[sort, max, sorted ....] → built in namespaces in python.

2) Global namespaces: (global variables)

Global namespaces are available

global variables comes under global namespaces category

By using 'globals()' function display global namespaces.

Ex:- a=10
b=20
s="gec"
globals()['a']
o/p:- 10
globals()['s']
o/p:- 'gec'

3) Enclosing namespaces:- These are available in nested functions.

Eg:- def add(a,b):
x=12 //enclosing namespace
y=20 // enclosing namespace
def sub(p,q):
return p+q+x+y
return sub

4) local namespaces:- These are specified within a function & by using locals() function display local namespaces

Eg:- def display():
no = 1200
name = "john"
address = "US"
} → local namespaces.
print(locals())
display()

o/p:- {no:1200, name:'john', address:'US'}

* Tuples :-

(UNIT - 3 Topic).

Q:- what is tuple ? How to change and delete a tuple

```
t = tuple(i for i in input().split()   # tuple comprehension
print(t)
```

Ex:- 78 8 9 5 6

o/p:- ('78', '8', '9', '5', '6')

```
⇒ t=tuple(i for i in input().split())
  print(t)
  m = list(t)
  print(m)
```

} converting tuple into list

⇒ delete operations on tuple

```
t = tuple(int(i) for i in input().split())
print(t)
m=list(t)
m.pop()  #delete operation
print(tuple(m))  #after deletion convert list into tuple.
```

Note:- It is not possible to update or delete existing values in tuple because by default tuple is immutable. So for the required change (update) and delete operations, we need to convert the tuple into list and apply operations. At last finally convert the updated list into tuple.

```
→ m[1]=100  #updating a tuple
  print(tuple(m))
```

→ write a python program to read list of names in the form of list. Display the given list of names in uppercase with its length.

Program:-

```
n=int(input('enter an integer'))
(or)
m=[i for i in input().split()
for i in m:
    print(i.upper(),'->',len(i))
```

O/P:- ECE IT CSE

ECE →3

IT - 2

CSE →3

→ write a python program to count no. of alphabets, digits, special symbols, spaces in a given string.

→ program:-

```
s=input('enter string')
a=d=s1=s2=0
for i in s:
    if i.isalpha():
        a=a+1
    elif i.isdigit():
        d=d+1
    elif i.isspace():
        s1=s1+1
    else
        s2=s2+1
print("alphabets count",a)
print("digits count",d)
print("spaces count",s1)
print("Special symbols count",s2)
```

⇒ n=3 → Query

3+33+333

n=4

4+44+444+4444

Program:-

```
n=int(input('enter an integer'))
for i in range(1,n+1):
```

```
        s = s + int(n*i)
    print(s)
```

(or)

```
n = input('enter digit')
k = int(n)
s = 0
for i in range(1, k+1):
    s = s + int(n*i)
print(s)
```

→ 3 + 33 + 333 = 369

Program:-

```
n = input('enter digit')
k = int(n)
s = 0
p = []
for i in range(1, k+1):
    p.append(n*i)
    s = s + int(n*i)
print('+'.join(p), '=', s)
```

o/p:- enter digit 3
3+33+333 = 369

→ n = 5

1/1 + 1/2 + 1/3 + 1/4 + 1/5 =

Program:-

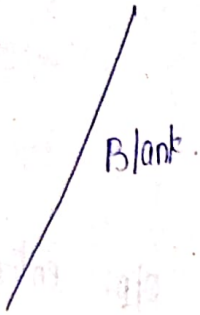
```
n = int(input('enter digit'))
s = 0
m = []
for i in range(1, n+1):
    z = '1/' + str(i)
    m.append(z)
    s = s + 1/i
print('+'.join(m), '=', s)
```

⇒ write a python program to count prime numbers from the given list of elements m = [2, 3, 7, 4, 8, 10]

Program:-

```
m = [int(i) for i in input().split()]
c = 0
for n in m:
    c = 0
    for i in range(1, n+1):
        if n%i == 0:
            c = c + 1
```

```
if c==2:
    c1=c1+1
print(c1)
```



- Modules

→ In python default type of return function is "tuple"

→ write a python program to find out sum of squares given 4 integers.

s=0

Program:-
```
def Sum(a,b,c,d):
    s=a**2+b**2+c**2+d**2
    return s
a,b,c,d = int(input()), int(input()), int(input()), int(input())
print(sum(a,b,c,d))
```

↓

Convert this program in the form of default arguments.

```
def sum(a=1, b=2, c=3, d=4):
    return a**2+b**2+c**2+d**2
a,b,c,d = int(input()), int(input()), int(input()), int(input())
print(Sum(a,b,c,d))
```

print(sum()) → it operates by giving default values

→ write a python program to swap given two values using function default arguments.

```
def swap(a=3,b=2):
    a,b = b,a
    return a,b
a,b = int(input()), int(input())
print(swap(a,b))
print(swap())
print(swap(1))
print(swap(5,6))
```

→ write a python program to find out the value b^2-4ac using functions with default arguments.

```
def quad(a=3, b=2, c=1):
    return b**b-(4**a**c)
a, b, c = int(input()), int(input()), int(input())
print(quad(a,b,c));
print(quad());
print(quad(1));
print(quad(4,8));
print(quad(2,5,6));
```

Set Data Structure

Set also contains sequence of elements with the following properties:

i, duplicates are not allowed into set.

ii, doesn't preserve any order.

iii, set enclosed within curly braces { }

iv, an empty set is represented by set()

⇒ List of methods:

1. add()
2. update()
3. clear()
4. pop()
5. discard()
6. remove()
7. del()
8. intersection()
9. union()
10. difference()
11. min(), max(), sum(), len(), sort(), sorted(), copy(), in, not in, any all, count()

Note:- sets does not allow slicing and no indexes for set items.

V. Imp

→ write a python program to find out reverse of a given string using double slicing.

```
str = input('enter steing')
print("Reverse is ", str[::-1])
```

O/p:- enter steing gec
Reverse is ceg

→ write a python program to check whether given steing is palindrome or not.

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

→ write a python program to check given list of strings are palindrome or not.

```
def palindrome(str):
    if str == str[::-1]:
        print('palindrome')
    else:
        print('not a palindrome')
[i for i in map(palindrome, input().split())]
```

→ write a python program to print each & every word/steing from the given list of steinge.

```
def palindrome(str):
    return ste[::-1]
n = [i for i in map(reverse, input().split())]
print(m)
```

→ write a python program to find out sum of given list of elements by using functions.

```
def add(n):
    return sum(n)
m=[i for i in map(int, input('Enter integer seperated by space').split())]
print('sum is', add(m))
```

* map contains (function name, sequence) → parameters.
* we do not apply more than one map function at a time.

→ Develop a python code to display binary values for given list of elements.

```
def binary(n):
    return bin(n)
m=[i for i in map(bin, input('Enter integer').split())]
print('binary number is', bin(m))
```

(or)

```
print(list(map(bin,[i for i in map(int,input().split())])))
```

Set :- It is a mutable type.

```
v={3,4,5}      s={1,2,3,100,4}
```

update:

```
s.update(v)
{1,2,3,100,4,5}
```

add():

↓

To add elements in the set.

2) delete operations:-

1) clear(): used to clear all the elements in the set

Ex:-
```
s={3,4,5}
s.clear()
o/p:- set()
```

2) pop() :- It returns / deletes first element from the set

Ex:- s = {2, 3, 5}

s.pop() (2) is deleted

o/p:- {2, 3}

Whereas in list, pop() refers to delete last element from the list

Ex:- s.pop(1) → Set do not accepts Arguments.

o/p:- TypeError

3) discard()

s = {3, 2, 1, 5, 56}

s.discard(1)

s

o/p:- {2, 3, 5, 56}

s = {3, 2, 1, 5, 56}

s.discard(4)

o/p:- {3, 2, 1, 5, 56} → if no element is present in given index, it returns no error & no changes in the set

4) remove :-

s = {1, 2, 3, 4, 5}

s.remove(3)

s

o/p:- {1, 2, 4, 5}

s.remove(10) → if element is not present, then it returns error

o/p:- KeyError

5) del() :-

s = {3, 4, 3}

del s

s

⇒ NameError :- s is not defined since s is deleted from the memory

6) Intersection() :- This function is used to display the common elements from the two or more sets.

Ex:- s1 = {1, 2, 3, 4}

s2 = {1, 3, 5}

s1.intersection(s2)

o/p :- {1, 3}

v1 = {2, 3}

v2 = {5, 6}

v1.intersection(v2)

o/p:- set()

→ It is applicable for strings also.

7) union():-

Ex:- $S_1 = \{1,2,3,4\}$

$S_2 = \{2,5,6,7\}$.

S_1.union(S_2)

O/p:- {1,2,3,4,5,6,7}

8) difference():- It is used to delete the

ex:- $S_1 = \{1,2,3,4\}$

$S_2 = \{1,3,5\}$

S_1.difference(S_2)	S_2.difference(S_1)
O/p:- {2,4}	O/p:- {5}

9) copy():- It is used to copy the existing values into other.

$S_1 = \{1,2,3,4\}$

$K = S_1.copy()$

K

O/p:- {1,2,3,4}

* dictionaries:-

dictionary also contains sequence of elements each and every contains pair of key & value. The key and value are seperated with colon(:). The dictionary elements are enclosed b/w { }. for representing empty dictionary d={}.

ex:- d={1:'IT', 2:'CSE', 3:'ECE'}

Note:-

1) Both keys and values allows any type of data type.

2) Conversion from list to dictionary:-

K=[1,2,3,4]

V=['IT','CSE','ECE','EEE']

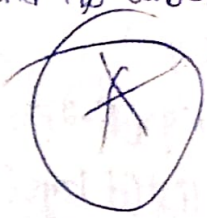
dict(zip(K,V))

O/p:- {1:'IT', 2:'CSE', 3:'ECE', 4:'EEE'}

list
dict(zip(k,v))

o/p: [(1,'IT'),(2,'cse'),(3,'ece'),(4,'eee')]

=> write a python program ~~that uses dictionary to return the name~~ to display sequence of members as keys and its cubes are its values.



```
n=int(input('enter n'))
d={i:i**3 for i in range(1,n+1)}
print(d)
```

o/p- enter n 3

{1:1, 2:8, 3:27}

=> Dictionaries does not contain indexing, so it does not support slicing.

Functions:-

1) ~~Set~~ get()
2) keys()
3) values()
4) items()
5) pop()
6) popitem()
7) del()
8) clear()
9) copy()
10) update()
11) setdefault()
12) fromkeys.
13) sum(), min(), in, any, all, sorted ....

Ex- d={1:'IT', 2:'cse', 3:'eee'}

d.get(1)	d.get(5)	d[1]	d[5]
o/p:- 'IT'	o/p:- Blank o/p.	o/p:- 'IT'	key Error.

Update:-

```
x = {4: 'iot'}
d.update(x)
o/p:- {1:'π', 2: 'cse', 3: 'ece', 4: 'eee'}.
>d.keys()
dict_keys([1,2,3,4,5,7])
>v = list(d.keys())
>v
o/p:- [1,2,3,4,5,7]
>d.values()
dict_values (['mE','cse','eee','ece'])
>d.items()
dict_items ([(1,'mE'),(2,'cse'),(3,'eee'),(4,'ece')])
```

⇒ Given the following dictionary:

```
inventory = {
'gold': 500,
'pouch': ['flint', 'twine', 'gemstone'],
backpack = ['xylophone','dagger','bedroll','breadloaf']
}
```

Try to do the followings:

i) Add a key to inventory called 'pocket'.

ii) Set the value of 'pocket' to be a list Consisting of the strings 'seaxe' and 'iron'.

iii) sort the items in the list.

⇒ Inventory = {'gold': 500, 'pouch': ['flint', 'twine', 'gemstone

⇒ write a python program to interchange values as keys & keys as 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(d)
d1 = {y:x for x,y in d.items()}
print(d1).
```

Methods:-

⇒ pop():- In dictionary pop() contains atleast one argument, empty pop() returns an Error & also returns Error if we give element/index that is not present in dictionary

Ex:- d = {1:'it', 2:'cse', 3:'eee'}

d.pop(1)

O/P:- d

{2:'cse', 3:'eee'}.

⇒ popitem():- It is used to delete last element from the dictionary

Ex:- d = {1:2, 2:3, 3:4, 4:5, 5:6} → through popitem() it is deleted.

d.popitem()

O/P:- {1:2, 2:3, 3:4, 4:5}

=> clear :- It is used to clear the dictionary totally.

Ex:- d.clear()
d
o/p:- { }

=> del() :- used to delete element from the dictionary

Ex:- d = {1:'a', 2:'b', 3:'c', 4:'d'}
del[1]
d
o/p:- {2:'b', 3:'c', 4:'d'}

=> del[5]
keyerror

=> del d → completely removed the reference of element d
d not defined.

=> del d[] → invalid syntax

=> copy() :- It is used to create exact copy of reference object

Ex:- d = {1:'a', 2:'b', 3:'c'}
c = d.copy()
c
o/p:- {1:'a', 2:'b', 3:'c'}.

> d.setdefault(6,'No value')
'No value'

setdefault method is used to add a key value pair (default) to existing dictionary.

=> d
{1:'a', 2:'b', 3:'c', 6:'No value'}

> d[6]
'No value'

> d.get(8, d[6]) (no key in dictionary → default value)
'No value'

d.get(10, d[6])
'No value'.

⇒ fromkeys(): This function is used to combine two lists or tuples into dictionary.

Ex:- t1 = (1,2,3)
t2 = ('IT')
dict.fromkeys(t1,t2)

O/P:- {1:'IT', 2:'IT', 3:'IT'}

>dict.fromkeys(t2,t1)

O/P:- {'I':(1,2,3), 'T':(1,2,3)}

>t1=(1,2,3)
>t2=('a','b','c','d','e')
>dict(zip(t2,t1))
o/p:- {'a':1, 'b':2, 'c':3}
>dict(zip(t1,t2))
o/p:- {1:'a', 2:'b', 3:'c'}

(*) ⇒ write a python program to sort dictionary elements by using keys.

```
k=[i for i in map(int,input('Enter key values').split())]
v=[i for i in input('Enter values').split()]
d=dict(zip(k,v))
s=sorted(d)
d1={}
for i in s:
    d1[i]=d[i]
print(d1)
```

O/P:-
Enter key values: 4 1 2
Enter values: a b c.

O/P:-
{1:'b', 2:'c', 4:'a'}

⇒ write a python program to display the following factors using dictionaries for n=5

```
*
* *
* * *
* * * *
* * * * *
```

Program:-
```
n = int(input('Enter an integer'))
d = {i:i*'*' for i in range(1,n+1)}
for i in d.keys():
    print(d[i])
```

O/p:- Enter an integer 5
```
*
* *
* * *
* * * *
* * * * *
```

⇒ write a python program to sort dictionary elements by using values

Sample input d = {1:'xyz', 2:'cd', 3:'ab'}

Output:- d = {3:'ab', 2:'cd', 1:'xyz'}

Program:
```
k = [i for i in map(int, input('Enter key values').split())]
v = [i for i in input('Enter values').split()]
d = dict(zip(k,v))
print(dict(sorted(d.items(), key = lambda x:x[1])))
```

Sorting occurs based on values through lambda function

(or)

```
k = [i for i in map(int, input('Enter key values').split())]
v = [i for i in input('Enter values').split()]
d = dict(zip(k,v))
```

```
d1 = {}
for i in sorted(d, key=d.get):
    d1[i] = d[i]
print(d1).
```

⇒ write a python program to accesr dictionary keys element by index.

```
k = [i for i in map(int, input('Enter key values').split())]
v = [i for i in input('Enter values').split()]
d = dict(zip(k,v))
for i in d:
    print(d[i])
```

⇒ How to access dictionary elements:-

(or)

looping over dictionary elements

1) using key()
2) using values()
3) using items()
4) using dictionary name
5) using get() method.

⇒ 1) using keys:-

```
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))
for i in d.keys():
    print(i).
```

o/p:- Enter keys 1 2 3
Enter values 'a', 'b', 'c'
o/p:- 1
2
3

⇒ 2) using values().

```
for i in d.values():
    print(i)
```

o/p:- Enter keys 1 2 3
Enter values a b c
o/p:- 'a'
'b'
'c'.

3) using items():-

```
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))
for x,y in d.items():
    print(x,'->',y)
```

4)
```
for i in d:
    print(d[i])
```

5)
```
for i in d:
    print(d.get(i))
```

⇒ Collections in python:-

Collection is a container to process list of elements with different functions and return the result. Every container holds or stores list of elements as input. The collection module contains the following methods:

1. Counter()
2. OrderedDict()
3. defaultdict()
4. ChainMap()
5. deque()

1) Counter(): this function is used to count number of occurences of elements from given sequence and sorted in dictionary. → counts how many time the steing is occurred including spaces.

Ex:-
```
from collections import *
print(Counter('welcome'))
```
O/P:- Counter({'e':2,'w':1,'l':1,'c':1,'o':1,'m':1})

2) OrderedDict(): It is a sub class of dictionary, It preserves keys in the given order.

```
from collections import *
d = OrderedDict() # collections dictionary
d1 = {} #normal dictionary.
d[1] = 'cse'
d[0] = 'IT'
d[2] = 'ece'
d[9] = 'aiml'
d[3] = 'iot'
Print(d)
print(d1)
```

o/p:- {'0':'IT', '2':'ece'}
OrderedDict([('9':'aiml'), '3', 'iot'])

→ defaultdictdict():

Eg:-

```
from collections import *
m = [10, 20, 35]
d = defaultdict(int)
for i in m:
    d[i] += 1
print(d)
```

diff → input → list

d[i] += 1 → aliasing

like a counter only diff in incrementing value

→ This function is used to provide default values for a dictionary.

→ chainmap():

using this function can combine two or more dictionaries into single dictionary.

→ Eg:-

```
from collections import *
d1 = {1:'a', 2:'b'}
d2 = {10:'c', 11:'d', 34:'v'}
d3 = {'it':12, 'cse':5}
print(chainmap(d1, d2, d3))
```

→ deque():-

→ It is doubleended queue:-
→ In deque both insertions, deletions performed two e. Points:

Eg:-
```
from collections import *
d = deque([34, 21, 1, 17, 89])
d.append(100)  #inserts at end of list.
d.appendleft(800) #this function inserts an element at beginning
print(d)
d.pop() # delete element at end point
print(d)
d.popleft() # delete element at beginning
print(d)
```

→ Slicing in python:-

Slicing is the process of accessing part of a sequence.

→ Generally python contains 2 types of slicing

1) single slicing
2) double slicing

The following data structures allow slicing mechanism

1) string 2) list 3) tuple

1) Single slicing

syntax:-

```
seqname[start:stop]
```

Eg:- m = [10, 20, 30, 14, 25]

```
m[0:1]
```

o/p:- [10, 20, ~~30~~ 15, ~~40~~, 25]

```
m[2:]
```

o/p:- [~~30~~ 15—40, ~~14~~, 25]

→ m[10:]

o/p :- []

→ m[:1]

o/p :- [10]

→ m[:3]

[10, 20, 15]

→ m[0:5]

[10, 20, 15, 40, 25]

→ m[:]

o/p :- [10, 20, 15, 40, 25]

→ m[1:1]

o/p :- [ ]

→ m[1:3]

o/p :- [20, 15]

Start = Stop o/p :- Blank

→ m[6:7]

o/p :- [ ]

→ m[7:7]

o/p :- [ ]

→ m[-5:-2]

o/p :- [10, 20, 15]

→ m[-10:]

o/p :- [10, 20, 15, 40, 25]

→ m[4:4]

o/p :- [ ]

→ m[-10.0:]

o/p :- [10, 20, 15, 40, 25]

→ m[:-1]

o/p :- [10, 20, 15, 40]

→ m[:-3]

o/p :- [10, 20]

→ m[-4:]

[20, 15, 40, 25]

→ m[-3:-1]

o/p :- [15, 40]

→ m[-1:-5]

o/p :- [ ]

→ m[-1:-1]

o/p :- [ ]

⇒ how to access tuple elements in python

1) using slicing
2) using indexes : +ve & -ve
3) using loops.

* tuple insertions, deletions are not possible.

set & dictionary :-

a) loops

⇒ Slicing :-

1) m = 'welcome'
   m[:5]
   o/p :- welco'

2) t = (1, 2, 3, "hello")
   t[3][:3]
   o/p :- hel'

* repetition operator (*) is applicable for all i.e list, string, set, tuple, ...

* Double Slicing :-

Syntax :- seq[start : stop : step]

* only positive indexes :-

Ex :- m = [10, 45, 20, 30, 5, 11]

m[0::] ⇒ m[0:6:] ⇒ m[0:6:1] → default step = 1

o/p : [10, 45, 20, 30, 5, 11] → default stop = 6

m[::] ⇒ m

o/p :- [10, 45, 20, 30, 5, 11]

m[:3:] → here default start = 0, step = 1.

o/p :- [10, 45, 20]

m[100::] * It do not throws error.

o/p :- [] → It gives how many are present before hundred and leave

m[:100:] step = 1

o/p :- [10, 45, 20, 30, 5, 11]

m[::1]

[10, 45, 20, 30, 5, 11]

m[::2]

[10, 20, 5]

m[::5]
o/p:- [10, 11]

m[::10]
o/p:- [10]

here, start > stop
not possible

m[1::3]
o/p:- [45, 5]

m[1:3:3]
o/p:- [45]

m[5:2:2]
o/p:- []

↳ Negative indices

m[-1::]
o/p:- [11]

m[-3:0:]
o/p:- []

m[-3::]
o/p:- [30, 5, 11]

m[-3:0:-1]
o/p:- [30, 20, 45]

m[:-3:] → m[-6:-3:]
o/p:- [10, 45, 20]

UNIT - IV Exception Handling & File Handling:

* Exception is also a type of error, but exception defines in run time

Exception keywords

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

Ex:

1)
```
try:
    print(10/0)
except ZeroDivisionError:
    print("error")
else:
    print("it is else block")
```

* If try fails, else do not executes.
if try dosent throws any exception, then else executes.

* List of Exceptions in Python:

1) ZeroDivisionError
2) IndexError
3) ModuleNotFoundError
4) NameError
5) ValueError.
6) TypeError
7) IndentationError
8) EOFError.
9) OverflowError
10) SyntaxError.
11) AttributeError.

unit-3

⇒numpy : numeric / numerical python.

⇒numpy → type of list and it is more efficient than list.

→ normal list follows double linked list whereas numpy memory is stored in continuous locations (spread throughout memory)

→ In numpy predefined functions are available

→ array based operations are more efficient with numpy

⇒ numpy is known as numerical python module to perform array based operations. So numpy elements are stored in continuous / consecutive memory locations.

→ In Double linked list (list) elements are stored in different memory locations. (→ normal list)

→ Numpy is 50 times faster than the list

Installing numpy module

1) Open Command prompt
2) pip install numpy

⇒How to create array in numpy:

by using array() method create arrays in numpy.

1. zero dimensional arrays
2. one dimensional arrays
3. two-dimensional arrays
4. Multi-dimensional arrays.

*1) Zero dimensional array:-

```
a = array(10)
```

⇒from numpy import *

```
a = array(10)
a.ndim
```

→ ndim is a property to display dimension of numpy

o/p: 0

2) One dimensional array:-

ex:- a = array([1,2,3,'IT'])

o/p:- array(['1', '2', '3', 'IT'],

> a.array([1,2,3], dtype = 'f')

o/p:- array([1., 2., 3.], dtype = float32)

3) Two dimensional arrays:-

ex:- a = array([[1,2,3],[4,5,6],[7,8,9]])

o/p:- ([1,2,3],
[4,5,6],
[7,8,9]])

⇒ matrix operations:-

1. add()
2. subtract()
3. multiply()
4. dot()
5. divide()
6. trace
7. diagonal()
8. T → Transpose.
9. linalg.matrix_rank()
10. linalg.inv()
11. linalg.det()
12. linalg.eig() → eigen values

1) add(): used to find addition of two matrices

Program:-

```
from numpy import *
a = array([1,2,3],[4,5,6][7,8,9]])
b = array([3,3,5], [8,3,4],[1,4,2]])
print("Addition is:\n", add(a,b))
```

o/p:- [[3,5,8]
[12,8,10]
[8,12,11]].

print("subteaction is: \n", subteact(a,b))
print("multiplication is: \n", multiply(a,b))
print("Dot product is: \n", dot(a,b))
print("Division is: \n", divide(a,b))
print("Trace of array a \n", teace(a))
print("Diagonal elements of a \n", diagonal(a))
print("Transpose of a is: \n", a.T);

linearalgebra methods:

print("det of matrix A is: \n", linalg.det(a))
print("Inverse of matrix B is: \n", linalg.inv(b))
print("Rank of matrix B is: \n", linalg.matrix_rank(b))
print("Eigen values & vectors of matrix B is: ", linalg.eig(b))

Multi-dimensional arrays:-

Ex:- a = array([[[1,2],[3,4]],[1,2,3],[5,6,7]])

b = a.reshape(2,2,2)

a: array([[[1,2],[3,4]],[[5,6],[8,9]]])

⇒ Shape() & reshape() methods:

shape(): It is used to display number of rows & columns of a given array

Ex:- a = array(3,4)
a.shape
o/p:- () → output of shape() is tuple.

2) a = array([12, 3])
a.shape
o/p:- (2,)

3) a = array([[[1,2],[3,4]],[[5,6],[7,8]],[[9,10],[11,12]]])
a.shape
o/p:- (2,2,2)

⇒ reshape():- It is a type of array used to convert one dimension to other

Eg:- a = array([1,2,3,4])
a.reshape(2,2)

o/p array([[1,2],
[3,4]])

a.reshape(3,3) → value error.

a.reshape(4,1)

o/p:- array([[1],
[2],
[3],
[4]])

UNIT-4

Topic:-

try:
print(5/2)
except ZeroDivisionError:
print('error')
else:
print('it is else')

o/p:- 2.5
it is else

⇒ from math import *

try:
s='gec'
print(s[5])
except:
print('error')
else:
print('it is else')

o/p:- error.

*Lab Programs

15) program:-

```
from math import *
a = int(input("enter first integer"))
b = int(input("enter second integer"))
c = input('enter text')
d = int(input("enter value for sqrt"))
try:
    print("division is :" a/b)
    print("Character is :", c[5])
    print("Square root is :", sqrt(d))
except (ValueError, ZeroDivisionError, IndexError):
    print("Error ....")
else:
    print("No exception")
```

16) User defined Exceptions program:-

Program:-

```
class ShortInputException(Exception)
    def __init__(self, err):
        self.err = err
t = input("Enter text")
try:
    if len(t) <3 :
        raise ShortInputException("text length should be more")
    else:
        print("Given text is:",t)
except ShortInputException as e :
    print(' err',e)
else :
    print('no error')
finally:
    print("finally block executed")
```

Error:- The problems or faults occurred in program are called as error. All these errors are verified by the compiler at Compile time.

→ It is also called as bug. Bugs create disturbance to the program

→ Types of Errors:-

1) Syntax Error.
↓
missing of paranthesis, double quotes, arguments missing

2) semantic error:-
↓
spelling mistakes of pre-defined functions.

eg:- in C programming
print('hello')

3) logical error:- It is user's mistake for input logics.

ex:-
```
n=3
for(i=1;i<=n;i+1)
f=f*i
```

→ Exception:- Exception is also an error occured during the execution of a program (runtime).

Types of Exceptions:-

1) pre-defined exceptions
2) user-defined exceptions.

Pre-defined Exceptions:-

In python. pre-defined exceptions are derived from 'Exception' base class.

→ List of pre-defined exceptions:-

1) ZeroDivisionError
2) NameError
3) ValueError.
4) ModuleNotFoundError.
5) IndexError
6) KeyError
7) OverFlowError.
8) SyntaxError.
9) IndentationError.
10) TypeError.
11) AtteibuteError
12) EOFError.

2) User-defined Exceptions:- Exceptions are created and used by the user are called as User defined exceptions.

⇒ Exception Handling Mechanisms:-

python contains following keywords to handle exceptions:

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

⇒ Advantages of Exception handling:-

1) to avoid abnormal termination of a program.
2) to seperate error code with normal program code.

1) ⇒ try:- try block is used to specify error possible code.

Syntax:- try:
    #error statement1
    #error statement2
    :

→ If an error is generated with try block then control goes to the Corresponding except block.

→ If except block is not present in the program then it is handled by default handler. (try → finally) not else (in b/w)

⇒ except:- This block is used to handle errors which are thrown by try block

Syntax:- 1) except Exceptionclass:
    #statement1
    :

2) except Exceptionclass as obj:
    #statement1
    #statement2

3) except (exceptionclass1, exceptionclass2, ..):
    #statements

4) except :
#statements

Programs:-

⇒ Develop a python program to handle ZeroDivisionError.

```
a = int(input("Enter first integer"))
b = int(input("Enter Second integer"))
try:
    print("Division is: ", a/b)
except ZeroDivisionError as e:
    print("Error...", e)
else:
    print("No error")
```

⇒ write a python program to handle a IndexError → Indexes

#for the index error, we list, tuples, strings.

Program:-

```
m = [i for i in map(int, input().split())]
n = int(input("enter index to display list item"))
try:
    print("list elements is :", m[n])
except IndexError as e:
    print("Error...", e)
else:
    print("No Error")
```

Enter elements

⇒ write a python program to handle ValueError. → other value -ve/floating

Program:-

```
a = int(input('Enter an integer'))
try:
    print("Given integer is :", a)
except ValueError as e:
    print("error...", e)
else:
    print("no error...")
```

1) import math
try:
a = int(input("enter an integer")) -ve & float
print("Factorial is:", math.factorial(a)) # input = -3 or 2.3
except ValueError as e: throws value error.
print("error...", e)
else:
print("No error...")

→ TypeError:- Type → Datatype / Type of Parameters.
# to handle TypeError
a = int(input("enter number")) # input = 123, then it throws error for len(a)
try:
print(len(a))
except TypeError as e:
print("Error...", e)
else:
print("no error")

2) a = int(input("Enter number"))
b = input("Enter String")
try:
print(a+b) # concatenation of string with int throws Type ~~Value~~Error
except TypeError as e:
print("error...", e)
else:
print("Error...")

→ NameError → not declared one is given.
# to handle NameError.
a = int(input("enter a value"))
b = int(input("enter another value"))
try:
if a>b:
print(a)
else
print(c). # variable name c is not declared, so it throws NameError.
except: NameError as e:
print("Error...", e)

```
else:
    print("No error...")
```

o/p:- c is not defined → NameError.

⇒ KeyError:- for this error, we use dictionaries

#to handle KeyError. (Enter keys)

```
k=[i for i in map(int,input().split())]
```
(Enter values)
```
v=[i for i in input().split()]
d=dict(zip(k,v))
print("Given dictionary elements are:",d)
key=int(input('enter key to display dictionary element'))
try:
    print("value for the given key is:",d[key])
except KeyError as e:
    print("Error....",e)
else:
    print("No error...")
```

⇒ Handle Multiple exceptions using multi-except block:-

In python program only one try block is allowed and multiple no. of except blocks are allowed.

example program (to handle TypeError, ZeroDivisionError, IndexError)

```
m=input('Enter string')
try:
    a=int(input('enter an integer'))      #type error.
    b=int(input('enter another integer'))  #type error.
    print("Division is",a/b)               #divide error.
    print('char in given index',m[5])      #index error.
except IndexError as e:
    print("divide error",e)
except TypeError as e:
    print("Data type wrong...",e)
else:
    print("No error...").
```

o/p:- Enter String singed

=> write a python program to handle NameError, KeyError, ValueError with multiple except blocks.

~~import math~~
~~try:~~
~~a = int(input('Enter an integer~~

```
import math
```

=> to handle multiple exceptions in single except block

```
m = input('Enter String')
try:
    a = int(input('enter an integer'))
    b = int(input('enter another integer'))
    print('Division is', a/b)
    print('char in given index', m[5])
except: # it handles all exceptions or except(TypeError, IndexError, ZeroDivisionError)
    print("Error...")
else:
    print("No Error...")
```

* 'raise':-

Uses of raise keyword.

1) used to raise an exception explicitly } noted
2) to handle user defined exceptions
3) used to reraise an exception

→ except:- used to raise an exception explicitly

Ex:

```
try:
    raise ZeroDivisionError("error...") # explicit raising
except ZeroDivisionError as e:
    print(e)
```

⇒ write a python program to raise an exception when given text length less than 3 characters by using explicit raise.

Ex:

```
s=input("enter steing")
try:
    if len(s)<3:
        raise keyError("text length >=3")
    else:
        print("Given text is",s)
except keyError as e:
    print(e)
```

2) using user-defined exceptions:-

For creating user-defined exceptions ex

1) create a class with user-defined exception name
2) Inherit base class 'Exception'
3) Creat a constructor with error statement as argument.

noted

⇒ write a python program to display wrong grades using "InvalidGradeError" exception

Program:-

```
class InvalidGradeError(Exception):
    def __init__(self,e):  # constructor in python
        self.e=e
grade=input("Enter marks Grade")
m=['A','B','C','D','E','F','A']
try:
    if grade not in m:
        raise InvalidGradeError("Wrong grade...")
```

```
        else:
            print("Grade is valid:", grade)
    except InvalidGradeError as e:
        print("Error:", e)
    else:
        print("No error...")
```

(X) Write a python program to handle ~~negative number~~ NegativeNumberException while calculating factorial of given number.

```
class NegativeNumberException(Exception):
    def __init__(self, e)
        self.e = e
n = int(input("enter an integer"))
try:
    if n<0:
        raise NegativeNumberException("Number is negative")
    else:
        print("Factorial is:", factorial(n))
except NegativeNumberException as e:
    print(e)
else:
    print("No Error...")
```

⇒ write a Python program to store n student names into text file.

```
n = int(input("Enter no. of student"))
f = open("d:/gec.txt","a")
for i in range(n):
    print("Enter student", i+1, "name")
    k = input()
    f.write(k)
print("Data Succesfully entered into file")
f.close()
```

⇒ write a python program to count no. of digits in a file.

```
f = open("d:/file1.txt","r")
k = f.read()
c = 0
for i in k:
    if i.isdigit():
        c = c+1
print("digits count is:", c)
```

Importance of file concept

* File Handling [If there is no file, all data stores in primary memory]

⇒ Significance of files :- file is a collection of related information stored in the form of records. files are stored in Permanent memory (secondary memory)

⇒ Types of files :-

1. text file
2. binary file

1. Text files :- text files data is stored in the form of ASCII code. This data is in human readable format. Text files accessing and other functions are slow. It contains new line, tab etc..

2. Binary files :- Data is represented in the form of bits. (encoding format). This data is machine readable.

⇒ Basic file operations :-
1) open file
2) read / write
3) close a file.

1) Open or create :- By using open() method, create file in python

Syntax :-

```
file obj = open (filename, "mode")
```

ex:
```
f = open("d:/it/gec/student.txt", "r").
f = open("d:/it/gec/student.txt")
```

→ If we donot mention any mode, by default it takes read mode.

→ absolute path:- complete path from start to end for a file.

⇒ List of file opening modes:

1. r → read
2. w → write
3. a → append
4. r+ → read/write
5. w+ → write/read
6. a+ → read/write.
7. rb – read binary
8. wb – write binary
9. ab – append binary
10. rb+ – read/write, binary.

notepad	.txt
ms word	.doc / .docx
ms excel	.xls / .xlsx
power point	.ppt / .pptx

⇒) read/write :-

read operation requires the following methods:

1. read()
2. readline()
3. readlines()

1) read() :- used to read entire data from file.

Syntax:- fileobj.read()

Syntax:- fileobj.read(n) # n indicates no of bytes.

Ex:- 1st create a file in specific location.

```
file = input("Enter file name with ext:")
try:
    f=of=open(file,"r")
    print("Content of file is", f.read())
except FileNotFoundError as e:
    print("File or directory not exist")
```

f.read() → It displays all data

If we use file(6) first 6 characters of file including spaces is considered.

len(f.read()) → It displays length of data

2) readline():- It reads only one line at a time.

Syntax:- fileobj.readline()

Ex:- Print statement in try: is only changes for before program.

```
print("First line is:", f.readline())
```

o/p:- CSE IT @

gec-notepad

CSE IT @
ECE EEE # 324
IOT AIML ME CE
IT * $

3) readlines():- used to read each & every line from the file and append to the list.

Syntax:-

fileobj.readlines()

```
print("Content is:", f.readlines())
```

o/p:- Content is: ['CSE IT @ \n', 'ECE EEE # 324\n', 'IOT AIML ME CE \n', 'IT * $']

⇒ write: The following are the list of methods to write().

1. write():- used to write information into file in the form of ASCII (text)

Syntax:- fileobj.write("information")

Ex program:-

```
no = int(input("Enter roll no"))
name = input("Enter name")
address = input("Enter address")
file = input("Enter file name with ext:")
try:
    f = open(file,'a')
    f.write("No.'s:"+str(no))
    f.write("Name is:"+name)
    f.write("address is:"+address)
f.close() print("data posted into file")
except FileNotFoundError as e:
    print("File or directory is not found")
```

2) writelines() :- It also relates to list. It is used to write information in file in the form of list

Ex:-
```
no = int(input("Enter roll no"))
name = input("Enter name")
address = input("Enter address")
file = input("Enter file name with ext.")
p = [str(no), name, address]
try:
    f = open(file, 'a')
    print(f.writelines(p))
    print("data posted into file ..")
    f.close()
except FileNotFoundError as e:
    print("File or dir not exist")
```

=> ~~write a pgth~~

3) close() :- It is used to close a file.

Syntax:
```
fileobj.close()
```

=> write a python program to print reverse of the file content.

Program:-
```
file = input("Enter file name with ext.")
try:
    f = open(file, "r")
    print("Reverse is :", f.read()[::-1])
    f.close()
except FileNotFoundError as e:
    print("File or directory not exist")
```

Lab program:-

write a python program to reverse each & every line in a text file.

Program:-

```
file = input("Enter file name with ext.")
try:
    f = open(file,"r")
    for i in f.readlines():
        print(i[::-1])
    f.close()
except FileNotFoundError as e:
    print("File or dir. not exist")
```

⇒ Implement python code to copy the contents from one file to another file.

Program:-

```
file1 = input("Enter file name to be read"))
file2 = input("Enter file name to write")
try:
    f1 = open(file1,'r')
    f2 = open(file2,'a')
    f2.write(f1.read())
    print("Copied Successfully")
    f1.close()
    f2.close()
except FileNotFoundError as e:
    print("File or dir. is not exist")
```

⇒ Implement a python code to calculate number of lines, no of words and no. of characters in the content of a file.

Program:-

```
file = input("Enter file name with .ext")
w=c=m=0
try:
    f = open(file,'r')
    for i in f.readlines():
```

```
        m = m+1
        for j in i.split():
            w = w+1
            for k in j:
                if k.isalpha():
                    c = c+1
print("lines count is :", m)
print("wood count is :", w)
print("char. count is :", c)
except FileNotFoundError as e:
    print("file or dir. not exist")
```

⇒ write a python program to count no of alphabets, digits, special symbols and spacer

```
file = input("Enter file name with ext.")
a = d = s = sp = 0.
try:
    f = open(file, 'r')
    for i in f.read():
        if i.isalpha():
            a = a+1
        elif i.isdigit():
            d = d+1
        elif i.isspace(): if i != '\n':
            sp = sp+1
        else:
            s = s+1
    print("alphabets count is", a)
    print("digits count is", d)
    print("spacer count is", sp)
    print("Special symbols count is", s)
except FileNotFoundError as e:
    print("File or directory not exist").
```

Lab Program:-

pip install pandas
pip install openpyxl } open command prompt. ← type

Open excel

```
import pandas as p
v = p.read_excel("d:/data.xlsx")
print("mean is :", v['s1'].mean())
print("Standard deviation is :", v['s1'].std())
```

⇒ write a python program to reverse each and every word for the content of the file.

```
file = input("Enter file name")
try:
    f = open(file,'r')
    for i in f.readlines():
        for j in i.split():
            print(j[::-1], end=' ')
        print()
    f.close()
except FileNotFoundError as e:
    print("File not exist")
```

⇒ Develop python program to read file content and display only even numbers from given file.

```
file = input("Enter file name")
try:
    f = open(file,'r')
    for i in f.read():
        if i.isdigit():
            k = int(i)
            if k%2 == 0:
                print(k, end=' ')
    f.close()
except FileNotFoundError as e:
    print("File not exist")
```

→ write a python program to merge contents of two & stored into another file.

file-1
gec.txt
CSE IT @ ECE FEE # 324 IOT AIML ME CE IT * &

file 2
srgec.txt
This is Python code

Program:-

```
file1 = input("Enter file1 name")
file2 = input("Enter file2 name")
file3 = input("Enter file3 name")
try:
    f1 = open(file1, 'r')
    f2 = open(file2, 'r')
    f3 = open(file3, 'a')
    f3.write(f1.read())
    f3.write(f2.read())
    print("File merge Successfully..")
    f1.close()
    f2.close()
    f3.close()
except FileNotFoundError as e:
    print("File not exist")
```

o/p:- Enter file1 d:\gec.txt
Enter file2 d:\srgec.txt
Enter file3 d:\itt.doc
File merge Successfully..

o/p window

itt.doc
CSE IT @ ECE FEE # 324 IOT AIML ME CE IT * & This is Python code.

⇒ Role of 'with' keyword in File handling :-

The use of with keyword is to use for closing a file without using close method. the file directly closes when completing of file.

Syntax:-

with open("filename", "mode") as obj: