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, llm , generati ai everywhere 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)

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 --> 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".

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\A3MAX SOFTWARE TECH\AppData\Local\Programs\Python\Python39\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 [ ]:
```
2 ** 5
```

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

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

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

In [ ]:
```
10 % 2
```

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

In [ ]:
```
3 + 'nit'
```

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('Max it')
```

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

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

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

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

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

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

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

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

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

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

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

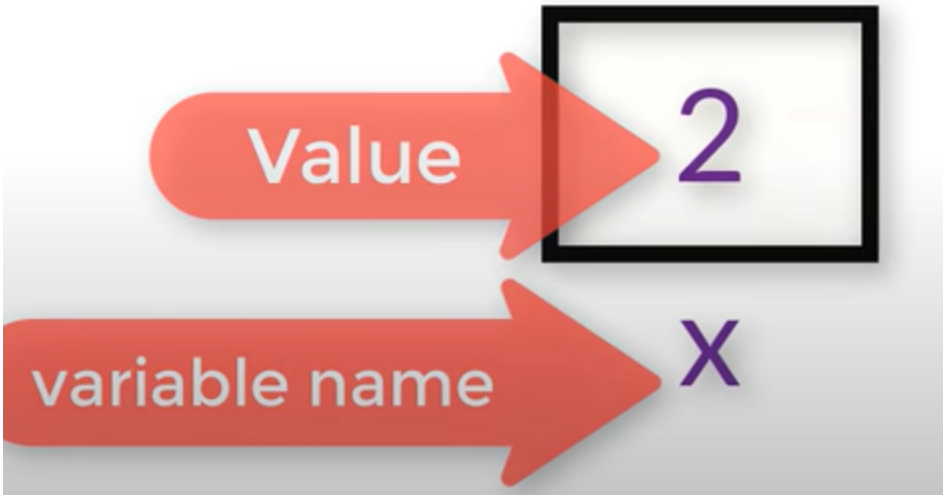
In [ ]:
```
#5 time print
5 * 'nit'
```

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 # _ understand the previous result of the
```

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

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

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

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[1:]
```

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

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

In [ ]:
```
name
```

In [ ]:
```
name1 = 'fine' # change the string fine to dine
name1
```

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

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

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 [ ]:
```
l = []
```

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 default l
```

In [ ]:
```
nums
```

In [ ]:
```
num1
```

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

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
```

In [ ]:
```
l = [1,2,3]
l
```

In [ ]:
```
l[0] = 100
l
```

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', 'datascienc
```

In [ ]:
```
prog
```

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

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

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

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

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 [ ]:
```
2 + 3
```

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)

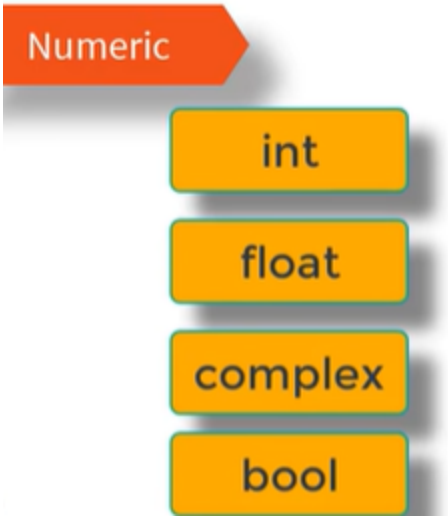
In [ ]:
```
PI = 3.14  #in math this is alway constant but python we can chang
PI
```

In [ ]:
```
PI = 3.15
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

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

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

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

In [ ]:
```
x2 = 3
y2 = 2
x2 ** y2
```

Assignment operator

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

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

In [ ]:
```
x
```

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

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

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

In [ ]:
```
x
```

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

In [ ]:
```
x
```

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

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

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

In [ ]:
```
a
```

In [ ]:
```
b
```

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

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

In [ ]:
```
n
```

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

Relational operator

we are using this operator for comparing

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

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

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

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

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

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

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

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

In [ ]:
```
a
```

In [ ]:
```
b
```

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

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

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

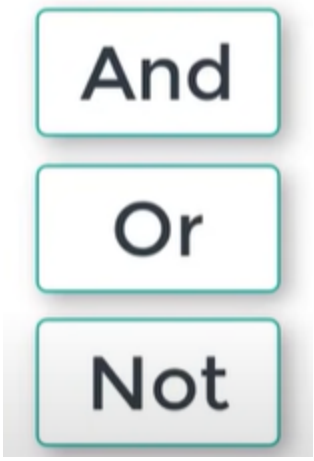
In [ ]:
```
a>b
```

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

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

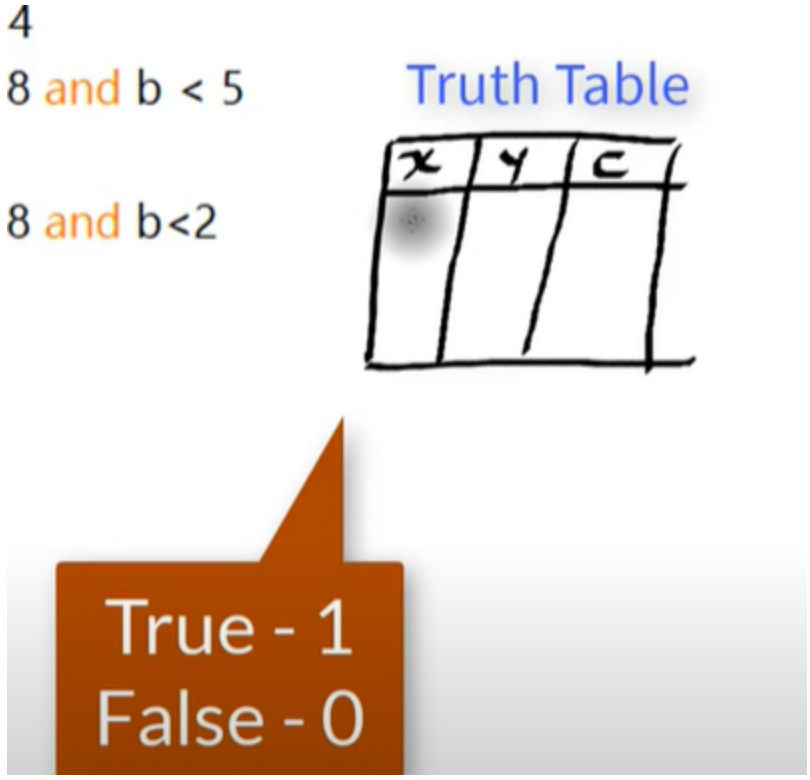
LOGICAL OPERATOR

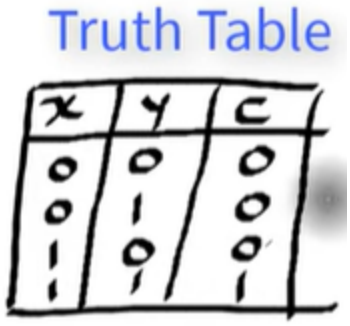
- logical operator you need to understand about true & false table

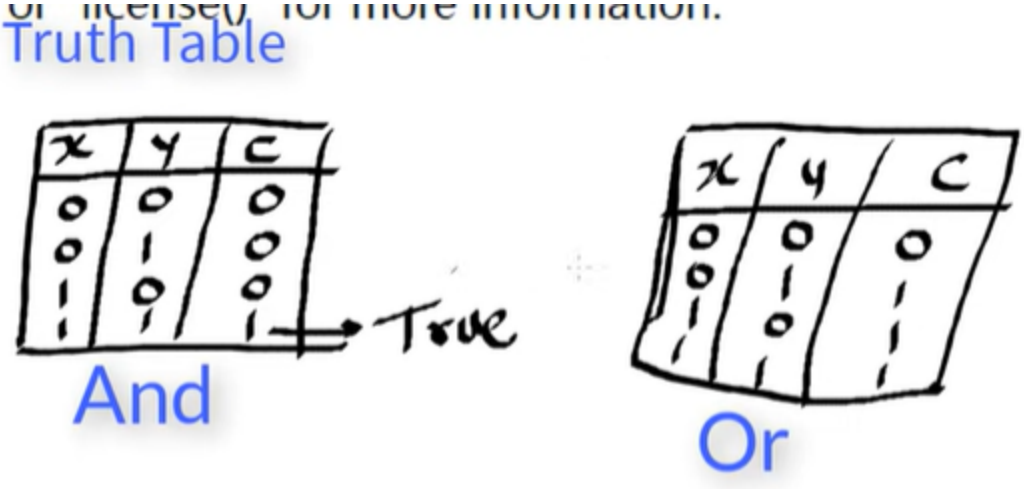


- 3 importand part of logical operator is --> AND, OR, NOT

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







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
```

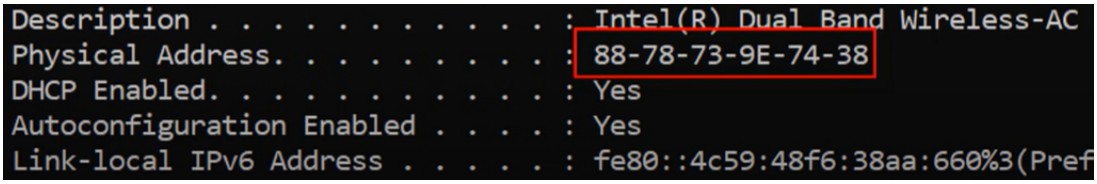
In [ ]:
```
x = not x
x
```

In [ ]:
```
x
```

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

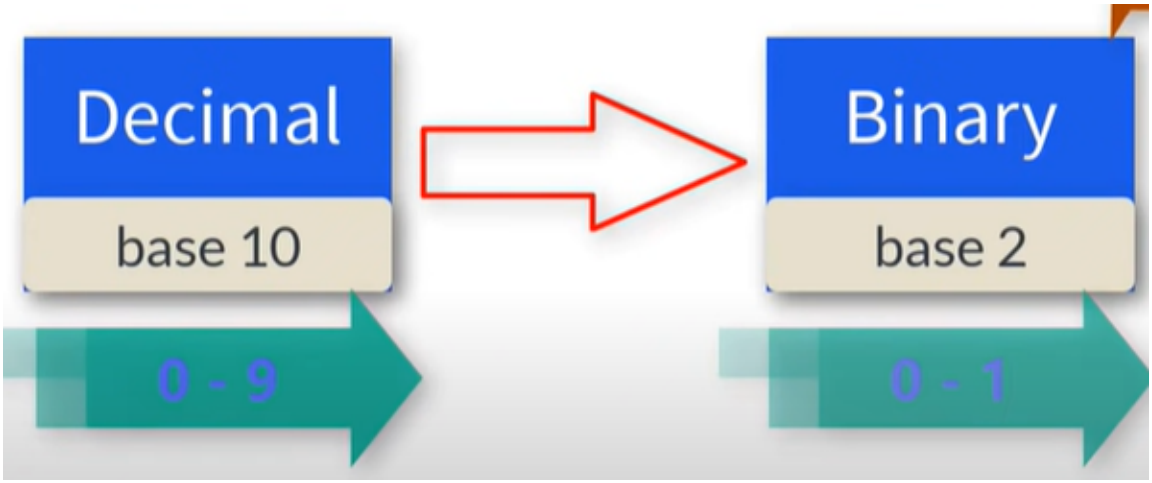
Number system coverstion (bit-binary digit)

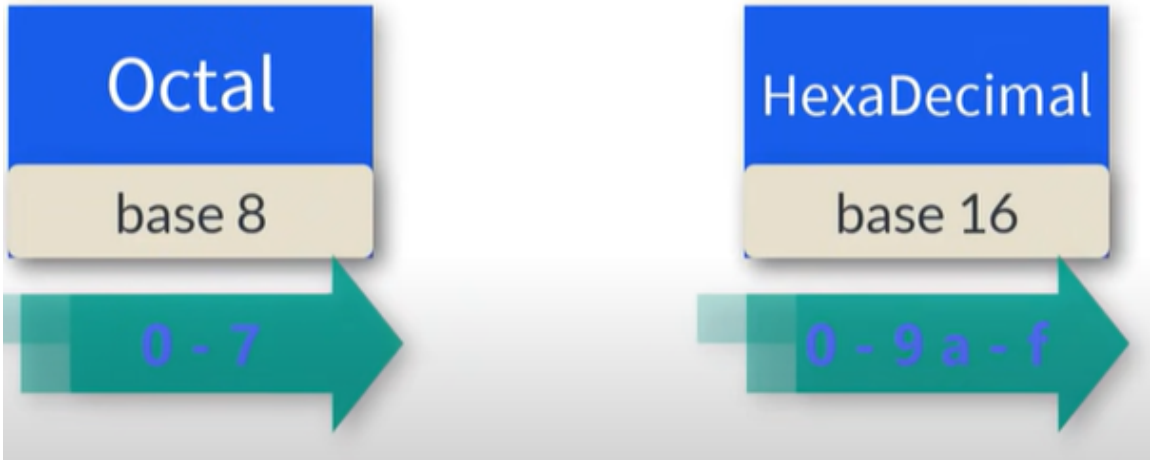
- In the programing we are using binary system, octal system, decimal system & hexadecimal system
- but where do we use this in cmd - you can check your ip address & lets understand how to convert from one system to other system
- when you check ipaddress you will these format --> cmd - ipconfig



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

hexadecimal : base (0-9 & then a-f)





In [1]:
```
25
```

Out[1]: 25

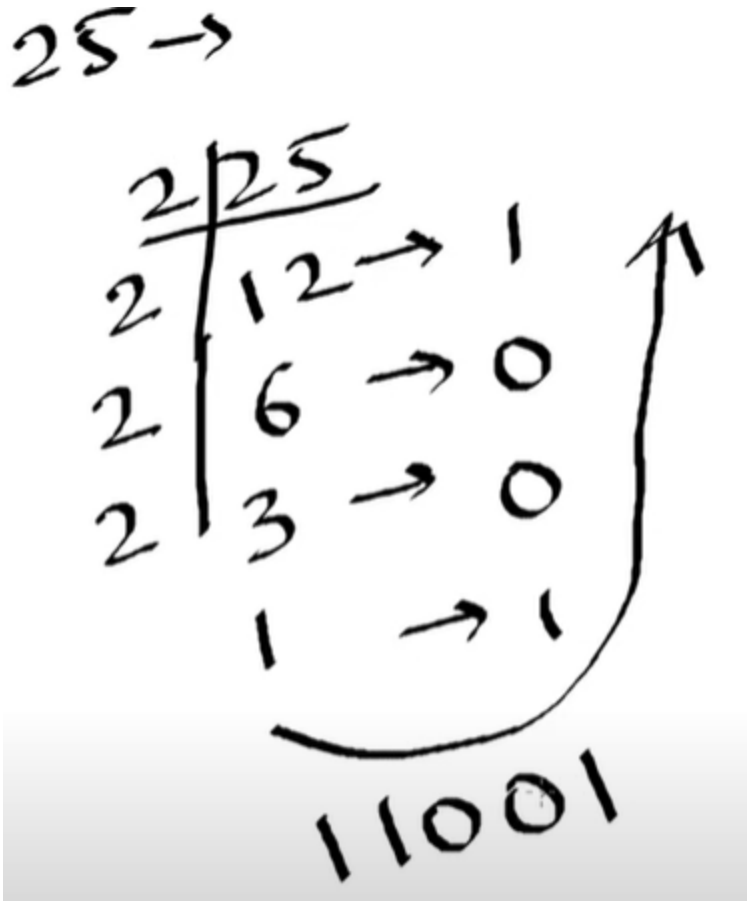
In [ ]:
```
bin(25)
```

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

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

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

In [ ]:
```
0o31
```



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

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

In [ ]:
```
bin(7)
```

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

In [ ]:
```
0o31
```

In [ ]:
```
int(0o31)
```

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

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

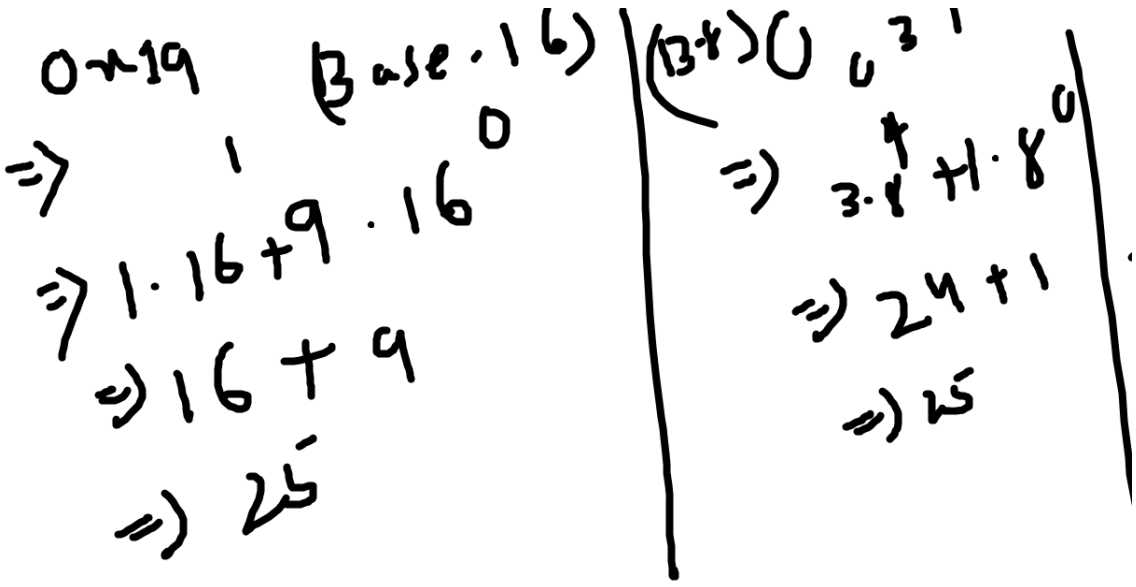
In [ ]:
```
hex(16)
```

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

In [ ]:
```
0xb
```

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

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



In [2]:
```
0x19
```

Out[2]: 25

In [3]:
```
0x15
```

Out[3]: 21

swap 2-variable in python

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

In [4]:
```
a = 5
b = 6
```

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

In [6]:
```
print(a)
print(b)
```

```
6
6
```

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

In [8]:
```
temp = a1
a1 = b1
b1 = temp
```

In [9]:
```
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 [13]:
```
a2 = 5
b2 = 6
```

In [14]:
```
#swap variable formulas without using 3rd formul
a2 = a2 + b2 # 5+6 = 11
```

```
b2 = a2 - b2 # 11-6 = 5
a2 = a2 - b2 # 11-5 = 6
```

In [15]:
```
print(a2)
print(b2)
```

```
6
5
```

In [18]:
```
0b110
```

Out[18]: 6

In [19]:
```
0b101
```

Out[19]: 5

In [16]:
```
print(0b110)
print(0b101)
```

```
6
5
```

In [21]:
```
print(0b101)
print(0b110)
```

```
5
6
```

In [20]:
```
#but when we use a2 + b2 then we get 11 that means we will get 4 bit which is 1 bit
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

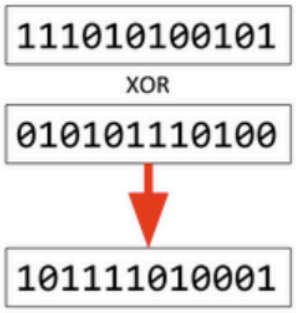
An XOR or *eXclusive OR* is a bitwise operation indicated by ^ and shown by the

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



In [22]:
```
print(a2)
print(b2)
```

```
6
5
```

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

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

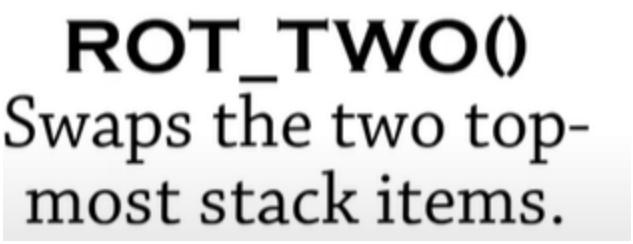
In [23]:
```
a2, b2
```

Out[23]:
```
(6, 5)
```

In [24]:
```
a2 , b2 = b2, a2 # how it work is b2 6 a2 is 5 first it goes into stack & then it r
```

In [25]:
```
print(a2)
print(b2)
```

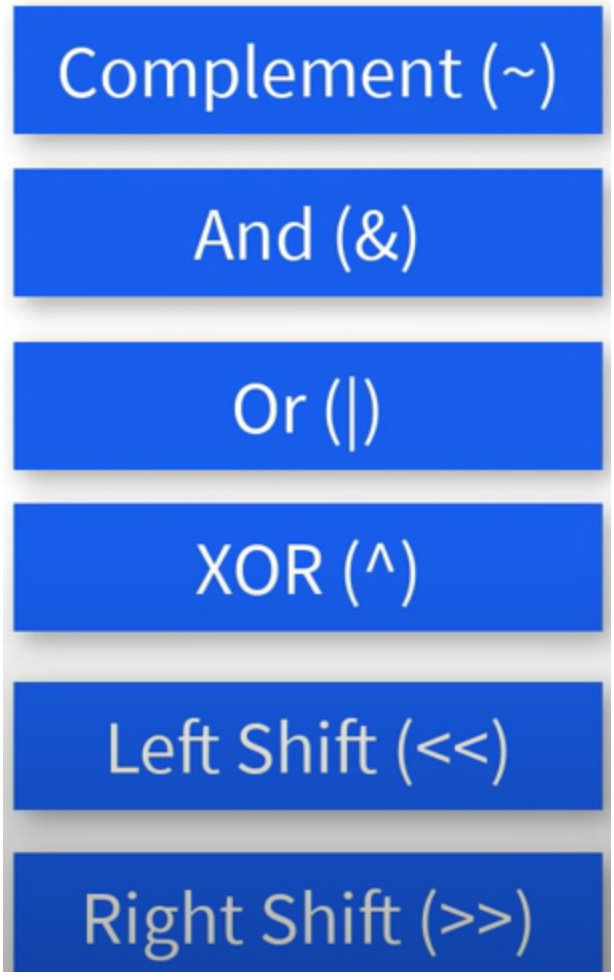
```
5
6
```



- internally it uses the rotational concept

BITWISE OPERATOR

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



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

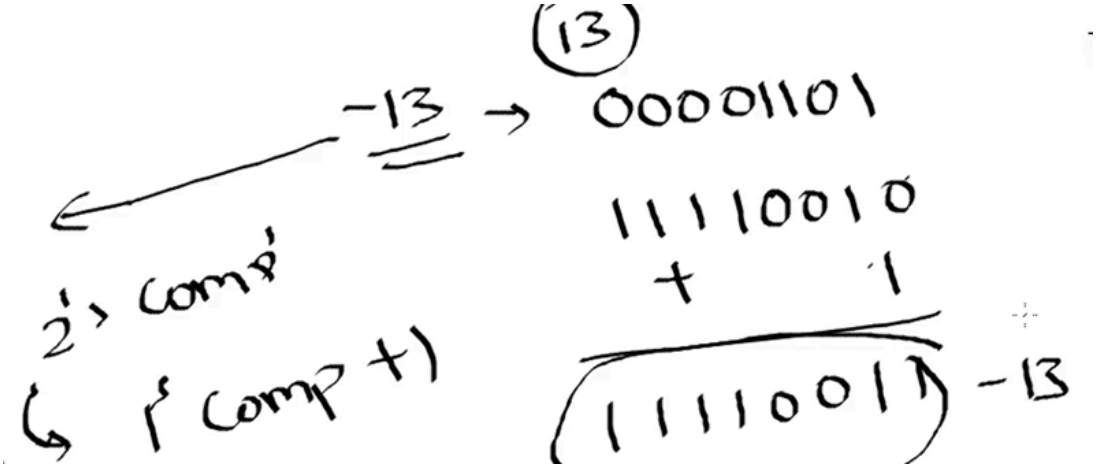
In [ ]:
```
0b1101
```

In [ ]:
```
0b1100
```

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 [1]:
```
# COMPLEMENT (~) (TILDE  OR TILD)
~12 # why we get -13 . first we understand what is complment means (reversr of bina
```

Out[1]:
```
-13
```

In [2]:
```
~46
```

Out[2]: -47

In [3]:
```
~54
```

Out[3]: -55

In [4]:
```
~10
```

Out[4]: -11

bit wise and operator

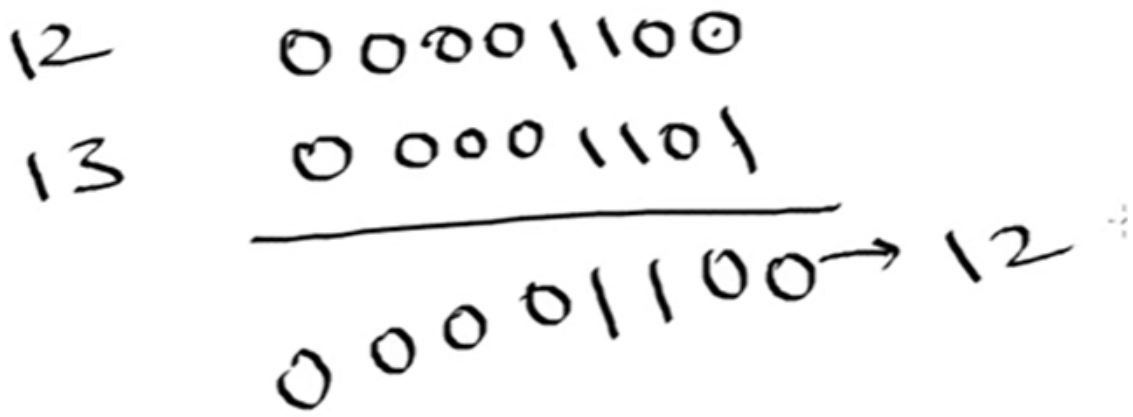
AND - LOGICAL OPERATOR ||| & - BITWISE AND OPERATOR
(we know that 1 & 1 is 1) 12 - 00001100 13 - 00001101 when we are add both then then outut we will get as 12

