

```
In [1]: #checking the verion
import sys
sys.version
```

```
Out[1]: '3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 15:03:56) [MSC v.1929 6
4 bit (AMD64)]'
```

```
In [5]: #syntax of the variable is variable=value, if we give value=variable then it shows t
a=3
print(a)
#here a is the variable
```

```
3
```

```
In [3]: #with writing of print statement we can get the statements that we need
a=3
b=4
print(a)
print(b)
```

```
3
```

```
4
```

sum of 2 num

```
In [8]: #sum of two numbers
num1=3
num2=6
sum=num1+num2
print(sum)
```

```
9
```

```
In [10]: #if we want to the sum statements in the output then
num1=3
num2=7
sum=num1+num2
print("the sum of",num1,"and",num2,"is",sum)
```

```
the sum of 3 and 7 is 10
```

```
In [12]: #python is case sensitive
#variable never start with number
#no special character is allowed in variable
#underscore will return the previous value
#python keyword never be a variable
```

integers

```
In [15]: i=4
i
```

```
Out[15]: 4
```

```
In [17]: type(i)
```

```
Out[17]: int
```

```
In [19]: a,b,c=4,5,6  
print(a,b,c)
```

```
4 5 6
```

float

```
In [22]: f=110.20  
f
```

```
Out[22]: 110.2
```

```
In [ ]:
```

```
In [24]: type(f)
```

```
Out[24]: float
```

```
In [26]: f1,f2,f3=2.3,3.4,5.1
```

```
In [ ]:
```

```
In [28]: f1=5
```

```
In [30]: f5=2.4e2 #float data type accepts "e"  
f5
```

```
Out[30]: 240.0
```

boolean

```
In [33]: b=True
```

```
In [35]: b1=False
```

```
In [37]: print(b)  
print(b1)
```

```
True  
False
```

```
In [39]: True+True+True+False-True
```

Out[39]: 2

In [41]: True+False #true is 1 and false 0

Out[41]: 1

In [43]: False*True

Out[43]: 0

In [45]: True*True

Out[45]: 1

In [47]: False/True

Out[47]: 0.0

complex

In [50]: c=10+20j

c

Out[50]: (10+20j)

In [52]: type(c)

Out[52]: complex

In [54]: c.imag

Out[54]: 20.0

In [56]: c.real

Out[56]: 10.0

In [58]: c1=10+20j
c2=30+40j
print(c1+c2)
print(c1-c2)

(40+60j)

(-20-20j)

In [60]: a=3+5j

a

Out[60]: (3+5j)

In [62]: a.imag

```
Out[62]: 5.0
```

```
In [64]: a.real
```

```
Out[64]: 3.0
```

strings

```
In [67]: s='nit'  
s
```

```
Out[67]: 'nit'
```

```
In [69]: type(s)
```

```
Out[69]: str
```

```
In [71]: s1="hello python"  
s1
```

```
Out[71]: 'hello python'
```

```
In [73]: s2='''nit python'''  
s2
```

```
Out[73]: 'nit python'
```

string indexing

```
In [76]: s1[0]
```

```
Out[76]: 'h'
```

```
s1[-4]
```

```
In [78]: s1[4]
```

```
Out[78]: 'o'
```

```
In [80]: s1[-4]
```

```
Out[80]: 't'
```

```
In [82]: s1[-7]
```

```
Out[82]: ''
```

```
In [84]: s
```

```
Out[84]: 'nit'
```

```
In [86]: 'nit'
```

```
Out[86]: 'nit'
```

```
In [88]: 'hello'
```

```
Out[88]: 'hello'
```

string slicing

```
In [91]: s1[:]
```

```
Out[91]: 'hello python'
```

```
In [93]: s1[2:7]
```

```
Out[93]: 'llo p'
```

data slicing

```
In [95]: s3="dataanalytic"
```

```
In [97]: s3[0:10]
```

```
Out[97]: 'dataanalyt'
```

```
In [99]: s3[12] # when we declare the index out of range it shows error
```

```
-----
```

```
IndexError  
Cell In[99], line 1  
----> 1 s3[12]
```

```
Traceback (most recent call last)
```

```
IndexError: string index out of range
```

```
In [101...]: s3[9:12]
```

```
Out[101...]: 'tic'
```

```
In [103...]: s3
```

```
Out[103...]: 'dataanalytic'
```

```
In [105...]: s3[0:11:2]
```

```
Out[105...]: 'dtaayi'
```

```
In [107... s3[2:-2]
```

```
Out[107... 'taanalyt'
```

```
In [109... print(s)
           print(s1)
           print(s2)
           print(s3)
```

```
nit
hello python
nit python
dataanalytic
```

type casting

other data types to int

```
In [111... int(2.3)
```

```
Out[111... 2
```

```
In [113... int(True) #boolean to int
```

```
Out[113... 1
```

```
In [115... int(2+4j)
```

```
-----
TypeError
Cell In[115], line 1
----> 1 int(2+4j)
```

```
Traceback (most recent call last)
```

```
TypeError: int() argument must be a string, a bytes-like object or a real number, no
t 'complex'
```

```
In [117... int("10")
```

```
Out[117... 10
```

```
In [119... int("nit")
```

```
-----
ValueError
Cell In[119], line 1
----> 1 int("nit")
```

```
Traceback (most recent call last)
```

```
ValueError: invalid literal for int() with base 10: 'nit'
```

```
In [121... s1=("hello")
           s2=("nit")
```

```
s3=(s1+s2)  
s3
```

Out[121... 'hellonit'

```
In [123... s=np.nan
```

```
NameError  
Cell In[123], line 1  
----> 1 s=np.nan
```

Traceback (most recent call last)

