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. 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.

1 + 1 # ADDITION

In [13]:
```
2+5
```

Out[13]: 7

In [14]:
```
9-6
```

Out[14]: 3

In [15]:
```
9*5
```

Out[15]: 45

In [16]:
```
55/5 #division
```

Out[16]: 11.0

In [17]:
```
55/5 #float division
```

Out[17]: 11.0

In [18]:
```
9//3 #integer division
```

Out[18]: 3

In [19]:
```
9+8- #synatax error
```

```
  Cell In[19], line 1
    9+8- #synatax error
         ^
SyntaxError: invalid syntax
```

In [183…
```
9+10
```

Out[183… 19

In [185…
```
9+9*2
```

Out[185… 27

In [187…
```
(9+9)*5 # BODMAS (Bracket || Oders || Divide || Multiply || Add || Substact)
```

Out[187… 90

In [189…
```
5*5*5*5*5 # exponentaion
```

Out[189… 3125

In [191…
```
5**2
```

Out[191… 25

In [193…
```
20/5
```

Out[193… 4.0

In [195…
```
10//3
```

Out[195… 3

In [197…
```
50%5 # Modulus
```

Out[197… 0

In [199…
```
18%2
```

Out[199… 0

In [ ]:
```

```

In [202…
```
a,b,c,d,e=9,7.3,'naidu',9+5j,True
print(a)
print(b)
print(c)
print(d)
print(e)
```

```
9
7.3
naidu
(9+5j)
True
```

In [204…
```
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'complex'>
<class 'bool'>
```

In [206…
```
type(c)
```

Out[206… str

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

In [209…
```
'Naresh IT'
```

Out[209… '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 [212…
```
print('naresh it')
```

```
naresh it
```

In [214…
```
"it technology"
```

Out[214… 'it technology'

In [216…
```
s1='naresh it'
s1
```

Out[216… 'naresh it'

In [218…
```
a=2
b=5
```

In [220…
```
c=a+b
c
```

Out[220… 7

In [222…
```
a=3
b='hi'
type(b)
```

Out[222… str

In [224…
```
a+b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[224], line 1
----> 1 a+b

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

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

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

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

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

In [230…
```
'nit' + 'nit'
```

Out[230…
```
'nitnit'
```

In [232…
```
# 5 times print
5* 'nit'
```

Out[232…
```
'nitnitnitnitnit'
```

In [234…
```
5*' nit'
```

Out[234…
```
' nit nit nit nit nit'
```

In [236…
```
print('c:\nit')
```

