[]:

In [1]:
```
1+1
```

Out[1]: 2

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

Out[2]: 1

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

Out[3]: 12

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

Out[4]: 2.0

In [5]:
```
8/4
```

Out[5]: 2.0

In [6]:
```
8 / 5
```

Out[6]: 1.6

In [7]:
```
8/4
```

Out[7]: 2.0

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

Out[8]: 10

In [9]:
```
8//4
```

Out[9]: 2

In [10]:
```
8+8
```

Out[10]: 16

In [11]:
```
5+5*5
```

Out[11]: 30

In [12]:
```
(5+5)*5
```

Out[12]: 50

In [13]:
```
2*2*2*2*2
```

Out[13]: 32

In [14]:
```
2**5
```

Out[14]: 32

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

Out[15]: 5.0

In [16]:
```
15//3
```

Out[16]: 5

In [17]:
```
14 % 2
```

Out[17]: 0

In [18]:
```
a,b,c,d,e = 15, 7.8, 'nit', 8+9j, True
```

In [19]:
```
print(a)
print(b)
print(c)
print(d)
print(e)
```

```
15
7.8
nit
(8+9j)
True
```

In [20]:
```
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 [21]:
```
type(c)
```

Out[21]: str

In [ ]:

In [ ]:

In [22]:
```
'Naresh IT'
```

Out[22]: 'Naresh IT'

In [23]:
```
print('Naresh IT')
```

```
Naresh IT
```

In [24]:
```
"max it technology"
```

Out[24]: 'max it technology'

In [25]:
```
s1='naresh i technology'
s1
```

Out[25]: 'naresh i technology'

In [26]:
```
a=2
b=3
a+b
```

Out[26]: 5

In [27]:
```
c=a+b
```

In [28]:
```
c
```

Out[28]: 5

In [29]:
```
a=3
b='hi'
```

In [30]:
```
type(b)
```

Out[30]: str

In [31]:
```
print('naresh it's 'technolgy')
```

```
  Cell In[31], line 1
    print('naresh it's 'technolgy')
          ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

In [32]:
```
print('naresh it\'s "technology"')
```

```
naresh it's "technology"
```

In [33]:
```
print('nareshit', 'technology')
```

```
nareshit technology
```

In [34]:
```
'nit' + ' nit'
```

Out[34]: 'nit nit'

In [35]:
```
'nit' ' nit'
```

Out[35]: 'nit nit'

In [36]:
```
5* 'nit'
```

Out[36]: 'nitnitnitnitnit'

In [37]:
```
5* ' nit'
```

Out[37]: ' nit nit nit nit nit'

In [38]:
```
print('c:\nit')
```

```
c:
it
```

In [39]:
```
print(r'c:\nit')
```

```
c:\nit
```

In [ ]:

VAriable | Identifiers |object

In [ ]:

In [40]:
```
2
```

Out[40]: 2

In [41]:
```
x= 2
x
```

Out[41]: 2

In [42]:
```
x + 3
```

Out[42]: 5

In [43]:
```
y = 3
y
```

Out[43]: 3

In [44]:
```
x +y
```

Out[44]: 5

In [45]:
```
x = 9
x
```

Out[45]: 9

In [46]:
```
x+y
```

Out[46]: 12

In [47]:
```
x+10
```

Out[47]: 19

In [48]:
```
y
```

Out[48]: 3

In [49]:
```
_ + y
```

Out[49]: 6

In [50]:
```
_ + y
```

Out[50]: 9

In [51]:
```
_ + y
```

Out[51]: 12

In [ ]:
```

```

In [52]:
```
#sring variable
```

In [53]:
```
name='nit'
name
```

Out[53]: 'nit'

In [54]:
```
name + 'technology'
```

Out[54]: 'nittechnology'

In [55]:
```
name + ' technology'
```

Out[55]: 'nit technology'

In [56]:
```
name 'technology'
```

```
  Cell In[56], line 1
    name 'technology'
         ^
SyntaxError: invalid syntax
```

In [57]:
```
name
```

Out[57]: 'nit'

In [58]:
```
len(name)
```

Out[58]: 3

In [59]:
```
name[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[59], line 1
----> 1 name[5]

IndexError: string index out of range
```

