Smart Knower Python Practice

features of python:

1) Easy to learn
2) High level language
3) supports a lot of library
4) Dynamically Typed
5) Interpreted language
6) object oriented
7) supports GUI
8) open source

Note:-

* press 'h' to display all shortcuts

Code

```
p,q,r = 10, True, "hello"
print(p)
print(q)
print(p,q)
```

o/p

```
10
True
10 True
```

Rules of naming a variable:-

1) start with A-Z, a-Z or underscore(_)
2) Should not start with number
3) should not contain special characters
4) " " " python keywords

Dynamic Typing

The data type of the variable is defined at run time

Data Types:

int, float, str, bool, NoneType, complex

Strings

1) Enclosed in single, double or triple quotes.
2) Triple quotes are used to write multiline strings.

" " – double quotes
' ' – single quotes
""" """ – Triple quotes
''' ''' – Triple quotes

P) a,b = 5,2

print(a/b) = 2.5 # normal division
print(a//b) = 2 # integer divison
print(a%b) = 1 # returns remainder

→ a**b # a^b

Assignment operator

a+=b # a=a+b
a*=b # a=a*b

Relation or Comparision operators:

a!=b # a is not equal to b
a==b # a is equal to b.

P)

a,b,c = 10,7,15
print(a>b and a>c)
print(a>b or a>c)
d = True
print(not d)

O/P:-

False
True
False

*Membership Operator (in, not in)
– returns True or false

*membership operators works with strings, List and Tuples, sets

P) a = '''Russia vs. Ukraine war has affected the global economy'''
print('war' in a)
print('when' in a)

O/P)

True
false

Note:-

\n – newline character.

P) print('welcome to \n python)

O/P)

welcome to
python

P) w,n = 78,45
print(w,'\n',n)
print(n,w)
print('Hello')

O/P)

78
45
45 78
Hello

P) a = "Hello"
b = "world"
c = "python"
print(a,b,c)
print(a,b,c, end="welcome")
print('Google')

O/P:
Hello world python
Hello world python welcome Google

P) print(a, end = " ")
print(7*3)

O/P:
Hello 21

"sep" - It is used to seperate different values that are passed in the print().

P) p,q,r = 'Hi', 'Bye', 'Bat'
print(p,q,r) # default sep is " " (empty space)
print(p,q,r, sep="_")
print(p,q,r, sep="**")

O/P:
Hi Bye Bat
Hi_Bye_Bat
Hi**Bye**Bat

P) P= "Google"
q= "meta"
print(P,q, sep="python")
print(P,q, sep="\n")

O/P: Google python meta
Google
meta

P) a,b,c = 10,20,30
print(a,b,c, sep=":", end="*_*")

O/P:
10:20:30*_*

Print formatting methods:

name = "Shreya"
age = 22
city = "Bangalore"

print(name, 'whose age is', age, 'lives in', city) #comma seperated method

print(f'{name} whose age is {age} lives in {city}') #f-formatting.

print("{} whose age is {} lives in {}".format(name,age,city)) #format method

O/P:
Shreya, whose age is 22 lives in Bangalore
Shreya whose age is 22 lives in Ban

Indentation:

1) 1 tab space is referred to as indentation.
2) Indentation is used in conditional statements, for and while loops, functions etc...

If-elif-else

```
if condition1:
    code1
elif condition2:
    code2
elif condition3:
    code3
else:
    code4
```

For loops:

In range(start, end, step) method

1) In ascending order, iteration proceeds from start till end-1, step size is positive
2) In descending order, iteration proceeds from start till end+1, step size is negative.
3) Default step value = +1
4) range works only on int.
5) If only one value is passed in range function, it is considered as end value.
6) Default start value is 0, if not mentioned.

P) for i in range(-5,3,2):
    print(i, end=" ")

O/P:-
-5 -3 -1 1

P) for i in range(14, 10, -1):
    print(i, end=" ")

O/P:-
14 13 12 11

Iterate on strings:

1) Positive indexes start from 0 and move from left to right in increasing order.
2) Negative indexes start from -1 and move from right to left in decreasing order.

P) s = "Today"
for i in s:
    print(i, end=" ")

O/P:-
T o d a y

P) s = "Today"
for i in range(len(s)):
    print(s[i], end=" ")

O/P:-
T o d a y

P) print(i, s[i])

O/P:-
0 T
1 o
2 d
3 a
4 y

P) s = "welcome"

```
for i in range (2, len(s)-2):
    print (s[i], end = " ")
```

O/P:

l c o

P) s = python

```
i=0
while i<len(s):
    print(s[i], end =" ")
    i = i+1
```

O/P:

p y t h o n

While LOOPS:

P)

```
i=0
while i<5:
    print (i, end =" ")
    i+=1
```

O/P:

0 1 2 3 4

P)

```
d=17
while d>4:
    print (d, end=" ")
    d -=2
```

O/P:

17 15 13 11 9 7 5

**

break - The moment break keyword in encountered, it takes program control out of the loop.

P)

```
for in in range (10,20):
    if i%3 ==0:
        break
    else:
        print (i)
```

O/P:

10
11

P)

```
i=1
while i<7:
    print(i)
    if i%3==0:
        break
    i+=1
```

O/P:

1
2
3

Iterate on strings:-

P) s = "welcome to python"

print(len(s))

#welcome-7, to-2, python-6, space chars-2

O/P:

17

Continue:

The moment continue is encountered, it takes the program control to the beginning of the loop.

P)
```
for i in range (10,20):
    if i%3 ==0:
        continue
    else:
        print (i, end =" ")
```

O/P)-
10 11 13 14 16 17 19

P)
```
i=10
while i<20:
    i+=1
    if i <15:
        print (i)
    else:
        continue
```

O/P)
11
12
13
14
15

List:

1) List are ordered collection of elements of same or different data types.
2) enclosed in [] square bracket & seperated by comma (,).
3) List is mutable (can be changed)
4) Indexing and slicing is allowed in list
5) List can store duplicate elements.

```
List = []
tuple = ()
set = {}
dictionary = {}
```

P)
```
r= [10,15,20,25,30,35,40,45]
   0  1  2  3  4  5  6  7
  -8 -7 -6 -5 -4 -3 -2 -1
print(r)
print(r[2:5])
print(r[1:6:2])
print(r[-1:-7:-2])
print(r[-1:-7:2])
```

O/P)
[10, 25 . . . . ,45]
[20, 25, 30]
[15, 25, 35]
[45, 35, 25]
[]

p) print(x[2::3])

#L→R, step=+ve ⇒[20,35]

print(x[5::-2])

#R→L, step=-ve

o/p:

[20,35]

[35, 25, 15]

Iteration in list:

p) w=[12,23,34,45,56]

for i in w:

print(i)

o/p:

12
23
34
45
56

p) for i in range(len(w)):

print(i,w[i])

o/p:

0 12
1 23
2 34
3 45
4 56

* List is mutable, it can be changed.

p) q=[1,2,3,4,5]

print(q[4])

q[4]="python"

print(q)

o/p:

5

[1,2,3,4,'python']

Methods of list:

1) Append

p) s=[1,2,3]

s.print(s)

s.append(4)

print(s)

o/p:

[1,2,3]

[1,2,3,4]

p) s.append([1,2])

print(s)

o/p:

[1,2,3,4,[1,2]]

2) extend()

It adds an element to the list at the end from an iterable (strings, list, tuple, set, dict, range())

P) d = [4,5,6]
print(d)
d.extend([11,12,13]) #extending a list
print(d, len(d))
d.extend((7,8)) #extending a tuple
print(d, len(d))
d.extend({17,18}) #extending a set
print(d, len(d))

O/P
[4,5,6]
[4,5,6,11,12,13] 6
[4,5,6,11,12,13,7,8] 8
[4,5,6,11,12,13,7,8,17,18] 10

P) d = [4,5,6,7]
print(d)
d.extend(10) → int is not iterable
print(d)

O/P
error because 'int' is not iterable

w = [7,8,9]
P) w.extend('Bye') #extending a string
print(w)
w.extend({'Name':'Ankit','Age':20})
print(w) #only keys will be extended
w.extend(range(20,23))
print(w)

O/P
[7,8,9]
[7,8,9,'B','y','e']
[7,8,9,'n','n','n','Name','Age']

3) clear()
Empties the list.
q = [1,2,3]
q.clear()
print(q)
O/P
[]

4) insert(index, element)
→ Insert an element at a specified index.
→ Elements previously present at that index are shifted one index towards right.

P) a1 = [1,2,3,4,5,6,7]
print(a1)
a1.insert(3,'Hello')
print(a1)
a1.insert(6,'Python')
print(a1)

O/P
[1,2,--,7]
[1,2,3,'Hello',4,5,6,7]
[1,2,3,'Hello',4,5,6,'Python',7]

5) index()
Returns the index of the first occurence of the element from the left. Indexes start at 0.

P)
a = ['Hi', 'bye', 'Hello', 'Python', 'C++', 'Java', 'Hello', 'C++', '@']

```
print(a2.index('Hello'))
print(a2.index('Hello', 3))
print(a2.index('C++'))
```

'Hello', 3 → index after Hello; to search for occurrence of next hello starting from next index.

O/P
2
6
4

*To find occurrence of same element

P) a2 = ['Hi' ... '@', 'Hello']

```
for i in range(len(a2)):
    if a2[i] == 'Hello':
        print(i)
```

O/P 2
6
9

(6) POP(index)

Removes the element at the specified index. If no index is mentioned, it removes the last element.

P) a = [0,1,2,3]

```
a.pop(1)
print(a)
```

O/P [0,2,3]

(7) remove(element)

It removes the first occurrence of the element from left which is passed as argument. Error if element is not in the list.

P) remove element

a = [-3, 34, 45, 56, -67, 2, 56, 1]

```
for i in a:
    if i==56:
        a.remove(56)
print(a)
```

O/P
[-3, 34, 45, -67, 2, 1].

8) count()

Returns the frequency of the element passed as an argument.

P) w = [1, 2, 1, 3, 1, 2, 4, 3, 3]

```
print(w.count(1))
print(w.count(3))
print(w.count('Hello'))
```

O/P
3
3
0

9) reverse()

It reverses the list. Arranges the list in right to left order

P) a = [1,2,3]

```
a.reverse(a)
print(a)
```

O/P

[3,2,1]

10) sort()

Sorts the list in ascending or descending order based on parameter reverse

P) s = [12,7,16,34,89,74,56]

```
print(s)
s.sort()        #ascending order
print(s)
s.sort(reverse=True)  #descending order
print(s)
```

O/P

[12,7,16,34,89,74,56]

[7,12,16,34,56,74,89]

[89,74,56,34,16,12,7]

P) print(s[::-1])

O/P

[89,74,56 ---, 7]

***

ASCII Values:

* A-Z = 65-90
* a-z = 97-122

P)

s = ['Hello', 'Python', 'Cricket', 'India', 'Zebra', 'Crypto']

```
s.sort()
print(s)
s.sort(reverse=True)
print(s)
```

O/P

['Cricket', 'Crypto', 'Hello', 'India', 'Python', 'Zebra']

['Zebra', 'Python', ---, 'Crypto', 'Cricket']

Methods min, max and sum

```
h = [12,13,15,25,10,40]
print(min(h))
print(max(h))
print(sum(h))
```

O/P

10
40
115

List Comprehension:

1) It is a shorthand syntax to create a list.
2) It provides a one-linear syntax to create a list.

P)
```
r = []
for i in range(4,17,3):
    r.append(i)
print(r)
```
O/P
[4,7,10,13,16]

P)
```
r1 = [i for i in range(4,17,3)]
print(r1)
```
O/P
[4,7,10,13,16]

Above both gives same O/P.

P)
```
r = []
for i in range(4,27,3):
    if (i%2 ==0):
        r.append(i)
print(r)
```
O/P
[4,10,16,22]

P)
```
r1 = [i for i in range(4,27,3) if i%2==0]
print(r1)
```
O/P
[4,10,16,22]

Above both gives same O/P.

P)
```
res = []
for i in range(2,5):
    for j in range(3,7):
        if (i+j)%2 ==0:
            res.append(i*j)
print(res)
```
O/P
[8,12,9,15,16,24]

P)
```
res1 = [i*j for i in range(2,5)
            j in range(3,7)
        if (i+j)%2 ==0
```
O/P
[8,12,9,15,16,24]

If-else in list comprehension

```
t=[]
for i in range(2,30,3):
    if i%2==0:
        t.append(i+5)
    else:
        t.append(i-5)
print(t)
```

O/P:

[7, 0, 13, 6, 19, 12, 25 ... 24]

```
t1=[i+5 if i%2==0 else i-5 for i in range(2,30,3)]
print(t1)
```

O/P:

[7, 0, 13, 6, 19, ... 24]

Dictionary

1) Dictionary elements are stored in key:value pair format.
2) Dictionary elements are enclosed in {} and are seperated by comma
3) Dictionary are unordered.
4) Indexing and slicing is not allowed on dictionary. Dictionary can be indexed based on its keys.
5) All the keys in the dictionary must be unique. Values may be duplicate.
6) Dictionary is mutable (it can be changed)
7) The key, value pair is called item in dictionary and each item is stored in form of tuple.

Empty dictionary:

```
d1={}
d2=dict()
print(d1)
print(d2)
```

O/P:

{}
{}

Examples of dictionary

```
wl={'name':'Harsh', 'age':20, 'city':'Mumbai'}
print(wl)
print(len(wl))
print(type(wl))
```

O/P:

{'name': 'Harsh', ...}
3
<class 'dict'>

Fetch values corresponding to a given key.

```
print(wl['age'])
print(wl['name'])
print(wl['city'])
```

O/P:

20
Harsh
Mumbai

**

Keys allowed = str, float, int, bool, complex, None, tuple – immutable data types.

** Keys not allowed = set, list, dictionary.

P)
w={'a1':10, 12.34:'Hello', 5:20, True:'ABC', 345:'Google', None:20, (10,20):'Meta'}
print(w)

O/P)
{'a1':10, 12.34:'Hello', ..., (10,20):'Meta'}

Add new key:value pair.

** Dict is mutable (it can be changed)

P)
d={'a1':10, 'a2':20, 'a3':30}
d['k1']=15
print(d)
d['k2']='function'
print(d)

O/P)
{'a1':10, ..., 'k1'=15}
{'a1':10, ..., 'k2'='function'}

Change value for a given key

P)
d={'a1':10, 'a2':20, 'a3':36}
d['a1']='Teja'
d['a2']=120
print(d)

O/P)
{'a1':Teja, 'a2':120, 'a3'=30}

** Indexing & slicing is not allowed.

P) d1={'a1':40, 'a2':20}
print(d1[0])
O/P) error because indexing is not allowed

**All the values in the dict must be unique. values can be duplicated.

P)
g1={'k1':5, 'k2':10, 'k3':15, 'k4':50, 'k2':20, 'k3':15, 'k2':65}
print(g1)
print(g1['k2'])

O/P)
{'k1':5, 'k2':65, 'k3':15, 'k4':5, 'k5':15}
65

if duplicate values are present in given dictionary, the last appeared value will be reflected in the output dictionary.

Methods of dict

1) get(key)

It is used to fetch value for a given key.

```
p) r={'k1':5,'k2':65,'k3':15}
print(r.get('k1'))
print(r['k1'])
print(r.get('k7'))
print(r['k7'])  # key error
```

both can use to fetch value

o/p
```
5
5
None
keyerror
```

2) update()

It is used to update a dict. with another dict. It can be used to merge 2 dict into one dict.

```
p) w1={'a1':10,'a2':20}
w2={'a3':30,'a4':40}
w1.update(w2)
w2.update(w1)
print(w1)
print(w2)
```

o/p
```
{'a1':10, --, --, 'a4':40}
{'a3':30, --, --, 'a2':20}
```

p) * updated value will reflect in list

```
w1={'a1':3,'a2':7,'a3':9}
w2={'a2':6,'a3':5,'a4':8}
w1.update(w2)
print(w1)
```

o/p
```
{'a1':3,'a2':6,'a3':5,'a4':8}
```

3) clear()

It empties the dictionary

```
p) w1.clear()
print(w1)
```

o/p
```
{}
```

4) pop(key)

Remove key-value pair from the dict based on key passed as an argument.

```
p) w1.pop('a3')
print(w1)
```

o/p
```
{'a1':3,'a2':7}
```

5) popitem()

Removes the key-value pair from the dict in LIFO (Last in first out) order.

```
p) w1.popitem()
print(w1)
```

o/p
```
{'a1':3,'a2':7}
```

(6) setdefault (key, value)

It is used to add a new key:value pair to the dict. If no value is passed, then the value will be updated as None.

P)

```
w1.setdefault('Name','Ankit')
print(w1)
w1.setdefault('Age')
print(w1)
```

O/P)

{'a1':10, 'a2':20, ---, 'Name':'Ankit'}

{---, ---, ---, 'Name':'Ankit', 'Age':None}

*** setdefault, cannot be used to update the existing value for a given key

7), 8), 9) keys, values and items

P)

```
print(w1.keys())
print(w1.values())
print(d.items())
```

O/P)

```
dict_keys(['a1', 'a2', 'a3'])
dict_values([10, 20, 30])
dict_items([('a1',10), ('a2',20), ('a3',30)])
```

keys() - Return keys of dict

values() - Return values of dict

items() - Return keys, values of the dict in tuple format.

Iteration in Dict.

By default dict allows iteration only on its keys.

P)

```
for i in w1:          # i corresponds to each & every key of dict
    print(i, w1[i], w1.get(i))
```

O/P)

```
a1 3 3
a2 7 7
a3 9 9
```

(or)

```
for i in w1.keys():
    print(i, w1[i], w1.get(i))
```

O/P) same o/p as above.

P)

```
for i in w1.values():   # i corresponds to each & every values of the dict.
    print(i)
```

O/P)

```
3
7
9
```

P)
```
for i in wl.items(): # i corresponds to each & every (key,value) of the dict
    print(i)
```
O/P
```
('a1', 3)
('a2', 7)
('a3', 9)
```

P)
```
for K,V in wl.items(): # K corresponds to keys & V corresponds to values
    print(K,V)
```
O/P
```
a1 3
a2 7
a3 9
```

Keys can be list, set or dict.

P)
```
d={}
for i in range(len(x)):
    d[x[i]]=x.count(x[i])
print(d)
```
O/P
```
{12:2,23:2,34:4,45:3,56:3,
 67:2,78:2,89:1}
```

Tuples:

1) Tuple elements are represented in () and the elements are separated by comma.
2) Tuple is also an ordered collection of elements of same or different data types.
3) Tuple is immutable (cant be changed)
4) Indexing and slicing is allowed
5) Duplicate elements are allowed.

Count of each and every element using dictionary:

P)
```
x=[23,12,34,23,45,56,45,
   34,56,67,78,67,45,34,56,
   34,78,89]
for i in range(len(x)):
    print(x[i],x.count(x[i]))
```
O/P
Prints how many values occurs how many times.

Create Empty Tuples

P)
```
t1=()
t2=tuple()
print(t1)
print(t2)
```
O/P
```
()
()
```

* Indexing and slicing follows the same pattern as with lists or str.

P)
```
q1 = (10,14,12,14,15,16,17)
print(q1[0])
print(q1[-1:-6:-2])
print(q1[3:6])
```

O/P

```
10
(17,15,12)
(14,15,16)
```

Iteration in Tuples

It is also same like that of list

Tuple methods:

1) index(element)

Returns the index of the first occurrence of the element from the left.

2) count(element)

Returns the frequency of the element passed as an argument.

P)
```
print(q1.index(10))
print(q1.count(14))
print(q1.index(14))
print(q1.index(14,2))
```

2nd occurrence of 14 ← The next occurrence of 14

O/P

```
0
2
1
3
```

P)
```
num = [1,2,3,4,5]
name = ('Ankit','Harsh','Shreya',
        'Ridhima','Mohit')
d = {}
for i in range(len(name)):
    d[num[i]] = name[i]
print(d)
```

O/P

{1:'Ankit', 2:'Harsh', ---, ---, 5:'Mohit'}

Set

1) It is unordered collection of elements of same or different data types.
2) set elements are represented in {} and are seperated by comma.
3) set does not support indexing or slicing (as it is unordered)
4) set doesnot allow duplicate elements
5) set is mutable (it can be changed)

Empty set

```
s1=set()
print(s1)
print(type(s1))
```

O/P
```
{}
<class 'set'>
```

```
s2={}
print(s2,type(s2))
```

O/P
```
{} <class 'dict'>
```

Iteration in sets :

```
z1={10,11,23,34,45,56}
for i in z1:
    print(i)
```

O/P
```
34
10
11
```

Duplicates are not allowed

```
d1={5,4,6,7,3,3,4,5,6,4,3,3}
print(d1)
```

O/P
```
{2,3,4,5,6,7}
```

Set methods :

1) union()

perform union of sets, works for multiple sets as well.

```
s1={5,6,7,8,9}
s2={1,3,5,7,10}
s3={11,0,12}
s4=s1.union(s2)
print(s4)
print(s1.union(s2,s3))
```

O/P
```
{1,3,5,6,7,8,9,10}
{0,1,3,5,7,10,11,12}
```

2) intersection()

→ perform intersection of sets, works for multiple sets as well
→ Returns common element from the sets used.

P) print(S1.intersection(S2))
print(S1.intersection(S2,S3))

O/P:-
{5,7}
{5}

3) difference()

→ S1.difference(S2) ⇒ Returns the elements of S1 which are not present in S2.
→ Can be applied on multiple sets at a time.

P)
print(S1.difference(S2))
print(S2.difference(S1,S3))
print(S1.difference(S1))

O/P:-
{6,8,9}
{10,3,1}
set()

4) symmetric difference

→ S1.symmetric_difference(S2) ⇒ difference operation b/w union & intersection.
→ Can take only 1 set as its argument.

P)
S4 = S1.union(S2)
S5 = S1.intersection(S2)
S6 = S1.symmetric_difference(S2)
print(S4)
print(S5)
print(S4.difference(S5))
print(S6)

O/P:-
{1,3,5,6,7,8,9,10}
{5,7}
{1,3,6,8,9,10}
{1,3,6,8,9,10}

5) update()

→ It also performs union of sets. But the set on which update operation has been performed is updated with the union of sets.
→ works with multiple sets as well

P) S1.update(S2)
print(S1)

O/P:-
{1,3,5,6,7,8,9,10}

P) S2.update(S1,S3)
print(S2)

O/P:-
{0,1,3,5,6,7,8,9,10,11,12}

6) pop()

Remove and return an arbitrary (random) set element.

P)
```
s1.pop()  #random element is deleted from set
print(s1)
```

O/P:-
{5,7,8,9}

8) clear()

Empties the set

P)
```
s1 = {1,2,3,4}
s1.clear()
Print(s1)
```

O/P:-
set()

9) remove(element)

Used to remove particular element from the set passed as an argument.

P)
```
s1 = {10,20,30,40,50}
s1.remove(30)
Print(s1)
```

O/P:-
{10,20,40,50}

7) Add()

It adds an element to the set.

P)
```
s1.add('messi')
Print(s1)
```

O/P:-
{10,20,'messi',30,40,50}

* It adds element at any place.
* No error when duplicate element is added.

min, max and sum on set

P)
```
s1 = {10,30,5,46,25}
Print(min(s1))
      (max(s1))
      (sum(s1))
```

O/P:-
10
46
124

To create a Rock Paper scissor game:

```
import random.
options = ['Rock', 'paper', 'scissor']
comp = random.choice(options)
user = input('select one : Rock, Paper or
                                scissor:')
print('user choice:', user)
print('comp choice:', comp)
print('*'*20)
if user == 'ROCK':
    if comp == 'paper':
        print('you lose. paper covers rock')
    elif comp == 'scissor':
        print('you win. Rock smashes scissor')
    else:
        print('Tie')
elif user == 'scissor':
    if comp == 'paper':
        print('you win. scissor cut paper')
    elif comp == 'rock':
        print('you lose. ROCK smashes
                                scissor')
    else:
        print('Tie')
elif user == 'paper':
    if comp == 'ROCK':
        print('you win. Paper covers rock')
    elif comp == 'scissor':
        print('you lose. scissor cut paper')
    else:
        print('Tie')
else:
    print('Invalid entry')
```

O/P:-

```
select one : Rock, Paper or scissor :
                                scissor
user choice : scissor
comp choice : scissor
**********
Tie
```

Dice game

1-1Point
2-2Point
3-3Point
4-4Point
5-5Point
6-6Point

In first 5 dice throw, whoever gets more points wins.

Two players are A, B

```
a_sum, b_sum = [], []
for i in range(5):
    a = random.randint(1,6)
    b = random.randint(1,6)
    a_sum.append(a)
    b_sum.append(b)
print(a_sum)
print(b_sum)
if sum(a_sum) > sum(b_sum):
    print(f'A wins with {sum(a_sum)}
                                points')
else:
    print(f'B wins with {sum(b_sum)}
                                points')
```

O/P:-

```
[1,4,3,3,4]
[3,4,4,5,5]
B wins with 21 points.
```

Functions

1) Functions facilitate reusability of code.
2) They prevent code repetition.

Types of functions

1) Built-in functions
   print(), input(), len(), type(), list(), append()
2) User defined functions
   They are created by user/ developers to solve a problem statement.

They consist of two parts
1) function definition
2) function calling - functions are called using function name.

P) def f1(a, b=3):
       print(a+b)

f1(4,5) # a=4, b=5
f1(8)   # a=8, b=3

O/P
9
11

P) def f2(a=2, b=3, c=5, d=4):
       r=(a+b)*(c-d)
       print(r)

f2(10,12,15,5)
f2(7,8,6)
f2(7,8)
f2(20)
f2()

O/P
220
30
15
23
5

P) def f4(a=10, b):
       r=a*b
       print(r)

f4(12,5)

O/P
error
non-default argument follows default argument.

P) def f4(b, a=10):
       r=a*b
       print(r)

f4(8,4)
f4(6)

O/P
32
60

Note:

** while defining the functions 'return' cannot generate sequence with arguments

1) Positional arguments have been defined first (in left to right order) and then followed by default arguments.

p) def f6():
    for i in range(5,10):
        print(i, end=" ")
f6()

o/p
5 6 7 8 9

return keyword:

1) return is only used inside functions.
2) return has to be the last statement in a function.
3) return can be used only once in a function (excluding the if-elif-else condition)
4) If multiple return statements are used, only the first one is the valid statement.
5) return cannot be used to generate a sequence.
6) Any code written below the return keyword will not get executed.

p) def f7():
    for i in range(5,10):
        return i
r=f7()
print(r)

o/p
5

p)
n=[10,8,12,9,15,21,14,7]
k=3
#right_rotate_list(n,k)
#result=[21,14,7,10,8,12,9,15]

def right_rotate_list(n,k): #k<len
    r=[]
    r.extend(n[-k:])
    r.extend(n[:(len(n)-k)])
    print(r)
right_rotate(n,k=3)

o/p
[21,14,7,10,8,12,9,15]

p)

#Given (x,y) coordinates of a person.

#Determine the distance travelled by the person after n steps.

#The person can move either L, R, T or B in each step.

#x,y – initial coordinates

#n – num of steps

#steps = [] consisting of L, R, T, B

#dist_travelled (x, y, n, steps).

#len(steps) = n

p) def dis_travelled (x,y,n,steps):

```
    x1,y1 = x,y
    for i in range(len(steps)):
        if steps[i] == 'L':
            x -= 1
        elif steps[i] == 'R':
            x += 1
        elif steps[i] == 'T':
            y += 1
        else:
            y -= 1
    x2,y2 = x,y
    dist = ((y2-y1)**2 + (x2-x1)**2)**0.5
    print('Initial coordinates', x1, y1)
    print('final u', x2, y2)
    print(dist)
dist_travel(2,3,5,'LTBRB')
```

O/P

Initial coordinates 2 3

final 1 4

dist travelled 4.1231

Pattern Based Questions!

see in youtube

some programs are practiced in Colab.

Python Inbuilt functions!

1) Lambda
2) map
3) reduce
4) filter
5) enumerate
6) zip
7) sorted

P)
```
w = lambda fn,ln : fn[0].upper()+'.'+
                   ln[0].upper()
print(w('elon','musk'))
print(w('Benedict','Cumberbatch'))
```
O/P
E.M
B.C

Lambda

→ It is used to create anonymous function (a function with no name)
→ It is used to create one-liner function.

Syntax:

lambda parameters : expressions

P)
```
y = lambda x,y : x+y
print(y(3,4))
print(y(6,7))
print(y(10,12))
```
O/P
7
13
22

if-else inside lambda

P)
```
d = lambda x,y : x*y-5 if(x%3==0
    and y%5!=0) else y*10-x
print(d(6,7))
print(d(12,10))
```
O/P
37
88

P)
```
w1 = lambda p : [i*2 for i in
                 range(1,p+1)]
print(w1(5))
```
O/P
[2,4,6,8,10]

2) map() :-

→ map() calls the specified function for each item of an iterable (list, string, tuple, dict etc) and returns a sequence of results.

→ lambda can be used inside map.

syntax: map(function, iterable)

* In map we can't use conditional operators.

P)
```
def square(x):
    return x*x
w = [1,2,3,4]
d = list(map(square,w))
print(d)
```
O/P
[1,4,9,16]

P)
```
x = [10,12,14,16,18]
d1 = list(map(lambda x: x**2, x))
print(d1)
d2 = list(map(lambda x: x*3+1 if x%4==0 else x*5-2, x))
print(d2)
```
O/P
[100,144,196,256,324]
[48,37,68,49,88]

P)
```
name = input().split(' ')
print(name)
```
O/P
input: 23 45 56 hello
['23', '45', '56', 'hello']

* split() function returns list as shown above

Q) Accept list of integers from the user using map function.

P)
```
d2 = list(map(int, input('Enter nos: ').split(' ')))
print(d2)
```
O/P
Enter nos: 23 67 89 15
[23, 67, 89, 15]

P)
```
x = "sean@Rash@michael"
x1 = x.split('@')
x2 = " ".join(x1)
print(x2)
print(x1)
```
O/P
Sean Rash Michael
['sean', 'Rash', 'michael']

3) filter()

→ Returns an iterator yielding those items from the iterable for which the function is True

Syntax: filter(function, iterable)

* In this only applies conditional operators.

P) def filter_odd(n):
    return n%2!=0
g=[12,23,....,20]
s2=list(filter(filter_odd,g))
print(s2)

O/P
[23,15,17,19]

P) g=[12,13,14,15,16,17,18,19,20]
s1=list(filter(lambda x: x%2==0,g))
print(s1)

O/P
[12,14,16,18,20]

P)
g=[12,23,14,....,20]
s2=list(filter(lambda x: x>17,g))
print(s2)
s3=list(filter(lambda x: x<40, g))
print(s3)
s4=list(filter(lambda x: x,g))
print(s4)

O/P
[23,18,19,20]
[12,23,14,15....20]
[12.....20]

Map vs filter

1) The length of the iterable obtained from map is always of the same length as that of original iterator onto which the map function has been applied
→ The length of iterable obtained from filter can be of the same length or less as that of original iterator onto which the filter function has been applied.

2) The filter operates only on those element of the iterable onto which the function returns True, whereas map operators on all the elements based on the function passed as an argument

P)

```
r = list(range(10,22,2))
print(r, len(r))
m1 = list(map(lambda x: x+5, r))
f1 = list(filter(lambda x: x%3!=0, r))
print(m1)
print(f1)
```

O/P)

[10, 12, 14, 16, 18, 20] 6

[15, 17, 19, 21, 23, 25]

[10, 14, 16, 20]

4) Reduce()

→ Reduce function reduces the iterable to a single value using the function passed as argument.

→ The function passed has to contain 2 parameters.

Syntax: reduce(function, iterator)

P)

```
from functools import reduce
b = [1,2,3,4,5]
t = reduce(lambda x,y : x+y, b)
print(t)
# S1 → x=1, y=2 ⇒ x+y=3
# S2 → x=3, y=3 ⇒ x+y=6
```

O/P)

15

5) enumerate()

→ It adds an extra counter to the iterables (strings, list, tuple, set, dict, range())

→ Counter element generated starts from zero.

→ Counter added to the list and tuple can be used as their index

→ Counter added to set should not be considered as index.

→ By default the count starts from zero.

P)

```
d = [10, 20, 30, 40, 50]
for i,j in enumerate(d):
    print(i,j,d[i])
```

O/P)

```
# i is the extra counter generated, j is the list element
0 10 10
1 20 20
2 30 30
3 40 40
4 50 50
```

P)

```
d = (11,12,13,14,15)
for x,y in enumerate(d, start=5):
    print(x,y)
```

O/P)

```
5 11
6 12
7 13
8 14
```

1) s = {11, 9, 13, 10, 8}

```
for x,y in enumerate(s):
    print(x,y)
```

O/P:

```
0 8
1 9
2 10
3 11
4 13
```

```
print(x1)
print(x2)
 "  (x3)
 "  (x4)
```

O/P:-

[('a',3), ('b',4), ---, ---, ---]

(('a',3), ('b',4), ---, ---, ---)

{('a',3), ('d',6), ('b',4), ---, ---}

2) d = {'a1':10, 'a2':25, 'a3':20, 'a4':15} {'a':3, 'b':4, 'c':5, ---, 'e':7}

```
for i,j in enumerate(d):
    print(i,j,d[j],d.get(j))
```

O/P:-

```
0 a1 10 10
1 a2 25 25
2 a3 20 20
3 a4 15 15
```

3) d = [11,12,13,14]

```
for x,y in enumerate(d[::-1]):
    print(x,y,d[x])
```

O/P:-

```
0 14 11
1 13 12
2 12 13
3 11 14
```

6) zip()

→ It aggregates (two or more) elements into tuple pairs except dict.

1)

```
d1 = ['a','b','c','d','e']
d2 = [3,4,5,6,7]
r1 = list(zip(d1,d2))
r2 = tuple(zip(d1,d2))
r3 = set(zip(d1,d2))
r4 = dict(  )
```

2) d3 = ['Hi', 'Bye', 'Tej']

```
w1 = list(zip(d1,d2,d3))
```

O/P:-

[('a',3,'Hi'), ------, ('c',5,'Tej')]

7) Sorted()

→ Sort the elements passed as argument

→ Sorted function returns a list

→ It is not a list function, however it can be applied on a list.

Eg)

```
w= [12,14,10,9,15,13]
x1 =sorted(w)
x2=sorted(w, reverse =True)
print(x1)
print(x2)
```

O/P:-

```
[9,10,12,13,14,15]
[15,14,13,12,10,9]
```

Note:-

→ sort() is only applied to list

→ sorted() can be applied to other data types as well.

Eg)

```
d={'a1':10, 'a3':8, 'a2':15, 'a4':25}
d1 = sorted(d.keys())
d2 = sorted(d.keys(), reverse=True)
d3 = sorted(d.values())
d4 = u (-1, reverse=True)
d5 = sorted(d.items())
d6 = sorted(d.items(), key=lambda x : x[0])   # sorting based on items
d7 = sorted(d.items(), key=lambda x : x[1])   # sorting based on value
                                              # reverse=True)
print(d1)
print(d7)
```

O/P:-

```
['a1', 'a2', 'a3', 'a4']
['a4', --, --, 'a1']
[8,10, 15, 25]
[25, 15, 10, 8]
[('a1',10), ('a2',15), --, ('a4',25)]
[('a1',10), u., ---, ('a4',25)]
[('a4',25), ('a2',15), ('a1',10), ('a3',8)]
```

Errors

→ Errors are the problems in a program due to which the program will stop the execution.

1) Error in python can be of two types.

(a) Syntax errors: They occur due to incorrect syntax.

(b) Exceptions: Exceptions are raised when some internal events occur which changes the normal flow of the program.

Exception Handling:

Exception handling doesn't remove or resolve the error, it simply avoids it.

1) try: we write the code which is suspected to contain exception (error)

2) except: Handle the error (exception). except block of code works only when there is an exception (error) in try block.

3) else: Code will execute if there is no error (exception) in try block.

4) finally: Code will work regardless of the presence or absence of error (exception) in the try block.

Note!:

(a) Atleast try and except block of code should be written while handling exception

(b) else and finally are optional

(1) ZeroDivisionError

```
try:
    a = int(input('Enter 1st number: '))
    b = int(input('Enter 2nd number: '))
except:
    print('No error encountered')
```

O/P)

Ent 1st no : 50

" 2nd no : 10

5.0

P)
```
try:
    a=int(input('1st no:'))
    b=int(input('2nd no:'))
    print(a/b)
except ZeroDivisionError as em:
    print('Error msg:', em)   #em=error msg
```

O/P:
1st no: 5
2nd no: 0
Error msg: division by zero.

2) TypeError

P)
```
a=10
b='5'
print(a+b)
```

O/P:
error
unsupported operand type(s) for +: 'int' and 'str'

P)
```
try:
    a=10
    b='7'
    print(a-b)
except TypeError as em:
    print('Error Msg-', em)
```

O/P:
Error Msg - unsupported operand type(s) for -: 'int' and 'str'

3) ValueError

P)
```
a=int(input('Enter number:'))
print(a)
```

O/P:
Enter a number: seven
ValueError
invalid literal for int() with base 10: 'seven'

P)
```
try:
    a=int(input('Enter 1st no:'))
    b=int(input('Enter 2nd no:'))
except ValueError as em:
    print('Error Msg-', em)
else:
    print(f'sum of {a} and {b} is {a+b}')
finally:
    print("This block will work regardless of occurrence of error")
```

O/P:
Enter 1st no: 7
Enter 2nd no: 8
sum of 7 and 8 is 15
This block will work regardless of occurrence of error.

O/P:
Enter 1st no: 40
Enter 2nd no: Eight
Error Msg - invalid literal for int() with base 10: 'Eight'
This block will work regardless of

4) IndexError

```
x = [23,25,27,29,3]
print(x[5])
```

o/p

IndexError
list index out of range

```
x = [23,25,27,29,3]
try:
    for i in range(len(x)):
        print(x[i], x[i+1])
except IndexError as em:
    print('Err msg -', em)
else:
    print('No error in try block')
```

o/p

```
23 25
25 27
27 29
29 31
Err msg - list index out of range
```

Note :-

* If 'try' block is true, 'else' block is executed.
* " " " " false, 'except' block gets executed.

Text file Handling :

Read the .txt file and display its content when file is in same folder in read mode.

File Handling methods

1) readlines() - displays all the content of the file (from the cursor position) in a list.
2) read() - Read at most n characters from stream or file.
3) seek() - Repositions the cursor to a particular index
4) tell() - Returns current cursor position.
5) close() - file is closed to deallocate (remove) the memory
6) write() - Used to write new content (data) to the file.

P)
```
#mode = 'r' => read mode
f = open ('file location', 'r')
rl = f.readlines() #returns a list
print(rl)
f.close() #to deallocate the memory
```

O/P)-

shows the content in the file

['welcome\n', 'Russia\n', 'Lenovo\n', 'HP']

P)
```
print ('Hello \\ world')
#\n - new line character
#\t - 1 tab space
```

O/P)-

Hello \world

*

#Below path will not work

#C:\Users\Dell\Downloads.....

→ If the address is only with one backslash then the program returns error.

→* It works when we use double forward slash also (//).

P)
```
fl = open ('C:\\Users\\Dell\\Down\ML_DL... -', 'r')
rl = fl.readlines()
print(rl)
fl.close()
```

O/P)-

['new file\n', 'thor and love']

P)
```
fl = open('C:\Users\Dell\Down... -', 'r')
rl = fl.readlines()
print(rl)
fl.close()
```

O/P)-

SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXX escape

** If we use single forward slash, it will work.

P)
```
fl = open ('C:/Users/Dell/Down...', 'r')
rl = fl.readlines()
print(rl)
fl.close()
```

O/P)- ['new file\n', 'thor & love']

P) f = open('demo1.txt','r')
r1 = f.read(21)
print(r1)
Print(f.tell())

O/P)
Welcome to python
RUS
21

→ here only 20 characters are present & 21st character is \n

P) r2 = f.read(12)
Print(r2)
print(f.tell()) #current cursor position

O/P)
Sia vs Ukrai
33

→ cursor is at 21th position, after that character are printed when read is called.

#seek() - it repositions the cursor to a particular index

P) f.seek(5)
r3 = f.read(7)
Print(r3)
print(f.tell())

O/P)
me to P
12

#close the file - To deallocate the memory.

f.close().

Reading the file in append mode

→ append mode is used to add some content (new data) to the file.

'a' implies append mode

P) w = open('file location', mode='a')
w.write('\nSept is about to start')
w.write('\nPython is fun to learn')
w.close()

Then

f = open('file location', 'r')
print(f.readlines())

O/P)
['Welcome to Python\n', 'Russia vs Ukraine\n', ---, 'Sept is about to start\n', 'Python is fun to learn']

Reading file in write mode

It is meant to create a new file and add content to the file.

Opening a pre-existing file in write mode ⇒ overwrites the content if write function is used.

P) f = open('demo2.txt', 'r')
print(f.readlines())

O/P)
['Hello World\n', 'File Handling in Python']

P) #w implies write mode
g = open('demo2.txt', 'w') #creating a new file - demo2.txt
g.write('Hello World')
g.write('\nFile Handling in Python')
g.close()

Then after creation
f = open('demo2.txt', 'r')
print(f.readlines())

O/P)
['Python training\n', 'SmartKnower']

Creating a new file using the write mode

P)
s = open('demo4.txt', 'w') #creating a new file in w mode
s.write('DS and ML\n')
s.write('Numpy and Pandas')
s.close()

Then after creation
f = open('demo4.txt', 'r')
print(f.readlines())

O/P)
['DS and ML\n', 'Numpy and Pandas']

Numpy Array

Array in numpy

→ Array is an ordered collection of homogenous elements in numpy.

→ All the elements in the numpy array must be of same data type.

Custom numpy array

1) np.zeros() - Used to create an array of zeroes of specified shape

2) np.ones() - Used to create an array of ones of specified shape.

P)

```
a1 = np.zeros((4,3))
print(a1)
print(a1.shape)
print(a1.dtype)
print(a1.ndim)
```

O/P:

```
[[0. 0. 0.
  0. 0. 0.
  0. 0. 0.
  0. 0. 0.]]
(4,3)
float64
2
```

P)

```
a1 = np.ones((3,5), dtype='float')
print(a1)
print(a1.shape)
print(a1.dtype)
print(a1.ndim)
print(a1.size)
```

O/P:

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
(3,5)
float64
2
15
```

P)

```
a1 = np.ones((2,3,5), dtype=int)
print(a1)
print(a1.shape)
```

O/P:

```
[[[1 1 1 1 1]
  [1 1 1 1 1]
  [1 1 1 1 1]]

 [[1 1 1 1 1]
  [1 1 1 1 1]
  [1 1 1 1 1]]]
(2,3,5)
int32
3
30
```

P)

```
a3 = np.ones((7), dtype=int)
print(a3)
```

O/P:

```
[1 1 1 1 1 1 1]
(7,)
int32
1
7
```

Note :-

A numpy array can be reshaped into product of its factors of its length (number of elements in the array)

P) v1 = np.array([2,4,6,8,10,12,14,16,18, 20,22,24])

print(v1)
print(len(v1))
print(v1.size)
print(v1.shape)
print(v1.ndim)

#factors of 12 - 1,2,3,4,6,12
#pairs are - (1,12),(12,1),(2,6),(6,2), (3,4),(4,3)

OP :-

[2 4 6 8 12 ... 24]
12
12
(12,)
1

Then

P) v1 = np.array([2,4,...,24]).reshape(4,3)

print(v1)
print(v1.shape)

OP :-

[[2 4 6]
 [8 10 12]
 [14 16 18]
 [20 22 24]]

(P)

try:
    a = int(input('Enter 1st no: '))
    b = int(input('Enter 2nd no: '))
    print(a/b)
except ValueError as err1:
    print('Error Msg -', err1)
except ZeroDivisionError as err2:
    print('Error Msg -', err2)
else:
    print(f'sum of {a} and {b} is {a+b}')
finally:
    print("This block will work regardless of occurrence of error")

OP :-

Enter 1st no : 8
Enter 2nd no : seven
Error msg - invalid literal for int()

```
d1 = np.arange(14,56,5)
print(d1)
print(type(d1))
```

O/P)

```
[14 19 24 29 ... 54]
<class 'numpy.ndarray'>
```

P)

```
d1=np.arange(10,28).reshape(2,3,3)
print(d1)
print(d1.shape)
```

O/P)

```
[[[10 11 12]
  [13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]
  [25 26 27]]]
(2,3,3)
```

P) d1.reshape(29)

O/P)

```
array([[10,11,12,...,18],
       [19,20,...,27]])
```

then print

P) print(d1.shape)

O/P) (...)

P) d1.shape = (3,2,3)

print(d1)

O/P)

```
[[[10 11 12]
  [13 ... 15]]

 [[16 17 18]
  [19 20 21]]

 [[22 23 24]
  [25 26 27]
  [28 29 30]]]
```

P) print(d1.shape)

O/P) (3,2,3)

P) s1.shape = (3,6)

print(s1)

print(s1.shape)

O/P)

```
[[10 ---- 15]
 [16 17 ... 21]
 [22 23 ... 27]]
(3,6)
```

* To change original shape of array "d.shape = (3,6)" is NOT

Attributes of a numpy array

1) dtype – Returns data type of the array

2) ndim – Returns the number of dimensions

3) shape – Returns the number of rows and columns (starting from 2D and onwards)

4) size – It returns the total number of elements.

P)
```
g1 = np.arange(1,10).reshape(3,3)
print(g1)
print(g1.dtype)
print(g1.ndim)
print(g1.shape)
print(g1.size)
```

O/P)
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
int32
2
(3,3)
9
```

np.linspace()

Generates linearly spaced elements over a specified range

P)
```
a = np.linspace(2,10,3)
print(a)
```

O/P)
```
[2. 6. 10.]
```

linearly spaced means difference between 6 and 2 and between 10 & 6 is same

P)
```
a1 = np.linspace(2,10,5)
print(a1)
```

O/P)
```
[2. 4. 6. 8. 10.]
```

P)
```
a2 = np.linspace(2,10,4)
print(a2)
```

O/P)
```
[2. 4.6666667 7.3333 10.]
```

P) a3 = np.linspace(5,30,)
print (a3.size)
print(a3)

O/P)-
50
[5. 5.5102040 6.0202040 ...
... 30.]

*By default it prints 50 linear space elements

P) a3 = np.linspace(5,3, retstep=True)

↓ it will gives the difference value.

5.5102040 − 5;
= 0.51020408...

↓ Based on this value the next elements are printed.

P) a4 = np.linspace(2,15,10)
print(a4)

O/P)
[2. 3.444 4.8889 ...
... 15.]

np.dot()

Compute the dot product of the arrays.

Matrix multiplication Rule

→ A(m,n) and B(p,q) is matrix multiplication compatible if n=p

→ Resultant shape of matrix after multiplication is (m,q)

P) a1 = np.arange(1,7).reshape(2,3)
print(a1)
print(a1.shape)
a2 = np.arange(-10,5).reshape(3,5)
print(a2)
print(a2.shape)

O/P)
[[1 2 3]
[4 5 6]]
(2,3)

[[-10 -9 -8 ... -6]
[-5 -4 -3 ... -1]
[0 1 2 3 4]]
(3,5)

Then
P) res = np.dot(a1,a2)
print(res)
print(res.shape)

O/P)
[[-20 -14 -8 -2 4]
[-65 -50 ... ...-5]]
(2,5)

P)

```
t1 = np.arange(1,7).reshape(2,3)
t2 = np.arange(-10,5).reshape(5,3)
```

#Now we cant perform dot Product of matrices

#for dot product, Transpose t2

```
print(t1.shape)
print(t2.T.shape)
res1 = np.dot(t1,t2.T)
print(res1)
```

O/P:-

(2,3)
(3,5)

```
[[-52 -34 -16  2  20]
 [-133 -88 -43  2  47]]
```

Q) Program to check armstrong number.

```
def armst(n):
    ln = len(str(n))
    temp = n
    arm = 0
    while temp > 0:
        rem = temp%10
        arm = arm + rem**ln
        temp //= 10
    if arm == n:
        print(f'{n} is armstrong')
    else:
        print(f'{n} is not armstrong')
```

O/P:-

153 is armstrong
1634 is ''
476 is not ''

Q) Program to check Palindrome

```
def Palin(n):
    temp = 0
    Pal = 0
    while temp > 0:
        rem = temp%10
        Pal = Pal*10 + rem
        temp //= 10
    if Pal == n:
        print(f'{n} is Palindrome')
    else:
        print(f'{n} is not an Palindrome')

Palin(12321)
Palin(1634)
```

O/P:-

12321 is Palindrome
1634 is not Palindrome.

Q) Program to check Palindrome in string.

```
def Pal_str(s):
    if s == s[::-1]:
        print(f'{s} is Palindrome')
    else:
        print(f'{s} is not an Palindrome')

Pal_str("racecar")
Pal_str("python")
```

O/P:-

racecar is Palindrome
Python is not an Palindrome.

Q) To check prime

```
def prime_chk(n):
    for i in range(2,n):
        if n%i ==0:
            return false
    return True
print (prime_chk(6))
print (prime_chk(2))
```

O/P:-
```
false
True
```

we also write.

```
for i in range(2,20):
    print(i, prime_chk(i))
```

O/P:-
```
2 True
3 True
4 false
5 True
```

P) Another method.

```
def prime_chk(n):
    c=0
    for i in range(2,n):
        if n%i ==0:
            c=0
            break
        else:
            c=1
    if c==1:
        print(f'{n} is prime')
    else:
        print(f'{n} is not prime')
prime_chk(6)
prime_chk(2)
```

O/P:-
```
6 is not prime
2 is not prime
```

P) Another method.

```
def prime_chk(n):
    c=0
    for i in range(1,n+1):
        if n%i ==0:
            c+=1
    if c==2:
        print(f'{n} is prime')
    else:
        print(f'{n} is not prime')
for i in range(2,20):
    prime_chk(i)
```

O/P:-
```
2 is prime
3 is prime
4 is not prime
5 is prime
```

④ Counting vowels in a string

P)
```
def vowel_count(s):
    v='aeiou'
    c=0
    for i in s.lower():
        for i in v:
            c+=1
    print(f'{s} has {c} vowels')
vowel_count("malayalam")
vowel_count("welcome to python coding")
```

OP)

Malayalam has 4 vowels

welcome to python coding has 7 vowels

P)
```
def vowel_count(s):
    d={'a':0,'e':0,'i':0,'o':0,'u':0}
    v='aeiou'
    for i in s:
        if i in v:
            d[i]+=1
    print(d)
vowel_count("welcome to Argentina")
```

OP)

{'a':1, 'e':3, 'i':1, 'o':2, 'u':0}

P)
```
s="welcome to Argentina"
v='aeiou'
r1=[i for i in s if i in v]
print(r1)
```

OP)

['e', 'o', 'e', 'o', 'e', 'i', 'a']

P)
```
res=Counter(r1)
print(dict(res))
```

OP)

{'e':3, 'o':2, 'i':1, 'a':1}

⑤ write a function to fetch any key corresponding to a value using the value as the argument. Assume any dictionary

P)
```
def fetch_key(d,val):
    for i,j in d.items():
        if j==val:
            print(i)
d={'k3':20,'k2':10,'k1':17,'k4':22,'k5':45}
fetch_key(d, val=17)
fetch_key(d, val=22)
```

OP)

k1

k4

k6

Q) write a function to find all the prime numbers b/w two numbers n1 and n2 where n1 & n2 are passed as arguments.

```
P) def prime_chk(n):
       for i in range(2,n):
           if n%i==0:
               return false
       return True
   def prime_bet(n1,n2):
       n1,n2 = min(n1,n2), max(n1,n2)
       num_n1n2 = list(range(n1,n2+1))
       yes = list(filter(prime_chk, num_n1n2))
       print(yes)
   prime_bet(32,77)
   prime_bet(50,20)
```

O/P

[37,41,43,47, ---,73]

[23,29,31,37,41,43,47].

Q) write a function to find the nth prime number higher than K, where n and K are arguments.

```
P) def prime_higher(n,k):
       for i in range(k, k**2):
           if prime_chk(i):
               primes.append(i)
       print(len(primes))
       print(f'{n}th highest prime after {k} is {primes[n-1]}')
   prime_higher(7,45)
```

O/P

[31,37, 41,43,47, ---, 113]

4th highest prime after 30 is 43.

P) prime_higher(84,63)

[67, 71, 73, ---- , 163]

84th highest prime after 63 is 557

OOPS :-

Class, object

```
P) class Student:          ##constructor
       def __init__(self, rno, name, dob, course=None):
           self.rno = rno          #instance variable
           self.name = name        #  "
           self.dob = dob          #  "
           self.course = course    #  "
       def greet(self):            #instance methods
           print(f'Hello {self.name}, welcome to oops session!')
   ST = student4(5, 'Ankit', '1998-01-01')
   #S1 is an object of class Student
```

```
print(s1.rno, s1.name, s1.dob, s1.course)
print(s1.greet())

print('*' *30)
s2 = student(7, 'Nitin', '1999-05-18',
                                'BTech')
print(s2.rno, s2.name, s2.dob, s2.course)
```

O/P :-

5 Ankit 1998-01-01 None

Hello Ankit, welcome to OOPS session.

None

****

7 nitin 1999-05-18 BTech