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 [5]:
```
12 & 13
```

Out[5]: 12

In [8]:
```
12 | 13
```

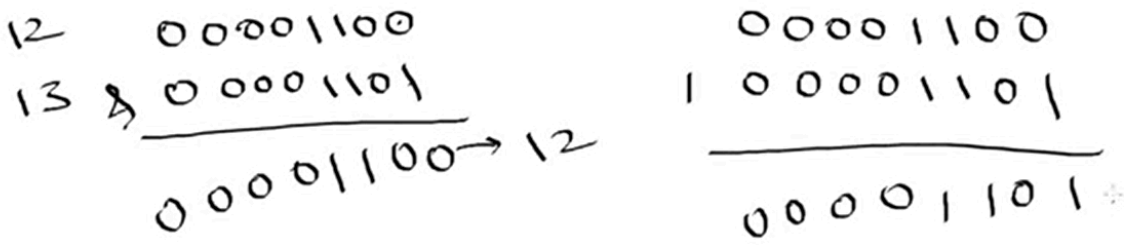
Out[8]: 13

In [9]:
```
1 & 0
```

Out[9]: 0

In [10]:
```
1 | 0
```

Out[10]: 1



In [11]:
```
bin(13)
```

Out[11]: '0b1101'

In [12]:
```
print(bin(35))
print(bin(40))
```

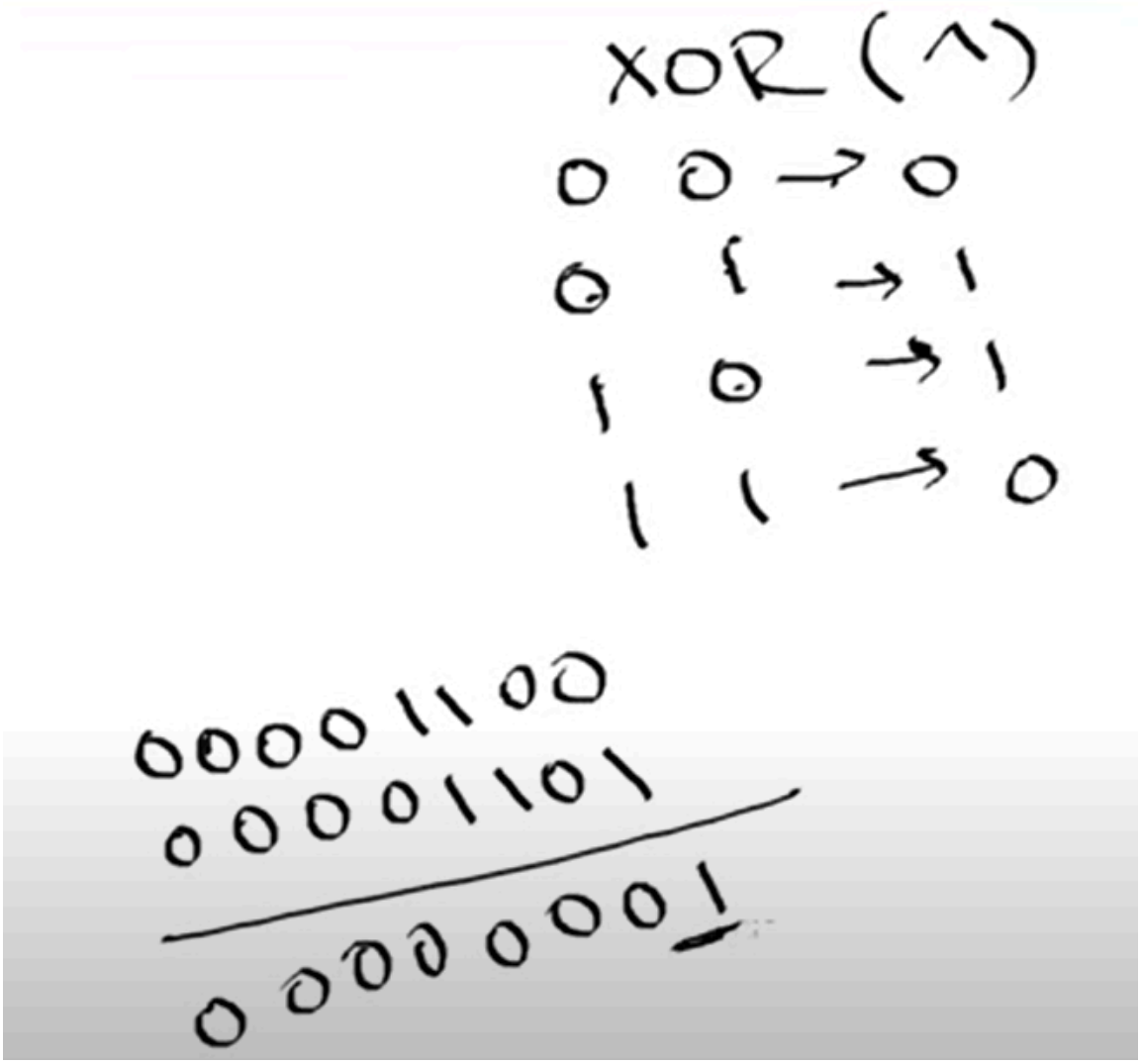
```
0b100011
0b101000
```

In [13]:
```
35 & 40 #please do the homework conververt 35,40 to binary format
```

Out[13]: 32

In [14]:
```
35 | 40
```

Out[14]: 43



In [15]:
```
# in XOR if the both number are different then we will get 1 or else we will get 0

12 ^ 13
```

Out[15]: 1

In [17]:
```
25^30
```

Out[17]: 7

In [ ]:
```
bin(7)
```

In [16]:
```
print(bin(25))
print(bin(30))
```

```
0b11001
0b11110
```

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

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

In [ ]:
```
0b00111
```

In [18]:
```
bin(10)
```

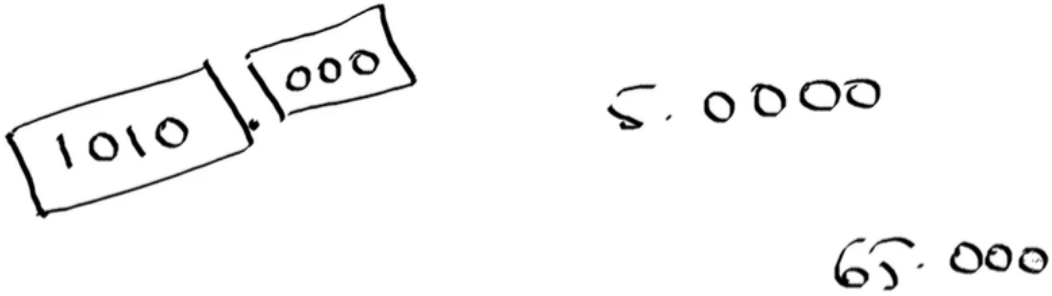
Out[18]: '0b1010'

In [19]:
```
10<<1
```

Out[19]: 20

In [20]:
```
10<<2
```

Out[20]: 40

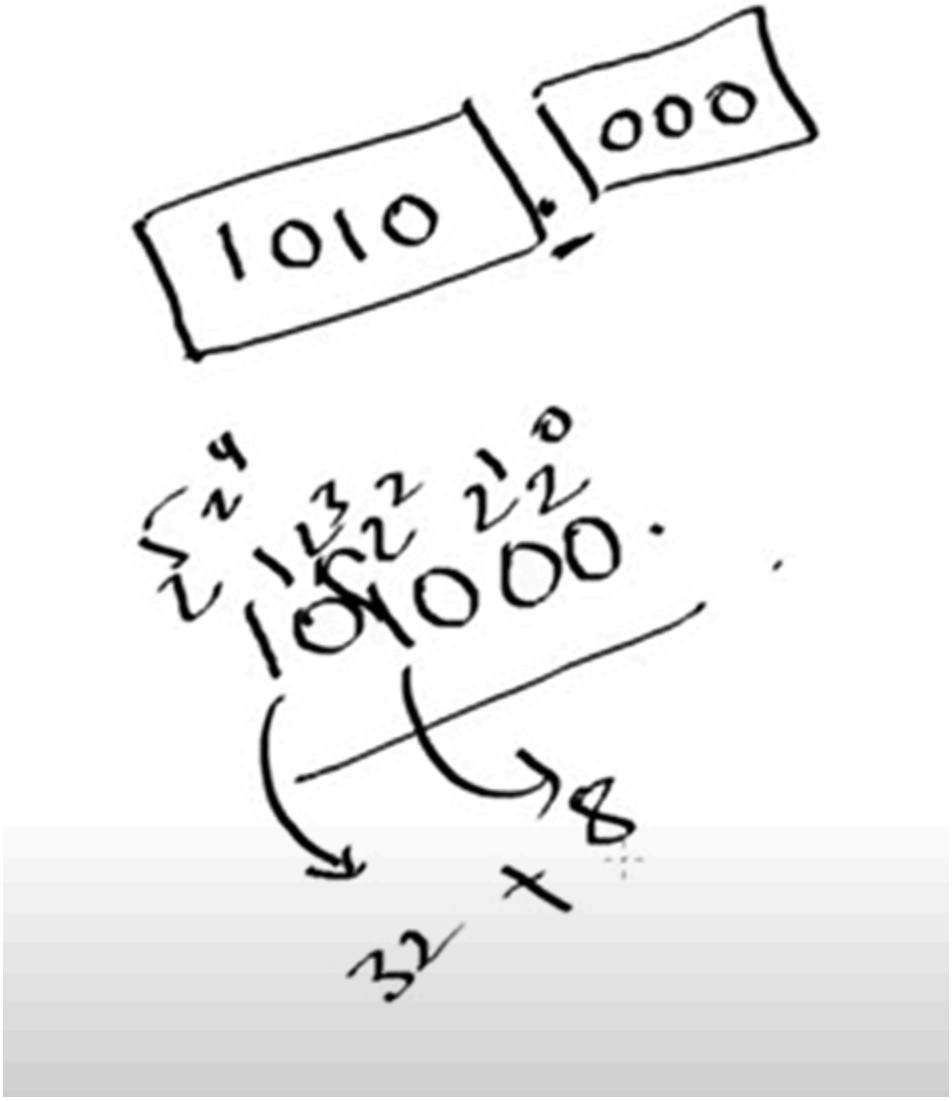


In [ ]:
```
bin(10)
```

In [ ]:
```
10<<1
```

In [ ]:
```
# BIT WISE LEFT SHIFT OPERATOR
# in left shift what we need to to we need shift in left hand side & need to shift
#bit wise left operator bydefault you will take 2 zeros ( )
#10 binary operator is 1010 | also i can say 1010
10<<2
```

In [ ]:
```
10<<3
```



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

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

BITWISE RIGHTSHIFT OPERATOR

- left side we are gaining the bits
- right side we are lossing bits

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

In [21]:
```
10 >> 1
```

Out[21]: 5

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

In [ ]:
```
10>>3
```

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

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

In [22]:
```
import keyword
keyword.kwlist
```

Out[22]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 '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']
```

import math function

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

In [23]:
```
x = sqrt(25) #sqrt is inbuild function
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 x = sqrt(25)

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

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

In [25]:
```
import math # math is module
```

In [26]:
```
x = math.sqrt(25)
x
```

Out[26]: 5.0

In [27]:
```
x1 = math.sqrt(15)
x1
```

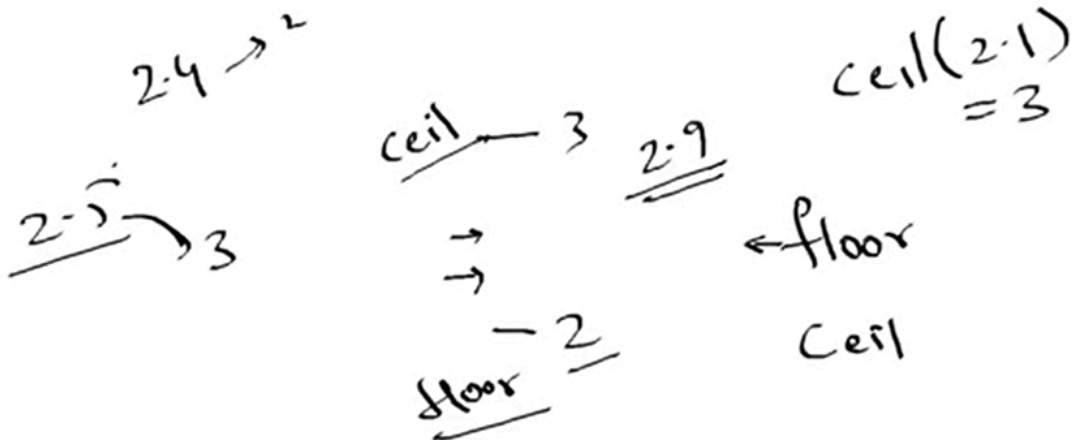
Out[27]: 3.872983346207417

In [28]:
```
print(math.floor(3.87)) #floor - minimum or least value
```

```
3
```

In [29]:
```
print(math.ceil(3.87)) #ceil - maximum or highest value
```

```
4
```



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

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

In [ ]:
```
print(math.e) # e - epsilon values
```

In [ ]:
```
m.sqrt(25)
```

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

In [ ]:
```
from math import sqrt, pow # math has many function if you want to import specific
print(pow(2,3))
print(m.sqrt(10))
```

In [ ]:
```
from math import sqrt, pow, floor, ceil # math has many function if you want to imp

print(pow(2,3))
print(sqrt(10))
```

```
print(floor(2.3))
print(ceil(2.3))
```

In [ ]:
```
from math import sqrt, pow

print(pow(2,3))
print(sqrt(10))
print(floor(2.3))
print(ceil(2.3))
```

In [ ]:
```
from math import *

print(pow(2,3))
print(sqrt(10))
print(floor(2.3))
print(ceil(2.3))
```

In [ ]:
```
from math import *

print(pow(2,3))
print(math.sqrt(10))
```

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

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

how to exeucute test file in using command line # pycharm run debug # how to install python idle # how to install pycharm & starts working on pycharm # run test1.py for run, debug # we are also using pycharm for loop & important deep learning & computer viso,Neural network concept

user input function in python || command line input

- how to get input from user

In [ ]:
```
x = input()
x
```

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

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 [ ]:
```
type(z)
```

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

```
z1 = x1 + y1
print(z1)
```

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

In [ ]:
```
x1 = input('Enter the 1st number') #whenevery you works in input function it always
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 [ ]:
```
st = input('enter a string')
print(st)
#print(type(ch))
```

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

In [ ]:
```
print(st[0:2])
```

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

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

In [ ]:
```
st = input('enter a string')[0]
print(st)
```

In [ ]:
```
st = input('enter a string')[1]
print(st)
```

In [ ]:
```
st = input('enter a string')[1:3]
print(st)
```

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

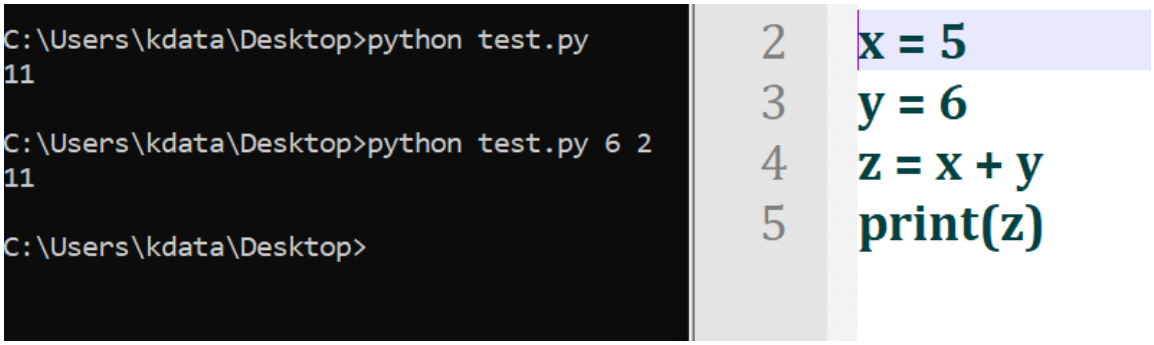
In [ ]:
```
result = input('enter an expr')
print(result)
```

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

EVAL function using input

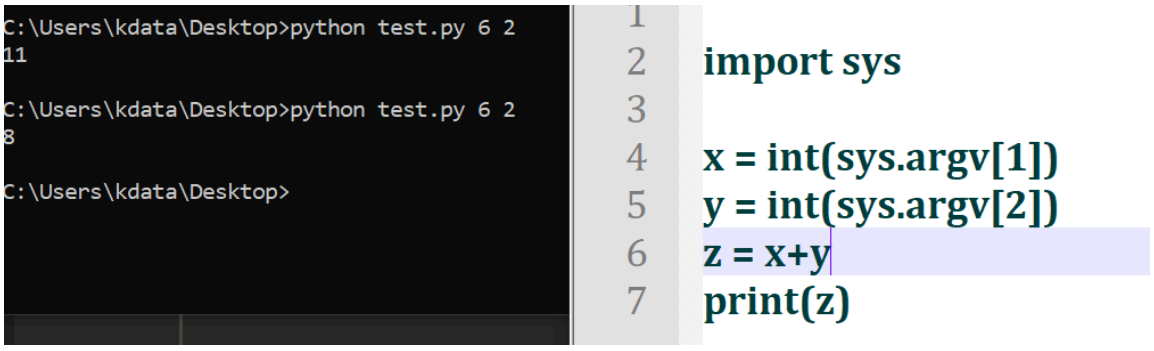
In [ ]:
```
result = eval(input('enter an expr'))
print(result)
```

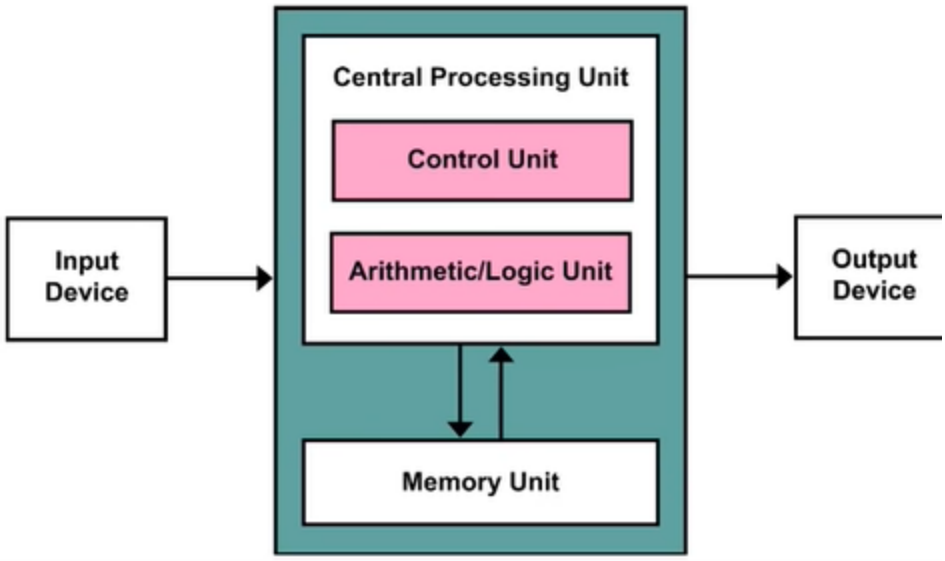
if you want to pass the value in cmd can we pass the value like this



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 we are use for file name , 1st index we are use for 1st argument, 2nd index we are use for 2nd argument





cpu

- cpu has 3 part - CU (Contol Unit) | MU(Memory Unit) | ALU (Arithmetic | Logic unit)
- MEMORY UNIT --> so far we work in memory unit-that means we save variable in memory & call the varible from memory & trying to save the data with the help of variable

- ALU (ARITHMETIC UNIT & LOGIC UNIT) --> ARITHMETIC UNIT --> when it comes to Arithmetic unit it performs calculation & we done so far addition, substraction, multiplication , division. LOGICAL UNIT --> when i talk about logical unit which makes your computer to think means user programed so that and let your computer think. in real life also we apply many condition to reach the goal. if this not possible then we can do that for example. e.g after complet +2 which field you can go for if not this field else what is other files. while we talk amoung each other many situation unknowily we are using if else condition.

LETS UNDERSTAND THE CONDITION TO COMPUTER TO THINK FOR THAT WE HAVE SOME SPECIAL KEYWORD -

In [1]:
```
if True: # indentiation is always 4 spaces
    print('Data Science')
```

```
Data Science
```

In [2]:
```
if True:
print('ds')
```

```
  Cell In[2], line 2
    print('ds')
    ^
IndentationError: expected an indented block after 'if' statement on line 1
```

In [ ]:
```
if False:
    print('Data Science')
print('bye for now')
```

In [ ]:
```
if True: # indentiation is always 4 spaces
    print('Data Science')
print('bye for now')
```

In [ ]:
```
if False: # indentiation is always 4 spaces
    print('Data Science')
else:
    print('bye for now')
```

Lets do one program as if divide by 2 then reminder is 0 then it is even number if reminder is not 0 then it is odd number

In [ ]:
```
#to print only even number
x = 4
r = x % 2

if r == 0:
    print('Even number')
```

In [ ]:
```
#to print only even number
x = 5
r = x % 2

if r == 0:
    print('Even number')
```

In [ ]:
```
x = 5
r = x % 2

if r == 0:
    print('Even number')
print('odd number')
```

In [ ]:
```
x = 6
r = x % 2

if r == 0:
    print('Even number')
print('odd number')
```

In [ ]:
```
x = 10
r = x % 2

if r == 0:
    print('Even number')
if r == 1:
    print('odd number')
```

In [ ]:
```
x = 11
r = x % 2

if r == 0:
    print('Even number')
if r == 1:
    print('odd number')
```

In [ ]:
```
x = 11
r = x % 2

if r == 0:
    print('Even number')
else:
    print('odd number')
```

In [3]:
```
x = 23
r = x % 2

if r == 0:
    print('Even number')
if r == 1:
    print('odd number')
```

```
odd number
```

In [ ]:
```
x = 17
r = x % 2

if r == 0:
    print('Even number')

if r != 0:
    print('odd number')
```

if we observe the code its too many line cuz many of the coder always they wanted to reduce the code lenght which is very good practise. instead of 2 if we can use if-- else

In [ ]:
```
x = 5
r = x % 2

if r == 0:
    print('Even number')
else:
    print('Odd Number')
```

In [ ]:
```
x = 4
r = x % 2

if r == 0:
    print('Even number')
else:
    print('Odd Number')
```

NESTED IF (if we have 2 condition so we need to implment with nested if)

In [ ]:
```
x = 3
r = x % 2

if r == 0:
    print('Even number')
    if x>5:
        print('greater number')

else:
    print('Odd Number')
```

In [ ]:
```
x = 4
r = x % 2

if r == 0:
    print('Even number')
    if x>5:
        print('greater number')

else:
    print('Odd Number')
```

In [ ]:
```
x = 4
r = x % 2

if r == 0:
    print('Even number')
    if x>5 :
        print('greater number')
    else:
       print('not greater ')
else:
    print('odd number')
```

In [ ]:
```
x = 6
r = x % 2

if r == 0:
    print('Even number')
    if x>5 :
        print('greater number')
    else:
        print('not greater ')
```

```
else:
    print('odd number')
```

We do have concept of (IF - ELIF- ELSE) e.g i want to print (1--> one , 2 --> two, 3--> three, 4--> four, 5- five)

In [ ]:
```
# when you use if it will check all condition but if we mention elif then it wont c
# when we use if condition it will check all every block of code  better debug in p
# you can debug with value 1 & d for both if & elif

x = 2

if x == 1:
    print('one')
if x == 2:
    print('Two')
if x == 3:
    print('Three')
if x == 4:
    print('four')
```

In [ ]:
```
# elif it wont check till the block once you find the output it wont go to next lin
# you can try with multiple parameter 1, 2 & 3 value in x

x = 4

if(x == 1):
    print('one')
elif(x == 2):
    print('Two')
elif(x == 3):
    print('Three')
elif(x == 4):
    print('four')
```

In [ ]:
```
x = 7

if(x == 1):
    print('one')
elif(x == 2):
    print('Two')
elif(x == 3):
    print('Three')
elif(x == 4):
    print('four')
```

In [ ]:
```
x = 7

if(x == 1):
    print('one')


elif(x == 2):
    print('Two')
```

```
elif(x == 3):
    print('Three')
elif(x == 4):
    print('four')

else:
    print('wrong output')
```

In [ ]:
```
x = 4

if(x == 1):
    print('one')
elif(x == 2):
    print('Two')
elif(x == 3):
    print('Three')
elif(x == 4):
    print('four')
else:
    print('wrong output')
```

In [ ]:
```
x = 10

if(x == 1):
    print('one')
elif(x == 2):
    print('Two')
elif(x == 3):
    print('Three')
elif(x == 4):
    print('four')

else:
    print('wrong output')
```

In [ ]:
```
#short hand if
a = 30
b = 20
if a > b: print("a is greater than b")
```

LOOPS -- in programing world some time we keep on repeating , may be you want to repeat 5 statement so one way is copy & paste multiple times or other way is

if you want to print the datascience 1000 times then what you will you cant copy for 1000 times , if you want to print 1000 times then you cant do manualy . that is the reason why we need to apply loop -> 2 type of loops -- While loop & For loop

In [4]:
```
print('data science')
print('data science')
print('data science')
print('data science')
print('data science')
```

```
data science
data science
data science
data science
data science
```

In [ ]:
```
i = 1          # initializing

while i<=5:    # condition
    print('data science')
    i = i + 1  # increment
```

In [ ]:
```
i = 5          # initializing

while i>=1:    # condition
    print('data science')
    i = i - 1  # decrement
```

In [ ]:
```
i = 1         # initializing
while i<=5:    # condition
    print('data science : ', i)
    i = i + 1  # increment
```

In [ ]:
```
i = 5          # initializing
while i>=1:    # condition
    print('data science :',i)
    i = i - 1  # decrement
```

can we use multiple while loop || nested while loop to understand nested while indepth understand you can use pycharm debug with f8 option

In [5]:
```
i = 1

while i<=5:
    print('data science') # when we mention end then new line will not create
    j = 1
    while j<=4:
        print('technology')
        j = j + 1

    i = i + 1
    print()

    # the output which we got is very lengty but how to make them one line lets ref
```

```
data science
technology
technology
technology
technology

data science
technology
technology
technology
technology

data science
technology
technology
technology
technology

data science
technology
technology
technology
technology

data science
technology
technology
technology
technology
```

In [6]:
```
i = 1
while i<=5:
    print(' datascience', end = "") # when we mention end then new line will not cr
    j = 1
    while j<=4:
        print(' technology', end="")
        j = j + 1

    i = i + 1
    print()
```

```
datascience technology technology technology technology
datascience technology technology technology technology
datascience technology technology technology technology
datascience technology technology technology technology
datascience technology technology technology technology
```

In [ ]:
```
# lets use while loop usig some numbers
i = 1

while i <= 2 :
    j = 0
    while  j <= 2 :
        print(i*j, end=" ")
        j += 1
```

```
    print()
    i += 1
```

In [ ]:
```
# lets use while loop usig some numbers
i = 1
while i <= 4 :
    j = 0
    while  j <= 3 :
        print(i*j, end=" ")
        j += 1
    print()
    i += 1
```

FOR LOOP - normally while loop it work with iteration or certaion some condition but for loop it will work with sequence (list, string,int)

In [ ]:
```
name = 'nit'

for i in name:
    print(i)
```

In [ ]:
```
name1 = [1,3.5,'hallo'] #i want print the value individualy

for i in name1:
    print(i)
```

In [ ]:
```
for i in [2, 3, 7.8, 'hi']:
    print(i)
```

In [ ]:
```
range(5)
```

In [ ]:
```
for i in range(5):
    print(i)
```

In [ ]:
```
for i in range(2,5):
    print(i)
```

In [ ]:
```
for i in range(1,10,3):
    print(i)
```

In [ ]:
```
# print the value which is divisible by 5
for i in range(1,21):
    print(i)
```

In [7]:
```
# print the value which is divisible by 5

for i in range(1,51):

    if i%5==0 :
        print(i)
```

```
5
10
15
20
25
30
35
40
45
50
```

In [8]:
```python
# print the value which is divisible by 5 i dont want that value
for i in range(1,51):

    if i%5!=0 :
        print(i)
```

1
2
3
4
6
7
8
9
11
12
13
14
16
17
18
19
21
22
23
24
26
27
28
29
31
32
33
34
36
37
38
39
41
42
43
44
46
47
48
49

LETS DISCUSS ABOUT 3 KEYWORDS -- BREAK || CONTINUE || PASS BREAK STATEMNT - if you apply break statment in a loop then it will end the loop # Pass = skips block of code(function, class etc) # Continue= skips 1 step/iteration during loop # Break= jumps out of the function/loop

In [ ]:
```
# write the code user ask chocklet from vendor machne write the basic code

x = int(input('How many choclets you want:?'))

i = 1
while i<=x:
    print('choclet')
    i += 1
```

- If the user says i need 10 choclet but vending machine dont have 10 choclate & machine has only 5 choclate so what you do on those scenario

- We have 3 choice now (eiter stop the transaction by you or you can give only 5 choclate) & may be vendor machine display the result as we are out of the stock
- Now lets try in the code

In [ ]:
```
ava = 5 # the machine has only 5 choclet

x = int(input('How many choclets you want:?'))

i = 1
while i<=x:
    print('choclet')
    i += 1
# if you check the user wants 10 choclets  but availabe choclet is 5 but we got out
# in this code we just declare but we dint apply any condition to it
```

In [ ]:
```
available_choclet = 5 # the machine has only 10 candis

x = int(input('How many choclets user want:?'))

i = 1
while i<=x:

    if i>available_choclet: # we stop the execution but which code execution not en
        break # break is statement | means jump out of the loop
    print('choclet')
    i += 1

print('bye for now')
```

In [ ]:
```
available_choclet = 5 # the machine has only 10 candis

x = int(input('How many choclets you want:?'))

i = 1
while i<=x:

    if i>available_choclet: # we stop the execution but which code execution not en
        print('out of stock')
        break # break is statement | means jump out of the loop
    print('choclet')
    i += 1

print('bye for now')
```

In [ ]:
```
for i in range(1,11):
    print(i)
```

- i dont want 11 number i want only 5 number for the range of 1 to 10

In [9]:
```
for i in range(1,11):
    if i == 6:
```

```
        break
    print(i)
```

```
1
2
3
4
5
```

in continue loop wont be terminate & exclue the assign number it give you entire output

In [10]:
```
for i in range(1,11):
    if i == 3:
        continue
    print(i)
```

```
1
2
4
5
6
7
8
9
10
```

In [11]:
```
for i in range(1,11):
    if i == 6:
        continue
    print('hello :',i)
```

```
hello : 1
hello : 2
hello : 3
hello : 4
hello : 5
hello : 7
hello : 8
hello : 9
hello : 10
```

#PASS Statement - pass the code & it wont go (code give you the error)

In [15]:
```
for i in range(1,11):
```

```
  Cell In[15], line 1
    for i in range(1,11):
                         ^
SyntaxError: incomplete input
```

In [16]:
```
for i in range(1,11):
    pass
```

you need to print the number from 1 to 50 but dont print the number which is divisible by 3 or 5

In [17]:
```
for i in range(1,51):

    if i%3 == 0:

        print(i)
print('end')
```

```
3
6
9
12
15
18
21
24
27
30
33
36
39
42
45
48
end
```

In [18]:
```
for i in range(1,51):

    if i%3 == 0:
        continue
    print(i)
print('end')
```

```
1
2
4
5
7
8
10
11
13
14
16
17
19
20
22
23
25
26
28
29
31
32
34
35
37
38
40
41
43
44
46
47
49
50
end
```

In [20]:
```python
for i in range(1,51):

    if i%3 == 0 or i%5 == 0:
        continue
    print(i)
#print('end')
# it will skip all the value which is divisible by 3 or 5
```

```
1
2
4
7
8
11
13
14
16
17
19
22
23
26
28
29
31
32
34
37
38
41
43
44
46
47
49
```

In [21]:
```python
for i in range(1,50):
    if i%3 == 0 and i%5 == 0:
        continue
    print(i)
print('end')
# when you apply and you wont get the value which is divisible by both 3 & 5 (15)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
16
17
18
19
20
21
22
23
24
25
26
27
28
29
31
32
33
34
35
36
37
38
39
40
41
42
43
44
46
47
48
49
end
```

In [22]:
```
# i dont want to print the values which are even numbers that means print only odd

for i in range(1,51):

    if (i%2 == 0):
        #print('even')
        continue
    else:
```

```
        print(i)
print('bye')
```

```
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
bye
```

PRINTING PATTERN IN PYTHON

In [23]:
```
print('# # # #')
print('# # # #')
print('# # # #')
print('# # # #')
```

```
# # # #
# # # #
# # # #
# # # #
```

In [26]:
```
for i in range(1,5):
    i=i+1
    print('# # # # ')
```

