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

Out[1]: 2

In [2]:
```
2+8
```

Out[2]: 10

In [3]:
```
9-2
```

Out[3]: 7

In [4]:
```
12+30
```

Out[4]: 42

In [5]:
```
10+15
```

Out[5]: 25

In [6]:
```
10-5
```

Out[6]: 5

In [7]:
```
35-10.8
```

Out[7]: 24.2

In [8]:
```
40-6.5
```

Out[8]: 33.5

In [9]:
```
40-45.9
```

Out[9]: -5.899999999999999

In [10]:
```
12*8
```

Out[10]: 96

In [11]:
```
2*8
```

Out[11]: 16

In [12]:
```
4*9
```

Out[12]: 36

In [13]:
```
35/7  #float division
```

Out[13]: 5.0

In [14]:
```
96/5
```

Out[14]: 19.2

In [15]:
```
96/9
```

Out[15]: 10.666666666666666

In [16]:
```
100/8
```

Out[16]: 12.5

In [17]:
```
100//8   #Integer division
```

Out[17]: 12

In [18]:
```
45//7
```

Out[18]: 6

In [19]:
```
45/7
```

Out[19]: 6.428571428571429

In [20]:
```
9+5-2
```

Out[20]: 12

In [21]:
```
12+5-
```

```
  Cell In[21], line 1
    12+5-
         ^
SyntaxError: invalid syntax
```

In [22]:
```
5+5*10
```

Out[22]: 55

In [23]:
```
10+2*5
```

Out[23]: 20

In [24]:
```
(10+5)*2*7
```

Out[24]: 210

In [25]:
```
(10+5)+2*5
```

Out[25]: 25

In [26]:
```
#exponentation
2*2*2*2*2
```

Out[26]: 32

In [27]:
```
2**5
```

Out[27]: 32

In [28]:
```
4*4*4*4
```

Out[28]: 256

In [29]:
```
4**4
```

Out[29]: 256

In [30]:
```
#Modulus
10%5
```

Out[30]: 0

In [31]:
```
10%2
```

Out[31]: 0

In [32]:
```
14%2
```

Out[32]: 0

In [33]:
```
14%3
```

Out[33]: 2

In [34]:
```
21%7
```

Out[34]: 0

In [35]:
```
21%3
```

Out[35]: 0

In [36]:
```
21%5
```

Out[36]: 1

In [37]:
```
21%%5
```

```
  Cell In[37], line 1
    21%%5
       ^
SyntaxError: invalid syntax
```

In [38]:
```
a,b,c,d,e=10,12.5,'Hello',True,10+5j
print(a)
print(b)
print(c)
print(d)
print(e)
```

```
10
12.5
Hello
True
(10+5j)
```

In [39]:
```
print(type(a))
print(type(b))
print(type(c))
print(type(d))
print(type(e))
```

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

In [40]:
```
'Naresh it'
```

Out[40]: 'Naresh it'

In [41]:
```
'Max it Technologies'
```

Out[41]: 'Max it Technologies'

In [42]:
```
a='Naresh it'
a
```

Out[42]: 'Naresh it'

In [43]:
```
a=2
b=3
a+b
```

Out[43]: 5

In [44]:
```
c=a+b
```

In [45]:
```
c
```

Out[45]: 5

In [46]:
```
type(c)
```

Out[46]: int

In [47]:
```
a=5
b='Hello'
c=a+b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[47], line 3
      1 a=5
      2 b='Hello'
----> 3 c=a+b

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

In [ ]:
```
c
```

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

In [ ]:
```
print('nareshit's 'Technology')
```

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

In [ ]:
```
print('Hello' 'Venky')
```

In [ ]:
```
print('Hello' ' Venky')
```

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

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

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

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

In [ ]:
```
print("c:\nit")
```

In [ ]:
```
print("C:\Users\evenk\OneDrive\Desktop\DS_NIT\18. Sample Superstore for M Functions
```

In [ ]:
```
print(r"C:\Users\evenk\OneDrive\Desktop\DS_NIT\18. Sample Superstore for M Function
```

Variable || Identifier || Object

In [ ]:
```
2
```

In [ ]:
```
x=5
x
```

In [ ]:
```
x+7
```

In [ ]:
```
y=5
y
```

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

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

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

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

In [ ]:
```
y
```

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

In [ ]:
```
_+x
```

In [ ]:
```
_+x
```

In [ ]:
```
_+3.5
```

In [ ]:
```
_
```

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

In [ ]:
```
_-10
```

In [48]:
```
_*5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[48], line 1
----> 1 _*5

