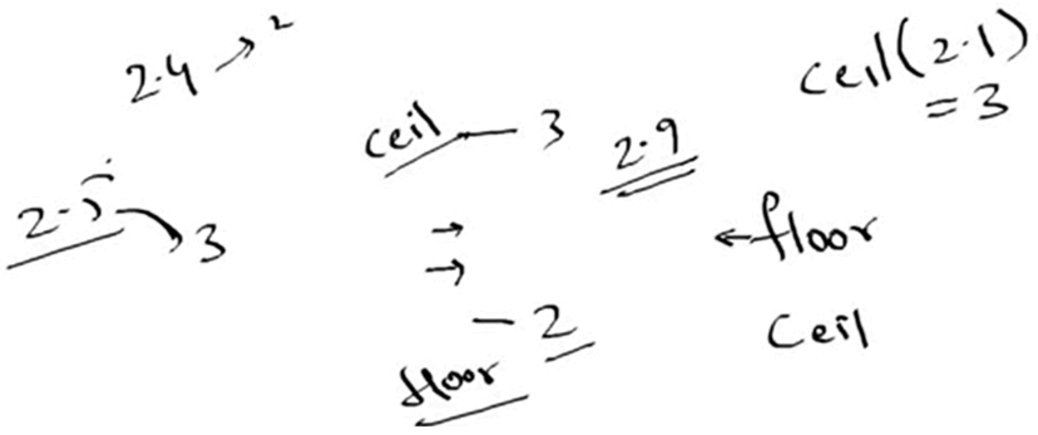
In [ ]:
```
import math # math is module
```

In [ ]:
```
x = math.sqrt(25)
x
```

In [ ]:
```
x1 = math.sqrt(15)
x1
```

In [ ]:
```
print(math.floor(2.9)) #floor - minimum or least value
```

In [ ]:
```
print(math.ceil(2.9)) #ceil - maximum or highest value
```



In [ ]:
```
print(math.pow(3,2))
```

In [ ]:
```
print(math.pi) #these are constant
```

In [ ]:
```
print(math.e) #these are constant
```

In [ ]:
```
import math as m
m.sqrt(10)
```

In [ ]:
```
from math import sqrt,pow # math has many function if you want to call specific
pow(2,3)
```

In [ ]:
```
from math import * # math has many function if you want to call specific functio

print(pow(2,3))
print(floor(2.3))
```

In [ ]:
```
round(pow(2,3))
```

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

# pycharm run debug # how to install python idle # how to install pycharm & starts working on pycharm

user input function in python || command line input

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

In [ ]:
```
x1 = input('Enter the 1st number') #whenevery you works in input function it alw
y1 = input('Enter the 2nd number') # it wont understand as arithmetic operator
z1 = x1 + y1
print(z1)
```

In [ ]:
```
type(x1)
type(y1)
```

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

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

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

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

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

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

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

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

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

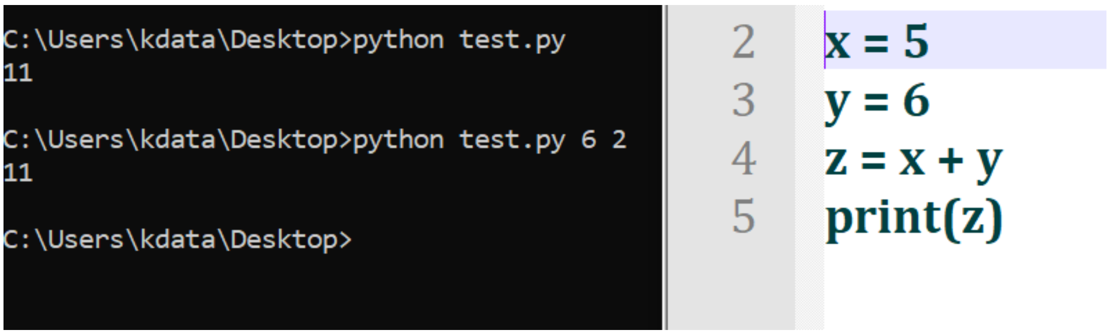
In [ ]:
```
ch = input('enter a char')[1:3]
print(ch)
```

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

EVAL function using input

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

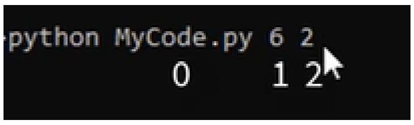
if you want to pass the value in cmd can we pass the value like this



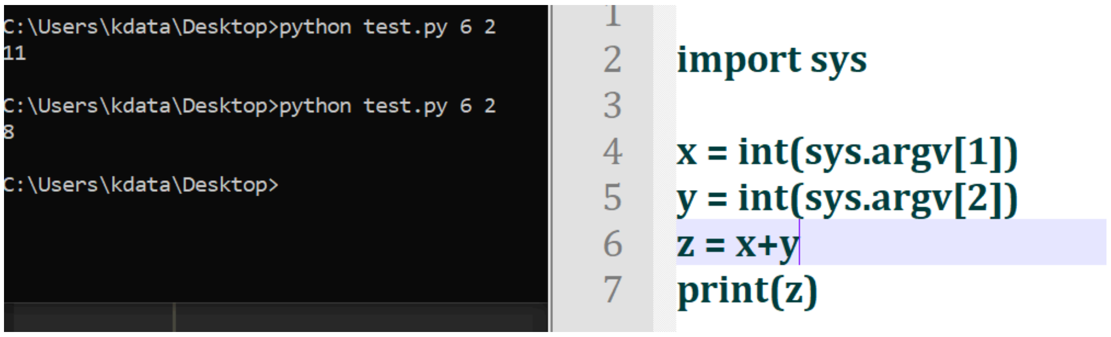
```
C:\Users\kdata\Desktop>python test.py
11

C:\Users\kdata\Desktop>python test.py 6 2
11

C:\Users\kdata\Desktop>
```

```
x = 5
y = 6
z = x + y
print(z)
```

when we run the above code in cmd then we get the value to 11 only but how to add them then we need to use very important concept called (argv) -- (argument values) what it does if you pass 1 value then 1 value it will display but if you pass 2 value then it will display 2 values

argv -- it will understand based on index number & bydefault index number 0 means that is file name 

```
python MyCode.py 6 2
       0         1 2
```



```
C:\Users\kdata\Desktop>python test.py 6 2
11

C:\Users\kdata\Desktop>python test.py 6 2
8

C:\Users\kdata\Desktop>
```

```python

import sys

x = int(sys.argv[1])
y = int(sys.argv[2])
z = x+y
print(z)
```