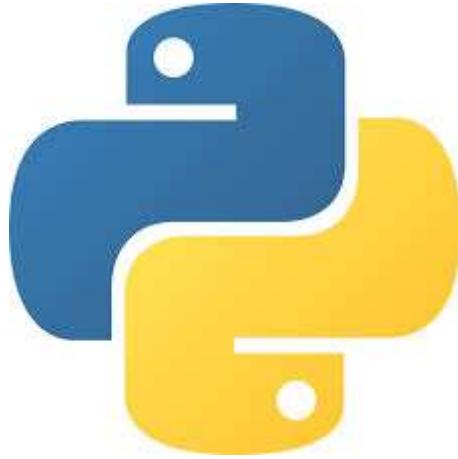
```
NameError: name 'np' is not defined
```

```
In [125... import numpy as np  
a=np.nan
```

```
In [127... type(a)
```

Out[127... float

import the image



keywords in python

```
In [129... import keyword  
keyword.kwlist
```

```
Out[129... ['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']
```

other data type to float

```
In [132... float(3)
```

```
Out[132... 3.0
```

```
In [134... float(True)
```

```
Out[134... 1.0
```

```
In [136... float(1+2j)
```

```
NameError Traceback (most recent call last)
Cell In[136], line 1
----> 1 float(1+2j)

NameError: name 'float' is not defined

In [138... float(3,4) # float accepts only one argument

TypeError Traceback (most recent call last)
Cell In[138], line 1
----> 1 float(3,4)

TypeError: float expected at most 1 argument, got 2

In [140... float("10")

Out[140... 10.0
```

other data type to complex

```
In [142... complex(10)
Out[142... (10+0j)

In [144... complex(10,20)
Out[144... (10+20j)

In [146... complex(2.3,10)
Out[146... (2.3+10j)

In [148... complex(True)
Out[148... (1+0j)

In [150... complex(False)
Out[150... 0j

In [152... complex("10")
Out[152... (10+0j)
```

other data types to bool

```
In [155... bool(1)
```

```
Out[155... True
```

```
In [157... bool(0)
```

```
Out[157... False
```

```
In [159... bool(2.3)
```

```
Out[159... True
```

```
In [161... bool()
```

```
Out[161... False
```

```
In [163... bool( )
```

```
Out[163... False
```

```
In [165... bool("nit")
```

```
Out[165... True
```

```
In [167... bool(10+2j)
```

```
Out[167... True
```

```
In [169... bool(0+0j)
```

```
Out[169... False
```

other datatypes to string

```
In [172... print(str(2))
print(str(2.3))
print(str(True))
print(str(1+2j))
```

```
2
2.3
True
(1+2j)
```

indexing and slicing

```
In [175... index="hellopython"
index
```

```
Out[175... 'hellopython'
```

```
In [177... index[::]
```

```
Out[177... 'hellopython'
```

```
In [179... index[::-1] #reverse the string
```

```
Out[179... 'nohtypolleh'
```

```
In [181... index[::-2]
```

```
Out[181... 'nhyolh'
```

```
In [183... index[::-4]
```

```
Out[183... 'nyl'
```

```
In [185... index[:4]
```

```
Out[185... 'hellopy'
```

```
In [187... index[1:10:3]
```

```
Out[187... 'eot'
```

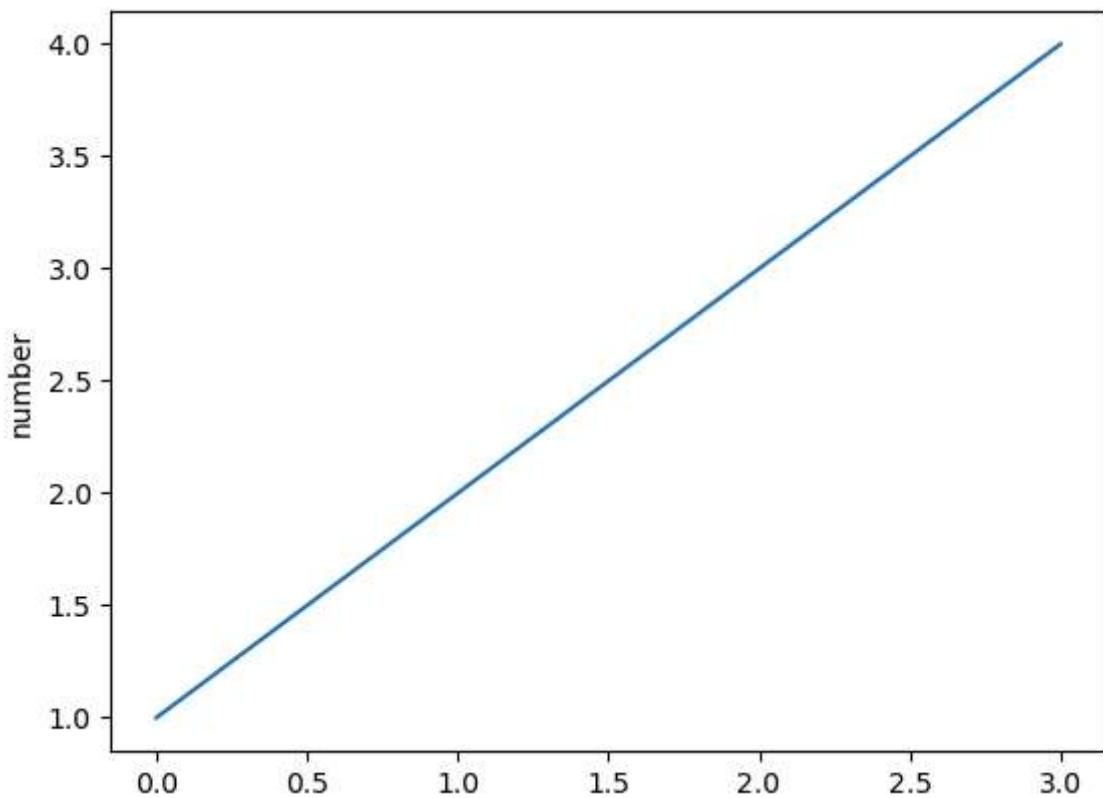
ploting the graph|

```
In [190... import matplotlib.pyplot as plt
```

```
plt.plot([1,2,3,4])
```

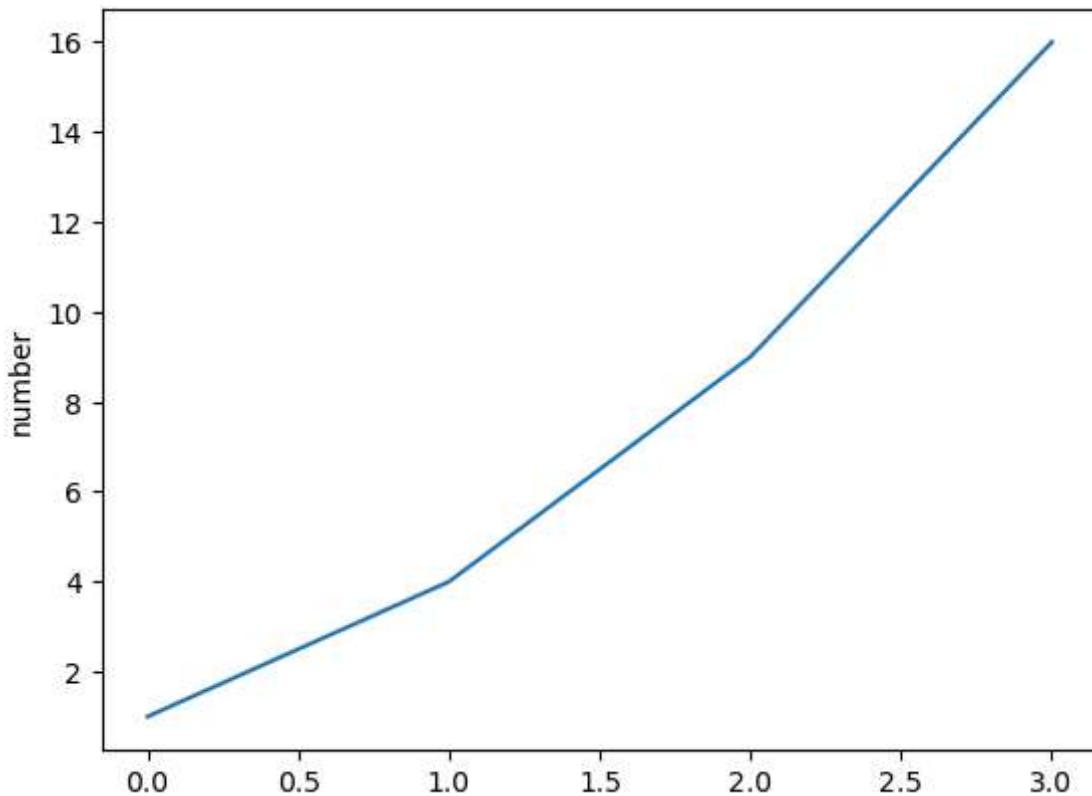
```
plt.ylabel('number')
```

```
plt.show()
```



In [191...]

```
import matplotlib.pyplot as plt
plt.plot([1,4,9,16])
plt.ylabel('number')
plt.show()
```



adding of 2 numbers can be printed in 3 different ways

In [195...]

```
num1=4
num2=5
add=num1+num2
print("addition of ",num1,"and",num2,"is=",add)
print("addition of the {} and {} is={}".format(num1,num2,add))
print(f"addition of the {num1} and {num2} is={add}")
```

addition of 4 and 5 is= 9
 addition of the 4 and 5 is=9
 addition of the 4 and 5 is=9

end statement

-if we want to attach the code in second line at the end of the first line there we use this statement

In [199...]

```
print("hello", end=" ")
print("how r u")
```

hello how r u

separator

- when we want to separate the terms we use it
- we can use special characters to separate the terms)

In [203...]

```
print("hii","hello","how r u",sep=" @ ")
```

hii @ hello @ how r u

In [205...]

```
print("hii","hello","how r u",sep=" $ ")
```

hii \$ hello \$ how r u

average of the numbers

In [208...]

```
num1=3
num2=4
average=num1+num2/2
average
```

Out[208...]

5.0

```
In [210...]: num1=3
num2=4
num3=5
average=(num1+num2+num3/3)
average1=(num1+num2+num3/3,2)
print( " the average of {},{}and{} is={}" .format(num1,num2,num3,average or average1))

the average of 3,4and5 is=8.666666666666666
```

string functions

```
In [213...]: s="sushmaramchander"
s.upper()
```

```
Out[213...]: 'SUSHMARAMCHANDER'
```

```
In [215...]: s="Sushmaramchander"
s.capitalize()
```

```
Out[215...]: 'Sushmaramchander'
```

```
In [217...]: print(s.lower())
print(s.title())
print(s.swapcase())
```

```
sushmaramchander
Sushmaramchander
sUSHMARAMCHANDER
```

```
In [1]: s1="hello"
```

data structures

- list
- tuple
- dict
- set

list

-denote with l -doesn't allow

```
In [223...]: l=[]
l
```

```
Out[223...]: []
```

```
In [225... type(1)
```

```
Out[225... list
```

append function in the list

```
In [228... l.append(10)
```

```
l
```

```
Out[228... [10]
```

```
In [230... l.append(20)
```

```
l.append(30)
```

```
l
```

```
Out[230... [10, 20, 30]
```

```
In [232... l.append(2.3)
```

```
l
```

```
Out[232... [10, 20, 30, 2.3]
```

```
In [234... l.append(1+2j)
```

```
l.append("nit")
```

```
l.append(True)
```

```
l
```

```
Out[234... [10, 20, 30, 2.3, (1+2j), 'nit', True]
```

length

```
In [237... len(l)
```

```
Out[237... 7
```

copy function

```
In [240... l1=l.copy()
```

```
l1
```

```
Out[240... [10, 20, 30, 2.3, (1+2j), 'nit', True]
```

```
In [242... l==l1
```

```
Out[242... True
```

count

```
In [245...]: l.count(20)
```

```
Out[245...]: 1
```

```
In [247...]: l
```

```
Out[247...]: [10, 20, 30, 2.3, (1+2j), 'nit', True]
```

```
In [249...]: l[:]
```

```
Out[249...]: [10, 20, 30, 2.3, (1+2j), 'nit', True]
```

```
In [251...]: l[4]
```

```
Out[251...]: (1+2j)
```

```
In [254...]: 12=[ ]
```

```
12
```

```
Out[254...]: []
```

```
In [256...]: 12.append(1)
12.append(2.3)
12.append(True)
12.append(1+2j)
12.append("nit")
12
```

```
Out[256...]: [1, 2.3, True, (1+2j), 'nit']
```

```
In [258...]: len(12)
```

```
Out[258...]: 5
```

clear

```
In [261...]: 12.clear
```

```
12
```

```
Out[261...]: [1, 2.3, True, (1+2j), 'nit']
```

```
In [263...]: len(12)
```

```
Out[263...]: 5
```

```
In [265...]: 12.append(1)
             12.append(2.3)
             12.append(True)
             12.append(1+2j)
             12.append("nit")
             12
```

```
Out[265...]: [1, 2.3, True, (1+2j), 'nit', 1, 2.3, True, (1+2j), 'nit']
```

remove

```
In [268...]: 12.remove(2.3)
             12
```

```
Out[268...]: [1, True, (1+2j), 'nit', 1, 2.3, True, (1+2j), 'nit']
```

```
In [270...]: 13=[]
             13.append(10)
             13
```

```
Out[270...]: [10]
```

```
In [272...]: 12
```

```
Out[272...]: [1, True, (1+2j), 'nit', 1, 2.3, True, (1+2j), 'nit']
```

extend

```
In [275...]: 13.extend(12)
             13
```

```
Out[275...]: [10, 1, True, (1+2j), 'nit', 1, 2.3, True, (1+2j), 'nit']
```

index

```
In [278...]: 13.index(1+2j)
```

```
Out[278...]: 3
```

```
In [280...]: 12.index("nit")
```

```
Out[280...]: 3
```

```
In [282...]: 13
```

```
Out[282...]: [10, 1, True, (1+2j), 'nit', 1, 2.3, True, (1+2j), 'nit']
```

```
In [284... 12
```

```
Out[284... [1, True, (1+2j), 'nit', 1, 2.3, True, (1+2j), 'nit']
```

insert

```
In [287... 12.insert(4,"technology")
12
```

```
Out[287... [1, True, (1+2j), 'nit', 'technology', 1, 2.3, True, (1+2j), 'nit']
```

```
In [289... 12.insert(3,False)
12
```

```
Out[289... [1, True, (1+2j), False, 'nit', 'technology', 1, 2.3, True, (1+2j), 'nit']
```

```
In [291... 13
```

```
Out[291... [10, 1, True, (1+2j), 'nit', 1, 2.3, True, (1+2j), 'nit']
```

pop

```
In [294... 13.pop()
```

```
Out[294... 'nit'
```

```
In [296... 13.pop(4)
```

```
Out[296... 'nit'
```

```
In [298... 13.pop(1)
```

```
Out[298... 1
```

```
In [300... 14=[10,100,3,45,76,24]
14
```

```
Out[300... [10, 100, 3, 45, 76, 24]
```

sort

```
In [303... 14.sort()
```

```
In [305... 14
```

```
Out[305... [3, 10, 24, 45, 76, 100]
```

```
In [307... 14.sort(reverse=True)
14
```

```
Out[307... [100, 76, 45, 24, 10, 3]
```

```
In [309... 15=["z","m","c","w"]
15
```

```
Out[309... ['z', 'm', 'c', 'w']
```

```
In [311... 15.sort()
15
```

```
Out[311... ['c', 'm', 'w', 'z']
```

```
In [313... 13
```

```
Out[313... [10, True, (1+2j), 1, 2.3, True, (1+2j)]
```

reverse

```
In [316... 13.reverse()
13
```

```
Out[316... [(1+2j), True, 2.3, 1, (1+2j), True, 10]
```

```
In [318... 12[3]
```

```
Out[318... False
```

```
In [320... 12=["hii",4,False]
```

```
In [322... 12
```

```
Out[322... ['hii', 4, False]
```

```
In [324... 14=["nit",False,2] # mutable:-we can modify or change ,remove the elements in the list
14
```

```
Out[324... ['nit', False, 2]
```

```
In [326... 14[1]=0
```

```
In [328... 14
```

```
Out[328... ['nit', 0, 2]
```

```
In [330... for i in 14:
    print(i)
```

nit

0

2

```
In [332...]: 16=["sbi","icic"]
17=["hdfs","kotak"]
family_bank=16+17
family_bank
```

Out[332...]: ['sbi', 'icic', 'hdfs', 'kotak']

```
In [334...]: for i in enumerate (14):
    print(i)
```

```
(0, 'nit')
(1, 0)
(2, 2)
```

tuple

- assigned vth "()"
- immutable

```
In [3]: t=()
t
```

Out[3]: ()

```
In [340...]: type(t)
```

Out[340...]: tuple

```
In [342...]: t1=tuple()
type(t1)
```

Out[342...]: tuple

```
In [5]: t=(10,10,20,30)
t
```

Out[5]: (10, 10, 20, 30)

```
In [7]: t1=(True,1.2,"nit",3,1+2j)
t1
```

Out[7]: (True, 1.2, 'nit', 3, (1+2j))

count

```
In [9]: t1.count(10)
```

```
Out[9]: 0
```

```
In [11]: t1.index("nit")
```

```
Out[11]: 2
```

```
In [15]: t1.count(True)
```

```
Out[15]: 1
```

for loop

```
In [354... for i in t1:  
      print(i)
```

```
True  
1.2  
nit  
3  
(1+2j)
```

```
In [356... t[:]
```

```
Out[356... (10, 10, 20, 30)
```

```
In [358... t4=t*4  
t4
```

```
Out[358... (10, 10, 20, 30, 10, 10, 20, 30, 10, 10, 20, 30, 10, 10, 20, 30)
```

set

- set and dict both are represented with "{}"
- if we need to create the set then we need to use "set()". so system take it as set otherwise by default it consider it as dict
- in set indexing and slicing is not acceptable

```
In [362... s={}  
s
```

```
Out[362... {}
```

```
In [364... type(s)
```

```
Out[364... dict
```

```
In [366... s1=set()
      type(s1)
```

Out[366... set

```
In [368... s1={100,20,3,15,47} # if same data type is used in the set then it automatically arr
      s1
```

Out[368... {3, 15, 20, 47, 100}

```
In [370... s2={2.3,4.5,1.3}
      s2
```

Out[370... {1.3, 2.3, 4.5}

```
In [372... s3={"z","m","a","x"}
      s3
```

Out[372... {'a', 'm', 'x', 'z'}

```
In [374... s4=[10,2.3,"a",5,True,6,7] #when set contains mixed data types in it then the arran
      s4
```

Out[374... {10, 2.3, 5, 6, 7, True, 'a'}

```
In [376... print(s1)
      print(s2)
      print(s3)
      print(s4)
```

{3, 100, 20, 47, 15}
 {1.3, 2.3, 4.5}
 {'a', 'x', 'm', 'z'}
 {True, 2.3, 5, 6, 7, 'a', 10}

```
In [378... for i in s1:
      print(i)
```

3
 100
 20
 47
 15

add function in set

```
In [381... s4.add(10)
      s4.add(20)
      s4.add(2.3)
```

In [383... s4

```
Out[383... {10, 2.3, 20, 5, 6, 7, True, 'a'}
```

```
In [385... s1.add(4)  
s1
```

```
Out[385... {3, 4, 15, 20, 47, 100}
```

```
In [387... len(s4)
```

```
Out[387... 8
```

clear

```
In [390... s4.clear()
```

```
In [392... s4
```

```
Out[392... set()
```

```
In [394... len(s4)
```

```
Out[394... 0
```

```
In [396... del s4
```

```
In [398... s4
```

NameError
Cell In[398], line 1
----> 1 s4

Traceback (most recent call last)

NameError: name 's4' is not defined

copy

```
In [401... s4=s1.copy()  
s4
```

```
Out[401... {3, 4, 15, 20, 47, 100}
```

```
In [403... s1==s4
```

```
Out[403... True
```

remove

```
In [406... s1.remove(100)
      s1
```

```
Out[406... {3, 4, 15, 20, 47}
```

pop

```
In [409... s1.pop() # in set pop remove a random irrespective of order
```

```
Out[409... 3
```

```
In [411... s2.pop()
```

```
Out[411... 1.3
```

```
In [413... s3
```

```
Out[413... {'a', 'm', 'x', 'z'}
```

```
In [415... "a" in s3
```

```
Out[415... True
```

set operation

```
In [418... A={1,2,3,4,5}
      B={4,5,6,7,8}
      C={8,9,10}
```

union

```
In [421... A.union(B)
```

```
Out[421... {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [423... d_union=A.union(B)
      d_union
```

```
Out[423... {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [425... print(A)
      print(B)
      print(C)
      print(d_union)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
{8, 9, 10}
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [427...]: `B.union(A,c)`

Out[427...]: `{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`

In [429...]: `A|B`

Out[429...]: `{1, 2, 3, 4, 5, 6, 7, 8}`

update

In [432...]: `A.update(B)`

In [434...]: `print(A)
print(B)
print(c)
print(d_union)`

```
{1, 2, 3, 4, 5, 6, 7, 8}
{4, 5, 6, 7, 8}
{8, 9, 10}
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [436...]: `A1={1,2,3,4,5}
B1={4,5,6,7,8}
c1={8,9,10}`

intersection

In [439...]: `A1.intersection(B1)`

Out[439...]: `{4, 5}`

In [441...]: `A2={1,2,3,4,5}
B2={4,5,6,7,8}
c12={8,9,10}`

difference

In [444...]: `A2-B2`

Out[444...]: `{1, 2, 3}`

In [446...]: `B2-A2`

```
Out[446... {6, 7, 8}
```

```
In [448... B2.difference(A2)
```

```
Out[448... {6, 7, 8}
```

symmetric difference

```
In [451... B2.symmetric_difference(A2)
```

```
Out[451... {1, 2, 3, 6, 7, 8}
```

```
In [453... s={2,3,4,5}
```

```
In [455... for i in s:  
    print(i)
```

```
2  
3  
4  
5
```

```
In [457... for i in enumerate(s):  
    print(i)
```

```
(0, 2)  
(1, 3)  
(2, 4)  
(3, 5)
```

```
In [459... a5={1,2,3,4,5,6,7,8,9}  
b5={3,4,5,6,7,8}  
c5={10,20,30,40}
```

```
In [461... a5.issuperset(b5)
```

```
Out[461... True
```

```
In [463... a5.issubset(b5)
```

```
Out[463... False
```

```
In [465... b5.issubset(a5)
```

```
Out[465... True
```

```
In [467... c5.isdisjoint(a5)
```

```
Out[467... True
```

```
In [469... c5.issubset(a5)
```

```
Out[469... False
```

```
In [471... a5
```

```
Out[471... {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
In [473... sum(a5)
```

TypeError

Cell In[473], line 1
----> 1 sum(a5)

Traceback (most recent call last)

TypeError: 'int' object is not callable

```
In [475... max(a5)
```

```
Out[475... 9
```

```
In [477... min(a5)
```

```
Out[477... 1
```

```
In [479... len(a5)
```

```
Out[479... 9
```

dict

- it is represented in key value pairs
- defined in {}
- keys can't be duplicated
- values can be duplicated

```
In [483... d={}
type(d)
```

```
Out[483... dict
```

```
In [485... d={1:"one",2:"two",3:"three",4:"four",5:"five"}
d
```

```
Out[485... {1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
```

```
In [487... d1={"one":1,"two":2,"three":3,"four":4,"five":5}
d1
```

```
Out[487... {'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
```

```
In [489... print(len(d))
      print(len(d1))
```

5

5

```
In [491... d[1]      # index and slicing is not accepted
```

```
Out[491... 'one'
```

```
In [493... d.keys()
```

```
Out[493... dict_keys([1, 2, 3, 4, 5])
```

```
In [495... d.values()
```

```
Out[495... dict_values(['one', 'two', 'three', 'four', 'five'])
```

```
In [497... print(d.keys())
      print(d.values())
```

dict_keys([1, 2, 3, 4, 5])

dict_values(['one', 'two', 'three', 'four', 'five'])

```
In [499... d2={1:2,"2.3":4.8,"nit":"nit",True:False,1+2j:4+5j}
      d2
```

```
Out[499... {1: False, '2.3': 4.8, 'nit': 'nit', (1+2j): (4+5j)}
```

```
In [501... d3={10:"ten",9.0:"nine"}
      d3
```

```
Out[501... {10: 'ten', 9.0: 'nine'}
```

```
In [503... d2.items() #it gives the keys and values
```

```
Out[503... dict_items([(1, False), ('2.3', 4.8), ('nit', 'nit'), ((1+2j), (4+5j))])
```

```
In [505... id(d2)
```

```
Out[505... 2256229796608
```

```
In [507... d2.pop(1)
```

```
Out[507... False
```

```
In [509... d2
```

```
Out[509... {'2.3': 4.8, 'nit': 'nit', (1+2j): (4+5j)}
```

```
In [511... d2.popitem() #random item will be deleted
```

```
Out[511... ((1+2j), (4+5j))
```

```
In [513... d
```

```
Out[513... {1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
```

```
In [515... for i in d:  
      print(i)
```

```
1  
2  
3  
4  
5
```

```
In [517... for i in d:  
      print(i,":",d[i])
```

```
1 : one  
2 : two  
3 : three  
4 : four  
5 : five
```

dict membership

```
In [520... d5={}
```

```
In [522... type(d5)
```

```
Out[522... dict
```

```
In [524... d5={2:"hii",3:"hello",4:four}
```

```
NameError                                                 Traceback (most recent call last)  
Cell In[524], line 1  
----> 1 d5={2:"hii",3:"hello",4:four}  
  
NameError: name 'four' is not defined
```

```
In [526... "hii" in d5
```

```
Out[526... False
```

```
In [528... range(5)
```

```
Out[528... range(0, 5)
```

```
In [530... r=range(5,10)
```

```
In [532... for i in r:  
      print(i)
```

```
5  
6  
7  
8  
9
```

```
In [534]: r1=range(0,11,2)  
r1
```

```
Out[534]: range(0, 11, 2)
```

```
In [536]: for i in r1:  
    print(i)
```

```
0  
2  
4  
6  
8  
10
```

```
In [538]: list(r1)
```

```
Out[538]: [0, 2, 4, 6, 8, 10]
```

```
In [1]: "nit"
```

```
Out[1]: 'nit'
```

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

```
Out[5]: 'nitnitnitnitnit'
```

```
In [7]: print('c:\nit')
```

```
c:  
it
```

```
In [9]: print( r'c:\nit')
```

```
c:\nit
```

```
In [11]: x=5
```

```
In [19]: name="nit"  
name
```

```
Out[19]: 'nit'
```

```
In [23]: name+"technology"
```

```
Out[23]: 'nittechnology'
```

```
In [27]: len(name)
```

```
Out[27]: 3
```

```
In [31]: name1="fine"  
name1
```

```
Out[31]: 'fine'
```

```
In [37]: name1[0:1]
```

```
Out[37]: 'f'
```

```
In [41]: name1[1:]
```

```
Out[41]: 'ine'
```

```
In [45]: "d"+name1[1:]
```

```
Out[45]: 'dine'
```

- arthimatic operator
- assignment operator
- logical operator

```
In [48]: x1=3  
y1=4
```

```
In [50]: x1+y1
```

```
Out[50]: 7
```

```
In [52]: x1-y1
```

```
Out[52]: -1
```

```
In [54]: x1/y1
```

```
Out[54]: 0.75
```

```
In [56]: x1//y1
```

```
Out[56]: 0
```

```
In [58]: x1|y1
```

```
Out[58]: 7
```

```
In [60]: x1**5
```

```
Out[60]: 243
```

```
In [62]: x1*y1
```

```
Out[62]: 12
```

assignment

```
In [65]: x=2  
x=x+2
```

```
In [67]: x
```

```
Out[67]: 4
```

```
In [69]: x+=2
```

```
In [71]: x
```

```
Out[71]: 6
```

```
In [75]: x*=2  
x
```

```
Out[75]: 24
```

```
In [77]: x-=2
```

```
In [79]: x
```

```
Out[79]: 22
```

```
In [81]: x/=2  
x
```

```
Out[81]: 11.0
```

unary

```
In [84]: n=7  
n
```

```
Out[84]: 7
```

```
In [86]: m=- (n)  
m
```

```
Out[86]: -7
```

relational

```
In [91]: a=5  
b=6
```

```
In [98]: a==b
```

```
Out[98]: True
```

```
In [100...]: a>b
```

```
Out[100...]: False
```

```
In [104...]: b>a
```

```
Out[104...]: False
```

```
In [106...]: b<a
```

```
Out[106...]: False
```

```
In [108...]: a<=b
```

```
Out[108...]: True
```

logical

```
In [111...]: a=5  
b=4
```

```
In [113...]: a<3 and b<3
```

```
Out[113...]: False
```

```
In [115...]: a>4 or b<4
```

```
Out[115...]: True
```

number system

```
In [117...]: 25
```

```
Out[117...]: 25
```

```
In [119...]: bin(25)
```

```
Out[119... '0b11001'
```

```
In [125... int(0b11001)
```

```
Out[125... 25
```

```
In [127... bin(7)
```

```
Out[127... '0b111'
```

```
In [129... oct(25)
```

```
Out[129... '0o31'
```

```
In [131... hex(25)
```

```
Out[131... '0x19'
```

```
In [133... 0xa
```

```
Out[133... 10
```

```
In [137... 0xb
```

```
Out[137... 11
```

```
In [ ]:
```

```
In [139... a=5  
b=6
```

```
In [143... a==b  
b==a  
print(a)  
print(b)
```

```
5
```

```
6
```

```
In [147... temp=a  
a=b  
b=temp
```

```
In [149... print(a)  
print(b)
```

```
6
```

```
6
```

```
In [151... a,b=b,a
```

```
In [153... print(a)  
print(b)
```

```
6  
6
```

bitwise

- complement ~
-

```
In [2]: print(bin(12))  
print(bin(13))
```

```
0b1100  
0b1101
```

```
In [4]: ~12
```

```
Out[4]: -13
```

```
In [6]: ~46
```

```
Out[6]: -47
```

and bitwise operator

```
In [11]: 12&13
```

```
Out[11]: 12
```

```
In [18]: 35&40
```

```
Out[18]: 32
```

```
In [20]: 35|40
```

```
Out[20]: 43
```

```
In [15]: 1&0
```

```
Out[15]: 0
```

xor

```
In [22]: 12^13
```

```
Out[22]: 1
```

```
In [24]: 25^30
```

Out[24]: 7

left shift and Right shift

In [27]: bin(10)

Out[27]: '0b1010'

In [29]: 10<<2

Out[29]: 40

In [31]: 10>>2

Out[31]: 2

In [33]: 10>>1

Out[33]: 5

In [35]: bin(2)

Out[35]: '0b10'

In [37]: 20>>2

Out[37]: 5

In [1]: import math
help(math)

Help on built-in module math:

NAME

math

DESCRIPTION

This module provides access to the mathematical functions defined by the C standard.

FUNCTIONS

acos(x, /)

Return the arc cosine (measured in radians) of x.

The result is between 0 and pi.

acosh(x, /)

Return the inverse hyperbolic cosine of x.

asin(x, /)

Return the arc sine (measured in radians) of x.

The result is between -pi/2 and pi/2.

asinh(x, /)

Return the inverse hyperbolic sine of x.

atan(x, /)

Return the arc tangent (measured in radians) of x.

The result is between -pi/2 and pi/2.

atan2(y, x, /)

Return the arc tangent (measured in radians) of y/x.

Unlike atan(y/x), the signs of both x and y are considered.

atanh(x, /)

Return the inverse hyperbolic tangent of x.

cbrt(x, /)

Return the cube root of x.

ceil(x, /)

Return the ceiling of x as an Integral.

This is the smallest integer $\geq x$.

comb(n, k, /)

Number of ways to choose k items from n items without repetition and without order.

Evaluates to $n! / (k! * (n - k)!)$ when $k \leq n$ and evaluates to zero when $k > n$.

Also called the binomial coefficient because it is equivalent to the coefficient of k-th term in polynomial expansion of the

expression $(1 + x)^{**n}$.

Raises `TypeError` if either of the arguments are not integers.
Raises `ValueError` if either of the arguments are negative.

`copysign(x, y, /)`
Return a float with the magnitude (absolute value) of x but the sign of y .

On platforms that support signed zeros, `copysign(1.0, -0.0)` returns -1.0 .

`cos(x, /)`
Return the cosine of x (measured in radians).

`cosh(x, /)`
Return the hyperbolic cosine of x .

`degrees(x, /)`
Convert angle x from radians to degrees.

`dist(p, q, /)`
Return the Euclidean distance between two points p and q .

The points should be specified as sequences (or iterables) of coordinates. Both inputs must have the same dimension.

Roughly equivalent to:
`sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))`

`erf(x, /)`
Error function at x .

`erfc(x, /)`
Complementary error function at x .

`exp(x, /)`
Return e raised to the power of x .

`exp2(x, /)`
Return 2 raised to the power of x .

`expm1(x, /)`
Return $\exp(x)-1$.

This function avoids the loss of precision involved in the direct evaluation of $\exp(x)-1$ for small x .

`fabs(x, /)`
Return the absolute value of the float x .

`factorial(n, /)`
Find $n!$.

Raise a `ValueError` if x is negative or non-integral.

`floor(x, /)`

Return the floor of x as an Integral.

This is the largest integer $\leq x$.

`fmod(x, y, /)`
Return $fmod(x, y)$, according to platform C.

$x \% y$ may differ.

`frexp(x, /)`
Return the mantissa and exponent of x, as pair (m, e).

m is a float and e is an int, such that $x = m * 2.^{e}$.
If x is 0, m and e are both 0. Else $0.5 \leq \text{abs}(m) < 1.0$.

`fsum(seq, /)`
Return an accurate floating point sum of values in the iterable seq.

Assumes IEEE-754 floating point arithmetic.

`gamma(x, /)`
Gamma function at x.

`gcd(*integers)`
Greatest Common Divisor.

`hypot(...)`
`hypot(*coordinates) -> value`

Multidimensional Euclidean distance from the origin to a point.

Roughly equivalent to:
`sqrt(sum(x**2 for x in coordinates))`

For a two dimensional point (x, y), gives the hypotenuse using the Pythagorean theorem: $\sqrt{x^2 + y^2}$.

For example, the hypotenuse of a 3/4/5 right triangle is:

```
>>> hypot(3.0, 4.0)
5.0
```

`isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)`
Determine whether two floating point numbers are close in value.

`rel_tol`
maximum difference for being considered "close", relative to the magnitude of the input values

`abs_tol`
maximum difference for being considered "close", regardless of the magnitude of the input values

Return True if a is close in value to b, and False otherwise.

For the values to be considered close, the difference between them must be smaller than at least one of the tolerances.

-inf, inf and NaN behave similarly to the IEEE 754 Standard. That is, NaN is not close to anything, even itself. inf and -inf are only close to themselves.

`isfinite(x, /)`
Return True if x is neither an infinity nor a NaN, and False otherwise.

`isinf(x, /)`
Return True if x is a positive or negative infinity, and False otherwise.

`isnan(x, /)`
Return True if x is a NaN (not a number), and False otherwise.

`isqrt(n, /)`
Return the integer part of the square root of the input.

`lcm(*integers)`
Least Common Multiple.

`ldexp(x, i, /)`
Return $x * (2^{**i})$.

This is essentially the inverse of frexp().

`lgamma(x, /)`
Natural logarithm of absolute value of Gamma function at x.

`log(...)`
`log(x, [base=math.e])`
Return the logarithm of x to the given base.

If the base is not specified, returns the natural logarithm (base e) of x.

`log10(x, /)`
Return the base 10 logarithm of x.

`log1p(x, /)`
Return the natural logarithm of 1+x (base e).

The result is computed in a way which is accurate for x near zero.

`log2(x, /)`
Return the base 2 logarithm of x.

`modf(x, /)`
Return the fractional and integer parts of x.

Both results carry the sign of x and are floats.

`nextafter(x, y, /, *, steps=None)`
Return the floating-point value the given number of steps after x towards y.

If steps is not specified or is None, it defaults to 1.

Raises a TypeError, if x or y is not a double, or if steps is not an integer

r.

Raises ValueError if steps is negative.

perm(n, k=None, /)

Number of ways to choose k items from n items without repetition and with or der.

Evaluates to $n! / (n - k)!$ when $k \leq n$ and evaluates to zero when $k > n$.

If k is not specified or is None, then k defaults to n and the function returns $n!$.

Raises TypeError if either of the arguments are not integers.

Raises ValueError if either of the arguments are negative.

pow(x, y, /)

Return $x^{**}y$ (x to the power of y).

prod(iterable, /, *, start=1)

Calculate the product of all the elements in the input iterable.

The default start value for the product is 1.

When the iterable is empty, return the start value. This function is intended specifically for use with numeric values and may reject non-numeric types.

radians(x, /)

Convert angle x from degrees to radians.

remainder(x, y, /)

Difference between x and the closest integer multiple of y.

Return $x - n*y$ where $n*y$ is the closest integer multiple of y.

In the case where x is exactly halfway between two multiples of y, the nearest even value of n is used. The result is always exact.

sin(x, /)

Return the sine of x (measured in radians).

sinh(x, /)

Return the hyperbolic sine of x.

sqrt(x, /)

Return the square root of x.

sumprod(p, q, /)

Return the sum of products of values from two iterables p and q.

Roughly equivalent to:

```
sum(itertools.starmap(operator.mul, zip(p, q, strict=True)))
```

For float and mixed int/float inputs, the intermediate products and sums are computed with extended precision.

```
tan(x, /)
    Return the tangent of x (measured in radians).

tanh(x, /)
    Return the hyperbolic tangent of x.

trunc(x, /)
    Truncates the Real x to the nearest Integral toward 0.

    Uses the __trunc__ magic method.

ulp(x, /)
    Return the value of the least significant bit of the float x.
```

DATA

```
e = 2.718281828459045
inf = inf
nan = nan
pi = 3.141592653589793
tau = 6.283185307179586
```

FILE

```
(built-in)
```

math module

```
In [3]: x=math.sqrt(25)
```

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

```
Out[5]: 5.0
```

```
In [7]: print(math.floor(2.3))
```

```
2
```

```
In [9]: print(math.ceil(2.3))
```

```
3
```

```
In [11]: print(math.pow(3,2))
```

```
9.0
```

```
In [13]: print(math.pi)
```

```
3.141592653589793
```

```
In [15]: print(math.e)
```

```
2.718281828459045
```

```
In [19]: math.sqrt(25)
```

```
Out[19]: 5.0
```

```
In [ ]:
```