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

In [ ]:
```
x
```

In [ ]:
```
y
```

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

In [49]:
```
name
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[49], line 1
----> 1 name

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

In [50]:
```
name+' technology'
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[50], line 1
----> 1 name+' technology'

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

In [51]:
```
name 'technology'
```

```
  Cell In[51], line 1
    name 'technology'
         ^
SyntaxError: invalid syntax
```

In [52]:
```
name
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[52], line 1
----> 1 name

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

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

In [53]:
```
name[0]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[53], line 1
----> 1 name[0]

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

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

In [54]:
```
name[:]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[54], line 1
----> 1 name[:]

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

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

In [55]:
```
name[2:8]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[55], line 1
----> 1 name[2:8]

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

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

In [56]:
```
name[-2]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[56], line 1
----> 1 name[-2]

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

Slicing

In [57]:
```
name='Hello welcome to python world!'
```

In [58]:
```
name
```

Out[58]: 'Hello welcome to python world!'

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

Out[59]: ' '

In [60]:
```
name[4]
```

Out[60]: 'o'

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

Out[61]: '!'

In [62]:
```
name[0:2]
```

Out[62]: 'He'

In [63]:
```
name[1:7]
```

Out[63]: 'ello w'

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

Out[64]: 'ello welcome to python world!'

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

Out[65]: 'Hello welcome to python world!'

In [66]:
```
name[3:5]
```

Out[66]: 'lo'

In [67]:
```
name[2:8]
```

Out[67]: 'llo we'

In [68]:
```
name[:8]
```

Out[68]: 'Hello we'

In [69]:
```
name[:5]
```

Out[69]: 'Hello'

In [70]:
```
name1='fine'
name1
```

Out[70]: 'fine'

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

Out[71]: 'f'

In [72]:
```
name1[0:1]='d'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[72], line 1
----> 1 name1[0:1]='d'

TypeError: 'str' object does not support item assignment
```

In [73]:
```
name1[0]='d'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[73], line 1
----> 1 name1[0]='d'

TypeError: 'str' object does not support item assignment
```

In [ ]:
```
name1
```

In [74]:
```
name1[1:]
```

Out[74]: 'ine'

In [75]:
```
'd'+name1[1:]
```

Out[75]: 'dine'

In [76]:
```
#write a python program to print fine to dine
name='fine'
name1='d'+name[1:]
print(name1)
```

```
dine
```

In [77]:
```
name='dine'
name1='w'+name[1:]
print(name1)
```

```
wine
```

In [78]:
```
name='Beer'
name1='D'+name[1:]
name1
```

Out[78]: 'Deer'

In [79]:
```
name='Bear'
name1="D"+name[1:]
name1
```

Out[79]: 'Dear'

In [80]:
```
name='Lion'
name1=name+' King'
name1
```

Out[80]: 'Lion King'

List

In [81]:
```
num=[10,20,30,40]
```

In [82]:
```
num
```

Out[82]: [10, 20, 30, 40]

In [83]:
```
num[0]
```

Out[83]: 10

In [84]:
```
num[2]
```

Out[84]: 30

In [85]:
```
num[-1]
```

Out[85]: 40

In [86]:
```
num[2:]
```

Out[86]: [30, 40]

In [87]:
```
num[:-1]
```

Out[87]: [10, 20, 30]

In [88]:
```
num[:4]
```

Out[88]: [10, 20, 30, 40]

In [89]:
```
num[1:3]
```

Out[89]: [20, 30]

In [90]:
```
num1=['hello','Venky']
```

In [91]:
```
num1
```

Out[91]: ['hello', 'Venky']

In [92]:
```
num2=['Venky',8.9,34,True,2+6j]
num2
```

Out[92]: ['Venky', 8.9, 34, True, (2+6j)]

In [93]:
```
num3=[num+num1]
num3
```

Out[93]: [[10, 20, 30, 40, 'hello', 'Venky']]

In [94]:
```
num3
```

Out[94]: [[10, 20, 30, 40, 'hello', 'Venky']]

In [95]:
```
num1
```

Out[95]: ['hello', 'Venky']

In [96]:
```
num2
```

Out[96]: ['Venky', 8.9, 34, True, (2+6j)]

In [97]:
```
num3
```

Out[97]: [[10, 20, 30, 40, 'hello', 'Venky']]

In [98]:
```
num4=[num,num1,num2]
num4
```

Out[98]:
```
[[10, 20, 30, 40], ['hello', 'Venky'], ['Venky', 8.9, 34, True, (2+6j)]]
```

In [99]:
```
num.append(45)
```

In [100…
```
num
```

Out[100…
```
[10, 20, 30, 40, 45]
```

In [101…
```
num.remove(30)
```

In [102…
```
num
```

Out[102…
```
[10, 20, 40, 45]
```

In [103…
```
num.pop()
```

Out[103…
```
45
```

In [104…
```
num
```

Out[104…
```
[10, 20, 40]
```

In [105…
```
num.insert(5,6)
```

In [106…
```
num
```

Out[106…
```
[10, 20, 40, 6]
```

In [107…
```
num.insert(0,5)
```

In [108…
```
num
```

Out[108…
```
[5, 10, 20, 40, 6]
```

In [109…
```
num1.insert(0,1)
```

In [110…
```
num1
```

Out[110…
```
[1, 'hello', 'Venky']
```

num

In [111…
```
num.insert(0,10)
```

In [112…
```
num
```

Out[112…
```
[10, 5, 10, 20, 40, 6]
```

In [113…
```
del num[2:]
```

In [114…
```
num
```

Out[114… [10, 5]

In [115…
```
num.extend([29,15,20])
```

In [116…
```
num
```

Out[116… [10, 5, 29, 15, 20]

In [117…
```
num3
```

Out[117… [[10, 20, 30, 40, 'hello', 'Venky']]

In [118…
```
del num3[3:]
```

In [119…
```
num3
```

Out[119… [[10, 20, 30, 40, 'hello', 'Venky']]

In [120…
```
min(num)
```

Out[120… 5

In [121…
```
max(num)
```

Out[121… 29

In [122…
```
sum(num)
```

Out[122… 79

In [123…
```
num.sort()
```

In [124…
```
num
```

Out[124… [5, 10, 15, 20, 29]

Tuple

In [125…
```
tup= tuple()
tup
```

Out[125… ()

In [126…
```
tup1=[2,5,10,15,20,9]
tup1
```

Out[126… [2, 5, 10, 15, 20, 9]

In [127…
```
tup1[0]
```

Out[127… 2

In [128…
```
tup[0]=10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[128], line 1
----> 1 tup[0]=10

TypeError: 'tuple' object does not support item assignment
```

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

In [129…
```
tup1
```

Out[129… [2, 5, 10, 15, 20, 9]

SET

In [130…
```
s={}
type(s)
```

Out[130… dict

In [131…
```
s=set()
```

In [132…
```
type(s)
```

Out[132… set

In [133…
```
s1={2,16,34,58,5}
s1
```

Out[133… {2, 5, 16, 34, 58}

In [134…
```
s3={50,35,53,'nit',53}
```

In [135…
```
s3
```

Out[135… {35, 50, 53, 'nit'}

In [136…
```
s1[1] #as we dont have proper sequencing thats why indexing not subscriptable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[136], line 1
----> 1 s1[1] #as we dont have proper sequencing thats why indexing not subscriptabl
e

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

DICTIONARY

In [137…
```
data={1:'apple',2:'banana',4:'ornage'}
data
```

Out[137…
```
{1: 'apple', 2: 'banana', 4: 'ornage'}
```

In [138…
```
data[4]
```

Out[138…
```
'ornage'
```

In [139…
```
data.items()
```

Out[139…
```
dict_items([(1, 'apple'), (2, 'banana'), (4, 'ornage')])
```

In [140…
```
data.values()
```

Out[140…
```
dict_values(['apple', 'banana', 'ornage'])
```

In [141…
```
data.keys()
```

Out[141…
```
dict_keys([1, 2, 4])
```

In [142…
```
data.get(2)
```

Out[142…
```
'banana'
```

In [146…
```
data.values()
```

Out[146…
```
dict_values(['apple', 'banana', 'ornage'])
```

In [148…
```
data.get(3)
```

In [149…
```
data
```

Out[149…
```
{1: 'apple', 2: 'banana', 4: 'ornage'}
```

In [150…
```
data.get(4)
```

Out[150…
```
'ornage'
```

In [151…
```
print(data.get(3))
```

```
None
```

In [152…
```
data.get(1,'Not Found')
```

Out[152…
```
'apple'
```

In [153…
```
data.get(3,'Not Found')
```

Out[153…
```
'Not Found'
```

In [154…
```
data[5]='five'
```

In [155…
```
data
```

Out[155…
```
{1: 'apple', 2: 'banana', 4: 'ornage', 5: 'five'}
```

In [156…
```
del data[5]
```

In [157…
```
data
```

Out[157…
```
{1: 'apple', 2: 'banana', 4: 'ornage'}
```

In [159…
```
#list in the dictionary
prog = {'python':['vscode', 'pycharm'], 'machine learning' : 'sklearn', 'datascienc
```

In [160…
```
prog
```

Out[160…
```
{'python': ['vscode', 'pycharm'],
 'machine learning': 'sklearn',
 'datascience': ['jupyter', 'spyder']}
```

In [161…
```
prog['python']
```

Out[161…
```
['vscode', 'pycharm']
```

In [162…
```
prog['machine learning']
```

Out[162…
```
'sklearn'
```

In [163…
```
prog['datascience']
```

Out[163…
```
['jupyter', 'spyder']
```

Help

In [164…
```
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".

```
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 them,
|      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
```

```
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.

No Python documentation found for 'dictionary'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.
```

```
Help on class dict in module builtins:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
 |      in the keyword argument list.  For example:  dict(one=1, two=2)
 |
 |  Built-in subclasses:
 |      StgDict
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
 |      True if the dictionary has the specified key, else False.
 |
 |  __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, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
```

```
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __or__(self, value, /)
|      Return self|value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the dict keys.
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(...)
|      D.__sizeof__() -> size of D in memory, in bytes
|
|  clear(...)
|      D.clear() -> None.  Remove all items from D.
|
|  copy(...)
|      D.copy() -> a shallow copy of D
|
|  get(self, key, default=None, /)
|      Return the value for key if key is in the dictionary, else default.
|
|  items(...)
|      D.items() -> a set-like object providing a view on D's items
|
|  keys(...)
|      D.keys() -> a set-like object providing a view on D's keys
|
|  pop(...)
|      D.pop(k[,d]) -> v, remove specified key and return the corresponding value.
|
|      If the key is not found, return the default if given; otherwise,
|      raise a KeyError.
|
|  popitem(self, /)
|      Remove and return a (key, value) pair as a 2-tuple.
|
|      Pairs are returned in LIFO (last-in, first-out) order.
|      Raises KeyError if the dict is empty.
|
|  setdefault(self, key, default=None, /)
|      Insert key with a value of default if key is not in the dictionary.
|
|      Return the value for key if key is in the dictionary, else default.
|
|  update(...)
|      D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
```

```
 |      If E is present and has a .keys() method, then does:  for k in E: D[k] = E
[k]
 |      If E is present and lacks a .keys() method, then does:  for k, v in E: D[k]
= v
 |      In either case, this is followed by: for k in F:  D[k] = F[k]
 |
 |  values(...)
 |      D.values() -> an object providing a view on D's values
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...)
 |      See PEP 585
 |
 |  fromkeys(iterable, value=None, /)
 |      Create a new dictionary with keys from iterable and values set to value.
 |
 |  ----------------------------------------------------------------------
 |  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

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 [165…

```
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 them,
|      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 [169…

```
help(set)
```

```
Help on class set in module builtins:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __ixor__(self, value, /)
 |      Return self^=value.
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
```

```
|      Return self!=value.
|
|  __or__(self, value, /)
|      Return self|value.
|
|  __rand__(self, value, /)
|      Return value&self.
|
|  __reduce__(...)
|      Return state information for pickling.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(...)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __xor__(self, value, /)
|      Return self^value.
|
|  add(...)
|      Add an element to a set.
|
|      This has no effect if the element is already present.
|
|  clear(...)
|      Remove all elements from this set.
|
|  copy(...)
|      Return a shallow copy of a set.
|
|  difference(...)
|      Return the difference of two or more sets as a new set.
|
|      (i.e. all elements that are in this set but not the others.)
|
|  difference_update(...)
|      Remove all elements of another set from this set.
|
|  discard(...)
|      Remove an element from a set if it is a member.
|
|      Unlike set.remove(), the discard() method does not raise
|      an exception when an element is missing from the set.
```