In [60]:
```
name[2]
```

Out[60]: 't'

In [61]:
```
name[-1]
```

Out[61]: 't'

slicing

In [62]:
```
name
```

Out[62]: 'nit'

In [63]:
```
name[0:2]
```

Out[63]: 'ni'

In [64]:
```
name[1:4]
```

Out[64]: 'it'

In [65]:
```
name[1:]
```

Out[65]: 'it'

In [66]:
```
name[:5]
```

Out[66]: 'nit'

In [67]:
```
name[3:9]
```

Out[67]: ''

In [68]:
```
name1='fine'
```

In [69]:
```
name1
```

Out[69]: 'fine'

In [70]:
```
name
```

Out[70]: 'nit'

In [71]:
```
name1[0:1]
```

Out[71]: 'f'

In [72]:
```
name1
```

Out[72]: 'fine'

In [73]:
```
name1[1:]
```

Out[73]: 'ine'

In [74]:
```
'd'+ name1[0:]
```

Out[74]: 'dfine'

In [75]:
```
len(name1)
```

Out[75]: 4

In [ ]:

List

In [76]:
```
nums=[10,20,30]
nums
```

Out[76]: [10, 20, 30]

In [77]:
```
nums[0]
```

Out[77]: 10

In [78]:
```
nums[-1]
```

Out[78]: 30

In [79]:
```
nums[1:]
```

Out[79]: [20, 30]

In [80]:
```
nums[:1]
```

Out[80]: [10]

In [81]:
```
nums1=['hi', 'hello']
nums1
```

Out[81]: ['hi', 'hello']

In [82]:
```
nums2=['hi', 8.9, 34]
nums2
```

Out[82]: ['hi', 8.9, 34]

In [83]:
```
nums3=[nums,nums1]
nums3
```

Out[83]: [[10, 20, 30], ['hi', 'hello']]

In [84]:
```
nums4=[nums,nums1,nums2]
```

In [85]:
```
nums4
```

Out[85]: [[10, 20, 30], ['hi', 'hello'], ['hi', 8.9, 34]]

In [86]:
```
nums
```

Out[86]: [10, 20, 30]

In [87]:
```
nums.append(45)
```

In [88]:
```
nums
```

Out[88]: [10, 20, 30, 45]

In [89]:
```
nums.pop(1)
```

Out[89]: 20

In [90]:
```
nums
```

Out[90]: [10, 30, 45]

In [91]:
```
nums.pop()
```

Out[91]: 45

In [92]:
```
nums
```

Out[92]: [10, 30]

In [93]:
```
nums1
```

Out[93]: ['hi', 'hello']

In [94]:
```
nums1.insert(2,'nit')
```

In [95]:
```
nums1
```

Out[95]: ['hi', 'hello', 'nit']

In [96]:
```
nums1
```

Out[96]: ['hi', 'hello', 'nit']

In [97]:
```
nums1.insert(0,1)
```

In [98]:
```
nums1
```

Out[98]: [1, 'hi', 'hello', 'nit']

In [99]:
```
nums2
```

Out[99]: ['hi', 8.9, 34]

In [100…
```
del nums2[2:]
```

In [101…
```
nums2
```

Out[101… ['hi', 8.9]

In [102…
```
nums2.extend([20,30,10])
```

In [103…
```
nums2
```

Out[103… ['hi', 8.9, 20, 30, 10]

In [104…
```
nums3
```

Out[104… [[10, 30], [1, 'hi', 'hello', 'nit']]

In [105…
```
nums3.extend(['a',5,6.6])
```

In [106…
```
nums3
```

Out[106… [[10, 30], [1, 'hi', 'hello', 'nit'], 'a', 5, 6.6]

In [107…
```
nums
```

Out[107… [10, 30]

In [108…
```
min(nums)
```

Out[108… 10

In [109…
```
max(nums)
```

Out[109… 30

In [110…
```
nums2
```

Out[110… ['hi', 8.9, 20, 30, 10]

In [111…
```
sum(nums)
```

Out[111… 40

In [112…
```
nums.sort()
```

In [113…
```
nums
```

Out[113… [10, 30]

In [114…
```
nums.sort(reverse=True)
```

In [115…
```
nums
```

Out[115… [30, 10]

In [ ]:
```

```

Tuple

In [116…
```
tup = (15,25,35)
tup
```

Out[116… (15, 25, 35)

In [117…
```
tup[1]
```

Out[117… 25

In [ ]:

Set

In [118…
```
s={}
```

In [119…
```
s1={21,2,54,36}
```

In [120…
```
s1
```

Out[120… {2, 21, 36, 54}

In [121…
```
s3={50,82,62,10,20}
```

In [122…
```
s3
```

Out[122… {10, 20, 50, 62, 82}

In [ ]:

Dictionary

In [123…
```
data = {1:'apple',2:'banana',3:'orange'}
data
```

Out[123… {1: 'apple', 2: 'banana', 3: 'orange'}

In [124…
```
data[2]
```

Out[124… 'banana'

In [125…
```
data.get(2)
```

Out[125… 'banana'

In [126…
```
print(data.get(3))
```

orange

In [127…
```
data.get(1,'not found')
```

Out[127… 'apple'

In [128…
```
data.get(4,'not found')
```

Out[128…
```
'not found'
```

In [129…
```
data
```

Out[129…
```
{1: 'apple', 2: 'banana', 3: 'orange'}
```

In [130…
```
del data[2]
```

In [131…
```
data
```

Out[131…
```
{1: 'apple', 3: 'orange'}
```

In [ ]:

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 [132…
```
help()
```

```
Welcome to Python 3.12's help utility! If this is your first time using
Python, you should definitely check out the tutorial at
https://docs.python.org/3.12/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To get a list of available
modules, keywords, symbols, or topics, enter "modules", "keywords",
"symbols", or "topics".

Each module also comes with a one-line summary of what it does; to list
the modules whose name or summary contain a given string such as "spam",
enter "modules spam".

To quit this help utility and return to the interpreter,
enter "q" or "quit".

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
```

In [133…
```
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return bool(key in self).
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, index, /)
 |      Return self[index].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
```

```
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(self, /)
 |      Return a reverse iterator over the list.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |
 |  __sizeof__(self, /)
 |      Return the size of the list in memory, in bytes.
 |
 |  append(self, object, /)
 |      Append object to the end of the list.
 |
 |  clear(self, /)
 |      Remove all items from list.
 |
 |  copy(self, /)
 |      Return a shallow copy of the list.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  insert(self, index, object, /)
 |      Insert object before index.
 |
 |  pop(self, index=-1, /)
 |      Remove and return item at index (default last).
 |
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(self, value, /)
 |      Remove first occurrence of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |
 |  sort(self, /, *, key=None, reverse=False)
 |      Sort the list in ascending order and return None.
 |
 |      The sort is in-place (i.e. the list itself is modified) and stable (i.e.
the
 |      order of two equal elements is maintained).
```

```
 |
 |      If a key function is given, apply it once to each list item and sort the
m,
 |      ascending or descending, according to their function values.
 |
 |      The reverse flag can be set to sort in descending order.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...)
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

In [134…

```
help(tuple)
```

```
Help on class tuple in module builtins:

class tuple(object)
 |  tuple(iterable=(), /)
 |
 |  Built-in immutable sequence.
 |
 |  If no argument is given, the constructor returns an empty tuple.
 |  If iterable is specified the tuple is initialized from iterable's items.
 |
 |  If the argument is a tuple, the return value is the same object.
 |
 |  Built-in subclasses:
 |      asyncgen_hooks
 |      UnraisableHookArgs
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return bool(key in self).
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
```