```
c:
it
```

In [238…
```
print(r'c:\nit')
```

```
c:\nit
```

variable || identifier || object

In [241…
```
#x is variable/identifier/object, 2 is the value
x=2
x
```

Out[241…
```
2
```

In [243…
```
x+5
```

Out[243…
```
7
```

In [245…
```
y=3
y
```

Out[245…
```
3
```

In [247…
```
x+y
```
Out[247… 5

In [249…
```
x+12
```
Out[249… 14

In [251…
```
y
```
Out[251… 3

In [253…
```
+y
```
Out[253… 3

In [255…
```
# string variable
name='naveen'
name
```
Out[255… 'naveen'

In [257…
```
name
```
Out[257… 'naveen'

In [259…
```
name +'Ram it'
```
Out[259… 'naveenRam it'

In [261…
```
name
```
Out[261… 'naveen'

In [263…
```
len(name)
```
Out[263… 6

In [265…
```
name[2] #python index begins with 0
```
Out[265… 'v'

In [267…
```
name[0]
```
Out[267… 'n'

In [269…
```
name[1]
```
Out[269… 'a'

In [271…
```
name[-1]
```
Out[271… 'n'

In [273…
```
name[-3]
```

Out[273… 'e'

In [275…
```
name[-2]
```

Out[275… 'e'

slicing

In [278…
```
#to print 2 character
name
name[0:2]
```

Out[278… 'na'

In [280…
```
name[1:4]
```

Out[280… 'ave'

In [282…
```
name[0:4]
```

Out[282… 'nave'

In [284…
```
name[1:]
```

Out[284… 'aveen'

name[2:]

In [287…
```
name[2:]
```

Out[287… 'veen'

In [289…
```
name[3:]
```

Out[289… 'een'

In [291…
```
name1='mine'
name1
```

Out[291… 'mine'

In [293…
```
name1[0:1]
```

Out[293… 'm'

In [295…
```
name1[0:1]
```

Out[295… 'm'

In [297…
```
name[1:0]
```

Out[297… ''

In [299…
```
name1='naidu'
name1
```

Out[299… 'naidu'

In [301…
```
name1[0:1]
```

Out[301… 'n'

In [303…
```
name1[0]
```

Out[303… 'n'

In [305…
```
name1
```

Out[305… 'naidu'

In [307…
```
name1[1:]
```

Out[307… 'aidu'

In [309…
```
name1[0:]
```

Out[309… 'naidu'

In [311…
```
name1[2:]
```

Out[311… 'idu'

In [313…
```
name1[3:]
```

Out[313… 'du'

In [315…
```
name1[4:]
```

Out[315… 'u'

In [317…
```
'm'+name1[1:]  #i want to change fine to dine
```

Out[317… 'maidu'

In [319…
```
len(name1) #python inbuild function
```

Out[319… 5

List

In [322…
```
# List
nums=[10,20,30,40]
nums
```

Out[322… [10, 20, 30, 40]

In [324…
```
nums[0]
```

Out[324… 10

In [326…
```
nums[1]
```

Out[326… 20

In [328…
```
nums[3]
```

Out[328… 40

In [330…
```
nums[-1]
```

Out[330… 40

In [332…
```
nums[0]
```

Out[332… 10

In [334…
```
nums[-4]
```

Out[334… 10

In [336…
```
nums[-3]
```

Out[336… 20

In [338…
```
num1=['hi' ,'happy']
num1
```

Out[338… ['hi', 'happy']

In [340…
```
num1
```

Out[340… ['hi', 'happy']

In [342…
```
num2=['hi',9.9,18] # we can assign multiple variable
num2
```

Out[342… ['hi', 9.9, 18]

In [344…
```
# can we have 2 list together
num3=[nums,num1,num2]
num3
```

Out[344… [[10, 20, 30, 40], ['hi', 'happy'], ['hi', 9.9, 18]]

In [346…
```
num4=[num3,nums]
num4
```

Out[346… [[[10, 20, 30, 40], ['hi', 'happy'], ['hi', 9.9, 18]], [10, 20, 30, 40]]

In [348…
```
nums.append(27)
nums
```

Out[348… [10, 20, 30, 40, 27]

In [350…
```
nums.remove(30)
nums
```

Out[350…
```
[10, 20, 40, 27]
```

In [352…
```
nums.remove(27)
nums
```

Out[352…
```
[10, 20, 40]
```

In [354…
```
nums.pop() #if you dont assign the index element then it will consider by defaul
nums
```

Out[354…
```
[10, 20]
```

In [356…
```
num1
```

Out[356…
```
['hi', 'happy']
```

In [358…
```
num1.insert(2,'nit') #insert the value as per index values i.e 2nd index we are
num1
```

Out[358…
```
['hi', 'happy', 'nit']
```

In [360…
```
num1.insert(0,1)
```

In [362…
```
num1
```

Out[362…
```
[1, 'hi', 'happy', 'nit']
```

In [364…
```
del num2[2:]
```

num2

In [367…
```
num2
```

Out[367…
```
['hi', 9.9]
```

In [369…
```
# if you need to add multiple values
num2.extend([29,15,18])
num2
```

Out[369…
```
['hi', 9.9, 29, 15, 18]
```

In [371…
```
num3.extend(['a',5,6.7])
num3
```

Out[371…
```
[[10, 20], [1, 'hi', 'happy', 'nit'], ['hi', 9.9, 29, 15, 18], 'a', 5, 6.7]
```

num3

In [374…
```
num3
```

Out[374…
```
[[10, 20], [1, 'hi', 'happy', 'nit'], ['hi', 9.9, 29, 15, 18], 'a', 5, 6.7]
```

In [376…
```
nums
```
Out[376… [10, 20]

In [378…
```
nums
```
Out[378… [10, 20]

In [380…
```
num1
```
Out[380… [1, 'hi', 'happy', 'nit']

In [382…
```
num5=(1,3,2,4,5,6,7,9) #inbuild function
min(num5)
```
Out[382… 1

In [384…
```
max(num5)
```
Out[384… 9

In [386…
```
sum(num5)
```
Out[386… 37

In [388…
```
sum(num5)
```
Out[388… 37

In [390…
```
nums.sort()
nums
```
Out[390… [10, 20]

In [392…
```
nums
```
Out[392… [10, 20]

tuple

In [395…
```
# Tuple
tup=(15,20,25,30)
tup
```
Out[395… (15, 20, 25, 30)

In [397…
```
tup[0]
```
Out[397… 15

In [399…
```
tup[1]
```
Out[399… 20

In [401…
```
tup[3]
```

Out[401…
```
30
```

In [403…
```
tup[2]
```

Out[403…
```
25
```

SET

In [406…
```
# Set
s = {}
```

In [408…
```
s1={3,6,9,12,15,18}
s1
```

Out[408…
```
{3, 6, 9, 12, 15, 18}
```

In [410…
```
s3={50,21,23,45,'nit',54}
s3
```

Out[410…
```
{21, 23, 45, 50, 54, 'nit'}
```

In [412…
```
s1[1] #as we dont have proper sequencing thats why indexing not subscriptable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[412], line 1
----> 1 s1[1]