```
|
|  intersection(...)
|      Return the intersection of two sets as a new set.
|
|      (i.e. all elements that are in both sets.)
|
|  intersection_update(...)
|      Update a set with the intersection of itself and another.
|
|  isdisjoint(...)
|      Return True if two sets have a null intersection.
|
|  issubset(self, other, /)
|      Test whether every element in the set is in other.
|
|  issuperset(self, other, /)
|      Test whether every element in other is in the set.
|
|  pop(...)
|      Remove and return an arbitrary set element.
|      Raises KeyError if the set is empty.
|
|  remove(...)
|      Remove an element from a set; it must be a member.
|
|      If the element is not a member, raise a KeyError.
|
|  symmetric_difference(...)
|      Return the symmetric difference of two sets as a new set.
|
|      (i.e. all elements that are in exactly one of the sets.)
|
|  symmetric_difference_update(...)
|      Update a set with the symmetric difference of itself and another.
|
|  union(...)
|      Return the union of sets as a new set.
|
|      (i.e. all elements that are in either set.)
|
|  update(...)
|      Update a set with the union of itself and others.
|
|  ----------------------------------------------------------------------
|  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
```

Introduce to ID

In [170…
```
num=5
id(num)
```

Out[170…
```
140734214978104
```

In [171…
```
name='venky'
id(name)
```

Out[171…
```
1847280831248
```

In [172…
```
a=10
id(a)
```

Out[172…
```
140734214978264
```

In [173…
```
b=a
```

In [178…
```
print(id(a))
print(id(b))
print(id(10))
```

```
140734214978264
140734214978264
140734214978264
```

In [175…
```
b==a
```

Out[175…
```
True
```

In [179…
```
id(10)
```

Out[179…
```
140734214978264
```

In [180…
```
k=10
id(k)
```

Out[180…
```
140734214978264
```

In [181…
```
a=20
id(a)
```

Out[181…
```
140734214978584
```

In [182…
```
id(b)
```

Out[182… 140734214978264

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 [184…
```
PI=3.14
PI
```

Out[184… 3.14

In [185…
```
PI=3.18
PI
```

Out[185… 3.18

In [186…
```
type(PI)
```

Out[186… float

DATA TYPES AND DATA STRUCTURES--->

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

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

In [187…
```
w=2.5
type(w)
```

Out[187… float

In [188…
```
a
```

Out[188… 20

In [189…
```
(a)
```

Out[189… 20

In [190…
```
w1=2+3j
type(w1)
```

Out[190… complex

In [191…
```
a=5.6
b=int(a)
b
```

Out[191… 5

In [192…
```
type(b)
```

Out[192…
```
int
```

In [193…
```
type(a)
```

Out[193…
```
float
```

In [198…
```
k
```

Out[198…
```
5.0
```

In [200…
```
print(a)
print(b)
print(k)
```

```
5.6
5
5.0
```

In [201…
```
k1=complex(b,k)
```

In [202…
```
k1
```

Out[202…
```
(5+5j)
```

In [203…
```
k2=complex(a,b)
k2
```

Out[203…
```
(5.6+5j)
```

In [204…
```
print(k2)
type(k2)
```

```
(5.6+5j)
```

Out[204…
```
complex
```

In [205…
```
b<k
```

Out[205…
```
False
```

In [206…
```
condition=b<k
condition
```

Out[206…
```
False
```

In [207…
```
type(condition)
```

Out[207…
```
bool
```

In [208…
```
int(True)
```

Out[208…
```
1
```

In [209…
```
int(False)
```

Out[209…
```
0
```

In [211…
```
float(True)
```

Out[211…
```
1.0
```

In [212…
```
complex(True)
```

Out[212…
```
(1+0j)
```

In [213…
```
l=[1,2,3,4]
print(l)
type(l)
```

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

Out[213…
```
list
```

In [214…
```
s={1,2,3,4}
s
```

Out[214…
```
{1, 2, 3, 4}
```

In [215…
```
type(s)
```

Out[215…
```
set
```

In [216…
```
s1={1,2,3,4,4,5,3,1,12}  #duplicate values are not allowed
s1
```

Out[216…
```
{1, 2, 3, 4, 5, 12}
```

In [217…
```
t=(10,20,30)
t
```

Out[217…
```
(10, 20, 30)
```

In [218…
```
type(t)
```

Out[218…
```
tuple
```

In [219…
```
str='Hello Venky'
type(str)
```

Out[219…
```
str
```

In [220…
```
str1='nit'
str1
```

Out[220… 'nit'

range()

In [221…
```
r=range(0,10)
r
```

Out[221… range(0, 10)

In [222…
```
type(r)
```

Out[222… range

In [223…
```
#if you want to print the range
list(range(0,10))
```

Out[223… [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [224…
```
r1=list(r)
r1
```

Out[224… [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [231…
```
#if you want to print the even numbers
even_number=list(range(2,10,2))
even_number
```

Out[231… [2, 4, 6, 8]

In [228…
```
#if you want to print the odd numbers
odd_numbers=list(range(1,20,2))
odd_numbers
```

Out[228… [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

In [234…
```
d={1:'one',2:'two',3:'three'}
d
```

Out[234… {1: 'one', 2: 'two', 3: 'three'}

In [235…
```
type(d)
```

Out[235… dict

In [237…
```
d.keys()
```

Out[237… dict_keys([1, 2, 3])

In [238…
```
d.values()
```

Out[238… dict_values(['one', 'two', 'three'])

In [239…
```
d[2]
```

Out[239…
```
'two'
```

In [242…
```
d.get(2)
```

Out[242…
```
'two'
```

Operators in python

1-ARITHMETIC OPERATOR(+,-,*,/,**)

In [244…
```
x1,y1=10,5
```

In [245…
```
x1+y1
```

Out[245…
```
15
```

In [246…
```
x1-y1
```

Out[246…
```
5
```

In [247…
```
x1/y1
```

Out[247…
```
2.0
```

In [248…
```
x1*y1
```

Out[248…
```
50
```

In [249…
```
x1//y1
```

Out[249…
```
2
```

In [250…
```
x1%y1
```

Out[250…
```
0
```

In [251…
```
x1**y1
```

Out[251…
```
100000
```

In [252…
```
2**3
```

Out[252…
```
8
```

In [253…
```
x1%%y1
```

```
  Cell In[253], line 1
    x1%%y1
       ^
SyntaxError: invalid syntax
```

In [255…
```
4**3
```

Out[255…
```
64
```

Assignment operator

In [280…
```
x=2
```

In [281…
```
x=x+2
x
```

Out[281…
```
4
```

In [282…
```
x
```

Out[282…
```
4
```

In [283…
```
x+=4
```

In [284…
```
x
```

Out[284…
```
8
```

In [285…
```
x*=2
```

In [286…
```
x
```

Out[286…
```
16
```

In [287…
```
x-=2
```

In [288…
```
x
```

Out[288…
```
14
```

In [289…
```
x/=2
```

In [290…
```
x
```

Out[290…
```
7.0
```

In [291…
```
x%=2
```

In [292…
```
x
```

Out[292… 1.0

In [294…
```
x1=20
x1
```

Out[294… 20

In [295…
```
x1%=2
```

In [296…
```
x
```

Out[296… 1.0

In [297…
```
a,b=5,6
```

In [298…
```
a
```

Out[298… 5

In [299…
```
b
```

Out[299… 6

Unary operator

Here we are applying unary mminus operator(-) on the operand n: the value of m becomes -7, which indicates it as a negative value

In [300…
```
n=7  #negation
```

In [301…
```
m=-(n)
```

In [302…
```
m
```

Out[302… -7

In [303…
```
n
```

Out[303… 7

In [304…
```
-n
```

Out[304… -7

Relational operator

We are using this operator for comparing

In [305…
```
a=5
b=7
```

In [306…
```
a==b
```

Out[306… False

In [307…
```
a<b
```

Out[307… True

In [308…
```
a>b
```

Out[308… False

In [309…
```
a>=b
```

Out[309… False

In [310…
```
a<=b
```

Out[310… True

In [313…
```
a=10
```

In [314…
```
a==b
```

Out[314… False

In [315…
```
a!=b
```

Out[315… True

In [316…
```
b=10
```

In [317…
```
a==b
```

Out[317… True

In [318…
```
a>=b
```

Out[318… True

In [319…
```
a<=b
```

Out[319… True

In [320…
```
a<b
```

Out[320… False

In [321…
```
a>b
```

Out[321…
```
False
```

In [322…
```
b=7
```

In [323…
```
a!=b
```

Out[323…
```
True
```

Logical operator

AND, OR, NOT

Truth table-----AND(X,Y(1,1=1) remaining all 0) Truth table----OR(X,Y(0,0=0) remaining all 1)

In [324…
```
a=5
b=4
```

In [325…
```
a<8 and b<5
```

Out[325…
```
True
```

In [326…
```
a<8 and b<2
```

Out[326…
```
False
```

In [327…
```
a<8 or b<2
```

Out[327…
```
True
```

In [328…
```
a>8 or b<2
```

Out[328…
```
False
```

In [329…
```
a>8 or b>2
```

Out[329…
```
True
```

In [330…
```
not x  #you can reverse the operation
```

Out[330…
```
False
```

In [332…
```
x=False
```

In [333…
```
x=not x
x
```

Out[333…
```
True
```

In [334…
```
x
```

Out[334…
```
True
```

In [335…
```
not x
```

Out[335…
```
False
```

Number system conversion (bit-binary digit)

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

Octal:base(0-7)

Hexadecimal:base(0-9 &a-f) when you check ipaddress you will find these format==> cmd-ipconfig

In [336…
```
25
```

Out[336…
```
25
```

In [337…
```
bin(25)
```

Out[337…
```
'0b11001'
```

In [338…
```
0b11001
```

Out[338…
```
25
```

In [339…
```
int(0b11001)
```

Out[339…
```
25
```

In [340…
```
bin(35)
```

Out[340…
```
'0b100011'
```

In [341…
```
int(0b100011)
```

Out[341…
```
35
```

In [342…
```
bin(20)
```

Out[342…
```
'0b10100'
```

In [343…
```
0b10100
```

Out[343…
```
20
```

In [344…
```
oct(20)
```

Out[344… '0o24'

In [345…
```
0o24
```

Out[345… 20

In [346…
```
hex(10)
```

Out[346… '0xa'

In [347…
```
0x190
```

Out[347… 400

In [349…
```
hex(400)
```

Out[349… '0x190'

In [350…
```
0xad
```

Out[350… 173

In [351…
```
0x12d
```

Out[351… 301

In [352…
```
0x19
```

Out[352… 25

In [353…
```
0x15
```

Out[353… 21

Swap variable in python

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

In [354…
```
a=5
b=6
```

In [357…
```
a=b
b=a
print(a)
print(b)
```

```
6
6
```

In [358…
```
a,b=b,a
```

In [359…
```
print(a)
print(b)
```

```
6
6
```

In [360…
```
a1=7
b1=8
```

In [361…
```
temp=a1
a1=b1
b1=temp
```

In [363…
```
print('After swapping a1:',a1)
print('After swapping b1:',b1)
```

```
After swapping a1: 8
After swapping b1: 7
```

In [373…
```
a2=5
b2=6
```

In [374…
```
#Swap variable formulas
a2=a2+b2
b2=a2-b2
a2=a2-b2
```

In [375…
```
print(a2)
print(b2)
```

```
6
5
```

In [376…
```
print(0b101)
print(0b110)
```

```
5
6
```

In [377…
```
print(bin(11))
print(0b1011)
```

```
0b1011
11
```

In [378…
```
a2=a2^b2
b2=a2^b2
a2=a2^b2
```

In [379…
```
print(a2)
print(b2)
```

```
5
6
```

In [382…
```
a2
```

Out[382… 6

In [383…
```
b2
```

Out[383… 5

In [380…
```
a2,b2=b2,a2
```

In [381…
```
print(a2)
print(b2)
```

```
6
5
```

BITWISE OPERATOR

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

In [385…
```
print(bin(12))
print(bin(13))
```

```
0b1100
0b1101
```

In [386…
```
~12
```

Out[386… -13

In [387…
```
~10
```

Out[387… -11

In [388…
```
~-13
```

Out[388… 12

In [389…
```
print(bin(-13))
```

```
-0b1101
```

In [392…
```
~45
```

Out[392… -46

In [393…
```
~6
```

Out[393… -7

In [394…
```
~-6
```

Out[394… 5

In [395…
```
~-1
```
Out[395…
```
0
```
In [396…
```
~1
```
Out[396…
```
-2
```

bitwise and operator

Truth table-----AND(X,Y(1,1=1) remaining all 0)---xy Truth table----OR(X,Y(0,0=0) remaining all 1)----x+y

In [398…
```
12&13
```
Out[398…
```
12
```
In [399…
```
1&1
```
Out[399…
```
1
```
In [400…
```
1|0
```
Out[400…
```
1
```
In [401…
```
1&0
```
Out[401…
```
0
```
In [402…
```
12|13
```
Out[402…
```
13
```
In [403…
```
35&40
```
Out[403…
```
32
```
In [404…
```
bin(35)
```
Out[404…
```
'0b100011'
```
In [405…
```
bin(40)
```
Out[405…
```
'0b101000'
```
In [406…
```
int(0b100011)
```
Out[406…
```
35
```
In [407…
```
int(0b101000)
```

Out[407… 40

In [408…
```
35|40
```

Out[408… 43

In [409…
```
# in XOR if the both number are different then we will get 1 or else we will get 0

12 ^ 13
```

Out[409… 1

In [410…
```
12^12
```

Out[410… 0

In [411…
```
25^30
```

Out[411… 7

In [412…
```
bin(25)
```

Out[412… '0b11001'

In [413…
```
bin(30)
```

Out[413… '0b11110'

In [414…
```
int(0b11110)
```

Out[414… 30

In [415…
```
int(0b000111)
```

Out[415… 7

In [416…
```
bin(7)
```

Out[416… '0b111'

In [417…
```
int(0b111)
```

Out[417… 7

In [418…
```
#BITWISE LEFT OPERATOR
#bit wise left operator bydefault you will tak 2 zeros()
# 10 binary operator is 1010 | also i can say 101000   32+8=40
10<<2
```

Out[418… 40

In [419…
```
5<<2 #10100 (16+4)
```

Out[419… 20

In [420…
```
bin(20)
```

Out[420… '0b10100'

Bitwise Rightshift Operator

In [421…
```
10>>2
```

Out[421… 2

In [422…
```
bin(10)
```

Out[422… '0b1010'

In [423…
```
bin(20)
```

Out[423… '0b10100'

In [424…
```
20>>4
```

Out[424… 1

import math module

In [426…
```
x=sqrt(25)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[426], line 1
----> 1 x=sqrt(25)

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

In [427…
```
import math
```

In [428…
```
x=math.sqrt(25)
x
```

Out[428… 5.0

In [429…
```
x1=math.sqrt(15)
x1
```

Out[429… 3.872983346207417

In [431…
```
print(math.floor(2.9)) #floor--minimum or least value
```

2

In [432…
```
print(math.ceil(2.9)) #ceil -- mac=ximum or highest value
```

```
3
```

In [433…
```
print(math.pow(3,2))
```

```
9.0
```

In [434…
```
print(math.pow(3,4))
```

```
81.0
```

In [435…
```
print(math.pi)
```

```
3.141592653589793
```

In [436…
```
print(math.e)
```

```
2.718281828459045
```

In [437…
```
import math as m
m.sqrt(10)
```

Out[437…
```
3.1622776601683795
```

In [440…
```
from math import sqrt, pow
pow(2,3)
```

Out[440…
```
8.0
```

In [441…
```
round(pow(2,3))
```

Out[441…
```
8
```

Use input function in python || command line input

In [442…
```
x=input()
y=input()
z=x+y
print(z)
```

```
hai,hello
```

In [444…
```
x1=input("Enter the first number")
y1=input("ENter the seocnd number")
z1=x1+y1
print(z1)
```

```
1012
```

In [445…
```
type(x1)
type(y1)
```

Out[445…
```
str
```

In [446…
```
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)
```

```
27
```

In [447…
```
x2 = int(input('Enter the 1st number'))
y2 = int(input('Enter the 2nd number'))
z2 = x2 + y2
z2
```

Out[447… 38

In [449…
```
ch = input('enter a char')
print(ch)
```

```
hello.
```

In [450…
```
print(ch[0])
```

```
h
```

In [451…
```
print(ch[1])
```

```
e
```

In [452…
```
print(ch[-1])
```

```
.
```

In [453…
```
ch = input('enter a char')[0]
print(ch)
```

```
h
```

In [454…
```
ch = input('enter a char')[1:3]
print(ch)
```

```
el
```

In [455…
```
ch = input('enter a char')
print(ch) # if you enter as 2 + 6 -1 we get output as 2 + 6-1 only
```

```
hello venky
```

In [456…
```
result = eval(input('enter an expr'))
print(result)
```

```
17.5
```

In [ ]:
```
```