Alignment methods

1. ljust()
2. rjust()
3. center()

These methods are used for formatting string
These methods justify or align string within given width

```
>>> str1="nit"
>>> str2=str1.ljust(10)
>>> print(str1)
nit
>>> print(str2)
nit
>>> str3=str1.ljust(10,"*")
>>> print(str3)
```

Example:

```
names=["naresh","ramesh","kishore","amar"]
for name in names:
    print(name.ljust(15,"*"))
```

Output

```
naresh*********
ramesh*********
kishore********
amar***********
```

Example:

```
>>> str1="nit"
>>> str2=str1.rjust(10)
>>> print(str1)
nit
>>> print(str2)
       nit
>>> str3=str1.rjust(10,"*")
>>> print(str3)
*******nit
```

Example

```
>>> str1="nit"
>>> str2=str1.center(15)
>>> print(str1)
nit
>>> print(str2)
      nit
>>> str3=str1.center(15,'*')
>>> print(str3)
******nit******
```

Sets

Sets are unordered collections, where insertion order is not preserved (OR) insertion order is not same (OR) insertion order changes from one insertion to another.

Sets not allows duplicate values/elements

In application developments set used,

1. To group individual objects or values, where duplicates are not allowed
2. To perform mathematical set operations (union, intersection, difference,…)
3. For membership testing (frequency count,…)
4. Remove duplicate values/elements from sequences

Python provides 2 set types

1. Set
2. Frozenset

Set

Set is mutable collection, after creating set changes can be done.
Set is mutable collection but elements or objects of set must be immutable (OR) set allows only immutable objects.
Set is non index based collection, where reading or writing cannot done using index. Set does not support indexing and slicing.
In set data is organized using hashing data structure.

How create set?

1. Set is created with elements using curly braces
   Syntax: {value,value,value,value,…}
2. Set is created using set comprehension
   {expression for variable in iterable}
   {expression for variable in iterable if test}
3. Set is created using set() type function
   set() :creating empty set
   set(iterable):creating set using elements or values generated by iterable (OR) converting existing iterable into set

Note: empty set cannot create using empty curly braces {}
In python empty curly braces represents dictionary

Example:

```
>>> A={}
>>> print(type(A))
<class 'dict'>
>>> A=set()
>>> print(type(A))
<class 'set'>
>>> print(A)
>>> set()
>>> B={10,20,30,40,50}
>>> print(B)
{50, 20, 40, 10, 30}
>>> C={10}
>>> print(C,type(C))
{10} <class 'set'>
>>> D={10,20,30,40,50,60}
>>> E={1.5,2.5,3.5,4.5,10,20,"naresh"}
>>> print(D,type(D))
{50, 20, 40, 10, 60, 30} <class 'set'>
>>> print(E,type(E))
{1.5, 2.5, 3.5, 4.5, 20, 'naresh', 10} <class 'set'>
>>> F={10,10,10,10,10}
>>> print(F,type(F))
{10} <class 'set'>
>>> G={10,20,30,10,20,30,10,20,30,40,50}
>>> print(G,type(G))
{50, 20, 40, 10, 30} <class 'set'>
```

```
>>> H={(10,20),(30,40)}
>>> H
{(30, 40), (10, 20)}
>>> I={[10,20],[30,40]}
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    I={[10,20],[30,40]}
TypeError: unhashable type: 'list'
>>> S1=set("PYTHON")
>>> print(S1)
{'P', 'H', 'Y', 'T', 'O', 'N'}
>>> S2=set(range(10,110,10))
>>> print(S2)
{100, 70, 40, 10, 80, 50, 20, 90, 60, 30}
>>> A=[10,20,30,40,50]
>>> print(A,type(A))
[10, 20, 30, 40, 50] <class 'list'>
>>> B=set(A)
>>> print(B)
{40, 10, 50, 20, 30}
>>> C=[1,2,3,10,20,30,40,1,2,3,4,10,20]
>>> print(C)
[1, 2, 3, 10, 20, 30, 40, 1, 2, 3, 4, 10, 20]
>>> D=set(C)
>>> print(D)
{1, 2, 3, 4, 40, 10, 20, 30}
>>> E=list(D)
>>> print(E)
[1, 2, 3, 4, 40, 10, 20, 30]
```

Set is non index based collection, which does not support indexing and slicing

```
>>> A={10,20,30,40,50}
>>> A[0]
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    A[0]
TypeError: 'set' object is not subscriptable
>>> A[0:3]
```

```
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    A[0:3]
TypeError: 'set' object is not subscriptable
```

How to read content of set?
The set content can be read using for loop
Syntax:

```
for variable-name in iterable:
      statement-1
      statement-2
```

Example:

```
A={10,20,30,40,50}

# Reading with for loop
for x in A:
   print(x)

# Reading with iterator
n=iter(A)
value1=next(n)
value2=next(n)
value3=next(n)
print(value1,value2,value3)
```

Output

```
50
20
40
10
30
50 20 40
```

Set is mutable collection and after creating set we can perform mutable operations. Set allows only 2 mutable operations

1. Adding
    a. add()
2. Removing

a. Remove()
b. Discard()
c. Clear()
d. Pop()

Adding element within set
After creating set elements/values can be added using add() method of set

Syntax: set-name.add(value)

This method is used to only single object

```
>>> A=set()
>>> print(A)
set()
A.add(10)
A.add(20)
>>> A.add(30)
>>> A.add(40)
>>> A.add(50)
>>> print(A)
{40, 10, 50, 20, 30}
>>> A.add(60,70)
Traceback (most recent call last):
  File "<pyshell#45>", line 1, in <module>
    A.add(60,70)
TypeError: set.add() takes exactly one argument (2 given)
```

https://www.hackerrank.com/challenges/py-set-add/problem?isFullScreen=false

```
N=int(input())
A=set()
for i in range(N):
    c=input()
    A.add(c)

print(len(A))
```

https://www.hackerrank.com/challenges/py-introduction-to-sets/problem?isFullScreen=false

```
A=set(array)
a=sum(A)/len(A)
 return a
```

Replacing values within set are not possible because set unordered collection and non index based

```
>>> A={10,20,30,40,50}
>>> A
{50, 20, 40, 10, 30}
>>> A[0]=100
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    A[0]=100
TypeError: 'set' object does not support item assignment
```