```
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(...)
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs)
|      Create and return a new object.  See help(type) for accurate signature.
```

In [ ]:

introduction to ID()

In [135…
```
num=5
id(num)
```

Out[135…
```
140730897545784
```

In [136…
```
name='nit'
```

In [137…
```
id(name)
```

Out[137…
```
2501124967040
```

In [138…
```
a=10
id(a)
```

Out[138…
```
140730897545944
```

In [139…
```
b=a
```

In [140…
```
id(b)
```

Out[140…
```
140730897545944
```

In [141…
```
PI = 3.14
PI
```

Out[141… 3.14

In [142…
```
PI =3.18
PI
```

Out[142… 3.18

In [143…
```
type(PI)
```

Out[143… float

In [ ]:

Data types and Data structures

In [144…
```
w = 2.5
type(w)
```

Out[144… float

In [145…
```
a
```

Out[145… 10

In [146…
```
(a)
```

Out[146… 10

In [147…
```
w2=2+3j
type(w2)
```

Out[147… complex

In [148…
```
a=5.6
b=int(a)
```

In [149…
```
b
```

Out[149… 5

In [150…
```
type(b)
```

Out[150… int

In [151…
```
type(a)
```

Out[151… float

In [152…
```
k = float(b)
```

In [153…
```
k
```

Out[153…
```
5.0
```

In [154…
```
print(a)
print(b)
print(k)
```

```
5.6
5
5.0
```

In [155…
```
k1 = complex(b,k)
k1
```

Out[155…
```
(5+5j)
```

In [156…
```
print(k1)
```

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

In [157…
```
type(k1)
```

Out[157…
```
complex
```

In [158…
```
b < k
```

Out[158…
```
False
```

In [159…
```
condition = b<k
condition
```

Out[159…
```
False
```

In [160…
```
type(condition)
```

Out[160…
```
bool
```

In [161…
```
int(True)
```

Out[161…
```
1
```

In [162…
```
int(False)
```

Out[162…
```
0
```

In [163…
```
l= [1,2,3,4]
```

In [164…
```
print(l)
type(l)
```

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

Out[164…
```
list
```

In [165…
```
s={1,2,3,4,5,6}
s
```

Out[165… {1, 2, 3, 4, 5, 6}

In [166…
```
type(s)
```

Out[166… set

In [167…
```
t=(10,20,30)
t
```

Out[167… (10, 20, 30)

In [168…
```
type(t
    )
```

Out[168… tuple

In [169…
```
str = 'nit'
type(str)
```

Out[169… str

In [ ]:

range

In [170…
```
r = range(0,10)
```

In [171…
```
r
```

Out[171… range(0, 10)

In [172…
```
type(r)
```

Out[172… range

In [173…
```
list(range(0,10))
```

Out[173… [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [174…
```
r1 = list(r)
```

In [175…
```
r1
```

Out[175… [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [176…
```
even_number = list(range(2,10,2))
even_number
```

Out[176… [2, 4, 6, 8]

In [177…
```
d= {1:'one', 2:'two', 3:'three'}
d
```

Out[177… {1: 'one', 2: 'two', 3: 'three'}

In [178…
```
type(d)
```

Out[178… dict

In [179…
```
d.keys()
```

Out[179… dict_keys([1, 2, 3])

In [180…
```
d.values()
```

Out[180… dict_values(['one', 'two', 'three'])

In [181…
```
d[2]
```

Out[181… 'two'

In [182…
```
d.get(2)
```

Out[182… 'two'

In [ ]:

Operators in python

In [ ]:

arithmetic operators

In [183…
```
x1, y1 = 10, 5
```

In [184…
```
x1
```

Out[184… 10

In [185…
```
y1
```

Out[185… 5

In [186…
```
x1 +y1
```

Out[186… 15

In [187…
```
x1 - y1
```

Out[187… 5

In [188…
```
x1 * y1
```

Out[188… 50

In [189…
```
x1 / y1
```
Out[189… 2.0

In [190…
```
x1 // y1
```
Out[190… 2

In [191…
```
x1 % y1
```
Out[191… 0

In [192…
```
x1 ** y1
```
Out[192… 100000

In [193…
```
2 ** 3
```
Out[193… 8

In [ ]:

In [194…
```
#Assignment operators
```
In [195…
```
x=2
```
In [196…
```
x= x+2
```
In [197…
```
x
```
Out[197… 4

In [198…
```
x += 2
```
In [199…
```
x
```
Out[199… 6

In [200…
```
x *= 2
x
```
Out[200… 12

In [201…
```
x -= 2
x
```
Out[201… 10

In [207…
```
x /= 2
```
In [208…
```
x
```
Out[208… 0.5

In [204…
```
x //= 2
x
```

Out[204… 2.0

In [209…
```
a,b = 5 ,6
```

In [210…
```
a
```

Out[210… 5

In [211…
```
b
```

Out[211… 6

In [ ]:

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 [212…
```
n = 7
```

In [213…
```
n
```

Out[213… 7

In [214…
```
m = -(n)
```

In [215…
```
m
```

Out[215… -7

In [216…
```
n
```

Out[216… 7

In [217…
```
-n
```

Out[217… -7

In [218…
```
-m
```

Out[218… 7

In [ ]:

Relational operator

we are using this operator for comparing

In [219…
```
a = 5
b= 7
```

In [220…
```
a == b
```

Out[220… False

In [221…
```
a>b
```

Out[221… False

In [222…
```
a<b
```

Out[222… True

In [223…
```
a == b
```

Out[223… False

In [224…
```
a = 10
```

In [225…
```
a
```

Out[225… 10

In [226…
```
a != b
```

Out[226… True

In [227…
```
b = 10
```

In [228…
```
a == b
```

Out[228… True

In [229…
```
a >= b
```

Out[229… True

In [230…
```
a <= b
```

Out[230… True

In [231…
```
a < b
```

Out[231… False

In [232…
```
a>b
```

Out[232… False

In [233…
```
b = 7
```

In [234…
```
b
```

Out[234… 7

In [235…
```
a != b
```

Out[235… True

In [ ]:

Logical operator

AND, OR, NOT

In [237…
```
a = 5
b = 4
```

In [238…
```
a
```

Out[238… 5

In [239…
```
b
```

Out[239… 4

In [240…
```
a < 8 and b < 5
```

Out[240… True

In [241…
```
a < 8 and b < 2
```

Out[241… False

In [244…
```
a<8 or b<2
```

Out[244… True

In [243…
```
a>8 or b<2
```

Out[243… False

In [245…
```
x = False
x
```

Out[245… False

In [246…
```
not x
```

Out[246… True

In [247…
```
x = not x
x
```

Out[247… True

In [248…
```
x
```
Out[248… True

In [249…
```
not x
```
Out[249… False

In [ ]:
```

```

Number system conversion (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 ip address you will these format --> cmd - ipconfig

In [250…
```
25
```
Out[250… 25

In [251…
```
bin(25)
```
Out[251… '0b11001'

In [252…
```
bin(35)
```
Out[252… '0b100011'

In [253…
```
int(0b100011)
```
Out[253… 35

In [256…
```
int(0b1111)
```
Out[256… 15

In [257…
```
oct(15)
```
Out[257… '0o17'

In [258…
```
hex(9)
```
Out[258… '0x9'

In [259…
```
0xf
```
Out[259… 15

In [260…
```
hex(25)
```
Out[260… '0x19'

In [261…
```
0x19
```

Out[261… 25

In [262…
```
0x15
```

Out[262… 21

swap variable in python

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

In [263…
```
a = 5
b = 6
```

In [264…
```
a = b
b = a
```

In [265…
```
a,b
```

Out[265… (6, 6)

In [266…
```
a,b = b,a
```

In [267…
```
print(a)
print(b)
```

```
6
6
```

In [268…
```
a1 = 7
b1= 8
```

In [269…
```
temp = a1
a1 = b1
b1 = temp
```

In [270…
```
print(a1)
print(b1)
```

```
8
7
```

In [271…
```
a2 = 5
b2 = 6
```

In [274…
```
#swap variable formulas
a2 = a2 + b2
b2 = a2 - b2
a2 = a2 - b2
```

In [273…
```
print(a2)
print(b2)
```

```
6
5
```

In [275…
```
print(0b101)
print(0b110)
```

```
5
6
```

In [276…
```
print(bin(11))
print(0b1011)
```

```
0b1011
11
```

In [277…
```
a2 = a2 ^ b2
b2 = a2 ^ b2
a2 = a2 ^ b2
```

In [278…
```
print(a2)
print(b2)
```

```
6
5
```

In [279…
```
print(a2)
print(b2)
```

```
6
5
```

In [280…
```
a2, b2 = b2, a2
```

In [281…
```
print(a2)
print(b2)
```

```
5
6
```

In [ ]:

Bitwise operator

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

In [ ]:

In [282…
```
print(bin(12))
print(bin(13))
```

```
0b1100
0b1101
```

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

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

In [ ]:

In [283…
```
~45
```
Out[283… -46

In [284…
```
~6
```
Out[284… -7

In [285…
```
~-6
```
Out[285… 5

In [286…
```
~-1
```
Out[286… 0

In [287…
```
~1
```
Out[287… -2

In [288…
```
~0
```
Out[288… -1

In [289…
```
~-0
```
Out[289… -1

In [ ]:

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 [290…
```
12 & 13
```
Out[290… 12

In [291…
```
1 & 1
```
Out[291… 1

In [292…
```
1 | 0
```
Out[292… 1

In [293…
```
1 & 0
```
Out[293… 0

In [294…
```
12 | 13
```
Out[294… 13

In [295…
```
35 & 40
```
Out[295… 32

In [296…
```
35 | 40
```
Out[296… 43

In [297…
```
bin(35)
```
Out[297… '0b100011'

In [298…
```
bin(40)
```
Out[298… '0b101000'

in XOR if the both number are different then we will get 1 or else we will get 0

In [ ]:
```

```

In [299…
```
25 ^ 30
```
Out[299… 7

In [300…
```
bin(25)
```
Out[300… '0b11001'

In [301…
```
bin(30)
```
Out[301… '0b11110'

In [302…
```
int(0b000111)
```
Out[302… 7

In [303…
```
# 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
```

Out[303… 40

In [304…
```
20<<4
```

Out[304… 320

In [305…
```
# BIT WISE RIGHT OPERATOR
```

In [306…
```
10>>2
```

Out[306… 2

In [307…
```
bin(20)
```

Out[307… '0b10100'

In [308…
```
 20>>4
```

Out[308… 1

In [ ]:

import math module

In [309…
```
x = sqrt(25)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[309], line 1
----> 1 x = sqrt(25)

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

In [311…
```
import math
```

In [312…
```
x = math.sqrt(25)
x
```

Out[312… 5.0

In [314…
```
x1 = math.sqrt(225)
x1
```

Out[314… 15.0

In [317…
```
print(math.floor(145.3))
```

```
145
```

In [318…
```
print(math.ceil(145.3))
```

```
146
```

In [ ]:

In [320…
```
print(math.pow(6,2))
```
```
36.0
```

In [321…
```
print(math.pow(6,4))
```
```
1296.0
```

In [322…
```
print(math.pi)
```
```
3.141592653589793
```

In [323…
```
print(math.e)
```
```
2.718281828459045
```

In [ ]:

In [324…
```
import math as m
m.sqrt (10)
```

Out[324… 3.1622776601683795

In [325…
```
from math import sqrt,pow
pow(2,3)
```

Out[325… 8.0

In [326…
```
round(pow(2,3))
```

Out[326… 8

In [ ]:

user input function in python || command line input

In [327…
```
x = input()
y = input()
z = x + y
print(z)
```
```
8020
```

In [328…
```
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)
```
```
8020
```

In [329…
```
type(x1)
type(y1)
```

Out[329… str

In [330…
```
x1 = input()
a1 = int(x1)

y1 = input()
b1= int(y1)

z1 = a1 + b1
print (z1)
```

```
100
```

In [331…
```
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)
```

```
8020
```

In [332…
```
x2 = int(input('Enter the 1st number'))
y2 = int(input('Enter the 2nd number'))
z2 = x2 + y2
z2
```

Out[332… 100

In [ ]:

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

In [333…
```
ch = input()
print(ch)
```

```
jjx
```

In [334…
```
print(ch[0])
```

```
j
```

In [335…
```
print(ch[1])
```

```
j
```

In [336…
```
print(ch[-1])
```

```
x
```

In [338…
```
ch = input()[0]
print(ch)
```

```
b
```

In [339…
```
ch = input('enter a char')[1:3]
print(ch)
```

```
gg
```

In [340…
```
ch = input('enter a char')
print(ch)
```

```
dbd
```

EVAL function using input

In [342…
```
result = eval(input('enter an expr'))
print(result)
```

```
5534
```

In [ ]:
```

```