juswinspj@gmail.com,
kunalaher0538@gmail.com,
itsvaradkodgire@gmail.com

and 8805200924
and 9594397472

why programming lang ?
==========
Problem solving

2 types of problem:-
==========
>>Compile Time Problem / Static - quantified
>>Run Time PRoblem / dynamic - cannot be quantified

What comes first ?
==========
CT - RT

My Problem in 100% run Time - Scripting Lang (python)
My Problem in 100% compile time - Programming lang(java)
My problem in 80% RT & 20% CT - python & C/C++/LUa
My PRoblen in 80% CT & 20% RT - programming lang(java)

Scripting Lang:-
==========
>> Unix Shell Script
>> Perl
>> Tcl Tk
>> Java Script
>> Python
>> Power Shell

Program Life Cycle:-
==========
app.py------>app.PYC----->output

app.PYC is also called as PYTHON BYTE CODES
PRE-Compiled python file
portable

Note:
python code are interpreted

==========
What is Heap Area ?
- Zero wastage of memory
- problem - Memory Leakage
- solution - GC

What is GC ?
- Memory House keeper

What is Reference ?
- only reference variable

What is NameSpace ?
- book keeper of reference variables
- display book keeper - dir()

What is shallow copy ?
- num = 10
- temp = num
- print(num) # 10
- print(temp) # 10
- print(num is temp) # True

- for mutable data structure - synchronization
- for im-mutable data structure

What is Reference counting ?
- when rc increment - shallow copy
- when rc decremented - variable goes out of scope

Guess:-

```
alst = ["hello", "this", "that", "then"]

blst = alst

alst[0] = "*"
alst[1] = "*"

print("-".join(blst))
```

A) ["hello","this", "that", "then"]
B) Error we cannot modify the list elements
C) ["that", "then"]
D) ["*", "*"]
E) "*-*-that-then"

==========

Terms:-

=======

class - udt - Table
object - instance of a class - Row
data mem - properties of an object - column labels
method - operations on an object - custom operations

how to create a object - var = new classname()

access data member - objectref.datamember

message passing - objectref.methodname()

class name = Account
Data members = num/name/type/balance
methods = check()/with()/dep()/isSavings()

we have this info/data
1234 siva sb 25000
1235 vivek cur 45000

step1 # create objects
acc1 = Account()

```
 acc2 = Account()

step2 # set the data
 acc1.num=1234
 acc1.name="siva"
 acc1.type="sb"
 acc1.bal = 2500
 acc2.num=1235
 acc2.name="vivek"
 acc2.type="cur"
 acc2.bal= 45000

step3 # withdraw 5k from vivek account & verify ?
 acc2.withdraw(5000)
 acc2.check()

step4 # deposit 2k to siva & verify ?
  acc1.deposit(2000)
  acc1.check()

step5 # is it valid / invalid  ?
  acc3.deposit(5000)  # invalid

step6 # is it valid / invalid
  acc1.delete_account() # invalid

step7 # transfer 3k from siva to vivek
  acc1.witdraw(3000)
  acc2.deposit(2000)
  acc1.check()
  acc2.check()



Story
we have readymade class named : list
                methods     : append(ITEM)
                            : pop(index)
                            : sort()
                            : reverse()

we have this data
    mango/grape/orange/berry
```

we have another data
   lizzy,shanavi,hana, nick

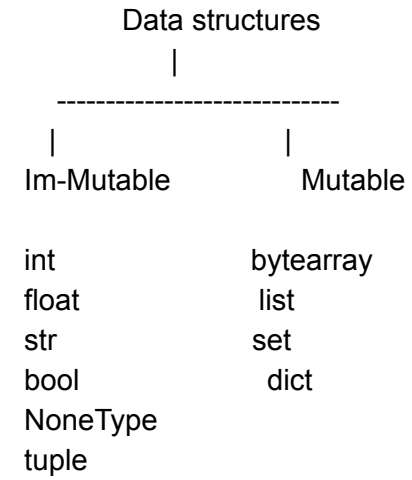
```
#step1:
fruits = list()
frnds  = list()

#step2 init
fruits.append("mango")
fruits.append("grapes")
fruits.append("orange")
fruits.append("berry")
print(fruits)

frnds.append("lizzy")
frnds.append("shanavi")
frnds.append("hana")
frnds.append("nick")
print(frnds)

#step3 delete lizzy
frnds.pop(0)
print(frnds)

# sort the fruits
fruits.sort()
print(fruits)

# reverse the frnds
frnds.reverse()
print(frnds)
```

==========
How to access the help in python?
==========

```
help(classname)
help(builtin_function)
help(classname.methodname)
```

==========
data type = programming lang

data structures - scripting lang



visually identify the data structures:-
==========
num=10
ht=2.5
result=True
output=None
name="KUNAL" # double quotes
name='KUNAL' # single quotes
name='"KUNAL"' # triple quotes

weeks=("sun","mon","tue","wed") # round brackets - tuple
numlst=[10,20,30,40] # square brackets - LIST
grps={"alpha", "beta", "delta"} # flower brackets - set
encode={ # dict
    "blr" : 10,
    "chn" : 20
    "mum" : 30
   }

what is the difference b/w list and tuple ?
  list - vector - mutable
  tuple - vector - im-mutable

what is the difference b/w string & byte string ?
   str - scalar - unicode
   byte - scalar - ascii

what is the diff b/w set & list ?
   list - vector - can have duplicates   - can be index/sliced
   set - vector - cannot have duplicates - cannot be index/sliced

Term:-
=======
Inplace operations
Outplace operations

How can we modify a Mutable data structure ? - Inplace
How can we modify a Im-Mutable   ? - Outplace

Rule1 : Scalar + Scalar
     : Vector + Vector

Rule2 : types should be same

ex1:-
=====

```
num1 = 10
num2 = 20

res = num1 + num2

print(res)  #
```

Ex2:-
====

```
num1 = 10
num2 = "lizzy"

#res = num1 + num2
```

```
#print(res)  # failed

res = str(num1) + num2

print(res)  # success
```

Ex3:-
=====

```
num1 = 25

num2 = "30"

res = num1 + num2

res = num1 + int(num2)  # success
```

Ex4:-
======

```
a = [10,20,30]
b = [40,50,60]

res = a + b

print(res)
```

Ex5:-
=====

>> Result should be a LIST

```
a = [10,20,30]   # list
b = (40,50,60)   # tuple

res = a + b

print(res)  # fails

res = a + list(b)
```

```
print(res)


Ex6:
=====
a = [10,20,30]
b = 40

res = a + b

print(res) # fails

res = a + [b]

print(res) # success
```

==========

top websites for python developers:-

==========

1) compiler - python.org - only python
anaconda.com - data sci dev (community edition)

2) online docs - www.docs.python.org

https://docs.python.org/3.12/reference/lexical_analysis.html#identifiers
https://docs.python.org/3.12/library/functions.html
https://docs.python.org/3.12/library/exceptions.html#exception-hierarchy

3) Coding Standard - pep8.org - pep257

4) Repo - play store - www.pypi.org

5) IDE - VS CODE + Co pilot
- PyCharm
- Jupyter notebook
- colab
- juputer lab

==========

==========
first python program:-
==========
```
import os    # load the library

num1 = input("Enter first : ")
num2 = input("Enter second : ")

res = num1 + num2

print(res)
print("RESULT = ",res)          # PREFERED PRACTICE
print("RESULT = "+str(res))     # stop using this
print("RESULT = "+repr(res))    # stop using this

print("Sum of %s and %s is %s" %(num1,num2,res))          # PREFERED WAY
print("Sum of {0} and {1} is {2}".format(num1,num2,res))
print(f"sum of {num1} and {num2} is {res}")
```

kunalaher0538@gmail.com and 9594397472
itsvaradkodgire@gmail.com and 8805200924

==========
==========
string-class:- (Nut & Bolts)
==========
>> Im-mutable data structure
>> char set = unicode - utf-8 (english & latin)
- utf-16
- utf-32

```
>> name="sri krishna"
   name='sri krishna'
   name='''sri krishna'
   name=r"neha\nhad\ndosa"
   name=R"neha\nhad\ndosa"
```

1) define a string : a="bengaluru"
2) string length : res=len(a)
3) Indexing : first element = a[0]
last element = a[-1]
4) Slicing : a[start:stop:step]

complete string : a[:]
alternate elem   : a[::2]/a[1::2]
reverse a string : a[::-1]
first 4   = a[:4]
last 4   = a[-4:]
Except first 4   = a[4:]
Except last 4   = a[:-4]

5) Search for substr   : if "substr" in a
                          if "substr" not in a
6) Split   : flst = data.split("DELIMITER")
7) join   : res = "DELIMITER".join(LIST)

Demo:-
======
Problem : only first letter should be converted to upper case
Hint   : python string class has a method - .upper()
                                           - .lower()

Given   : name="aditya"
Expected: res="Aditya"

solution:-

```
name="aditya"

first  = name[0].upper()
remain = name[1:]

res = first + remain

print(res)
```

Demo:-
======
Problem : convert last 2 letters to uppercase
Hint   : python string class has a method - .upper()
                                           - .lower()

Given   : name="aditya"
Expected: res="aditYA"

solution:-

```
name="aditya"

el2  = a[:-2]
last2 = a[-2:].upper()
res = el2 + last2
print(res)
```

Task:-
=======
Given:-

```
name = "pradyumn"
```

Expected:-

```
res = "PRadyuMN"
```

Problem : for a given string we have to convert first 2 & last2 letter

Duration : 5 mins
Time   : 11.50 to 11.55

solution:

```
name = "pradyumn"

f2 = name[:2].upper()
l2 = name[-2:].upper()
ef2el2 = name[2:-2]

res = f2 + ef2el2 + l2

print(res)
```

Search for a substr:-
==========
>> "in" is an membership operator in python
>> linear search

```
      #          1         2         3
      #0123456789012345678901234567890123456789012
sent = "today is a monday workday weekday"

print(sent.index("is"))   # 6
print(sent.index("day"))  # 2
print(sent.rindex("day")) # 30
print(sent.find("day"))   # 2


print("day" in sent) # True

print("hello" in sent) # False

see later:-
------------
 sent.count("day")
 sent.index("day")
 sent.rindex("day")



date="22-10-2024" what is the delimiter/fieldseparator/column separator? hyphen

what is the delimite in time ?   =
  time="12:29:30"  colon

what is the delimiter in
  data = "deepansh loves to speak"  # white space

what is the delimiter in
  data="192.168.1.15"   # fullstop/dot

data="10#20#30#40"
    pound


Demo for split:-
================
dob = "15-aug-1947"

flst = dob.split("-")
```

```
print(flst)   # ["15", "aug", "1947"]
```

Task:-
======
Given:-
=======
```
sent = "kunals favourite dish is chicken"
```

Expected:-
==========
first word = kunals
last word = chicken
2nd word from the last = is
second words last letter = e
last words first 3 letter = chi

duration : 5 mins
Time   : 12:40 to 12:45

solution:-
==========
```
sent = "kunals favourite dish is chicken"

flst = sent.split()

print("First word = ",flst[0])
print("Last word  = ",flst[-1])
print("2nd word from last = ",flst[-2])
print("secnd words last letter = ",flst[1][-1])
print("Last words first 3      = ",flst[-1][:3])
```

Demo:-
======
```
data = "ravi-blr,chn,mum,hyd"

flst = data.split("-")

print(flst)   #["ravi",  "blr,chn,mum,hyd"]
```

```
res = flst[1].split(",")

print(res)   #[ "blr",  "chn",   "mum", "hyd"]
```

OR

```
res = data.split("-")[-1].split(",")
```

OR

```
res = data.replace("-",",").split(",")[1:]
```

Demo:-

```
=======

nums = ["10", "20", "30", "40"]

print(sum(nums)) #  throw an exception - TypeError
                 #  since it is a collection of string
```

Demo for Join:-

```
================

data = ["alpha","beta","delta","omega"]

res = "-".join(data)

print(res)  # "alpha-beta-delta-omega"
```

key take away from string-class:-

```
--------------------------------
>> indexing
>> slicing
>> search
>> split
>> join
```

Branching statements:-
==========
>> relational operator - a==b,a!=b,a>b,a>=b, a<b, a<=b
>> logical operator - if rashid and mobile
if rashid or mobile

```
age = int(input("Enter u r age : "))

if age>=10 and age<=20:
  print("yes")
elif age>=21 and age<=30":
  print("no")
else:
  print("try again")
```

>> iterable - collection
>> iterator - cursor which move on the collection

Iterator:-
==========

```
cities = ["blr", "mum", "chn", "del", "hyd", "kol"]

for elem in cities:
  print("Hello", elem)
```

elem - print("Hello",elem)
blr - print("hello",blr)
mum - print("Hello",mum)
chn - print("Hello",chn)
del - print("Hello",del)
hyd - print("HEllo",hyd)
kol - print("Hello",kol)

write a python program to generate nos from 1 to 10

```
for index in range(1,11,1):
    print(index)
```

forward iterator
reverse iterator
index based iterator - range()
parallel iterator   - zip()
enumerate iterator   - enumerate()

keyword:-

>>break
>>continue
>>while loop

```
num=1

while num<=10:

  if num%3==0:
     continue

  print(num)
  num+=1
```

key take away:-
==========
>> if-else
>> for iterator
==========
==========
Task:-
======
Given
cities = ["blr", "mumbai", "chn", "delhi", "hyd", "kol", "pune"]

Expected:-
==========
hello BR
hello MI
hello CN

hello DI
hello HD
hello KL
hello PE

Duration : 5 mins
Time   : 3.15 to 3.20

```
cities = ["blr", "mumbai", "chn", "delhi", "hyd", "kol", "pune"]

for elem in cities:
    print("Hello", elem[0].upper()+elem[-1].upper())
```

Demo1:-
======

```
arr = [10,11,12,13,14]

#for(i=0;i<5;i++)
            #[0,1,2,3,4]
for index in range(0,len(arr),1):
  arr[index] = arr[index] ** 2
  #print(index, arr[index])
print(arr)
```

When do we index based for iterator in python ?
   >> only to modify a list

FAQ:-
======

```
range(1,5,1)   ->   [1,2,3,4]
range(1,5,2)   ->   [1,3]
range(1,5)     ->   [1,2,3,4]
range(5)       ->   [0,1,2,3,4]
```

example for parallel iterator:-

```
list1 = ["fr", "idli", "idli", "vada", "upma"]
list2 = ["akhand","kmil", "shivesh","vinod","kunal"]
```

```
for person,dish in zip(list2,list1):
    print("%s had %s" %(person,dish))
```

example for enumerate:-
==========
```
b = ["akhand","kmil", "shivesh","vinod","kunal"]

print(list(enumerate(b))

[(0, 'akhand'),
 (1, 'kmil'),
 (2, 'shivesh'),
 (3, 'vinod'),
 (4, 'kunal'),
]


for index, value in enumerate(b):
    print(index,value)
```

Lab Task:-
==========
Task1:-
=======
Given:-
=======
```
name = "harshavardhan"
```

Expected:-
==========
```
res1 = "harshavardhaN" # convert the last letter to upper case
res2 = "HarshavardhaN" # convert the first & last letter to upper case
res3 = "harshavar-DHAN"# Convert the last 4 letters to upper case
res4 = "harshavar-NDHD"# Convert the last 4 letters to upper case & reverse it
```

Task2:-

=======
sent = "today is monday workday weekday"

using .count method display no of times "day" is repeated
using .index method get the index all the "day"

Expected:-
==========
Total no time "day" is repeated is = 4

first occurance index = 2
second occurance index = 12
third occurance index = 20
fourth occurace index = 28

solution-
=========
sent = "today is monday workday weekday"

pos=-1
for _ in range(sent.count("day")):
    pos = sent.index("day",pos+1)
    print(pos)

Task3:-
=======
data = "15-aug-1947 10:30:45"

using split method split the above data

display the following data

Expected:-
----
date is = 15-aug-1947
day = 15
month = aug
year = 1947
time is = 10:30:45
hours = 10

mins = 30
secs = 45

Task4:-
========
data = "ravi-sales-20,10,40,30"

Expected:-
----
total sales = 100
min = 10
max = 40
avg = 20.00

```
marks = [int(e) fore in data.split("-")[-1].split(",")]
print(min(marks))
print(max(marks))
print(sum(marks)/len(marks))
```

Task5:-
=======
Using for loop, write and run a Python program for this algorithm.
Here is an algorithm to print out n! (n factorial) from 0! to 10! :
1. Set f = 1
2. Set n = 0
3. Repeat the following 10 times:
a. Output n, "! = ", f
b. Add 1 to n
c. Multiply f by n

Task6:-
========
Modify the program above using a while loop so it prints out all of the factorial values that are less than 2 billion. (You should be able to do this without looking at the output of the previous exercise.)

Task7:-
=======
Write a program that asks the user how many days are in a particular month, and what day of the week the month begins on (0 for Monday, 1 for Tuesday, etc), and then

prints a calendar for that month. For example, here is the output for a 30-day month that begins on day 4 (Thursday):

```
S  M  T  W  T  F  S
          1  2  3
4  5  6  7  8  9  10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

Task8:-
========
Define a procedure histogram() that takes a list of integers and prints a histogram to the screen. For example, histogram([4, 9, 7]) should print the following:

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

Task9:-
=======
Given:-
========
data = [10,20,30,40,50]

Expected:-
---
res="10-20-30-40-50"

Task10:-
=========
Given:-
========
data = "10-50-25-12-85"

Expected:-
==========
res = "11-51-26-13-86"

Task11:-

========
using for loop
print the following outputs

output1:-

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

output2:-

A
B B
C C C
D D D D

==========
names = ["shiva", "pradyumn", "deeksha", "vinod", "sam"]

```
for elem in names:
   print("%s" %(names))   # left aligned


for elem in names:
   print("%10s" %(names))   # right aligned
```

==========
tuple-class:-
==========
>> im-mutable
>> read-only vector

1) define a tuple   : weeks=("sun","mon","tue","wed")
                        weeks="sun","mon","tue","wed"
2) length   : res = len(weeks)
3) search for "mon" : if "mon" in weeks

4) indexing : same as string
5) slicing : same as string
6) compare 2 tuples : if a==b
7) merge : c=a+b

tuple unpacking:-
---
a=10
b=20
c=30

OR

1) a,b,c = 10,20,30

2) a,b,c = 10,20,30,40,50 # fails

3) a,b = 10,20
a,b = b,a

4) *a,b,c = 10,20,30,40,50

a,*b,c = 10,20,30,40,50

a,b,*c = 10,20,30,40,50

*a,b,c = 10,20,30

*a,b,c = 10,20

*a,*b,c = 10,20,30,40

Demo:-
=======

dob = "15-aug-1947"

flst = dob.split("-")

print(flst[0])
print(flst[1])

```
print(flst[2])
```

OR

```
day,month,year = dob.split("-")

print(day)
print(month)
print(year)
```

Guess:-
========

```
data = "arun-sales-10-20-30-40-blr"


name,dept,sales,loc = data.split("-")
```

FAQ:-
=====
Given:-
----
num=4530

Expected:-
----
four five three zero

solution:-
===========
```
#   0     1    2     3     4     5    6     7     8     9

num=4530
a=["zero","one","two","three","four","five","six","seven","eight","nine"]


          #["4","5","3","0"]
for elem in str(num):
   print(a[int(elem)],end=" ")
```

==========
list-class:-
==========
>> mutable data structure
>> inplace operations

1) define a empty list : alst=[]/alst=list()
2) define a list : alst=[10,20,30,40,50]
3) length : res=len(alst)
4) search for 30 : if 30 in alst
: alst.index(30)
: alst.count(30)
5) append at end : alst.append(ITEM)
6) del based on index : alst.pop(INDEX) / alst.pop()
del based on value : alst.remove(10) # first occurance
del a slice : del alst[:3]
7) sort asc order : alst.sort()
desc order : alst.sort(reverse=True)
8) reverse a list : alst.reverse()

Note:
---
alst.extend(blst)
alst.insert(0,25)
compare 2 lists alst==blst
merge 2 lists c=alst+blst

sum(alst)
max(alst)
min(alst)
sorted(alst)

filter the data
transformation
collect the required data

Demo:-

Given:-
---
numlst = [10,15,13,20,18,16]

Expected:-
---
print(odds) # [15,13]
print(evens) # [10,20,18,16]

Solution:-
---

numlst = [10,15,13,20,18,16]
odds,evens = [],[]

for num in numlst:
    if num%2==0:
        evens.append(num)
    else:
        odds.append(num)

print(odds)
print(evens)

Task:-
======
names = ["harish", "manava", "abhishek", "amruth", "yash"]

Expected:-
==========
print(res) # ["Hh", "Ma", "Ak", "Ah", "Yh"]

Duration : 5 mins
Time   : 12.05 to 12.10

solution1:-
---
names = ["harish", "manava", "abhishek", "amruth", "yash"]
res=[]
for name in names:

```
    temp = name[0].upper() + name[-1].lower()
    res.append(temp)
print(res)
```

solution2:-

```
names = ["harish", "manava", "abhishek",  "amruth",  "yash"]
res=[ name[0].upper() + name[-1].lower()  for name in names ]
print(res)
```

Solution3:-

```
names = ["harish", "manava", "abhishek",  "amruth",  "yash"]
res=list(map(lambda name : name[0].upper()+name[-1], names))
print(res)
```

Demo:-
======
```
names = ["hari-sales", "manava-accts", "abhi-finan",  "amrut-purch"]
```

Expected:-
==========
```
res = ["sales", "accts", "finan", "purch"]


sol1 = [e.split("-")[1] for e in names]
sol2 = list(map(lambda x : x.split("-")[1], names))
```

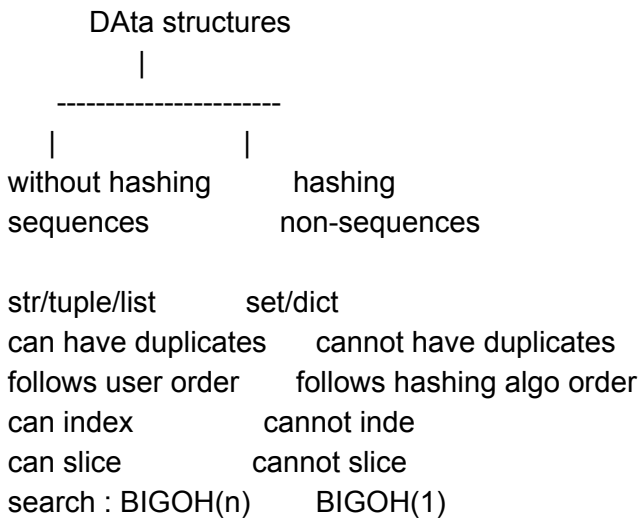
Custom Sort:-
==========
```
datlst = ["ravi-85","manu-80","arun-75","hari-60","guru-50","mani-65"]

# sort on the first field/first column
datlst.sort()
print("\n".join(datlst))

# sort based on marks
datlst.sort(key = lambda x : int(x.split("-")[1]) )
print("\n".join(datlst))
```

Task:-
=======

data = ["Q1=50", "Q2=60" , "Q3=40", "Q4=45"]

Expected:-
---
print(res) ["Q3=40", "Q4=45", "Q1=50", "Q2=60"]
==========
What is hashing ?
==========
1) UNIQUE KEYS
2) HASHING FUNCTION
3) BUCKETS

4) Collissions
5) re-hashing



set-class:-
==========
>> mutable collection
>> cannot have duplicates
>> no indexing
>> no slicing
>> SET members should Im-Mutable

1) define a empty set : a=set()

2) define a set : a={10,20,30}
b={20,25,40}
3) union : res = a.union(b) / c=a|b
4) intersection : res = a.intersection(b) / c=a&b
5) diff : res=a-b
6) symm diff : res=a^b

Demo1:-
---
a = { 10,20,30,10,20,30,40}

print(len(a))

A) 7
B) error
C) 4
D) 0

Demo2:-
========

a={10,"hai", 2.5, (5,6), [7,8]}

print(len(a))

A) 5

B) 8

C) Error

D) 0

dict-class:-
==========
>> collection of key-value pairs
>> key should be unique - Im-Mutable

value can have duplicates - Mutable/Im-Mutable

>> Key------>Value BIGOH(1)
Value---->Key BIGOH(n)

1) define a empty dict : atab={}
2) define a dict : atab={"red" : 10,
"blue" : 20,
"green" : 30}
3) length : res=len(atab)
4) get value "blue" : atab["blue"]/atab.get("blue")
5) delete "red" : atab.pop("red")
6) search for black : if "black" in atab
7) overwrite green - 55 : atab["green"] = 55
8) add new key pink-60 : atab["pink"] = 60

Demo:-

```
dishes = { "idli"  : 30,
           "poori" : 50,
           "dosa"  : 60,
           "upma"  : 40,
           "poha"  : 45
         }

select  = input("Enter u r selection : ")
if select in dishes:
    print("Selected item = ",select)
    print("Prices        = ",dishes[select])
else:
    print("Selected item not found", keys(dishes))
```

Task:-
======
```
studs = {
          "neha" : "dbda-10-20-30-blr",
          "riju" : "dac-40-50-60-pune",
          "hari" : "emb-70-80-90-del",
          "john" : "web-12-13-14-mum"
         }
```

Expected:-

Enter student name : hari
marks = [70, 80, 90]
Total = ?

solution:-

```
name = input("Enter student name :")

if name in studs:
   marks = studs[name].split("-")[1:-1]
   marks = list(map(int, marks))
   print("Marks = ",marks)
   print("Total = ",sum(marks))
```

Loop over dict:-

```
atab = {
        "kar" : "blr",
        "tn"  : "chn",
        "ker" : "tpuram",
        "mah" : "mum"
       }
```

MEthod1:- Loop over the dictionary using key

```
           #["kar", "tn", "ker", "mah"]
for key in  atab:
    print(key, atab[key])
```

```
   key     atab[key]
1) "kar"   atab["kar"]  ---> "blr"
2) "tn"    atab["tn"]   ---> "chn"
3) "ker"   atab["ker"]  ---> "tpuram"
4) "mah"   atab["mah"]  ---> "mum"
```

MEthod2:-
==========

```
atab = {
        "kar" : "blr",
        "tn"  : "chn",
        "ker" : "tpuram",
        "mah" : "mum"
       }
                #  key,  value
                #[("kar", "blr"), ("tn", "chn"), ("ker", "tpuram"), ("mah", "mum")]
for key,value  in  atab.items():
    print(key,value)
```

Demo:-

```
ptab = {
        "DVD" :  55,
        "MON" :  60,
        "PRN" :  30,
        "CPU" :  25,
        "HDD" :  75
       }
```

Expected:-

==========

```
prod_name_above_50 = ["MON", "HDD", "DVD"]
```

```
ptab = {
        "DVD" :  55,
        "MON" :  60,
        "PRN" :  30,
        "CPU" :  25,
        "HDD" :  75
       }
prod_name_above_50 = []

for key,value in ptab.items():
    if value >= 50:
        prod_name_above_50.append(key)

print(prod_name_above_50)
```

Frequency count:-
==========
Given:-
---
datalst = ["alpha", "beta", "alpha", "delta", "beta","omega"]

Expected:-
---
print(freqcnt) # {
"alpha" : 2,
"beta" : 2,
"delta" : 1,
"omega" : 1
}

solution:-

# elem
datalst = ["alpha", "beta", "alpha", "delta", "beta","omega"]

# define a empty a dict to store the frequency count
freqcnt = {}

# loop over the data list
for elem in datalst:
    # check if the elem is in freqcnt dict
    if elem in freqcnt:
        # increment the value by 1
        freqcnt[elem] = freqcnt[elem] + 1
    else:
        # add elem as key & value as 1
        freqcnt[elem] = 1

print(freqcnt)

Key Take away from dict:-

>> Design a a dict
>> get value a given key
>> loop on the dict
>> frequency cnt

Lab Task:-
==========
Task1:-
========
Given:-
---
data = "ravi-blr-math=50,sci=40,soc=30"

Problem:-

>> find the total from the above data

Expected:-

Total marks = 120

Task2:-
========
Given:-
---
names = ["ravi", "arun", "raja", "amit", "Ankur", "hari"]

Problem:-

>>filter out names starting with "a"
>>Ignore the case
>>Store the resultant in a new list

Expected:-
---

res = ["arun", "amit", "Ankur"]

Task3:-
=======
sales = ["dvd-50", "prn-30", "mon-10", "hdd-55", "cpu-20"]

Problem:-

>>filter out product name if quantity >=40
>>Store the resultant in a new list
>>filter out product name if quantity <40
>>Store the resultant in a new list

Expected:-
----
above40 = ["dvd", "hdd"]
below40 = ["prn", "mon", "cpu"]

Task4:-
=======
cities = ["blr", "chn", "mum", "hyd", "del"]
grps = ["blr", "del", "noida"]

Problem:-

>>find the common city names between two lists
>>Don't use set operations

Expected:-
----
common = ["blr", "del"]

Task5:-
=======
Given:-
----
nums = [1,2,3,4,5]

Problem:-

>> square each number - INPLACE OPERATION

Expected:-
---
print(nums) # [1,4,9,16,25]

Task6:-
========
grp1 = ["red=10", "blue=20", "green=30", "black=40"]
grp2 = ["orange=50", "brown=45", "red=5", "black=33"]

Problem:-

>> using set operations - find the common colours b/w them

Expected:-
==========
print(res) # {"red", "black"}

Task7.1:-
========
Given:-
---
num=4503

PRoblem:-

>> using dictionary

Expected:-
---
four five zero three

Task7.2:-

Given="eight-zero-one-four"

Expected:-

8014

Task8:-
=======
grps = ["alpha", "beta", "delta", "alpha", "beta", "omega","alpha"]

Problem:-
----
find the unique & duplicate values in above list

Expected:-
----
print(unique)   # ["delta", "omega"]
print(duplicates) # ["alpha", "beta"]

Task9:-
=======
Given:-
----
names = ["ravi", "arun", "raja", "amit", "ankur", "harish"]

Problem:-

>> Convert the first and last letter to upper case each string

Expected:-
----
print(res) # ["RavI", "AruN", "RajA", "AmiT", "AnkuR", "HarisH"]

Task10:-
========
grp1 = {"blr" : 5, "chn": 5, "hyd": 5 , "del" : 5 }
grp2 = {"blr" : 1, "mum": 2, "noida": 3, "del" : 4 }

Expected:-
==========
print(res) # {"blr" : 6,
             "chn" : 5,
             "hyd" : 5,
             "del" : 9,
             "mum" : 2,

```
        "noida": 3}
```

Nested data:-
==========
a=((10,20),(30,40),(50,60))

```
>> tuple within a tuple/ nested tuple
>> print(a[0]) (10,20)
>> print(a[-1][-1])    # 60
>> a[0] = 0            # INVALID
>> a.append((0,0))     # INVALID
>> a[0][0] = 25        # INVALID
```

a=([10,20], [30,40], [50,60])

```
>> list within a tuple
>> print(a[-1])     # [50,60]
>> print(a[0][0])   # 10
>> a[-1] = 55       # INVALID
>> a.append([1,2])  # INVALID
>> a[0].append(55)  # VALID
>> a[0][0] = 25     # VALID
```

a=[(10,20), (30,40), (50,60)]

```
>> tuple within a list
>> print(a[-2])    #  [30,40]
>> print(a[0][-1]) #  20
>> a[-1] = 55      #  VALID
>> a.append([1,2]) #  VALID
>> a[0].append(55) #  INVALID
>> a[0][0] = 25    # INVALID
```

a=[[10,20],[30,40],[50,60]]

```
>> list within a list
>> print(a[1])     # [30,40]
>> print(a[-2][0]) # 30
>> a[-1] = 55      # valid
```

```
>> a[0].append(55) # valid
>> a[0][0] = 25    # valid


a={
   "aviral" : [10,20],
   "sahil"  : [30,40],
   "varda"  : [50,60],
   "balaji" : [70,80],
  }
>> list within a dict
>> print(a["aviral"][0]) ---> 10
>> print(a["varda"])    ---> [50,60]
>> add 25 to balaji     ---> a["balaji"].append(25)

>> a["varda"][-1] ===> 60
>> a["avrial"])   ====> [10,20]
>> if "agna" in a ====> False
>> a["sahil"].pop() ====>  delete 40


a={
   "loc1" : {"city"  : "blr",
             "mem"   : ["a", "b", "c" ]
            },
   "loc2" : {"city"  : "chn", "mem" : ["p", "q"]},
   "loc3" : {"city"  : "hyd", "mem" : ["x","y", "z"]},
  }


>> dict within a dict - nested dict
>> a["loc1"] --->  {"city"  : "blr", "mem" : ["a", "b", "c" ]}
>> a["loc2"]["city"]
>> a["loc1"]["mem"][0]
>> search for "p" in loc2 in mem   -  if "p" in a["loc2"]["mem"]


File Handling:-
===============
>> to save the data permanently for the future use
>> copy the RAM DATA to HARD DISK
>> Every file will have 3 reference points
   BOF
```

```
   CUR
   EOF
>> File Agent unders stands only STRING
>> int   -> str
   float -> str
   tuple -> str
   list  -> str
   set   -> str
   dict  -> str

>> File modes
    r - readonly   -  BOF
    w - overwrite  -  BOF
    a - append     -  EOF
    r+ - read-write   -BOF
    w+ - overwrite read-BOF
    a+ - append-read   -EOF


How to write into a file:-
--------------------------
#fileagent_name = open(filename, file_opening_mode)

f1 = open("data.txt", "w") # open the file in teh current directory
#f1 = open(r"c:\that\this\data.txt", "w")   #
#f1 = open("c:\\that\\this\\data.txt", "w")   #
#f1 = open("c:/that/this/data.txt", "w")   #

f1.write("hello\n")
f1.write("hai\n")
f1.write("20\n")
f1.write("30\n")

f1.close()


How to read from the file :-
============================
f1 = open("data.txt", "r")

for elem in f1:
    print(elem)
```

```
f1.close()
```

Auto Close the File / Context Mgr:-
==========

```
f1 = open("data.txt", "r")
res = f1.read()
print(res)
f1.close()
```

OR

```
with open("data.txt", "r") as f1:
    res = f1.read()
    print(res)
```

File handling & Exception Handling:-
==========

```
try:
   f1 = open("data.txt",)

except FileNotFoundError as e1:
   print("What to do")

else:
   buffer = f1.read()
   print(buffer)

finally:
   f1.close()
```

Other fns:-

```
strbuffer = f1.read()     # complete file & return it as a string
strbuffer = f1.read(1024) #
strbuffer = f1.readline()
lstbuffer = f1.readlines()# read complete file & returns it as a LIST
```

data.txt:-

==========
ramya is getting ready for deepavali
deeksha is also getting ready
ashok ready to eat sweets
vinay is ready to fire crackers in sasural

```
res=["ramya is getting ready for deepavali",
"deeksha is also getting ready",
"ashok ready to eat sweets",
"vinay is ready to fire crackers in sasural"]
]
```

```
res = fob.readlines()
print(res[0])     # "ramya is getting ready for deepavali",
print(res[-1])    #
print(res[0].split()[-3])  #
print(res[-1].split()[0]) #
```

```
res="ramya is getting ready for deepavali"

res = fob.readline()
print(res[0])     # r
print(res[-1])    # i
print(res.split()[0])  # ramya
print(res.split()[-1]) # deepavali
```

```
res='''ramya is getting ready for deepavali
deeksha is also getting ready
ashok ready to eat sweets
vinay is ready to fire crackers in sasural
'''

res = fob.read()
print(res[0])     # r
print(res[-1])    # l
print(res.split()[0])  # ramya
print(res.split()[-1]) # sasural
```

Demo:-
=======

```
fob = open("one.txt", w")
fob.write("arun-50-blr\n")
fob.write("ravi-10-chn\n")
fob.write("john-20-mum\n")
fob.write("yash-40-hyd")
fob.close()
```

Task:-
======

```
f1 = open("data.txt", "w")

f1.write("ravi kumar\n")
f1.write("harish prasad\n")
f1.write("sameer simha\n")
f1.write("arun kumar")
f1.close()
```

RK - kumar
HP - prasad
SS - simha
AK - kumar

solution:-
==========

```
with open("Data.txt", "r") as fob:
    for line in fob:
        first,last = line.split()
        print((first[0]+last[0]).upper(), last)
```

key take away from file handling

>> open & Close a file
>> Write
>> read
>> exception handling & Files
>> with keyword

==========
==========
Function:-
==========
>> collection of statement
>> reduces the complexity of the program
>> sub job/sub program/ sub task

>> based on arguments
    positional args - def fun(src,dest,oper):
    default args   - def fun(src="one.txt", dest="two.txt", oper="copy"):
    hybrid args   - def fun(src, dest, oper="copy")

>> postional args
>> nested fns

Demo1:-
========
```
def add2nums(num1,num2):        # function signature
   res = num1 + num2            # function body
   return res


a=10
b=20
ans = add2nums(a,b)
print(ans)
```

Note:
 all the variable defined within function
 and
 function argument are LOCAL VARIABLES

1) function-name   - add2nos
2) how many does it accept - 2

3) type args - first arg - int/float
second arg - int/float
4) type of function - positional/compulsory
5) does it return - yes
6) local vars - num1,num2,res
7) how to call the fn - ans = add2nums(5,6)

Demo2:-
========
```
def square(alst):
  newlst = []
  for num in alst:
     newlst.append(num*num)
  return newlst
```

1) function-name - square
2) how many does it accept - 1
3) type args - first arg - list
4) positional/default/hybrid- positional
5) does it return - yes
6) local vars - alst, newlst, num
7) how to call the fn - ans = square([5,6,7,8])

Demo3:-
========
```
def filter_dict(atab,limit=50):
  newlst = []
  for key,value atab.items():
     if value>=limit:
       newlst.append(key)
  return newlst
```

1) function-name - filter_dict
2) how many does it accept - 2
3) type args - first arg dict
second arg int
4) positional/default/hybrid- hybrid
5) does it return - yes
6) local vars - key,value,atab,limit,newlst
7) how to call the fn - ans=filter_dict({"a":60,"b":40,"c":80})

```
ans=filter_dict({"a":60,"b":40,"c":80},70)
```

Demo4:-

```
========
def get_word(filename="one.txt",letter="a"):
  res=[]
  with open(filename,"r") as fob:
      for line in fob:
          for word in line.split():
              if word[0]==letter:
                 res.append(word)
  print(res)
```

one.txt

```
--------
arun works on
data above and
also this done
```

1) function-name - get_wrod
2) how many does it accept - 2
3) type args - first arg - string
   - second args - string
4) positional/default/hybrid- default args
5) does it return - no
6) local vars - filename,letter,res,fob,line,word
7) how to call the fn -

Rule:

any variable declared/defined outside the function are GLOBAL VARIABLE
any variable declared/defined inside the function are LOCAL VARIABLE

Guess:-

```
========
def fun():
   print("Inside function = ",num)
```

```
num=10
print("Main num =",num)
fun()
```

A) Main num = 10
  Inside function = 10
B) Main num = 10
  Inside function = 0
C) Main num = 10
  Error
D) Error

Guess:-
========

```
def fun():
    num=20
    print("Inside function = ",num)

num=10
print("Main num =",num)
fun()
```

Guess:-
========

```
def fun():
    global num
    num=20
    print("Inside function = ",num)

num=10
print("Main num =",num)
fun()
print("END of main num =",num)
```

output
---
main num = 10
inside function = 20

end of main num = 20

Keyword args:-

==========

>> pass args - random

>> while calling a function we specify the parameter explicitly

```
def operations(num1=5,num2=10,oper="add"):
    print("Num1 = ",num1)
    print("Num2 = ",num2)
    print("Oper = ",oper)


operations(oper="quot")       #  num1=5   num2=10  oper="quot"
operations(num2=50, num1=40)  #  num1=40  num2=50  oper="add"
```

Variable non-keyword Args:-

==========

```
def fun(args):
   print(args)

fun(1,2,3)          #  error
fun(1,2,3,4,5)      #  error
fun(1)              #  works
fun()               #  error
fun(1,2)            #  error


def fun(*args):
   print(args)

fun(1,2,3)          #  works
fun(1,2,3,4,5)      #  works
fun(1)              #  works
fun()               #  works
fun(1,2)            #  works
```

Variable Keyword Args:-
==========
```
def fun(kwargs):
  print(kwargs)

fun(src=10, dest=20)      # error
fun()                     # error
fun(a=10,b=20,c=30,d=40)  # error
fun(kwargs=5)             # work


def fun(**kwargs):
  print(kwargs)

fun(src=10, dest=20)      # works
fun()                     # works
fun(a=10,b=20,c=30,d=40)  # works
fun(kwargs=5)             # works
```

Variable args:-
==========
```
def fun(*args, **kwargs):
  print(args)
  print(kwargs)

fun(10,20,a=30,b=40)  # args=(10,2) kwargs={"a":30,"b":40}
fun(5,6)
fun()
fun(p=5,q=6)
```

Guess:-
========
```
def fun(alst):
   alst[:3] = [0]*3       #  alst[0] = 0  alst[1] = 0  alst[2] = 0


numlst = [10,20,30,40,50,60]
print("before = ",numlst)
fun(numlst)
```

print("after = ",numlst) # what is the outptut here

A) [10,20,30,40,50,60]
B) [0,0,0,40,50,60]
C) None of the above
D) Error

such function are called as "CALL BY REFERENCE"

Nested Functions:-

```
def outer():
   a=10
   b=20
   def inner():
      c=30
      res=a+b+c
   inner()

outer()
inner()  # error
```

Lab Task:-
==========
Task1:-

```
depts={
       101 : "sales",
       102 : "purch",
       103 : "accts",
       104 : "finan"
      }
emps = {
        "arun" : "blr-101-alpha",
        "ravi" : "chn-104-beta",
```

```
  "hari" : "hyd-102-delta",
  "manu" : "del-103-omega"
 }
```

Expected:-

```
Enter the emp name : ravi
    location     : blr
    dept id      : 104
    dept name    : finan
    proj name    : beta

Enter the emp name : john
Error - Invalid emp name
```

Task2:-

Define a procedure histogram() that takes a list of integers and prints a histogram to the screen. For example, histogram([4, 9, 7]) should print the following:

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

Task3:-

Write a version of a palindrome recognizer that also accepts phrase palindromes such as "Go hang a salami I'm a lasagna hog.", "Was it a rat I saw?", "Step on no pets", "Sit on a potato pan, Otis", "Lisa Bonet ate no basil", "Satan, oscillate my metallic sonatas", "I roamed under it as a tired nude Maori", "Rise to vote sir", or the exclamation "Dammit, I'm mad!". Note that punctuation, capitalization, and spacing are usually ignored.

Task4:-

A pangram is a sentence that contains all the letters of the English alphabet at least once, for example: The quick brown fox jumps over the lazy dog. Your task here is to write a function to check a sentence to see if it is a pangram or not.

Task5:-

In cryptography, a Caesar cipher is a very simple encryption techniques in which each letter in the plain text is replaced by a letter some fixed number of positions down the alphabet. For example, with a shift of 3, A would be replaced by D, B would become

E, and so on. The method is named after Julius Caesar, who used it to communicate with his generals. ROT-13 ("rotate by 13 places") is a widely used example of a Caesar cipher where the shift is 13. In Python, the key for ROT-13 may be represented by means of the following dictionary:

key = {'a':'n', 'b':'o', 'c':'p', 'd':'q', 'e':'r', 'f':'s', 'g':'t', 'h':'u', 'i':'v', 'j':'w', 'k':'x', 'l':'y', 'm':'z', 'n':'a', 'o':'b', 'p':'c', 'q':'d', 'r':'e', 's':'f', 't':'g', 'u':'h', 'v':'i', 'w':'j', 'x':'k', 'y':'l', 'z':'m', 'A':'N', 'B':'O', 'C':'P', 'D':'Q', 'E':'R', 'F':'S', 'G':'T', 'H':'U', 'I':'V', 'J':'W', 'K':'X', 'L':'Y', 'M':'Z', 'N':'A', 'O':'B', 'P':'C', 'Q':'D', 'R':'E', 'S':'F', 'T':'G', 'U':'H', 'V':'I', 'W':'J', 'X':'K', 'Y':'L', 'Z':'M'}

Your task in this exercise is to implement an encoder/decoder of ROT-13. Once you're done, you will be able to read the following secret message:

Pnrfne pvcure? V zhpu cersre Pnrfne fnynq!

Note that since English has 26 characters, your ROT-13 program will be able to both encode and decode texts written in English.

Task6:-

Write a Python program to sort a tuple by its float element.
Sample data:
[('item1', '12.20'), ('item2', '15.10'), ('item3', '24.5')]
Expected Output:
[('item3', '24.5'), ('item2', '15.10'), ('item1', '12.20')]

Task7:-

Write a Python program to count the elements in a list until an element is a tuple.

Sample input : list = [10, 20, 30, (40,50), 60]

Sample output = 3

Task8:-

Write a Python program to compute element-wise sum of given tuples, using “zip()” function

Original tuples:

(1, 2, 3, 4)
(3, 5, 2, 1)
(2, 2, 3, 1)

Element-wise sum of the said tuples:
(6, 9, 8, 6)

Task9:-

Given a dictionary of students and their favourite colours:
people={'Arham':'Blue','Lisa':'Yellow',"Vinod:'Purple','Jenny':'Pink'}

1. Find out how many students are in the list
2. Change Lisa’s favourite colour to "Purple"
3. Remove 'Jenny' and her favourite colour
4. Sort and print students and their favourite colours alphabetically by name

Task10:-

Write a function translate() that will translate a text into "rövarspråket" (Swedish for "robber's language"). That is, double every consonant and place an occurrence of "o" in between. For example, translate("this is fun") should return the string "tothohisos isos fofunon".

function - collection of statements
module - collection of functions
package - collection of modules

Modules:-
==========
>> collection of fns,variables,classes
>> .PYC
>> load a module
1) import module_file_name

2) from module_file_name import *
>> fully qualified name (FQN)
relative name (REL)

what is the FQN of my city "blr" ?
-worldmap.asia.india.kar.blr
-is independent of the current location

What is the REL of my city "blr"?
-is dependent of the current location
-if we are asia - india.kar.blr
india - kar.blr
kar - blr

How to write user defined module:-
---
mathoper.py

```
def add(a,b):
  print(a+b)

def square(num):
  print(num**2)

def greet():
  print("Hello from Ramya")
```

another.py:-

```
import mathoper   # we are outside mathoper.py

mathoper.add(10,20)
mathoper.square(5)
mathoper.greet()
```

Diff ways to load a library:-
---
1) import mathoper

```
  mathoper.add(10,20)
  mathoper.square(5)
  mathoper.greet()
```

2) import mathoper m

```
  m.add(10,20)
  m.square(5)
  m.greet()
```

3) from mathoper import *   # we are inside mathoper

```
  add(10,20)
  square(5)
  greet()
```

4) from mathoper import add # we are inside mathoper

```
  add(10,20)
```

Demo:-

------

alpha.py

```
def greet():
    print("Happy Morning")
```

beta.py

```
def greet():
    print("Happy Evening")
```

main.py

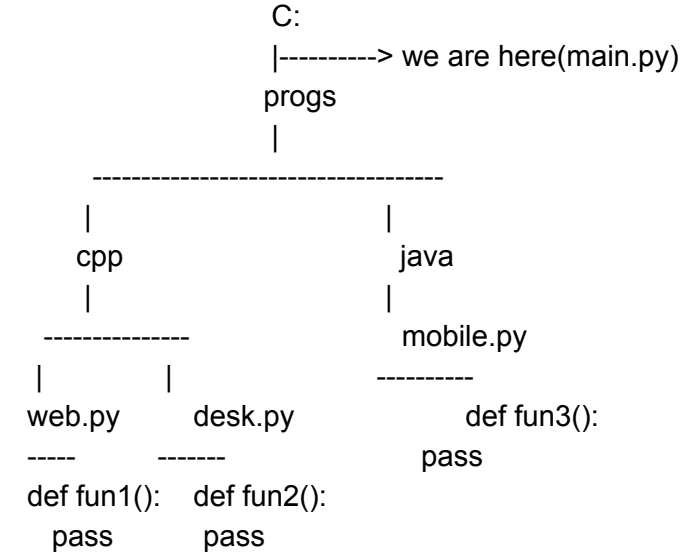
```
from alpha import *   # we are inside alpha.py
from beta import *    # we are inside beta.py

greet()
```

see later:-

>>name clashes
>>what is env variable "PYTHONPATH" / sys.path   - Library search path

Packages:-
==========
>> folder
>> collection of modules & sub packages
>> __init__.py

>> from package import *   # load a package



we need to call fun1,fun2,fun3:-
---
from progs import *   # we are inside "progs"

cpp.web.fun1()
cpp.desk.fun2()
java.mobile.fun3()

we need to call fun1,fun2

```
from progs.cpp import *    # we are inside "cpp"

web.fun1()
desk.fun2()
```

we need to call fun1

```
import progs.cpp.web

progs.cpp.web.fun1()
```

Generator:-
==========
>> Python function can return only once

```
def invest():
  return 1
  return 2
  return 3
  return 4
  return 5

res = invest()
print(res)
```

>> python generator function can return multiple times
```
def invest():
  yield 1
  yield 2
  yield 3
  yield 4
  yield 5

res = invest()
print(res)

print(list(res))#  LUMPSUM
```

OR

```
next(res) # 1
next(res) # 2
next(res) # 3
next(res) # 4
next(res) # 5
next(res) # StopIteration
```

Decorator:-

==========

banking.py

```
def alert(fnref):
    def wrapper(*args,**kwargs):
        print("before")
        res = fnref(*args,**kwargs)
        print("after")
        return res
    return wrapper

@alert
def withdraw():
  pass

@alert
def deposit():
  pass

@alert
def transfer():
  pass

def checkbalance():
  pass
```

Database Connectivity:-
==========

```
import sqlite3    # loading db driver

con = sqlite3.connect("master.db")

cur = con.cursor()

cur.execute("create table query")

cur.execute("insert the query")

con.commit()

cur.execute("Select * from table")

for elem in cur.fetchall():
    print(elem)

cur.close()
con.close()
```

NOSQL
NEWSQL
VECTORDB

Regular Experssion:-
==========

```
name = "varun"

if name == "arun"      #   False   - String Matching

if "arun" in name      #    True   - static pattern matching
```

regex/re - dynamic pattern matching
regex meta chars
. = a char

[] = range of chars
^   = line starts with
$ = line ends with
?   = zero/one - {0,1}
*   = zero/more - {0,}
+   = one/more - {1,}
               - {2,6}
               - {5}
{m,n} =
regex char classes
[0-9]   = \d
[0-9a-zA-Z] = \w
[\t\n]   = \s

how write a regex for date dd/mm/yyyy

\d{2}-\w{3}-\d{4}

```
import re   # load a readymade library (CORE LIBRARY)

dob = input("Enter u r dob : ")

if re.search("^\d{2}-\w{3}-\d{4}$",dob):
   print("u have entered date correctly")
else:
   print("u made a mistake")
```

==========

html
xml

```
htmldata='''
<html>
<head> hello </head>
<title> world </title>
<body>
<h1>  first </h1>
</body>
</html>'''
```

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(htmldata, "html.parser") # prepare a tree

print(soup.head.text)
print(soup.title.text)
print(soup.body.text)
print(soup.body.h1.text)
```

Lab Task:-
==========
Task1:-
========

```
f1 = open("data.csv", "w")
f1.write("name,loc,salary\n")
f1.write("arun,blr,25000\n")
f1.write("hari,chn,45000\n")
f1.write("john,mum,30000\n")
f1.write("manu,hyd,35000")
f1.close()
```

>>run the above program - it will create a csv file
>>read the csv file store this data in a DATABASE TABLE

Task2:-
=======
From the DATABASE TABLE, READ THE TABLE contents and
store them in a text file

Task3:-
=======
Write program to find the most repeated word in a file

Assume the file "data.txt" contains
---
this that this this this that
then that that that
that this then that

Expected:-

Most repeated word in the file is = this

Task4:-
=======
Define a function overlapping () that takes two lists and returns True
if they have at least one member in common, False otherwise.

Task5:-
=======
Write a function find_longest_word() that takes a list of words and returns the
length of the longest one.

Task6:-
=======
Write a function filter_long_words() that takes a list of words and an integer n
and returns the list of words that are longer than n

Task7:-
=======
Define a simple "spelling correction" function correct ()
that takes a string and sees to it that

1)two or more occurrences of the space character is compressed into one, and

2)inserts an extra space after a period if the period is
directly followed by a letter.

e.g. correct ("This is very funny and cool.Indeed!") should return

"This is very funny and cool. Indeed!"

Task8:-
=======
In English, present participle is formed by adding suffix -ing to infinite form: go ->
going. A simple set of heuristic rules can be given as follows:
a)If the verb ends in e, drop the e and add ing
  (if not exception be, see, flee, knee, etc.)
b) If the verb ends in ie, change ie to y and add ing
c)For words consisting of consonant-vowel-consonant, double the final
  letter before adding ing
d) By default, just add ing

Your task in this exercise is to define a function make_ing_form() which given a verb in

infinitive form returns its present participle form. Test your function with words such as lie, see, move and hug. However, you must not expect such simple rules to work for all cases.

Task9:-
=======

Write a program to display the first and last word of a given file

Enter a filename : one.txt

contents of one.txt:-
---
hello world of unix was
the output of the above
program which was given

Expected output is :-
---
hello - was
the - above
program - given

Task10:-
========

write a program to convert the file contents of upper case

Enter the file name : one.txt

Enter the output file : out.txt

contents of one.txt:-
---
hello world of unix was
the output of the above
program which was given

contents of out.txt:-

HELLO WORLD OF UNIX WAS
THE OUTPUT OF THE ABOVE
PROGRAM WHICH WAS GIVEN

BS4:-(web scrapping)

==========

numpy:-

=======

pandas:-

=========

>>class & objects
>>Exception handling