```
# # # #
# # # #
# # # #
# # # #
```

In [27]:
```
for i in range(1,5):
    if i<=5:
        print('# # # #')
```

```
# # # #
# # # #
# # # #
# # # #
```

In [28]:
```
for j in range(4):
    print('#')
```

```
#
#
#
#
```

In [33]:
```
for j in range(4):
    print('# # # #')
```

```
# # # #
# # # #
# # # #
# # # #
```

In [ ]:
```
for j in range(4):
    print('#', end = " ")
```

In [34]:
```
for j in range(4):
    print('#', end=" ")

for j in range(4):
    print('#', end=" ")
```

```
# # # # # # # #
```

In [35]:
```
for j in range(4):
    print('#', end=" ")

print()

for j in range(4):
    print('#', end=" ")
```

```
# # # #
# # # #
```

In [ ]:
```
for j in range(4):
    print('#', end="  ")

print()

for j in range(4):
```

```
    print('#', end="  ")

print()

for j in range(4):
    print('#', end="  ")

print()

for j in range(4):
    print('#', end="  ")
```

In [ ]:
```
for i in range(4):
    for j in range(4):
        print('#', end="  ")
    print()
    # pease use debug mode in pycharm
```

In [36]:
```
for i in range(4):
    for j in range(i+1):
        print('#', end = " ")
    print()
```

```
#
# #
# # #
# # # #
```

In [37]:
```
for i in range(1,5):
    print("# "*i)
```

```
#
# #
# # #
# # # #
```

In [38]:
```
for i in range(1,5):
    for j in range(4):
        if i>j:
            print("#",end=" ")
    print()
```

```
#
# #
# # #
# # # #
```

In [39]:
```
list(range(5))
```

Out[39]:
```
[0, 1, 2, 3, 4]
```

In [40]:
```
for i in range(4):
    for j in range(i):
        print('#', end="  ")
    print()
```

```
#
#  #
#  #  #
```

In [ ]:
```
for i in range(4):
    for j in range(i+1):
        print('#', end="  ")
    print()
```

In [41]:
```
for i in range(4):
    for j in range(4-i):
        print('#', end="  ")
    print()
```

```
#  #  #  #
#  #  #
#  #
#
```

In [42]:
```
for i in range(1,5):
    print("# "*(5-i))
```

```
# # # #
# # #
# #
#
```

- For|Else in python
- In other language for else not supportable but in python it is supportable

eg- lets print the number from 1- 20 & we dont want print number which is divisible by 5

In [43]:
```
nums = [12,15,18,21,26]

for num in nums:
    if num % 5 == 0:
        print(num)
```

```
15
```

In [44]:
```
nums = [12,14,18,21,25,30,35]

for num in nums:
    if num % 5 == 0:
        print(num)
```

```
25
30
35
```

In [45]:
```
nums = [12,14,18,21,25,20]

for num in nums:
    if num % 5 == 0:
        print(num)
```

```
25
20
```

In [46]:
```
nums = [12,14,18,21,20,25]

for num in nums:
    if num % 5 == 0:
        print(num)
        break
```

```
20
```

In [47]:
```
nums = [12,14,18,21,20,25]

for num in nums:
    if num % 5 == 0:
        print(num)
        break
```

```
20
```

In [48]:
```
nums = [10,14,18,21,5,10]

for num in nums:
    if num % 5 == 0:
        print(num)
        break #it will print only 1 number then it break
```

```
10
```

In [ ]:
```
nums = [7,14,18,21,23,27] #hear there is no number which is divisible by 5 we got o

for num in nums:
    if num % 5 == 0:
        print(num)
        break
```

In [51]:
```
nums = [7,14,18,21,23,27] #hear there is no number which is divisible by 5 we got o

for num in nums:
    if num % 5 == 0:
        print(num)
        break
    else:
        print('Number Not Found') #every iteration it cheking condition
```

```
Number Not Found
```

In [52]:
```
nums = [7,14] #hear there is no number which is divisible by 5 we got output as bla
for num in nums:
    if num % 5 == 0:
        print(num)
        break
    else:
        print('Number Not Found') #every iteration it cheking condition
```

```
Number Not Found
Number Not Found
```

In [53]:
```
nums = [7,14,18,21,23,27] #hear there is no number which is divisible by 5 we got o

for num in nums:
    if num % 5 == 0:
        print(num)
        break
else:
        print('Number Not Found') # hear else we dont write in if block but we can
```

```
Number Not Found
```

In [55]:
```
nums = [10,14,18,21,20,27] #hear there is no number which is divisible by 5 we got

for num in nums:
    if num % 5 == 0:
        print(num)
        break
```

```
else:
        print('Not Found')
```

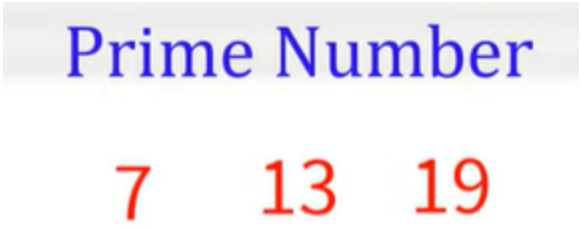
```
10
```

In [ ]:
```
nums = [10,14,18,21,20,27,30] #hear there is no number which is divisible by 5 we g

for num in nums:
    if num % 5 == 0:
        print(num)
        #break
else:
        print('Not Found')
```

In [ ]:
```
nums = [10,14,18,21,20,27] #hear there is no number which is divisible by 5 we got
for num in nums:
    if num % 5 == 0:
        print(num)
        break
else:
        print('Not Found')
```

- prime number - how to check given number is prime number or not



In [59]:
```
num = 14

for i in range(2,num):
    if num % i == 0:
        print('Not prime Number')
        break
else:
    print('Prime Number')
```

```
Not prime Number
```

In [ ]:
```
num = 13
for i in range(2,num):
    if num % i == 0:
        print('Not prime Number')
        break
else:
    print('Prime Number')
```

In [ ]:
```
from array import *
arr = array('i',[])

n = int(input('Enter the length of the array'))

for i in range(5):
    x = int(input('Enter the next value'))
    arr.append(x)
print(arr)
```

In [ ]:
```
from array import *
arr = array('i',[])

n = int(input('Enter the length of the array'))

for i in range(5):
    x = int(input('Enter the next value'))
    arr.append(x)
print(arr)
```

In [ ]:
```
from array import *
arr = array('i',[])

n = input('Enter the length of the array')

for i in range(5):
    x = input('Enter the next value')
    arr.append(x)
print(arr)
```

Way of creating array using numpy

In [ ]:
```
from numpy import *
arr = array([1,2,3,4,5])
print(arr)
type(arr)
```

In [ ]:
```
print(arr.dtype)
```

In [ ]:
```
arr = array([1,2,3,4,5.9])
print(arr)
```

In [ ]:
```
print(arr.dtype)
```

In [ ]:
```
arr2 = array([1,2,3,4,5.9],float)
arr2
```

In [ ]:
```
arr3 = array([1,2,3,4,5.6],int)
arr3
```

In [ ]:
```
import numpy as np
```

In [ ]:
```
arr4 = np.linspace(0, 16, 10) # break the code between 10 spaces between 0 to 16 bu
arr4
```

In [ ]:
```
arr5 = np.arange(0,10,2) # arange - as range
arr5
```

In [ ]:
```
arr6 = np.zeros(5)
arr6
```

In [ ]:
```
arr7 = np.ones(5)
arr7
```

In [ ]:
```
import tkinter as tk
# Function to be called when the button is clicked
def on_button_click():
    label.config(text="Button clicked!")

# Create the main application window
root = tk.Tk()
root.title("Simple Tkinter App")

# Create a label widget
label = tk.Label(root, text="Hello, Tkinter!")
label.pack(pady=20)

# Create a button widget
button = tk.Button(root, text="Click Me", command=on_button_click)
button.pack(pady=20)

# Run the application
root.mainloop()
```

In [ ]:
```
import tkinter as tk
from tkinter import messagebox

# Function to be called when the button is clicked
def on_button_click():
    user_input = entry.get()
    messagebox.showinfo("Information", f"You entered: {user_input}")

# Create the main application window
root = tk.Tk()
root.title("Simple Tkinter App")

# Create a label widget
```

```
label = tk.Label(root, text="Enter something:")
label.pack(pady=10)

# Create a text entry widget
entry = tk.Entry(root, width=30)
entry.pack(pady=10)

# Create a button widget
button = tk.Button(root, text="Submit", command=on_button_click)
button.pack(pady=10)

# Run the application
root.mainloop()
```