TypeError: 'set' object is not subscriptable
```

DICTIONARY

In [415…
```
# Dictionary
data={1:'naidu',2:'naveen',3:'ramunaidu',4:'satyavathi'}
data
```

Out[415…
```
{1: 'naidu', 2: 'naveen', 3: 'ramunaidu', 4: 'satyavathi'}
```

In [417…
```
data[4]
```

Out[417…
```
'satyavathi'
```

In [419…
```
data[1]
```

Out[419…
```
'naidu'
```

In [421…
```
data[3]
```

Out[421…
```
'ramunaidu'
```

In [423…
```
data[2]
```

Out[423… 'naveen'

In [425…
```
data.get(2)
```

Out[425… 'naveen'

In [427…
```
data.get(4)
```

Out[427… 'satyavathi'

In [429…
```
data.get(1)
```

Out[429… 'naidu'

In [431…
```
data.get(3)
```

Out[431… 'ramunaidu'

In [433…
```
print(data.get(2))
```

naveen

In [435…
```
print(data.get(4))
```

satyavathi

In [437…
```
print(data.get(2))
```

naveen

In [439…
```
print(data.get(1))
```

naidu

In [441…
```
print(data.get(3))
```

ramunaidu

In [443…
```
data.get(1,'not found')
```

Out[443… 'naidu'

In [445…
```
data.get(2,'not found')
```

Out[445… 'naveen'

In [447…
```
data[5]='five'
data
```

Out[447… {1: 'naidu', 2: 'naveen', 3: 'ramunaidu', 4: 'satyavathi', 5: 'five'}

In [449…
```
data
```

Out[449… {1: 'naidu', 2: 'naveen', 3: 'ramunaidu', 4: 'satyavathi', 5: 'five'}

In [451…
```
del data[5]
data
```

Out[451… {1: 'naidu', 2: 'naveen', 3: 'ramunaidu', 4: 'satyavathi'}

In [453…
```
data
```

Out[453… {1: 'naidu', 2: 'naveen', 3: 'ramunaidu', 4: 'satyavathi'}

In [455…
```
#list in the dictionary
prog={'python':['vscode','pycharam'], 'machine' : 'sklearn', 'datascience':['jup
prog
```

Out[455…
```
{'python': ['vscode', 'pycharam'],
 'machine': 'sklearn',
 'datascience': ['jupter', 'spyder']}
```

In [457…
```
prog['python']
```

Out[457… ['vscode', 'pycharam']

In [459…
```
prog['machine']
```

Out[459… 'sklearn'

In [461…
```
prog['datascience']
```

Out[461… ['jupter', 'spyder']

how to creat a 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

In [465…
```
#help()
```

In [467…
```
# help(tuple)
```

In [469…
```
# help(list)
```

introduce to ID()

In [472…
```
# variable address
num=5
id(num)
```

Out[472… 140715253312056

In [474…
```
name='nit'
id(name)  #Address will be different for both
```

Out[474…
```
1849542055216
```

In [476…
```
a=10
id(a)
```

Out[476…
```
140715253312216
```

In [478…
```
b=a  #thats why python is more memory efficient
```

In [480…
```
id(b)
```

Out[480…
```
140715253312216
```

In [482…
```
id(10)
```

Out[482…
```
140715253312216
```

In [484…
```
k=10
id(k)
```

Out[484…
```
140715253312216
```

In [486…
```
a=20 # as we change the value of a then address will change
id(a)
```

Out[486…
```
140715253312536
```

In [488…
```
id(b)
```

Out[488…
```
140715253312216
```

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 [491…
```
PI= 3.14
PI
```

Out[491…
```
3.14
```

In [493…
```
type(PI)
```

Out[493…
```
float
```

DATA TYPES & DATA STRUCTURES --->

NUMERICAL DATA

int

float

complex

bool

In [496…
```
a = 5
type(a)
```

Out[496…
```
int
```

In [498…
```
a = 2.7
type(a)
```

Out[498…
```
float
```

In [500…
```
b = 9+9j
type(b)
```

Out[500…
```
complex
```

In [502…
```
a=True
b=False
type(a)
```

Out[502…
```
bool
```

list

tuple

set

string

range

dictionary

In [505…
```
w=2.5
type(w)
```

Out[505… float

In [507…
```
(a)
```

Out[507… True

In [509…
```
w2 = 2 + 3j  #so hear j is represent as root of -1
type(w2)
```

Out[509… complex

In [511…
```
#convert flot to integer
a=5.6
b=int(a)
b
```

Out[511… 5

In [513…
```
type(a)
```

Out[513… float

In [515…
```
type(b)
```

Out[515… int

In [517…
```
k=float(b)
k
```

Out[517… 5.0

In [519…
```
k1=complex(b,k)
print(k1)
```

(5+5j)

In [521…
```
type(k1)
```

Out[521… complex

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

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

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

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

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

In [ ]:
```
1=[1,2,3,4,5]
print(1)
```

