Numbers

the operators +, -, * and / can be used to perform arithmetic; parentheses (()) can be used for grouping.

```
2+2
```

```
4
```

```
2*2+4
```

```
8
```

```
(150 - 60) /4
```

```
22.5
```

Division (/) always returns a float.

To do floor division and get an integer result you can use the // operator.

the remainder you can use %

division(/) vs floor division(//)

```
17/3  # division will return as float
```

```
5.666666666666667
```

```
17//3 # floor division will give the integer part and discards the decimal part
```

```
5
```

** operator to calculate the power

```
5**2 # 5 squared
```

```
25
```

Test

Python text (represented by type str, so-called “strings”

```
'spam eggs' #  single quotes
```

```
'spam eggs'
```

```
"spam egg" #  double quotes
```

```
'spam egg'
```

print() funtion

```
s = 'First line.\nSecond line.'  # \n means newline
print(s)
```

```
First line.
Second line.
```

String concatination with (+) operator and repeated with (*)

```
3* 'p' + 'ython'
```

```
'pppython'
```

```
'py' +'thon'
```

```
'python'
```

particularly useful when you want to break long strings

```
text = ('Put several strings within parentheses '
        'to have them joined together.')
text
```

```
'Put several strings within parentheses to have them joined together.'
```

string index (positive and negative indexing)

```
word ='python'
word[0]
```

```
'p'
```

```
word[1]

'y'

word[-1]

'n'

word[-6]

'p'
```

string slicing (forward and backward)

```
word[0:2]

'py'

word[-6:6]

'python'

word[0:2] + word[2:6]

'python'
```

python string is immutable

we cant change the strings after assigning

```
#word[0] ='j'  # TypeError: 'str' object does not support item
assignment
```

len() ------> The built-in function len() returns the length of a string:

```
len(word)

6
```

Data Structure

LIST

- List is a mutable

- list allows duplicates
- list allows indexing and slicing

```
list=[]  # empty list
list
```

```
[]
```

predefined functions in list

- append() --add the element at end of the list
- insert() --insert the value based on given index position .eg:insert(index,value)
- copy() --copy ome to another list (shallow copy)
- index() --return the value of first occurance of given index
- clear() --remove all elements from th elist
- count() --count the values based on frequence
- reverse() --reverse the elements of the list
- sort() --sort the elements (default accending order)
- pop() --first return the value and remove the default last element of the list
- pop(index) --remove the element based on given index [gives index error]
- remove() --remove the elements on given value [valueerror]
- extend() --merging the one list to another list

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple',
'banana']
fruits
```

```
['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
```

```
# append()
fruits.append('apple')
print(fruits)
```

```
['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana',
'apple']
```

```
# insert()
fruits.insert(2,'mango')
print(fruits)
```

```
['orange', 'apple', 'mango', 'pear', 'banana', 'kiwi', 'apple',
'banana', 'apple']
```

```
# copy()
fruits1=fruits.copy() # shallow copy
print(fruits1)
```

```
['orange', 'apple', 'mango', 'pear', 'banana', 'kiwi', 'apple',
'banana', 'apple']
```

```
fruits=fruits1 # deep copy
print(fruits)
```

```
['orange', 'apple', 'mango', 'pear', 'banana', 'kiwi', 'apple',
'banana', 'apple']
```

```
# index()
fruits.index('mango')
```

```
2
```

```
# count()
fruits.count('mango')
```

```
1
```

```
fruits.count('apple')
```

```
3
```

```
# reverse()
fruits.reverse()
print(fruits)
```

```
['apple', 'banana', 'apple', 'kiwi', 'banana', 'pear', 'mango',
'apple', 'orange']
```

```
# sort()
fruits.sort()  # default accending order
print(fruits)
```

```
['apple', 'apple', 'apple', 'banana', 'banana', 'kiwi', 'mango',
'orange', 'pear']
```

```
# decending order
fruits.sort(reverse=True)
print(fruits)
```

```
['pear', 'orange', 'mango', 'kiwi', 'banana', 'banana', 'apple',
'apple', 'apple']
```

```
# accending order
fruits.sort(reverse=False)
print(fruits)
```

```
['apple', 'apple', 'apple', 'banana', 'banana', 'kiwi', 'mango',
'orange', 'pear']
```

```
# pop()
fruits.pop()
```

```
'pear'
```

```
print(fruits)

['apple', 'apple', 'apple', 'banana', 'banana', 'kiwi', 'mango',
'orange']

fruits.pop(3)
print(fruits)

['apple', 'apple', 'apple', 'banana', 'kiwi', 'mango', 'orange']

# remove()
fruits.remove('kiwi')
print(fruits)

['apple', 'apple', 'apple', 'banana', 'mango', 'orange']

# extend()

fruits1=[1,3,4,5,6]
fruits1

[1, 3, 4, 5, 6]

fruits.extend(fruits1)
print(fruits)

['apple', 'apple', 'apple', 'banana', 'mango', 'orange', 1, 3, 4, 5,
6]

# clear()
fruits1.clear()

print(fruits1)

[]
```

del statement

del used to delete entire variable memory space

```
del fruits1

#fruits1 # NameError: name 'fruits1' is not defined
```

Tuple

- tuple allows the duplicates
- tuple allows indexing and slicing
- tuple is immutable

```
t=()  # empty tuple
type(t)
```

```
tuple
```

assigning the values to tuple in different ways

```
t=1,2,3,4,5,6,7,2,5,6,7,7
type(t)
```

```
tuple
```

```
t1=([1,2,3],[4,5,6])
type(t1)
```

```
tuple
```

```
t2=((1,2,3,4,'hello',2.4))
type(t2)
```

```
tuple
```

Pre defined functions in tuple

- index() --return the value of first occurance of given index
- count() --count the values based on frequence

```
t
```

```
(1, 2, 3, 4, 5, 6, 7, 2, 5, 6, 7, 7)
```

```
t.index(4)
```

```
3
```

```
t.count(7)
```

```
3
```

set()

1) Unordered & Unindexed collection of items. 2) Set elements are unique. Duplicate elements are not allowed. 3) Set elements are immutable (cannot be changed). 4) Set itself is mutable. We can add or remove items from it.

```
s1=set()  #empty set
s1
```

```
set()
```

```
myset={1,2,3,4,5} #  set of numbers
myset
```

```
{1, 2, 3, 4, 5}
```

```
len(myset)
```

```
5
```

```
my_set ={1,2,3,4,5,6,7,7,8} # duplicates are not allowed
my_set
```

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

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

```
1
2
3
4
5
6
7
8
```

```
for i in enumerate(my_set):
    print(i)
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 5)
(5, 6)
(6, 7)
(7, 8)
```

set predefined functions

- add() --Adds a given element to a set randomly
- clear() -- Removes all elements from the set.
- copy() --copy one set to another set
- remove() --Removes the specified element from the set(return error)
- discard() --emoves the specified element from the set (return no error)
- pop() --Removes and returns a random element from the set

- update() --merge the both elements and update to A part

set operations

- union() -- merge the two sets without duplicates
- intersection() -- this function gives which are common elements in two sets
- difference() --removes the common elements and return the remaining elements of set1
- symmetric_difference()--removes the common elements and return both set elements
- symmetric_difference_update()--remove the common elements and update to A part
- issuperset()-- return True contains all elements of both sets
- issubset() --returns True contains set1 present in set2 or gives False
- isdisjoint() -- returns True when there is no elements present in set1 and set2

```
s1 ={1,2,3,4,5,6,7}
s2={'a','b','d','f','y','h','l'}

# add()
s1.add('hyd')
s1.add(22.3)
s1

{1, 2, 22.3, 3, 4, 5, 6, 7, 'hyd'}

# remove()
s1.remove(2)
s1

{1, 3, 4, 5, 6, 7, 'hyd'}

# discard()
s2.discard('h')
s2

{'a', 'b', 'd', 'f', 'l', 'y'}

# pop()
s2.pop()
s2

{'a', 'b', 'd', 'f', 'y'}

# update()
s1.update(s2)
s1
```

```
{1, 3, 4, 5, 6, 7, 'a', 'b', 'd', 'f', 'hyd', 'y'}
```

set operations

```
# union()
s3=s1.union(s2)
s3
```

```
{1, 3, 4, 5, 6, 7, 'a', 'b', 'd', 'f', 'hyd', 'y'}
```

```
# intersection()
s4=s2.intersection(s1)
s4
```

```
{'a', 'b', 'd', 'f', 'y'}
```

```
# difference()
s3=s1.difference(s2)
s3
```

```
{1, 3, 4, 5, 6, 7, 'hyd'}
```

```
a={1,2,3,4,5}
b={6,7,8,9,10,11,12,13}
```

```
# symmetric_difference()
c=a.symmetric_difference(b)
c
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}
```

```
a={1,2,3,4,5,11,23,25,67}
b={6,7,8,9,10,11,12,13}
```

```
# symmetric_difference_update()
a.symmetric_difference_update(b)
print(a)
```

```
{1, 2, 3, 4, 5, 67, 9, 8, 12, 7, 6, 10, 13, 23, 25}
```

```
# issuperset() | issubset()  | isdidjoint()
```

```
a={1,2,3,4,5,6,7,8,9,10}
b={4,5,6,7,8}
c={11,12,13,14,15}
```

```
a.issuperset(b)
```

```
True
```

```
b.issuperset(a)
```

```
False
```

```
a.issubset(b)
```

```
False
```

```
b.issubset(a)
```

```
True
```

```
a.isdisjoint(b)
```

```
False
```

dictionary

dict is a mutable

in key will be more priority

keys must be unique and values accecpt duplicates

the data store in the form of {key : value}

```
d1={}
type(d1)
```

```
dict
```

```
d={'hyd':'telangana','banglore':'karnataka','chennai':'tamil','pune':'mumbai'}
d
```

```
{'hyd': 'telangana',
 'banglore': 'karnataka',
 'chennai': 'tamil',
 'pune': 'mumbai'}
```

```
d['hyd']
```

```
'telangana'
```

```
d.get('hyd')
```

```
'telangana'
```

```
d.values() # return dict values
```

```
dict_values(['telangana', 'karnataka', 'tamil', 'mumbai'])
```

```
d.items() # return key-value of a dict
```

```
dict_items([('hyd', 'telangana'), ('banglore', 'karnataka'),
('chennai', 'tamil'), ('pune', 'mumbai')])
```

dict predifined funtions

- clear() --clear all dict elements
- pop() --remove the values by passing the (key)
- popitem() --remove the last inserted (key,value) from dict
- copy() -- coping the content of one dict to anothe dict
- get() -- return value by given key
- keys() -- it returns the keys of a dict
- values() --it returns the values of a dict
- items() --return (key,value)
- update() -- merging one dict to another dict

```
d

{'hyd': 'telangana',
 'banglore': 'karnataka',
 'chennai': 'tamil',
 'pune': 'mumbai'}

#values()
d2=d.values()
print(d2)

dict_values(['telangana', 'karnataka', 'tamil', 'mumbai'])

# items()
d2=d.items()
print(d2)

dict_items([('hyd', 'telangana'), ('banglore', 'karnataka'),
('chennai', 'tamil'), ('pune', 'mumbai')])

for i in d2:
    print(i)

('hyd', 'telangana')
('banglore', 'karnataka')
('chennai', 'tamil')
('pune', 'mumbai')

# update()
d1={1:2,2:20,3:30}
d2={30:1}
d1.update(d2)
d1

{1: 2, 2: 20, 3: 30, 30: 1}
```

```
d
```

```
{'hyd': 'telangana',
 'banglore': 'karnataka',
 'chennai': 'tamil',
 'pune': 'mumbai'}
```

```
# keys()
cap = d.keys()
print(cap)
```

```
dict_keys(['hyd', 'banglore', 'chennai', 'pune'])
```

```
# get()
val=d.get('pune')
val
```

```
'mumbai'
```

```
# pop()
d.pop('pune')
print(d)
```

```
{'hyd': 'telangana', 'banglore': 'karnataka', 'chennai': 'tamil'}
```

```
# popitem()
d.popitem()
print(d)
```

```
{'hyd': 'telangana', 'banglore': 'karnataka'}
```

```
# clear()
d.clear()
```

```
d
```

```
{}
```