9/3/24

Unit - III : Data Structures:

List Data structure:

It is a mutable data structure, containing different data type elements. List is always enclosed within square brackets [ ].

Properties:

1) It allows duplicates.
2) It preserves the order of elements.
3) list elements are accessed by using index/loop/slicing.
4) List also allows insertions, deletions and updates.
5) List may also contains sub-list (nested list).

all the elements in list are seperated by comma.

ex: m = [10, 20, None, 33.3, "IT"].

List Operations: (All datastructures, only list, operatoors)

1) sort() function: used to arrange elements in ascending order.

ex: 1) m = [1, 2, 10, 3]
>m.sort()
>m
[1, 2, 3, 10]

2) m = ['it', 'cse']
>m.sort()
>m
['cse', 'it']

3) m = [1, 'it']
m.sort()
TypeError : '<' not supported between instances of 'str' and 'int'.

4) >m = [1, True, false, 0]
>m.sort()
>m
[false, 0, 1, True]

2) reverse(): It is used to reverse the list elements.

ex: 1) m=[1,2,3]

```
>m.reverse()
>m
[3,2,1]
```

2) m=[1,'it']

```
>m.reverse()
['it',1]
```

3) index():- used to find out index of given element in the list.

ex: 1) m=[10,20,30]

```
>m.index(10)
0
```

2) m.index(99)

ValueError: 99 is not in list.

insertion:- → multiple elements - single index

4) append(): used to add an element at the end of list. based on index.

```
>> m=[10,20,30]
>> m.append('IT')
>> m
[10,20,30,'IT']
```

```
>> m.append([3,4,5])
>> m
[10,20,30,'IT',[3,4,5]]
```

5) extend():- It also adds elements into end of the list.

```
>>> m=[10,20,30]
>>> m.extend([1,2,3])
[10,20,30,1,2,3]
```

multiple elements - multiple indices.

we can also add single element using extend.

6) insert(): used to insert elements based on index.

Ex: >> m = [10,20,30,1,2,3]
>> m.insert(1,45)
>> m
[10,45,20,30,1,2,3]

2) m.insert(12,100)
↳ If index is not present it doesn't throws any exception, it inserts at last.
>> m
[10,45,20,30,1,2,3,100]

>m[0] = 23
>m
[23,45,20,30,1,2,3,100]

>m[-1] = 300
>m
[23,45,20,30,1,2,3,300]

>m[1:3] = [8,9]
>m
[23,8,9,30,1,2,3,300]

>> m = [1,2,3]
>> m.insert(65,4)
>> m
[1,2,3,4]
↓ It will not give any error.

Deletion:

7) pop(): used to delete last element from the list.
→ two types - 1) default (no parameters)
2) with parameter.

>> m.pop()
300
>>m
[23,8,9,30,1,2,3]
>> m.pop(2)
9
>>m
[23,8,30,1,2,3]
>>> m.pop(20)
IndexError: pop index out of range.

8) remove(): It removes the elements based on given value.

>> m = [10,20,20,30]
>> m.remove(20)
>>> m
[10,20,30]
↑ left occurrance of 20 will be deleted.
removes
>>> m.remove(100)
ValueError: List.remove(x): x not in list.

9) clear(): It clears the data in list.

10) del(): It removes permanently from memory.

↓ apply to all data structures.

Clear(): It removes all elements from the list.

Ex: 1) >m=[1,2,3,'IT']

```
>m
[1,2,3,'IT']
>>m.clear()
>>m
[ ]
>>m.clear()
>>m
[ ]
```

del(): It is a common for all data structures, by using this function remove single element or set of elements or remove completely from the memory.

Ex:

```
>>a=10
>>del a
>>a
```

Name Error: name 'a' is not defined.

```
>>> a=[3,4,5]
>>> del (a[0])
>>> a
[4,5]
```

```
>>> a=[3,4,5,6,7]
>> del a[1:3]
>> a
[3,6,7]
>>>del a
>> a
error.
```

```
>>a=[1,2,3,4,5,6,76,7]
>> del a[1:3:4]
>> a
[1, 3, 4, 5, 6, 76, 7]
```

```
>>> del a[1:20:3]
>>> a
[1, 4, 5, 76, 7]
```

```
>> for i in range(1, 20,3):
       print(i)

1
4
7
10
13
16
19
>>> a
[1, 4, 5, 76, 7]
```

```
>>> a[100: 200]
[ ]
>>> a
[1, 4, 5, 76, 7]
>>> a[:100]
[1, 4, 5, 76, 7]
>>> a=[10, 20, 30, 40, 50, 60]
>>> del a[:100:2]
[10, 30, 50]
```

ii) copy(): shallow copy and deep copy

shallow copy means it creates another reference for the existing list. (alias)

Deep copy means it creates seperate copy.

ex: shallow copy():

```
1)> a = [10, 20, 30, 40, 50]
  >b= a  # shallow copy
  >a
  [10, 20, 30, 40, 50]
  > b
  [10, 20, 30, 40, 50]
  => id(a), id(b)
  (2583167337928, 2583167337928)
  >> b[0] = 99
  >> a
  [99, 20, 30, 40, 50]
  >> b
  [99, 20, 30, 40, 50]
```

```
>> a =[10, 20, 30]
>> b =a.copy()
>> a
[10, 20, 30]
>> b
[10, 20, 30]
```

copy. id(a), id(b) not equal

```
>> b[0] = 99
>> a
[10, 20, 30]
>> b
[99, 20, 30]
```

```
>> a = [10, [20, 30, 40]]
>> b = a.copy()
>> a
[10, [20, 30, 40]]
>>> b
[10, [20, 30, 40]]
>>> b[1][0] = 99
>>> b
[10, [99, 30, 40]]
>>> a
[10, [99, 30, 40]]
```

```
>> import copy
>> a = [10, [20, 30, 40]]
>> b = copy.deepcopy(a)
>> b
[10, [20, 30, 40]]
>>> b[1][0] = 99
>>> a
[10, [20, 30, 40]]
>>> b
[10, [99, 30, 40]]
```

```
> a=[10,20,30]
> b=copy.deepcopy(a)
> a
[10,20,30]
> b
[10,20,30]
> a[0]=98
> a
[98,20,30]
> b
[10,20,30]
```

12) len(): used to find out length of given sequence.

Quiz

```
>> a
[10, [20, 30, 40]]
>> len(a)
2
>> len(a[1])
3
>>> len(a[-1])
3
```

```
>>len(a[0])
TypeError: object of type 'int' has no len()
>>len('welcome')
7
>>> m=['eee', 'it', 'cse', 'mech.']
>>len(m[0])
3
>>>len(m[0][0])
1
>>len(m[1][2])
IndexError: string index out of range.
```

13) min(): used to findout minimum value from the given sequence.

Eg:
```
> min(10)
TypeError: 'int' object is not iterable.
>>min(21,32)
21
>>> a
[10, [20, 30, 40]]
>>> min(a)
TypeError: '<' not supported blw instances of 'list' and 'int'.
```

```
>>> mn(a[0])
20
>> a = [3,4,5,6,7]
>> mn(a)
 3
>> a = ['it','cse','eee']
>> min(a)
 'cse'
>> a = [1,'cse']
>> mn(a)
TypeError: '<' not ...
>> a = [True,1,False,10]
>> min(a)
 False.
```

14) **max()**: It is used to findout maximum value from the given sequence.

15) **sum()**: It is used to findout sum of given list of elements.

only for integers floating boolean

```
>> list(123)
  error.
>> a = [1,2,3]
>> sum(a)
  6
>> a = [True,3]
>> sum(a)
  4
>> a = ['I1','cse']
>> sum(a)
  error.
>> sum(list(map(int,list(str(456)))))
  15
>> list(str(456))
  ['4','5','6']
>> a = 'welcome'
>> a.sort()
   error.
```

```
>sum(list(map(int,str(4,5,6))))
 15
```

```
>> a=list('welcome')
>> a
['w','e','l','c','o','m','e']
>> a.sort()
>> a
['e','e','e','l','m','o','w']
>>> ''.join(a)
'eeelmow'
>> a.reverse()
>> ''.join(a)
'womleee'
```

16) count(): used to count no. of times an elements occurred in sequence.

Ex:
```
> a=[10,20,20,20,10]
>a.count(20)
  3
>>> a.count(100)
  0
```

17) sorted(): It is used for all data structures. This function is used to sort elements in either in ascending order or descending order.

→ return value is list.

Ex:
```
>a= [1,2,0,-4]
>sorted(a)
 [-4,0,1,2]
>> a
 [1,2,0,-4]
>>sorted(a,reverse=True)
 [2,1,0,-4]
```
↓ by default it is false

descending order.

18) any(): any() function returns boolean value.

ex: any(a)
True.

If any one of the element is true then it is true.
If all are false then only it is false.

>> a = [0, 0, 0]
>> any(a)
false.

19) All():

a = [1, 2, 3]
>> all(a)
True.
>> a = [0, 1, 2]
>> all(a)
false.

Operators:

20) *, +

>> a = [10, 20]
>> a * 5
[10, 20, 10, 20, 10, 20, 10, 20, 10, 20]
>> a[0]
>> a * 10
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>> a = [1, 2]
>> b = ['it', 'cse']
>> a + b
[1, 2, 'it', 'cse']
>> b + a
['it', 'cse', 1, 2]

21) <, >, <=, >=, ==, !=

>> a = [10, 20]
>> b = [20, 30]
>> a < b
True.

```
>> a = [10, 20, 30]
>> b = [10, 20, 1, 5, 6]
>> a > b
True
>> a = ['it', 'cse']
>> b = [4, 5]
>> a < b
TypeError:
>> a = ['it', 'cse']
>> b = ['eee', 'ece']
>> a < b
false
```

→ write a pp to read six subjects marks and findout total and percentage of marks using list. (without comprehension).

```
n=int(input("enter n value"))
m=[]
for i in range(n):
    print('enter subject', i+1, 'marks:')
    k=int(input())
    m.append(k)
print(m)
print('Total marks', sum(m))
print('percentage of marks', sum(m)/(n*100)*100)
print("percentage of marks: %.2f"%(sum(m)/(n*100)*100))
```

8/4/24

set Data Structure:-

Set is a sequence of data items of any type.

→ These are also called Iterable objects.

→ set elements are always enclosed with in curly braces { }.

Properties:

1. set is always unordered

ex: m = {4,1,2}
m
{1,2,4}

2. It does not allow duplicates.

ex: s = {1,2,1,2}
s
{1,2}

3. set is mutable. (removal, insertions are possible in set)

4. There is no index for set elements, so it doesnot support slicing.

ex: s = {4,1,2}
s[0]
TypeError:

5. Using for loop access the set elements.

Syntax:

obj = {elem1, elem2 ....}

ex: s = {'it', 'cse', 23, false, 55.5}
s
{false, 'it', 23, 55.5, 'cse'}

How to declare an empty set?

using set conversion function. set() method.

ex: s = set()
>type(s)
<class 'set'>

don't use s = { } → represents an empty dictionary

>type(s)
<class 'dict'>

sequences (or) Iterable objects. { strings, list, tuple, set, dictionary }

How to perform some operations on set?

1) Insertion:

1) add(): used to add an element into the set.

syntax: setobj.add(ele)

Ex:

```
>s={4,2,1,'IT'}
>s
{1,2,4,'IT'}
>s.add('CSE')
>s
{1,2,'CSE','IT'}
>s.add(3,4,5)
  Error
>s.add([3,4,5])
  Error.
```

2) update(): used to add more no. of items into the set in the form of new set.

syntax:

setobj.update(new set)

Ex:

```
>s.update({5,6})
>s
{1,2,'CSE',4,'IT',5,6}
>s.update({4,5,6})
>s
{1,2,'CSE',4,5,6,'IT',5,6}
>s1={7,8,9}
>s.update(s1)
>s
{1,2,'CSE',7,8,9,4,5,6,'IT',5,6}
```

```
>s.update({[4,5,6]})
  Error:
```

2) Deletions: How to delete an element from set.

1) remove(): used to remove an element from the set.

Syntax: setobj.remove(ele)

Ex: >s.remove(1)

```
>s
{2,'CSE',4,5,6,7,8,9,'IT',56}
>s.remove(10)
KeyError:
>s={3,4,5,6,7}
>s.remove(4)
>s
{3,5,6,7}
```

```
>s.remove(100)
KeyError:
```

2) pop(): It is also used to remove an element from the set.

Syntax: setobj.pop()

Ex:

```
>s={5,6,7,8}
>s
{8,5,6,7}
>s.pop()
8
>s
{5,6,7}
>s={9,2,3,4}
>s.pop()
9
>s
{2,3,4}
>s.pop()
2
>s.pop()
3
>s.pop()
4
```

```
>s.pop()
KeyError:
```

3) discard(): used to remove an element, if element is not present then it doesn't throw any exception.

→ remove() and pop() methods throws an exception whenever an element is not present in set, but discard() method never throws any exception.

syntax: setobj.discard(ele)

Ex: >s={2,6,7,8}
>s.discard(7)
>s
{8,2,6}
>s.discard(2)
>s.discard(8)
>s.discard(6)
>s

4) clear(): used to remove all the elements from the set.

syntax:
setobj.clear()

→ clear only removes elements from the set but reference is available.

Ex: >s={4,5,6}
>s.clear()
>s
set()
>s.discard(44)
set()

5) del():

syntax: del(setobj)

8) union():

Ex:

```
>S1 = {1,2,3,4,5}
>S2 = {10,2,3,4,11}
>S1.union(S2)
{1,2,3,4,5,10,11}
>S3 = {3,4,20}
>S1.union(S2,S3)
{1,2,3,4,5,10,11,20}
>S4 = {'it','cse'}
>S1.union(S2,S3,S4)
{1,2,3,4,5,'it',10,11,20,'cse'}
```

```
>>S1.union([6,7,8])
{1,2,3,4,5,6,7,8}
>>m = [3,8,10]
>>S1.union(m)
{1,2,3,4,5,8,10}
>>v1 = 'welcome'
>>v2 = 'welcome to gec'
>>set(v1)|set(v2)
{'e','o','m','l','t','g','w','c',' '}
>>k = set(v1)|set(v2)
>>k
{'e','o','m','l','t','g','w','c',' '}
>>''.join(k)
'eomltgwc'
>>v1
'welcome'
>>set(v1)
{'e','o','m','l','w','c'}
```

```
>S1|S2
{1,2,3,4,5,10,11}
>S1|S2|S3
{1,2,3,4,5,10,11,20}
>S1|S2|S3|S4
{1,2,3,4,5,'it',10,11,20,'cse'}
```

→ union operation doesnot effect on contents of S1.

```
>>S1|m
Type Error:
```

→ union is used to combine two or more sets.

→ we can use '|' instead of union.

```
>>''.join(set(v1))
'eomlwc'
>>n = 12223
>>len(set(str(n)))
```

9) Intersection():

```
>>S1
{1,2,3,4,5}
>> S2
{2,3,4,10,11}
>> S1.intersection(S2)
{2,3,4}
> S3
{3,4,20}
> S1.intersection(S2,S3)
{3,4}
>> S1 & S2
{2,3,4}
> m                    > S1.intersection(m)
[3,8,10]                {3}
> S1 & m
  Error.
```

10) difference():

```
>> S1.difference(S2)        >> S1-S2
{1,5}                       {1,5}
>> S2.difference(S1)
{10,11}
>> m
  [3,8,10]
>> S1.difference(m)
{1,2,4,5}
```

11) set symmetric_difference():

>> S_1
{1,2,3,4,5}
>> S_2
{2,3,4,10,11}
>> S_1.symmetric_difference(S_2)
{1,5,10,11}
>> $S_1 \wedge S_2$
{1,5,10,11}

12) isdisjoint(): It returns boolean value.

>> S_1
{1,2,3,4,5}
>> S_2
{2,3,4,10,11}
>> S_1.isdisjoint(S_2)
false

>> C_1 = {1,2}
>> C_2 = {5,6,7}
>> C_1.isdisjoint(C_2)
True.

13) issubset():

>> S_1.issubset(S_2)
false
>> C_1 = {1,2}
>> C_2 = {1,2,3,4}
>> C_1.issubset(C_2)
True
>> $C_1 <= C_2$
True.

14) issuperset():

>> C_1.issuperset(C_2)
false
>> C_2.issuperset(C_1)
True.

>> $C_2 >= C_1$
True.

15) len():

16) max():

17) min():

18) sum()

19) any():

20) all():

21) in, not in

22) <=, >=

23) copy():

Comprehensions:

```
m=[i for i in range(10) if i%2==1]
```

[1, 3, 5, 7, 9]

```
m={i for in range(10) if i%2==1}
print(m)
```

{1, 3, 5, 7, 9}

```
m=tuple(i for i in range(10) if i%2==1)
print(m)
```

(1, 3, 5, 7, 9)

```
m={i for in map(int, input().split())}
print(m)
```

o/p:

1) 2 2 2 56 6 3 1 8 2

{1, 2, 3, 6, 8, 56}

2) 1201 1202 1203 1201 1202

{1201, 1202, 1203}

```
n=int(input("enter range:"))
s=set()
for i in range(n):
        s.add(int(input()))
print(s)
```

o/p: enter range:4
7
9
7
9
{7,9}

How to access the set elements in python?

using loops - access the set elements.

ex: for i in s:
        print(i)

7
9

```
>>dir(set())
```

→ Dictionaries:-

→ It is also represent in the form of { }

It always contains key value pairs

key and value are seperated with: colon symbol.

syntax:

dictobj={key1:value1, key2:value2, ...}

→ key may be any datatype, value also any datatype

→ keys never repeated, values may repeated.

→ dictionaries also unordered.

NO indices and slicing.

i.e, we cannot access the dictionaries using indexing or slicing

1) Insert:

i) using [ ]:

Syntax:

dictobj[key]=value

Ex:

```
>d = {1:'it', 2:'cse'}
>d
{1:'it', 2:'cse'}
>> d[5] = 'eee'
>>d
{1:'it', 2:'cse', 5:'eee'}
>>d[2] = 'ce'
>>d
{1:'it', 2:'ce', 5:'eee'}
>>d['gec'] = 48
>>d
{1:'it', 2:'ce', 5:'eee', 'gec':48}
>>d[22.3] = 56
>>d
{1:'it', 2:'ce', 5:'eee', 'gec':48, 22.3:56}
```

ii) update():

Syntax: dictobj.update(dict)

Ex: y={5:'mech', 7:'aiml', 'gec':'it'}

```
> d.update(y)
>d
{1:'it', 2:'ce', 5:'mech', 'gec':'it', 22.3:6, 7:'aiml'}
>d.update({6:9})
{1:'it', 2:'ce', 5:'mech', 'gec':'it', 22.3:6, 7:'aiml', 6:9}
```

2) delete:

3) pop(): pop() definitely requires one parameter in dictionary

>d.pop(2) syntax: dictobj.pop(key)

'ce'

>d

{1:'it', 5:'mech', 'ec':'ib', 22-3:56, 7:'aiml', 6:9}

>d.pop(10)

KeyError:

When KeyError arises?

If we access set of dictionary element, if that element is not present, then it throws key Error.

>> d = {20481201: {'abc', 'us', 99.99, 9898989}}

>>d[20481201]

{'us', 'abc', 99.99, 9898989}

>> d = {(1,2) : (3,4)}

>> d

{(1,2): (3,4)}

>> d[(1,2)]

(3,4)

4) popitem(): popitem() always remove element from the end of the dictionary

syntax: dictobj.popitem()

ex: >d = {1:'ce', 2:'eee', 3:'me', 4:'ece', 5:'cse'}

>d.popitem()

(5,'cse')

The o/p is in tuple format.

5) clear():

```
>d.clear()
>d
{ }
```

6) del():

```
d={1:'a', 2:'b', 3:'c'}
>del d[1]
>>d
{2:'b', 3:'c'}
>>del d[8]
KeyError:
>>del d
>>d
Name Error:
```

7) copy():

```
>d={1:'a', 2:'b'}
>a=d.copy()
>a
{1:'a', 2:'b'}
```

How to access dictionary elements!

8) get():

Syntax: dictobj.get(key)

```
Ex:>d
  {1:'a', 2:'b'}
  >d.get(1)        >>d.get(6)
  'a'              >>
 >>d[1]
  'a'
>>d[6]
  KeyError:
```

Difference between [ ] and get() method.

get() method doesn't throw any exception, if key is not present in the dictionary.

9) set default() method:

It requires for get() method

Ex:

```
>d.setdefault(11, None)
>d
{1:'a', 2:'b', 11: None}
>>d.get(11)
>>d.get(2)
'b'
>>d.get(7)
>>d.get(7,d[11])
>>d.setdefault(11,"No key")
'No key'
>>d
{1:'a', 2:'b', 11:'No key'}
>>d.get(7,d[11])
'No key'
>>d.get(2,d[11])
'b'
```

setdefault() method is used to provide default values for no key elements.

10) keys():

Syntax: dictobj.keys(), list(dictobj.keys())

Ex: d.keys()

dict_key([1, 2, 11])

>list(d.keys())

[1, 2, 11]

11) values():

>d.values()

dict_values(['a', 'b', 'No key'])

12) items():

Ex: >d.items()

dict_items([(1,'a'), (2,'b'), (11,'No key')])

>list(d.items())

[(1,'a'), (2,'b'), (11,'No key')]

How to iterate dictionaries:

```
d = {1:'a', 2:'b', 3:'c', 4:'d'}
for i in d:
        print(i, d[i])
```

O/P: 1 a
2 b
3 c
4 d

13) min(), max(), sum(), sorted(), del(), any(), all()

```
d: {1:'a', 2:'b', 3:'c', 4:'d'}
print("1st method")
for i in d:
    print(i,d[i])
print("2nd method")
for x,y in d.item():
    print(x,y)
print("3rd method")
for i in d.keys():
    print(i,d[i])
print("4th method")
for i in d.values():
    print(i)
print("5th method")
for in in d:
    print(i,d.get(i))
```

output:

```
1st method
1 a
2 b
3 c
4 d
2nd method
1 a
2 b
3 c
4 d
3rd method
1 a
2 b
3 c
4 d
4th method
a
b
c
d
5th method
1 a
2 b
3 c
4 d
```

```
>d
{1:'a', 2:'b', 3:'c', 4:'d'}
>>min(d)
1
>>max(d)
4
>>sum(d)
10
>>min(d.values())
'a'
>>sum(d.values())
TypeError:
>>d*3
TypeError:
>>d+{4:5}
TypeError:
```

Comprehensions:

```
→ d={i:i*i for i in range(1,5)}
print(d)
```

o/p: {1:1, 2:4, 3:9, 4:16}

```
→ d={i:len(i) for i in input().split()}
print(d)
```

o/p: gec cse it a

{'gec':3, 'cse':3, 'it':2, 'a':1}

1/4/24

LAB Program (13)

```
import math
def ball_collide(t1,t2):
    x1,y1,r1 = t1[0], t2[1], t1[2]
    x2,y2,r2 = t2[0], t2[1], t2[2]
    d = math.sqrt((x2-x1)**2 + (y2-y1)**2)
    c = d/2
    r = r1+r2
    if c<=r:
        return True
    else:
        return False
b1 = tuple(int(i) for i in input("enter x1,y1,r1").split())
b2 = tuple(int(i) for i in input("enter x2,y2,r2").split())
if(ball_collide(b1,b2)):
    print("collide")
else:
    print("Not collide")
```

LAB Program (14)

```
st = input("enter a string")
d = {}
for i in st:
    d[i] = st.count(i)
print(d)
print("count is:")
for x,y in d.items():
    print(y,'->',x)
```

1. Write a pp to sort dictionary using keys.

key may be integeal/float/string

Ex: d={3:'it', 5:'cse', 1:'eee'}

o/p:

{1:'eee', 3:'it', 5:'cse'}

→
```
k=[i for i in map(int,input('enter keys:').split())]
v=[i for i in input('enter values:').split()]
d=dict(zip(k,v))
print("Before sorting")
print(d)
for i in sorted(d.keys()):
        p[i]=d[i]
print("After sort:")
print(p)
```

2. Develop python program to sort dictionary elements using values.

```
k=[i for i in map(int,input('enter keys:').split())]
v=[i for i in input('enter values:').split()]
d=dict(zip(k,v))
print("Before sort:")
print(d)
s={}
b=sorted(d.values())
for i in b:
        for x,y in d.items():
                if y==i:
                        s[x]=i
print("After sort:")
print(s)
```

[without using pre-defined functions]

* using predefined functions:

```
k= [i for i in map(int, input('enter keys:').split())]
v= [i for i in input('enter values:').split()]
d= dict(zip(k,v))
print("Before sort:")
print(d)
s={}
for i in sorted(d.items(),key=lambda x: x[1]):
        s[i[0]] = i[1]
print("After sort")
print(s)

for i in sorted(d, key=d.get):
        s[i]=d[i]
```

⇒ Strings:- in python:-

Python strings are immutable.

functions: Applications - data validations.

1) captalize(): This function is used to convert the first letter into kapital) upper case remaining all letters are lower case.

Ex: >s = 'welcome'
>s.captalize()
'Welcome'
>S = 'WELCOME'
>s.captalize()
'Welcome'
>S
'WELCOME'
>s='123welcome'
>s.captalize()
'123welcome'

syntax:
stringobj.capitalize()

```
>S.= 'welcome to it'
> s.capitalize()
'Welcome to it'
```

2) title(): This function is used to convert every word first character into upper case.

Syntax: stringobj.title()

Ex:

```
>S='welcome'
>s.title()
'Welcome'
>S='welcome to it'
>s.title()
'Welcome To It'
>S='WELCOME TO IT'
>s.title()
'Welcome To It'
```

3) casefold(): This function is used to convert all letters into lower case.

Syntax: stringobj.casefold()

Ex:

```
>S='welcome'
>s.casefold()
'welcome'
>S='WELcome'
>s.casefold()
'welcome'
```

4) lower(): This function is used to convert uppercase alphabets into lowercase alphabets.

Syntax: stringobj.lower()

Ex: >s='SRGEC'
>s.lower()
'srgec'

5) upper(): This function is used to convert lowercase alphabets into upper case alphabets.

Syntax: stringobj.upper()

Ex: >s='SRgec'
>s.upper()
'SRGEC'

6) swapcase(): to interchange lowercase letters into upper case and uppercase into lower case.

Syntax: stringobj.swap()

Ex: >s='WeLcOmE'
>s.swapcase()
'wELCoMe'

> S='WELCOME'
> s.swapcase()
'welcome'

7) encode(): The encode() method encodes the string, using the specified encoding. If no encoding is specified, UTF-8 will be used.

Syntax: stringobj.encode() string.encode(encoding=encoding, errors=errors)

Ex: > s.encode()
b'WELCOME'

8) len(): used to find out length of given string.

9) count(): used to count number of occurences of given character.

>p='srgec'
>len(p)
5

>p.count('e')
1

>s='welcome'
>s.count('e')
2

10) center(): Appending blank spaces before and after the string

Syntax: stringobj.center(length)

Ex: >p='srgec'
>p.center(20)
srgec
>p.center(10)
srgec

11) startswith(): This method returns True if the string starts with the specified value, otherwise false.

Syntax: stringobj.startswith(value)

Ex: >p.startswith('sr')
True
>p.startswith('gec')
false

12) endswith(): This method returns True if the string ends with the specified value, otherwise True.

Syntax: stringobj.endswith(value)

Ex: >p.endswith('gec')
True
>p.endswith('sr')
True

13) index():

Syntax:

Ex: >p='srgec'
>p.index('r')
1
>p.index('gec')
2
>>p.index('z')
value Error:

14) rindex():

syntax:

Ex: s = 'welcome'
>
>s.index('e')
1
>s.rindex('e')
6

15) find():
It searches for a character in the string and return index. If the character is not present in the string, it doesn't throw any exception.

ex: >S = 'welcome'
>s.find('e')
1
>s.find('z')
-1

16) rfind(): It finds from right side. same as like find() it doesn't throw any exception if the character is not present in the string

syntax:
ex: >s.rfind('e')
6

17) lstrip(): This function is used to remove blank spaces from left side.

syntax:
EX: >S = ' welcome to it '
> s.lstrip()
'welcome to it '

18) rstrip(): Removes blank spaces from right side.

syntax: stringobj.rstrip()

Ex: >s=' welcome to it '

>s.rstrip()

' welcome to it'

19) strip(): Removes blank spaces from both the sides.

syntax: stringobj.strip().

Ex: >s.strip()

'welcome to it'

20) join(): This method takes all items in an iterable and joins them into one string. A string must be specified as seperator.

Syntax: string.join(iterable)

Ex: >s='welcome'

>'srgec'.join(s)

'wsrgecesrgecmsrgecesrgec'

>' '.join(s)

'w e l c o m e'

18/4/24

21) format(): to display the output in given format.

Ex: a=10

> b=20

>'{}+{}={}'.format(a,b,a+b)

'10+20=30'

>'{a}'+'{b}'='{a+b}'

'{a}+{b}={a+b}'

>'{0}+{1}={2}'.format(a,b,a+b)

'10+20=30'

>'{0:x}'.format(56)

'38'

22) replace(): used to replace existing string with new string.

ex:

```
>s='welcome to it department'
>s.replace('t','z')
'welcome zo iz departmenz'
>s.replace('q','z')
'welcome to it departemt'
```

// if the given string is not present it doesn't throw any exception the result is same string only.

if the string is present then it replace all the strings with new string.

```
> s.replace('t','z',1)
'welcome zo it department'
>s
'welcome to it department'
```

23) split(): used to split the given string into list of strings based on separator.

ex:

```
>s.split(' ')
['welcome', 'to', 'it', 'department']
>s.split('t')
['welcome ', 'o i', 'depar', 'men', '']
> s.split('')
value Error: empty separator.
>s.split()
['welcome', 'to', 'it', 'department']
>s.split('q')
['welcome to it depaotment']
> s1='22U81A1200='
```

24) splitlines():

Ex:

```
>s= '''welcome
 to
 it'''
>s.splitlines()
['welcome', 'to', 'it']
```

25) zfill():

Ex: s='srgec'

```
>s.zfill(6)
'0srgec'
```

String validation functions:

26) isalpha(): If all characters are alphabets, then it returns true.

Ex:

```
>s='welcome'
>s.isalpha()
True
>s='welcome it @12'
>s.isalpha()
false
>s='sr gec'
>s.isalpha()
false
```

27) isdigit() or isdecimal(): If all characters are digits then it returns true.

```
s='121'
s.isdigit()
True
```

```
s='a123'
s.isdigit()
false
```

28) isalnum(): If string contains all characters are both alphabets and digits then it returns true.

Ex:
```
>s='ab'
>s.isalnum()
 True
```

29) isspace():

Ex:

30) isidentifier():

Ex: s='welcome'
```
>s.isidentifier()
 True
>s='welcome to'
>sisidentifier()
 false
```

31) islower(): all characters are lowercase

32) isupper(): all characters are uppercase.

4/24

Example:

```
a=int(input("enter an integer"))
b=int(input("enter another integer"))
try:
    print("Division is:",a/b)
except ZeroDivisionError:
    print("Denominator is zero:",e)
else:
    print("Else block ... ")
```

output:

1) Enter an integer: 6
Enter another integer: 2
Division is : 3.0
Else block ....

2) enter an integer: 6
enter another integer: 0
Denominator is zero: division by zero.

i) Predefined or built-in exceptions.

ZeroDivisionError /0
IndexError tuple, list, string index out of range
KeyError dict
ValueError int(input()) - string, wrong input, sqrt(-5), factorial
OverflowError e^2000
NameError invalid variables
ModuleNotfoundError. wrong module import

2. write a pp to handle index exception.

```
m=['ece','it','cse','eee']
i=int(input("enter an index:"))
try:
    print("given index value is:",m[i])
except IndexError as e:
    print("out of bounds:",e)
else:
    print("else block....")
```

3. write a pp to handle value Error:

```
m= ['ece', 'it', 'cse', 'eee']
try:
    print(i
    i = int(input("enter an index:"))
    print("given index value is:",m[i])
except ValueError as e:
    print("Invalid index:",e)
else:
    print("It is else block...")
```

LAB (15)

4. write a pp to handle multiple errors with one except statement.

```
m = ['ece','cse','it','eee']
try:
    a=int(input("enter first integer"))
    b=int(input("enter another integer"))
    i=int(input("enter an index:"))
    print("Division is:", a/b)
    print("List index value is:",m[i])
except (zeroDivisionError, IndexError, ValueError) as e:
    print("Error...", e)
else:
    print("It is else...")
finally:
    print("finally block...")
```

LAB 16:

```
class ShortInputException(Exception):
    def __init__(self,err):
        self.err=err
st=input("enter a text:")
try:
    if len(st)<3:
        raise ShortInputExcept("input is too short")
    else:
        print("Text is:",st)
except ShortInputException as e:
    print("Error...:",e)
else:
    print("else block..")
finally:
    print("finally block...")
```

22/4/24

Slicing:

Slicing is the process of accessing sequence of elements based on certain range, from tuple, list and string.

Note: Slicing is not applicable for sets and dictionaries.

Slicing is nothing but part of data.

Types of slicing:

1. single slicing
2. Double slicing.

Single slicing:

Syntax: seq_name [start:stop]

a) positive Index:

Ex: >a = [10, 20, 30, 40, 50, 60, 70]

```
> a [3]
 40
>a [40]
IndexError: list index out of range.
```

	-7	-6	-5	-4	-3	-2	-1
a = [10,	20,	30,	40,	50,	60,	70]	
	0	1	2	3	4	5	6

```
>a [3:7]
[40, 50, 60, 70]
```

when we use +ve indices always start < stop

```
> a [4:10]
[50, 60, 70]
```

→ slicing never throws exception.

```
>a [:]
[10, 20, 30, 40, 50, 60, 70]
```

→ default start is zero, default stop is length of sequence

i) only with start index:

```
>a [0:]
[10, 20, 30, 40, 50, 60, 70]
>a [3:]
[40, 50, 60, 70]
>a [10:]
 [ ]
>a [6:]
 70
```

ii) only with stop index:

```
> a[:100]
[10, 20, 30, 40, 50, 60, 70]
> a [:30]
[10, 20, 30]
> a[:0]
 [ ]
```

iii) both indexes start and stop:

```
>a[0:1]
[10]
>a[3:2]
[]
>a[3:3]
[]
>a[4:0]
[]
```

b) Negative Index: start<stop

i) only with start index:

```
> a[-1:]
[70]
> a[-2:]
[60, 70]
>a[-7:]
[10, 20, 30, 40, 50, 60, 70]
>a[-10:]
[10, 20, 30, 40, 50, 60, 70]
```

ii) only with stop index:

default start is -7, upto -2 exclude -3.

```
> a[:-3]
[10, 20, 30, 40]
>a[:-20]
[]
```

-7 < -20 X

iii) both indexes start and stop:

```
>a[-1:-6]
[]
>a[-3:-1]
[50, 60]
>a[-10:-3]
[10, 20, 30, 40]
```

c) Both positive and negative index:

```
>a[3:-3]
 40
>a[4:-5]
 []
>a[-7:6]
[10,20,30,40,50,60]
>a[0:-1]
[10,20,30,40,50,60]
```

```
          -7 -6 -5 -4 -3 -2 -1
a = [10, 20, 30, 40, 50, 60, 70]
     0   1  2  3  4  5  6
```

26/4

Double slicing:

Syntax:

seq_name [start : stop : step] start=0, stop=n, step=1.

1) Positive index: start < stop

```
          -8 -7 -6 -5 -4 -3 -2 -1
Eg: a = [10, 20, 30, 40, 50, 60, 70, 80]
         0  1  2  3  4  5  6  7
```

```
>>a[::]  → default step is '1'.
          default start is '0'
          default stop is '8'
                 ↑8
>>a[3::]→1
[40,50,60,70,80]
>>a[40::] → slicing never gives IndexError.
[]
>>a[3:5:]
[40,50]
>>a[1:6:1]
[20,30,40,50,60]
>>a[1:6:2]
[20,40,60]
```

```
>>a[3:6:40]
[40]
>>a[::3]
[10,40,70]
>>a[:4:]
[10,20,30,40]
```

```
>>a[10:39:80]
[]
>>s='welcome'
>>s[::2]
'wloc'
>>
```

2. Negative index:

```
>>>a
[10,20,30,40,50,60,70,80]
 0  1  2  3  4  5  6  7
-8 -7 -6 -5 -4 -3 -2 -1
>>a[-1::]
[80]
>>a[-3::]
[60,70,80]
>>a[-9::]
[10,20,30,40,50,60,70,80]
>>a[-1::-1]
[80,70,60,50,40,30,20,10]
>>a[::-1]
[80,70,60,50,40,30,20,10]
>>s='welcome'
>>s[::-1]
'emoclew'
```

```
>>a[-8:-5:]
[10,20,30]
```

default start: -1, stop: -9

1. write a pp to find out given string is palindrome or not.

```
st=input('enter a string')
if st==st[::-1]:
    print(st,'is palindrome')
else:
    print(st,'is not palindrome')
```

-1
-9
>a[::-2]

[80,60,40,20]

>>a[:-5:] [10,20,30]
-8 1

3) Both positive and negative index!

>a[-8:-5:1]

[10,20,30]

>a[3:-1:1]

[40,50,60,70]

> a[-3:1:-1]

[60,50,40,30]

>>a[-3:8:2]

[60,80]

>>a[-3:8:-1]

[]

Numpy: → Numeric Python.

1. How to install numpy in IDLE?

>pip install numpy

for creating arrays in python 'numpy' module is available..

→ 'numpy' is 50 times faster than list, due to internal storage.

→ numpy elements are stored in continuous memory locations

→ list elements are stored in continuous (or) Random memory allocations.

→ dataset is more then only prefer numpy module.

3. numpy contains more no.of predefined functions for matrix operations and linear algebral relational operations.

4. numpy allows any no.of dimension arrays.

5. Most of the numpy libraries are written in c and c++.

→ How to create array in numpy?

```
from numpy import *
a=array([1,2,3])
print(a)
```

o/p: [1 2 3]

* Types of arrays in numpy:

1. 0 Dimension
2. 1 Dimension
3. 2 Dimension
4. N Dimension

i) How to create zero dimension array in numpy module?

```
>from numpy import *
>a=array(10)
>a.ndim → to display no.of dimensions.
0
```

ii) one dimensional array:

```
>from numpy import *
>a=array([1])
>a.ndim
1
```

Attributes in numpy:

ndim

Two dimension Array:

```
>from numpy import *
>d = array([[1,2],[3,4]])
>d.ndim
 2
 >d
array([[1,2],[3,4]])
```

Three dimension Array:

```
>from numpy import *
>e = array([[[1,2],[3,4]],[[5,6],[6,7]]])
>e.ndim
 3
 >e
 array([[[1,2],
         [3,4]],
         [[5,6],
          [6,7]]])
```

```
>a.shape
 ()
>b.shape
 (4,)
>d.shape
 (2,2)
> e.shape
 (2,2,2)
```

```
>a
 array(10)
> b
 array([1,2,3,4])

>a.size
 1
>b.size
 4
> d.size
 4
> e.size
 8
```

```
>>a.dtype
dtype('int32')
>>b.dtype
dtype('int32')
```

How to create data type:

```
>k=array([3,4,54], dtype='i')
>k
array([ 3, 4, 54], dtype = int32)
>k.astype('f')
array([3., 4., 54.], dtype=float32)
```

List of data types in numpy:

i = integer
f = float
s = string
c = complex
b = boolean
u = unsigned

reshape() method:

```
>a=array([1,2,3,4,5])
>a.reshape(2,2)
array([[1, 2],
       [4, 5]])
>a.ndim
1
>a
array([1,2,3,4,5])
```

```
>b=a.reshape(2,2)
>a.ndim, b.ndim
(1,2)
>x=array([1,2,3,4,5])
>y=x.reshape(2,2)
```

ValueError: cannot reshape array of size.

```
>a=array([[1,2,3,4],[6,7,8,9],[12,13,54,21]])
>a
array([[1,2,3,4],
       [6,7,8,9],
       [12,13,54,21]])
[10 11 12 13]
>a[1,2:4]
array([8,9])
>a[1:2,2:4]
array([[8,9]])
>a[0:3,2:4]
array([[3,4],
       [8,9],
       [54,21]])
>where(a%2==0)
(array([0,0,1,1,2,2],dtype=int64), array([1,3,0,
                                          2,0,2],dtype=int64))
>where(a==12)
(array([2],dtype=int64), array([2],dtype=int64))
```

Matrix operations using numpy:

```
>> a = array([[1,2],[3,4]])
>> b = array([[5,6],[7,8]])
>>add(a,b)
array([[6,8],
       [10,12]])
>> subtract(a,b)
array([[-4,-4],
       [-4,-4]])
>>multiply(a,b)
array([[5,12],
       [21,32]])
>>dot(a,b)
array([[19,22],
       [43,50]])
>> divide(a,b)
array([[0.2 ,0.33333333],
       [0.4285 7143, 0.5   ]])
>a.T
array([[1,3],
       [2,4]])
>a+b
array([[6,8],
       [10,12]])
```

```
>a-b
array([[-4,-4],
       [-4,-4]])
>a*b
array([[5,12],
       [21,32]])
>a/b
 array([[0.2       , 0.33333333],
        [0.42857143, 0.5       ]])
>a.trace()
 5
>a.diagonal()
array([1,4])

>linalg.matrix_rank(a)
2
>linalg.det(a)
-2.0000000004
>> linalg.eig(a)
(array([-0.37228132, 5.37228132]),
```

```
>> std(a)
 1.1180339887499895
>> linalg.inv(a)
array([[-2. , 1. ],
       [1.5 , -0.5]])
```

```
>> a=array([[2,1],
            [4,3]])
>sort(a)
 array([[1,2],
        [3,4]])
```

```
>sort(a, axis=0)
array([[2,1],
       [4,3]])
>sort(a, axis=-1)
array([[1,2],
       [3,4]])
```

How to convert numpy to list?

```
>a
array([[1,2],
       [3,4]])
>p=a.tolist()
>p
[[1,2],[3,4]]
```

How to convert list to numpy.

The simplest way to convert a Python list to a Numpy array is by using the 'numpy.array()' function.

29/4

Collections module in Python:-

Collection module is used to develop data structure related operations in python.

→ Most of the collection classes are derived from dictionary.

List of collection module function:

1. counters
2. Ordered Dict
3. Default Dict
4. ChainMap
5. deque

1. Counter(): This function is used to count no. of occurances [in a given list or string] of a character or integer from a given sequence.

→ from collections import *

```
S='welcome'
print (Counter(S))
```

o/p: counter {'e':

2. OrderedDict: This Dictionary same as with normal dictionary but it follows the insertion order.

eg: from collections import *

```
d = OrderedDict()
d['a'] = 10
d['b'] = 21
d['c'] = 77
print(d)
Print(type(d))
```

o/p:

3. Default Dict:

It is used to create Dictionary keys with default data type and this default dictionary doesn't throw any exception when key is not found.

ex:

```
from collection import *
d = defaultdict(int)
d[3] = 'a'
d[4] = 'b'
print(d[8])   # zero
print(d)  # all values from dict
print(d['it'])  # blank output.
```

2) from collections import *

```
d = defaultdict(int)
m = [1,2,1,2,3,2]
for i in m:
    d[i] += 1
print(d)
```

o/p defaultdict(<class 'int'>, {1: 2, 2: 3, 3: 1})

4. ChainMap: It is used to merge multiple dictionaries into single dictionary.

ex:

```
from collections import *
d1 = {1:'a', 2:'b', 3:'c'}
d2 = {7:'z', 9:'x', 10:'y'}
d3 = {55:'p'}
c = chainMap(d1,d2,d3)
print(c)
print(list(c.keys()))
print(list(c.values()))
```

ChainMap({1:'a', 2:'b',

5. dequeue():

deque is a q

insert and delete elements at both end points.

Ex:

```
from collections import *
d= dequeue ( [3, 4, 5, 6, 7])
d.append (12) #adding an element at end
print (d)
d.appendleft(33) # adding an element at start
print(d)
d.pop() #removing an element at end
print(d)
d.popleft()#removing an element at start.
print (d)
```

3014

Exception handling and file Handling (Unit-4)

Exception: Exception is an error raised during the execution of a program.

To handle these exceptions python contains following keywords..

1. try
2. except
3. else
4. raise
5. finally

try & syntax: This try block contains exception code. whenever an exception is raised in try block, it throw to the corresponding except block.

```
try:
    #error code
    #error code
```

2) Except block: Except block is like a catch block in java. It is used to catch the exceptions thrown by the try block.

Syntax-1:

```
except except_classname as obj:
    #statements
    #statements.
```

Syntax-2: except (except_class1, except_class2..) as obj:

```
    #statements
    #statements.
```

Syntax-3: except.:

```
    #statements
    #statements.
```

Advantages of Exception handling:

1. To avoid abnormal termination of the program.
2. Exception code is seperated in one block and normal code is seperated in another block.

why python is robus? - It contains builtin handling mechanism.

Else block: Else block is executed whenever try block doesn't throw any exception. Else block is always placed after the except block.

syntax:

```
else:
    #statements
    #statements.
```

4) raise:

1) used to raise an exception explicitly.
2) used to handle user defined exception.
3) to re-raise an exception in nested try block.

Syntax1: raise exception_class_name ('error-statement')

syntax 2: raise obj;

5) finally block: It is an optional block and it is mentioned at the end of the all exception blocks. Used to release un-used resources in a program (file closing, data base closing and garbage collection).

Syntax:
```
finally:
    #statements
    #statements.
```

Note:
1. A python program contains any no.of except block and it contains only one try block.
2. Try block definitely requires either except block or finally block.
3. for all exception handling programs follows the sequence:
   try → except → else → finally (or) try → finally.

2/5/24

1. write a pp to handle index error for the given string.

```
st = input("enter string:")
i= int(input("enter index to display character:"))
try:
    print("character is:", st[i])
except IndexError as e:
    print("error...", e)
else:
    print("else is executed...")
```

O/P:

```
Enter string: srgec
enter index... : 2
character is : g
else is executed.
```

```
Enter string : gec
enter index... : 20
error... string index out of range.
```

2. Write a pp to handle keyError..

```
d = {1:'a', 10:'b', 3:'c'}
k=int(input("Enter key"))
try:
    print("Dictionary value is:", d[k])
except keyError as e:
    print("key not exist ...", e)
else:
    print("Else block...")
finally:
    print("finally block... ")
```

3. write a pp to handle overFlow-Error

```
p=int(input("enter power value"))
try:
    import math
    print("Exponent raising power values is:", math.e**p)
except OverFlowError as e:
    print("Too many precisions ...", e)
else:
    print("Else ...")
finally:
    print("finally ....")
```

o/p: enter power value: 2000
Too many precisions....

4. How to handle NameError:

```
a=int(input("enter first integer:"))
b=int(input("enter second integer:"))
try:
    if a!=0 and b!=0:
        print("Division is:", a/b)
    else:
        print("Addition is:", a+b+c)
except NameError as e:
    print("variable not declared...", e)
else:
    print("else...")
finally:
    print("finally block...")
```

5. How to handle ModuleNotFoundError.

```
P=int(input(" Enteor power value!"))
try:
    import maths
    print("Exponent raising poweor value is:", maths.eff,
except ModuleNotfound Error as e:
    print("Invalid Module ...", e)
else:
    print("Else ..")
finally:
    print("finally...")
```

maths is an invalid module. So, it throws an exception

6. write a pp to handle multiple exceptions using multiple except blocks.

```
try:
    a=int(input("enteor first integeor")) #value Error
    b=int(input("enteor second integeor")) #value Error
    print("Division is:", a/b) # zeroDivisionError.
    print(k)
    d={12:'IT'}
    k=int(input("Enteor key:")) # key Error.
    print("key value is:", d[k])
except ValueError as e:
    print("wrong input ...", e)
except KeyError as e:
    print("key not found error ...", e)
except zeroDivisionError as e:
    print("Divide Error...", e)
else:
    print("Else Executed...")
finally:
    print("(finally) End of the program..")
```

3/5/24

File Handling in Python:

To store data permanently in H.D we use file handling

File is a collection of related records.

Generally, there are two types of files in python.

1. text files
2. Binary files

1. Text files: In text files the data is stored in ascii format. This data is easily understand by the user. In ascii format. Every character is represented in 8-bits format.

The line end contains '\n'. central character of file EOF " " " string \0

2. Binary files:

The information in the binary files is stored in the form of sequence of bits. This data is not readable by the humans. Only machines can understand this data.

Basic file operations in python:

1. create or open
2. read or write
3. close

1. create or open:

By using open() method file is created in python.

Syntax: obj = open("path of file", "mode")

List of file opening modes:

1) r - read mode
2) w - write mode (overwrite)
3) a - appen mode (joinladd)
4) r+ = read/write
5) w+ = read/write
6) a+ = read/write (in append mode)
7)

7) rb - read binary
8) wb - write binary
9) ab - append binary
10) rb+ -
11) wb+ - } binary in read and write mode.
12) ab+ -

2) Read/write operations:

How to perform read operations in python?

i) read()
ii) readline()
iii) readlines()

write operations in python:

i) write()
ii) writelines()

3) close():

After finishing the read/write operations file must be closed by using close() function.

obj.close():

1. write a pp to store student information into text file.

File extensions:

- .txt - text --> notepad
- .doc or .docx - msword - document or document extension
- .xls or .xlsx - excel sheet --> msexcel
- .ppt or .pptx --> power point or power point extension --> ms power point
- .c --> c language
- .java --> java "
- .py --> python "
- .cpp --> c++ "
- .js --> java script
- .html or .htm --> hypertext markup language

```
no=int(input("enter roll number"))
name=input("enter name of the student:")
address=input("enter address of the student")
address=input("enter file with path and extension")
 path      name
try:
    f=open(path,'a')
    f.write("Roll number is:"+str(no))
    f.write("Name is:"+name)
    f.write("Address is:"+address)
    print("Data is successfully stored in file...")
    f.close()
except FileNotFoundError as e:
    print("file path not exist...",e)
```

write():

This function is used to write data in the form of string.

syntax: obj.write(st)

writelines() method:

This function is used to write data in the form of list into the file.

syntax: obj.writelines(list-obj)

2. write pp to store student information into text file using write lines.

```
no= int(input("enter roll number"))
name= input("enter name of the student")
address= input("enter file with path and extension")
                address of the student
path= input("enter file name with path and extension")
m=[str(no), name, address]
try:
    f=open(path,'a')
    f.writelines(m)
    print("Data is successfully stored in the file...")
    f.close()
except FileNotFoundError as e:
    print("File path not exist",e)
```

File read functions:

i) read() function:

It is used to read entire content from the file.

read() function is having two different syntaxes.

syntax1: fileobj.read()

syntax 2: fileobj.read(n)

Example:

write a pp to read the contents of text file.

Input:

eng.text - notepad

```
IT   CSE   ECE
EEE  IOT   AIML
12   05    04
CE   ME    AI&DS
```

→ Now open IDLE open new file.

```
fname = input('enter file name with extension:')
try:
    f=open(fname,'r')
    print("file content is:")
    print(f.read())
except FileNotFoundError as e:
    print("file error--", e)
```

o/p: Enter file name with extension: d:\srgec.text

file content is

```
IT   CSE   ECE
EEE  IOT   AIML
12   05    04
CE   ME    AI&DS
```

(or) print(f.read(10))

o/p: IT CSE ECE

o/p: >>IT CSE ECE

>>>

>>>

>print(f.read()[::-1])

o/p:

print(f.read()[2:20:2])

o/p:

ii) readline() function:

→ It is used to read single line at a time.

syntax: fileobj.readline()

```
fname =
try:
    f= open(f
    print
    print(f.readline())
except fileNotfound Error as e:
    print("file error...", e)
```

o/p: IT CSE ECE
>>

How to find file size in python?

print(len(f.read()))

iii) readlines() function:

```
fname=
try:
    f
    P
    print(f.readlines())    print(len(f.readlines()))
except fileNotfound Error as e:
    print("file error...", e)
```

o/p: ['IT CSE ECE \n', 'EEE IOT AIML\n', '12 05 04\n', 'CE ME AIDS\n', '\n', '\n', '\n']

→ If we want to copy data from one file to another file.

```
f1  r
f2  a
f2.write(f1.read())
```

→ write a PP to copy contents from one file to another file.

```
file1 = input("enter existing file name to copy:")
file2 = input("Enter new file to copy")
try:
    file1 = open(file1, 'r')
    f2 = open(file2, 'a')
    f2.write(f1.read())
```

```
        print("file copied ...."); f1.close(), f2.close()
except  FileNotFoundError as e:
        print("file error-- ", e)
```

→ write a pp to copy no.of lines, no.of words, no.of characters from given file.

```
file1=input('enter file name with extension')
w=c=l=0
try:
    f=open(file1,"r")
    for i in f.readlines():
        l=l+1
        for j in i.split():
            w=w+1
            c=c+len(j)
    print("lines count is:",l)
    print("word count is:", w)
    print("character count is:", c)
except FileNotFoundException as e:
    print("file error---",e")
```

each line reverse

```
print(i[::-1])
```

```
for j in i.split():
    print(j[::-1])
```

words reverse

role of 'with' keyword in files:

This block is used to open a file and at the end of block file is automatically closed without writing close() function.

Syntax:

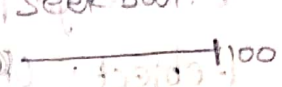
```
with open("path","mode") as obj:
        #statements
        #statements.
```

```
fname = input("enter file name with extension")
try:
    with open(fname, 'r') as f:
        print(f.read())
except FileNotFoundError as e:
    print("File Error..", e)
```

Random Access files in Python:

Random Access (files) contains the following file methods.

1) seek()
2) tell()



1) seek(): This function is used to move file pointer to given position. syntax: fileobj.seek(offset, whence)

whence
0 - from starting position of file
1 - from current position of file
2 - from end position

ex: f.seek(10, 0)
f.seek(5, 1)
f.seek(-3, 2)

2) tell(): display the current position of a file pointer.
fileobj.tell()

→
```
fname = input("enter file name with extension")
try:
    f = open(fname, 'r')
    f.seek(10)
    print(f.read())
    print("current file pointer is:", f.tell())
    f.close()
except FileNotFoundError as e:
    print("File Error...", e)
```