```
type(1)
```

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

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

In [ ]:
```
s1{1,2,3,4,3,4,5,6,7,6} #duplicates are not allowed
s1
```

In [ ]:
```
s={10,20,30}
s
```

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

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

Range ()

In [ ]:
```
r=range(0,20)
r
```

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

In [ ]:
```
list(range(0,20))
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 [ ]:
```
d.keys()
```

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

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

In [ ]:
```
d.get(1)
```

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

OPERATORS IN PYTHON

arithmetic

assignment

relational

logical

unary

Arithmetic operator

x1 ,y1 = 10, 5 x1 + y1

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

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

In [ ]:
```
n
```

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

Relational operator

we are using this operator for comparing

In [ ]:
```
a = 5
b = 9
```

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
```

LOGICAL OPERATOR

AND, OR, NOT

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

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

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

bin(25)

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

In [ ]:
```
bin(55)
```

In [ ]:
```
type(0b110111)
```

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

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

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

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

In [ ]:
```
hex(11)
```

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

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

swap variable in python

(a,b = 6,9) After swap we should get ==> (a, b = 9,6)

In [ ]:
```
a = 6
b = 9
```

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

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

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

in above scenario we lost the value 6 a1 = 4 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)
print(0b110)
```

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 [ ]:
```
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(13))
print(bin(14))
```

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 [ ]:
```
~12
```

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

In [ ]:
```
~9
```

In [ ]:
```
~99
```

In [ ]:
```
~-81
```

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

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 [ ]:
```
25^26
```

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

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

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

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

In [ ]:
```
1000>>4
```

BITWISE RIGHTSHIFT OPERATOR

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

In [ ]:
```
1000<<6
```

import math module

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

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

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

In [ ]:
```
x1 = math.sqrt(35)
x1
```

In [ ]:
```
x2 = math.sqrt(13)
x2
```

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

In [ ]:
```
print(math.ceil(2.4))  #ceil - maximum or highest value
```

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

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

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

In [ ]:
```
import math as m
```

In [ ]:
```
m.sqrt(2)
```

In [ ]:
```
m.sqrt(10)
```

In [ ]:
```
from math import sqrt,pow # math has many function if you want to call specific
pow(3,4)
```

round(pow(3,4))

In [ ]:
```
x
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: