Day3

- Loops
- Functions
- Strings
- Data Structures

Loops

- for iterator in iterable
- range()

In [4]:

```
# read a string from user and find the length of it
# LBR Engineering College #i/p
# o/p:
clg=input("I am from ")
cnt=0 #
for ch in clg:
    #print(ch,end=" ")
    cnt+=1
print('length of string=',cnt)

```

...

In [2]:

```
len(clg)
```

Out[2]:

```
23
```

In [5]:

```
LBR Engineering College
```

...

In [6]:

```
# print
# i/p:LBR Engineering College
#o/p:L**********************e
```

In [11]:

```
n=len(clg)
```

In [12]:

```
clg[0]
```

Out[12]:

```
's'
```

In [16]:

```
clg[-1]
```

Out[16]:

```
'r'
```

In [27]:

```
clg=input("I am from ")
n=len(clg)
for ix in range(n):
    if ix==0 or ix==n-1:
        print(clg[ix],end="")
    else:
        print("*",end="")
```

```
I am from LBR Engineering College
L********************e
```

while

- a condition based loop
- user incrementation
- iterator should initialize before starting loop
- it will go for infinite no.of times until you stop
- *syntax*
  - while condition:
    - statements

In [28]:

```
# print the multiplication table of a number
```

In [29]:

```
val=int(input("enter the number:"))
for dig in range(1,11):
    print(val,'x',dig,"=",val*dig)
```

In [30]:

```
# using while loop
num=1
while num<=10:
    print(num,end=" ")
    num+=1
```

```
1 2 3 4 5 6 7 8 9 10
```

In [31]:

```
#  nth multiplication table
n=int(input())
it=1
while it<=10:
    print(n,"x",it,'=',n*it)
    it+=1
```

. . .

In [32]:

```
# find the digits in a number
#12345:5
num=input()
cnt=0
for dig in num:
    cnt+=1
print("no.of digits=",cnt)
```

```
897653
no.of digits= 6
```

In [33]:

```
s=0 # finding the sum of digits in a number
for dig in num:
    s+=int(dig)
print("sum:",s)
```

```
sum: 38
```

In [37]:

```
# looping in reverse order
stop=25
while stop>=1:
    print(stop,end=" ")
    stop-=2
```

```
25 23 21 19 17 15 13 11 9 7 5 3 1
```

In [38]:

```
for num in range(50,10,-1):
    print(num,end=" ")
```

...

In [41]:

```
# printing even numbers present in a range
n=int(input())
if n%2==0:
    for num in range(n,50,2):
        print(num,end=" ")
else:
    for num in range(n,50,2):
        print(num,end=" ")
```

...

In [43]:

```
# using while
n=int(input())
if n%2==0:
    while n<=50:
        print(n,end=" ")
        n+=2
else:
    while n<=50:
        print(n,end=" ")
        n+=2
```

...

In [44]:

```
# print the above values in reverse order
```

Nested Loops

- loop existed in another loop
  - inner loop in outer loop
- example
  - a matrix will consists of m cols in n rows

In [45]:

```
# 8 9 0
# 5 6 7
```

In [46]:

```
for num in range(1,6): # 5 times
    print(num,end=" ")
```

```
1 2 3 4 5
```

In [48]:

```
for num in range(1,29): # 28 times
    #print(num,end=" ")
    if num%2==0:
        print(num,end=" ")
```

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28
```

In [49]:

```
for num in range(1,6): # 5 times
    for clg in range(1,4): # w r t 4
        # num=1,num=2,num=3,3 # 15
        print(clg,end=' ')
    print()
```

. . .

In [50]:

```
for row in range(1,6):
    for col in range(1,row+1): # 1,2
        print(col,end=" ")
    print()
```

. . .

In [53]:

```
n=int(input())
for row in range(1,n+1): #12345
    for col in range(1,row+1): #
        if row%2==0: #
            print("*",end=" ")
        else:
            print("#",end=" ")
    print()
```

. . .

In [61]:

```
# print the pattern in reverse order
n=int(input())
for num in range(n,0,-1): # 10,9,8,7..1
    for col in range(num,0,-1):
        if col%2==0 and col<n//2:
            print("*",end=" ")
        elif col%2==0 and col>n//2:
            print("#",end=" ")
        else:
            print("@",end=" ")
    print()
```

. . .

Functions

- A block of statements to peform a specific task
- def is the keyword that represents the function
- 2types of functions

    1. predefined function
        - were defined when the language is designed
        - no need to define those again
        - simply we can call and use those functions
        - ex:print(),input(),type(),int(),bin(),ord(),len()
    2. user defined
        - defined by the user/programmer
- **syntax**
    - def function_name(args):
        - statements
            - return
    - in the function definition
        - the main purpose writing code/actual code
        - formal values/dummy varibles
    - in function call
        - we will use that function
            - parameters are there
            - actual data
- reusability of code
- functions follows the modularity

In [62]:

```
## we can define the function in 4 ways
def add(x,y): # list of argument and return
    return x+y

add(9,4)
```

Out[62]:

```
13
```

In [63]:

```
def function():
    pass
```

In [65]:

```
function()
```

In [66]:

```
# we can define the fun without args and return
def product():
    return a*b

a,b=int(input()),int(input())
# global
product()
```

...

In [67]:

```
# function without arguments without return
def length_of_str():
    st=input()
    print(len(st))

length_of_str()
```

...

In [69]:

```
st #local variable
```

...

In [70]:

```
a
```

...

In [71]:

```
b
```

...

In [87]:

```
# function with args and without return
def i_am(name):
    print("I am ",name)

i_am('kAVYA')
```

```
I am  kAVYA
```

In [73]:

```
name
```

...

In [75]:

```
# you should follow some rules
# default and non-default args
# *args and **kwargs
```

In [80]:

```
def number(*values): # variable length of args
    for ch in values:
        print(ch,end=" ")

number(*[1,2,4,5,6,7,10])
```

```
1 2 4 5 6 7 10
```

In [81]:

```
# default and non-default
def arguments(a=10,b):
    return a+b
arguments(8)
```

...

In [91]:

```
# default and non-default
def args(a,b=9):
    return a+b

args(6)
```

...

In [88]:

```
# default and non-default
def example(a=8,b=9): #
    return a+b

example(int(input("first:")),int(input('second:')))
```

In [93]:

```
example() # empty
```

...

In [94]:

```
example(10)
```

...

In [97]:

```
# define a function that checks whether
# given number is prime or not
num=int(input())
factors=0
# 8:1,2,4,8:0
for dig in range(1,num+1):
    if num%dig==0:
        print(dig,end=" ")
```

...

In [99]:

```
# define a function that checks whether
# given number is prime or not
num=int(input())
factors=0
# 8:1,2,4,8:0
for dig in range(1,num+1):
    if num%dig==0:
        factors+=1
if factors==2:
    print("prime")
else:
    print("not prime")
```

...

In [100]:

```
# using function
def is_prime(n):
    count=0
    for num in range(1,n+1):
        if n%num==0:
            count+=1
    if count==2:
        return True
    else:return False

is_prime(13)
```

Out[100]:

```
True
```

In [101]:

```
# print the prime numbers present in range
for val in range(int(input()),int(input())):
    if is_prime(val):
        print(val,end=" ")
```

...

In [102]:

```
def prime_range(lw,up):
    for num in range(lw,up+1):
        if is_prime(num):
            print(num,end=" ")
prime_range(1,100)
```

...

In [103]:

```
'k'# what is the perfect number
# sum of the factors of a given number equals to itself
# 6:1,2,3=1+2+3=6
def is_perfect(p):
    sm=0
    for dig in range(1,p//2+1):
        if p%dig==0:
            sm+=dig
    if sm==p:
        return True
    else:return False

is_perfect(9)
```

...

In [104]:

```
is_perfect(6)
```

...

In [105]:

```
28:1,2,4,7,14
```

...

Strings

- Group/collection of characters
- anything that is enclosed in the quotations called as string
- str() that represents the string data
- can declare an empty str with either '' or ""
- it is immutable
- dir() to check the list of str methods

In [107]:

```
1 str(909899) # converted int into str
```

Out[107]:

```
'909899'
```

In [108]:

```
1 dir(str)
```

...

In [110]:

```
1 print() # can be empty
```

In [117]:

```
1 name=input('name:') # dynamic str
2 print("I am",name)
```

```
name:Electrical Department
I am Electrical Department
```

In [118]:

```
1 name
```

Out[118]:

```
'Electrical Department'
```

In [119]:

```
1 len(name)
```

Out[119]:

```
21
```

In [120]:

```
1 for ch in name:
2     print(ch)
```

...

Index

- index is the memory address of the char/data
- 2types
    1. Normal Indexing
        - 2ways of traversing

A. +ve indexing
- starts from 0 to +inifinite/len(iterable)-1
- travers from left to right

B. -ve indexing
- travers from right to left
- starts from -1 and upto -inifinite

2. Fancy Indexing
- condition based indexing

In [121]:
```
1 name
```

Out[121]:
```
'Electrical Department'
```

In [122]:
```
1 name[0] # iterable[index]
```

Out[122]:
```
'E'
```

In [123]:
```
1 name[1]
```

Out[123]:
```
'l'
```

In [124]:
```
1 name[5] # 6th char
```

Out[124]:
```
'r'
```

In [125]:
```
1 name[19]
```

Out[125]:
```
'n'
```

In [133]:
```
1 name[::]
```

Out[133]:
```
'Electrical Department'
```

In [134]:

```
name[:]

```

Out[134]:

```
'Electrical Department'
```

In [135]:

```
name[::-1]
```

Out[135]:

```
'tnemtrapeD lacirtcelE'
```

In [136]:

```
name
```

Out[136]:

```
'Electrical Department'
```

In [137]:

```
name[::2] # alternate chars
```

Out[137]:

```
'Eetia eatet'
```

In [131]:

```
name[::-2] # alternate chars in reverse order
```

Out[131]:

```
'tetae aiteE'
```

slicing

- cutting in pieces
- splitting sub part of string using index as a boundary
- name[start:stop:step_count]

In [138]:

```
name[5:14] # retrieving the chars from 5th to 14th
```

Out[138]:

```
'rical Dep'
```

In [139]:

```
1 name
```

...

In [140]:

```
1 name[5:] # 5th to last
```

Out[140]:

```
'rical Department'
```

In [141]:

```
1 name[:16] # from 0 to 15th
```

Out[141]:

```
'Electrical Depar'
```

In [143]:

```
1 name[5:12] # index 5th to 11th
```

Out[143]:

```
'rical D'
```

In [144]:

```
1 name[-1]
```

Out[144]:

```
't'
```

In [145]:

```
1 name[-2] # last but one char
```

Out[145]:

```
'n'
```

In [146]:

```
1 -5,-9
```

Out[146]:

```
(-5, -9)
```

In [147]:

```
1 name[-5:-9] # empty
```

Out[147]:

```
''
```

In [148]:

```
1 name
```

...

In [149]:

```
1 name[-9:]
```

...

In [150]:

```
1 name[4:10]
```

...

In [151]:

```
1 name[-9:-3]
```

...

In [156]:

```
1 name[-3:-9:-1] # in reverse order
```

...

In [153]:

```
1 name
```

...

string methods

- the functions can only be applied on strings
  - str.replace(old,new)
  - str.count(ch)
  - split()
  - strip()
  - lstrip()
  - rstrip()
  - join()
  - format()
  - center()
  - title()
  - capitalize()

- casefold()
- swapcase()
- lower()
- upper()
- islower()
- isupper()

In [157]:

```
1 name
```

Out[157]:

```
'Electrical Department'
```

In [158]:

```
1 name.lower() # converts the str into lowercase alphabets
```

Out[158]:

```
'electrical department'
```

In [159]:

```
1 name.upper()
```

Out[159]:

```
'ELECTRICAL DEPARTMENT'
```

In [160]:

```
1 name.swapcase()
```

Out[160]:

```
'eLECTRICAL dEPARTMENT'
```

In [161]:

```
1 new='EngINeerING At MylAVaraM'
```

In [162]:

```
1 new.swapcase()
```

Out[162]:

```
'eNGinEERing aT mYLavARAm'
```

In [163]:

```
1 new.upper()
```

Out[163]:

```
'ENGINEERING AT MYLAVARAM'
```

In [168]:

```
1 up=new.center(40) # centralize the str with length
2 up
```

...

In [169]:

```
1 len(new)
```

...

In [170]:

```
1 len(up)
```

...

In [171]:

```
1 '@'.join(new) # joins the string with delimeter
```

...

In [174]:

```
1 st='aeiour e v o w e l s'
2 'p'.join(st)
```

...

In [175]:

```
1 up
```

Out[175]:

```
'        EngINeerING At MylAVaraM        '
```

In [180]:

```
1 new.capitalize() # capitalizes the first char of str
```

...

In [181]:

```
1 new.title() # capitalizes the first char of each word
```

...

In [182]:

```
1 up
```

Out[182]:

```
'        EngINeerING At MylAVaraM        '
```

In [184]:

```
1 # remove the spaces on both sides
2 lp=up.lstrip()
3 lp
```

...

In [185]:

```
1 up.rstrip()
```

...

In [186]:

```
1 up.strip() # removes the both side spaces
```

...

In [187]:

```
1 names=input().split() # splits the str at ' '
2  # list()
3 names
```

...

In [188]:

```
1 np="RUthu.apssdc.located at Guntur . hi hello".split('.')
2 np
```

...

In [189]:

```
1 np
```

...

In [190]:

```
1 type(np)
```

...

In [191]:

```
1 names
```

...

In [195]:

```
# .format
nm,org,des='Ruthu','APSSDC','TD'
"I am {0} from {1} working as {2}".format(nm,org,des)
```

...

In [196]:

```
# .format
nm,org,des='Ruthu','APSSDC','TD'
"I am {} from {} working as {}".format(nm,org,des)
```

...

In [199]:

```
name='Nandini' # f-string
roll_id=123908
f"I am {name} and my id is {roll_id}"
```

...

In [212]:

```
# str.replace
# str.index()
# starts.endswith()
# startswith()
ch=input("char:")
st=input("string:")
if st.endswith(ch):
    print("yes")
```

```
char:l
string:Electrical
yes
```

In [214]:

```
st.startswith('E')
```

Out[214]:

```
True
```

In [228]:

```
ch=input("char:")
st=input("string:")
new=input("replacing char:")
if ch in st:
    st=st.replace(ch,new) # str updation
st
```

```
char:o
string:engineering
replacing char:b
```

Out[228]:

```
'engineering'
```

In [203]:

```
dir(str)
```

...

In [204]:

```
help(str.replace)
```

...

In [206]:

```
res
```

Out[206]:

```
'electrical'
```

In [207]:

```
st
```

Out[207]:

```
'electrical'
```

In [208]:

```
name
```

Out[208]:

```
'Nandini'
```

In [210]:

```
name.replace('i','new')
```

Out[210]:

```
'Nandnewnnew'
```

In [211]:

```
1 name
```

Out[211]:

```
'Nandini'
```

In [222]:

```
1 st
```

Out[222]:

```
'elehtrihal'
```

In [223]:

```
1 st[0].isalpha()
```

Out[223]:

```
True
```

In [224]:

```
1 st[3].isdigit()
```

Out[224]:

```
False
```

In [225]:

```
1 st[5].isnumeric()
```

...

In [233]:

```
1 comb='Ruthu@123#890,lavanya%ramu'
2 comb[4].isdigit()
```

Out[233]:

```
False
```

In [247]:

```
1 comb.index('@')
```

Out[247]:

```
5
```

In [248]:

```
comb.index('a')
```

...

In [ ]:

```

```

In [232]:

```
comb[-1].isdigit()
```

...

In [237]:

```
# read a str from user and print the special chars
cmb=input()
new=''
for ch in cmb:
    if ch.isdigit() or ch.isalpha() or ch.isspace():
        new+=ch
    else:
        print(ch,end=" ")
```

...

In [235]:

```
dir(str)
```

...

In [239]:

```
specials="!@#$%^&*,./()<>?"
for ch in specials:
    if ch in cmb:
        print(ch,end=" ")
```

...

In [241]:

```
for ix in range(len(cmb)):
    print(ix,cmb[ix],sep="=")
```

...

In [243]:

```
# read n space separated integers from user
# and find the sum
# 90 45 56 89 23 91 45 3 5 60 23
sm=0
mul=input().split()
for num in mul:
    sm+=int(num)
sm
```

. . .

In [245]:

```
# find the sum of middle digits
# of n space separated values
# n:
# 890 456 834 109 396 290
# sm aa o/p
total=int(input())
nums=input().split()[:total]
sm=0
for dig in nums:
    sm+=int(dig[1])
sm
```

. . .

Introduction to Data Structures

- way of organising the data in a perticular format
- 4types
    1. Tuple
    2. List
    3. Set
    4. Dict

Tuple

- It is one of the data structures in python
- it is heterogenous data structure
- represented by ()
- tuple(ele)
- it is immutable in nature

In [246]:

```
dir(tuple)
```

. . .

tuple methods

- count

- index

In [249]:

```
1 comb
```

Out[249]:

'Ruthu@123#890,lavanya%ramu'

In [254]:

```
1 # tuple
2 tp=tuple(comb)
3 print(tp)
4 t=(12,'hi','new')
5 t
```

...

In [263]:

```
1 tp.count('a')
```

Out[263]:

4

In [268]:

```
1 char='a'
2 count=0
3 for ch in tp:
4     if ch==char:
5         count+=1
6 print(count)
```

4

list

- it is heteroneous data structure
- it is mutable in nature
- list()
- represented by []

In [269]:

```
1 nums
```

Out[269]:

['890', '456', '834', '109', '396', '290', '834']

In [272]:

```
li=[3,4,5,'hi','electrical','python',
   'new','list',90.45]
print(li)
```

```
[3, 4, 5, 'hi', 'electrical', 'python', 'new', 'list', 90.45]
```

In [274]:

```
marks=[97,89,79,90,67,10,70,59,35]
marks[0]
# marks<80
```

...

In [275]:

```
marks[3:]
```

Out[275]:

```
[90, 67, 10, 70, 59, 35]
```

In [276]:

```
marks[-1]
```

...

In [277]:

```
marks[::-2]
```

Out[277]:

```
[35, 70, 67, 79, 97]
```

In [278]:

```
for num in marks:
    print(num)
```

...

In [279]:

```
for ix in range(len(marks)):
    print(ix,marks[ix])
```

...

In [281]:

```
new=[] # new list contains the values<80
for num in marks:
    if num<80:
        new.append(num)
new
```

Out[281]:

```
[79, 67, 10, 70, 59, 35]
```

list methods

- li.append()
- pop()
- remove(value)
- pop(index)
- insert(index,data)
- count()
- sort()
- reverse()
- extend()
- copy()
- clear()
- index()

In [286]:

```
marks.append(345)
```

In [290]:

```
marks.insert(4,500)
```

In [291]:

```
marks
```

Out[291]:

```
[97, 89, 79, 90, 500, 100, 67, 10, 70, 59, 35, 345, 345]
```

In [294]:

```
marks.remove(35) # data/item
```

. . .

In [296]:

```
marks.remove(345)
```

In [298]:

```
marks.remove(marks[3])
```

```
marks
```

In [300]:

```
marks.pop()
```

Out[300]:

345

In [302]:

```
marks.pop(4) # 5th element
```

...

In [303]:

```
marks
```

Out[303]:

[97, 89, 79, 500, 67, 10, 70, 59]

In [ ]:

```
marks.extend([3,4,5])
marks
```

In [305]:

```
marks.append([1,2,3]) # can add any another ds
```

In [307]:

```
marks.pop()
```

Out[307]:

[1, 2, 3]

In [309]:

```
marks.sort()
```

In [311]:

```
marks
```

Out[311]:

[3, 4, 5, 10, 59, 67, 70, 79, 89, 97, 500]

In [312]:

```
# reverse sort
marks.reverse()
```

In [313]:

```
marks
```

...

In [316]:

```
 # second largest element
marks[1]
```

Out[316]:

```
97
```

In [317]:

```
new=marks.copy()
new
```

Out[317]:

```
[500, 97, 89, 79, 70, 67, 59, 10, 5, 4, 3]
```

In [318]:

```
new.index(67)
```

Out[318]:

```
5
```

In [319]:

```
new.clear()
```

In [321]:

```
#new
for ch in input():
    new.append(ch)
print(new)
```

...

In [328]:

```
# how to remove duplicate elements
marks.extend([4,90,70,89,100])
```

In [329]:

```
print(marks)
```

...

In [330]:

```
for item in marks:
    if marks.count(item)>=2:
        marks.remove(item)
print(marks)
```

```
[500, 97, 79, 67, 59, 10, 5, 3, 4, 90, 70, 89, 100]
```

In [333]:

```
first=input().split()
unq=[]
for val in first:
    if val not in unq:
        unq.append(val)
print(unq)
```

```
90 89 hi hello 67 80 45 234 23 89 hi
['90', '89', 'hi', 'hello', '67', '80', '45', '234', '23']
```

SET

- well defined collection of objects
- represented by {}
- set()
- removes the duplicates data items
- mutable in nature

In [334]:

```
dir(set)
```

...

In [ ]:

In [261]:

```
vowels[3]
```

...

In [262]:

```
vowels[-1]
```

...

In [335]:

```
A={1,2,3,4,'hi','hello',9,4,5,'hey',5}
A
```

...

In [337]:

```
A[0] # NOT POSSIBLE
```

...

In [342]:

```
B={4,5,9,3,4,6,'HI','hi',2,3,'python','data'}
B
```

Out[342]:

```
{2, 3, 4, 5, 6, 9, 'HI', 'data', 'hi', 'python'}
```

In [343]:

```
A.intersection(B) # common elements
```

Out[343]:

```
{2, 3, 4, 5, 9, 'hi'}
```

In [344]:

```
A.union(B) # ALL THE ELEMENTS IN BOTH SETS
```

...

In [345]:

```
A.difference(B) #A-B
```

Out[345]:

```
{1, 'hello', 'hey'}
```

In [346]:

```
A
```

Out[346]:

```
{1, 2, 3, 4, 5, 9, 'hello', 'hey', 'hi'}
```

In [347]:

```
A.symmetric_difference(B) # non-smilar elements
```

...

In [348]:

```
1 A.issuperset(B)
```

Out[348]:

```
False
```

In [349]:

```
1 B.issuperset(A)
```

Out[349]:

```
False
```

In [350]:

```
1 A.pop() # first element is removed by default
```

...

In [352]:

```
1 A.remove(9)
```

In [353]:

```
1 A
```

Out[353]:

```
{2, 3, 4, 5, 'hello', 'hey', 'hi'}
```

In [354]:

```
1 A.remove(9)
```

...

In [356]:

```
1 A.difference_update(B)
```

In [357]:

```
1 A
```

Out[357]:

```
{'hello', 'hey'}
```

In [359]:

```
1 A.intersection_update(B) # {PI}
```

In [361]:

```
1 A.union(B)
```

Out[361]:

```
{2, 3, 4, 5, 6, 9, 'HI', 'data', 'hi', 'python'}
```

In [362]:

```
1 dir(set)
```

...

In [363]:

```
1 B.discard('HI')
```

In [364]:

```
1 B
```

Out[364]:

```
{2, 3, 4, 5, 6, 9, 'data', 'hi', 'python'}
```

In [365]:

```
1 B.discard(8) # non-existed
```

In [366]:

```
1 B.add('new')
```

In [367]:

```
1 B
```

...

In [ ]:

```
1
```