#VARIABLES

#SINGLE ASSIGNMENT  
name="MAHESH"  
age=19  
height=5.6  
weight=60  
print(name,age,height,weight)  
  
#MULTIPLE ASSIGNMENT  
name,age,height,weight="Mahesh",19,5.6,60  
print(name,age,height,weight)

#arithmetic operations  
print(2+3)  
print(2-3)  
print(2\*3)  
print(2/3)  
print(2%3)  
  
#normal division  
print(5/2)  
#floor division  
print(5//2)  
  
#combination of arithmetic operations  
print(2+3\*10)  
print((2+3)\*10)  
  
#power  
print(2\*2\*2\*2\*2)  
print(2\*\*5)  
  
import math  
print(math.pow(2,5))  
  
import math as m  
print(m.pow(2,5))  
  
print("hello world")  
print("HELLO WORLD")  
print('HELLO WORLD')  
  
#strings can be printed using "" or ''  
print("SURYA'S LAPTOP")  
print('MAHESH"S book')  
  
#string with both " , '  
#\ backslash simply changes the meaning of the '  
print('mahesh\'s "LAPTOP" is lenovo')  
  
print(" MAHESH is \"good\" boy ")  
  
# printed many strings seperated by commas  
  
print("surya",'mahesh',"HELLO",'mowa')  
  
#strings can be concatenated simply using + sign  
  
print("surya"+'mahesh'+"HELLO"+'mowa')  
  
#printing same string multiple times using \*  
print(5\*"\nSURYA ")  
  
# \n goes to new line  
print("\n hello murugan ")  
print("\n syndicate member \n pushpa pesire")  
  
#for printing the raw string use 'r before that  
  
print(r"\n syndicate member \n pushpa pesire")  
print(r'\n syndicate member \n pushpa pesire')

#variables  
x=10  
print(" value of x = ",x)  
print(" value of x = "+str(x))  
print(" value of x = %d"%(x))  
  
print(x+5)  
print(x-1)  
print(x)  
  
name="SURYA MAHESH.kolisetty"  
print(name)  
# individual characters can be retrieved from the string  
print(name[0])  
  
#back indexing also possible  
print(name[-1])  
  
#slicing  
print(name[:3])  
print(name[1:])  
print(name[2:4])  
  
print(" strings are immutable , but subsets can be retrieved / sliced acc to our wish")  
  
print("MYNAME IS :",name)  
  
print("length of '"+name+"' is : ",len(name))

#LISTS

# Adds List Element as value of List.  
list = ['Mathematics', 'chemistry', 1997, 2000]  
list.append(20544)  
print(list)  
  
List1 = [1, 2, 3]  
List2 = [2, 3, 4, 5]  
  
# Add List2 to List1  
List1.extend(List2)  
print(List1)  
  
# Add List1 to List2 now  
List2.extend(List1)  
print(List2)  
  
list = [1, 2, 3, 4, 5]  
print(sum(list))  
  
list = [1,2, 3]  
print(sum(list))  
  
list = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]  
print(list.count(1))  
  
list = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]  
print(len(list))  
  
list = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]  
print(list.index(2))  
  
list = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]  
print(list.index(2, 2))  
  
list = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]  
  
"""index(element, start, end) : It will calculate till index end-1. """  
  
# will check from index 2 to 4.  
print("After checking in index range 2 to 4")  
print(list.index(1))  
  
# will check from index 2 to 3.  
print("After checking in index range 2 to 3")  
print(list.index(2))  
list = [1,6,2,4,3,9]  
print(min(list))  
  
list = [1,6,2,4,3,9]  
print(max(list))  
  
list = [1,6,2,4,3,9]  
  
# Reverse flag is set True  
list.sort(reverse=True)  
  
# List.sort().reverse(), reverses the sorted list  
print(list)  
  
list = [1,6,2,4,3,9]  
sorted(list)  
print(list)  
  
list = [1,6,2,4,3,9]  
print(list.pop())  
  
list = [1,6,2,4,3,9]  
del list[0]  
print(list)  
  
list = [1,6,2,4,3,9]  
list.remove(3)  
print(list)

#LIST  
  
nums=[1,4,2,3,4  
 ,66,7,8,22,1]

#list can hold any types  
print(nums)  
names=["abc","def","hello","murugan","syndicate","member","pushpa","pesire"]  
print(names)  
#list can be sliced  
print(nums[:4])  
print(nums[2:])  
print(nums[1:5])  
  
  
#list can be sliced  
  
print(names[:4])  
print(names[2:])  
print(names[1:5])  
  
#list can hold multiple types  
values=[1,2,3,"pushpa","syndicate","member",1.0,1+2j]  
print(values)  
print(values[-1])  
  
  
#list can be have another lists as members  
  
mymix=[nums,names,values]  
print(mymix)  
  
print(nums)  
nums.append(1000)  
print(nums)  
nums.insert(3,20000)  
print(nums)  
nums.remove(66)  
print(nums)  
print(nums.pop())  
print(nums)  
nums.pop(3)  
print(nums)  
print(nums.pop())  
  
del nums[5:]  
print(nums)  
  
del nums[:3]  
  
print(nums)  
#extend a list with a list  
nums.extend([1,2,3,4,5,6,7,8,9,0])  
print(min(nums))  
print(max(nums))  
print(sum(nums))  
  
nums=[6,2,5,1,3,8,9,4]  
nums.sort()  
print(nums)  
nums.sort(reverse=True)  
print(nums)  
  
l=[]  
l=nums.copy()  
print(l)  
  
print(nums.\_\_getitem\_\_(1))  
nums.\_\_setitem\_\_(1,100)  
print(nums)  
nums.sort()  
print(nums)  
  
nums.append(1)  
nums.append(1)  
nums.append(1)  
  
  
print(nums.count(1))

import random  
  
#lists are mutable in python  
a=[1,2,3]  
print(" list if ekements in a is: ",a)  
for i in a:  
 print(i)  
 # max , min  
print(max(a))  
print(min(a))  
l=[100,20,1,200,25,1000]  
print(l)  
#sort  
l.sort()  
print(l)  
s=sum(l)  
print("SUM OF ELEMENTS IN THE LIST %d"%(s))  
print("SUM OF ELEMENTS IN THE LIST "+str(s))  
l=[[100,20,1,200,25,1000],[10,20,1,200,25,1000],[100,2,1,20,25,1000]]  
print(l)  
print(l[0])  
print(l[1])  
print(l[2])  
print(l[0][2])  
print(l[2][2])  
j=0  
k=0  
for j in range(len(l)):  
 for k in range(len(l[0])):  
 print(l[j][k])  
  
print(1000 in l[0])  
b=[1,2,3,4,5,6,7,8,9]  
print(b)  
#membership in  
for i in range(len(b)):  
  
 print(b[i])  
  
#built in methods  
b.append(10)  
b.append(20)  
b.append(30)  
b.append([60,70])  
#append takes only argument single one  
print(b)  
b.extend([80,90,100])  
b.extend([110,120,130,[140,150,160]])  
print(b)  
#insert  
b.insert(3,1000)  
print(b)  
#remove  
b.remove(30)  
print(b)  
b.remove(1000)  
print(b)  
#pop  
b.pop()  
print(b)  
b.pop()  
print(b.pop(3))  
#del  
del b[2]  
print(b)  
#shuffle  
c=[1,2,3,4,5]  
random.shuffle(c)  
print(c)  
c.sort()  
print(c)  
#printing reverse  
c=[1,2,3,4,5]  
random.shuffle(c)  
print(c)  
c.sort(reverse=True)  
print(c)  
  
print(c\*2)  
c.sort()  
print(c)  
c.sort(reverse=True)  
print(b)  
  
c.append(10)  
c.append(10)  
c.append(10)  
c.append(10)  
print(c)  
#count  
print(" TOTALLY THERE ARE %d 10'S IN THE LIST"%(c.count(10)))  
#index  
print(c.index(10))  
print(c.index(2))  
  
#reverse  
c.reverse()  
print(c)

a=[1]  
  
a.append([2,3,4,5])  
print(len(a))  
print(a)  
  
#append only adds one element to the list , so a after this is with list [2,3,4,5] as second element  
  
print("Hello Mike".split())

for i,x in enumerate(['A','B','C']):  
 print(i, x)

#LIST SORTED FUNCTION

L=[1,3,2]  
print(sorted(L,reverse=False))  
print(sorted(L))  
print(sorted(L,reverse=True))  
print(L)

#list comprehensions  
d=[]  
print(d)  
d=[ele for ele in range(10)]  
print(d)  
  
d=[ele\*ele for ele in range(10)]  
print(d)  
  
d=[ele\*ele for ele in range(10) if ele%2==0]  
print(d)  
  
d=[ele\*ele for ele in range(10) if ele%2!=0]  
print(d)  
  
e=[]  
for ii in range(20):  
 e.append(ii)  
print(e)  
  
  
#printing even numbers upto 50 using list comprehensions  
  
f=[ele for ele in range(50) if ele%2==0]  
print(f)  
  
print(f.\_\_getitem\_\_(1))  
print(f.\_\_setitem\_\_(2,1000))  
print(f)  
#tuple immutable  
#() comma seperated values in paranthesis  
#multiple datatypes can be hold in the tuple  
  
t=(1,2,3,4,5)  
print(" TUPLE IS : ",t)  
#list can be assigned to tuples  
#TUPLES can be assigned to LISTS  
t=f  
e=t  
print("TUPLE T IS WITH NOW WITH LIST F ELEMENTS ",t)  
e=t  
print("LIST E IS WITH NOW WITH TUPLE ELEMENTS ",e)  
  
t2=tuple(a)  
print(t2)  
#t2[0]="BYE" not possibl in tuples  
print(t2.\_\_getitem\_\_(2))  
  
l2=list(t)  
print(l2)  
l2[0]="hello"  
print(l2)  
  
t=(10)  
print(" NOW TUPLE T IS WITH SINGLE ELEMENT LETS CHECK THE TYPE ",type(t))  
  
t=(10,)  
print(" NOW TUPLE T IS WITH SINGLE ELEMENT BUT WE FOLLOWED THE SYNTAX LETS CHECK THE TYPE ",type(t))  
print("IT's WORKING"+'COOOOOOL')

#LISTS  
elements=[1,"ahgsg",1.1,1+2j]  
print(type(elements))  
for i in range(len(elements)):  
 print(elements[i],end=" ")  
  
elements.append(1000)  
print(elements)  
#lists can take all mixed data types  
#one list can hold another list , tuple ,set strings all the primitive built in data types  
elements=[1,"ahgsg",1.1,1+2j,[1,"ahgsg",1.1,1+2j],(1,2,3,4,5,6),{1,2,3,4,5,6,7,8,9,0}]  
print(elements)  
  
for i in range(len(elements)):  
 print(elements[i])

#COMPREHENSIONS  
  
#list comprehensions  
import math  
  
l=list()  
print(type(l))  
l=[]  
for i in range(10):  
 l.append(i)  
  
print(l)  
  
l=[]  
l=[i for i in range(0,20,2) ]  
print(l)  
  
l=[]  
l=[i for i in range(1,20,2) ]  
print(l)  
  
l=[]  
l=[i for i in range(40) if i%4==0 ]  
print(l)  
  
  
#tuple comprehensions  
  
t=()  
print(type(t))  
  
t=tuple(i for i in range(0,20,2) )  
print(t)  
  
t=tuple(i for i in range(1,20,2) )  
print(t)  
  
t=tuple(i for i in range(0,20,2) if i%4==0 )  
print(t)  
  
  
l=list(i for i in range(0,20,2) if i%4==0 )  
print(l)  
  
s=set(i for i in range(0,20,2) if i%4==0 )  
print(s)  
  
d={i:i\*\*2 for i in range(1,20,2) if i%2==1}  
print(d)  
  
import math  
l=[ i for i in range(0,10)]  
t=tuple( i for i in range(0,10))  
s={ i for i in range(0,10)}  
d={ i:math.pow(i,2) for i in range(0,10)}  
  
  
print(l)  
print(t)  
print(s)  
print(d)

#STRINGS

#strings  
#strings are immutable  
  
s="PYTHON"  
print(s)  
  
#accesing string  
for i in range(len(s)):  
 print(s[i])  
  
print(s," TYPE --->",type(s))  
  
s1="124003143@sastra.ac.in"  
print(s1)  
print(type(s1))  
  
for i in range(len(s1)):  
 print(type(s1[i]))  
  
#i=len(s1)-1  
for i in range(len(s1)):  
 print(s1[0-i],end="")  
  
#slicing  
print("\n SLICING",s[0:])  
print("SLICING",s[:len(s)])  
print("SLICING",s1[:len(s1)])  
  
  
"""hello  
 world"""  
  
s2="""hello  
 world"""  
print(s2)  
  
#.format method  
  
print("{0} \n {1} ".format(1,"hell"))  
# 0 for first 1 for second ..so .on positiinal arguments  
  
print("{0}\n{1} ".format(10,"hell"))  
print("{0}\n{1} ".format(10,10.0))  
print("{0}\n{1}\n{2} ".format(10,10.0,1+2j))  
print("{0}\n{1}\n{2}\n{3} ".format(10,10.0,1+2j,"PESIRE"))

str1="python"  
str2="progamming"  
print(" HEY SHIV COOOL ",str1+" "+str2)  
  
  
print("MIN OF",str1,"BASED ON THE ASCII VALUE ",min(str1))  
print("MIN OF",str2,"BASED ON THE ASCII VALUE ",min(str2))  
print("MAX OF",str1,"BASED ON THE ASCII VALUE ",max(str1))  
print("MAX OF",str2,"BASED ON THE ASCII VALUE ",max(str2))  
  
#whether a chracter in the string present or not  
  
print("p"in str1)  
print("py"in str1)  
print("prog"in str1)  
print("prog"in str2)  
  
for i in range(len(str1)):  
 print(str1[i])  
  
for x in str2:  
 print(x)  
  
#methods in strings  
  
s="python"  
print(s)  
  
print(s.center(20,"\*"))  
print(str2.center(20,"\*"))  
  
  
s="welcome to python programming"  
print("o repeated ",s.count("o")," TIMES in ",s)  
  
  
print(s,"is a string starts",s.startswith("py"))  
print("STRING ENDSWITH ing :",s.endswith("ing"))  
  
print(s.capitalize())  
  
  
# we can find the frequency of each character from particular index to particular index of the string  
s="OOPSSSSSOOOOOOOOPPPPPSSSSOOOOPPPSSS"  
print(len(s))  
print(s.count("O",0,10))  
print(s.count("P",5,len(s)))  
print(s.count("S",5,10))  
  
s="PYHON IS MY LANGUAGEY PYPYPYPPYPY "  
print(" INDEX OF 'Y' IN STRING S IS ",s.find("Y"))  
print(" INDEX OF 'Y' IN STRING S WHEN SEARCHED FROM BACK IS ",s.rfind("Y"))  
print(s.find("Z"))  
  
print("index of 'P' in the string searched from the start : ",s.index("P"))  
print(len(s))  
print("index of 'P' in the string searched from the REVERSE : ",s.rindex("P"))

#REPLACE METHOD

s="AAAAAAAA"  
print(s.replace('A','B'))  
  
s="AAHAHAHAHA"  
print(s.replace('A','B'))  
  
  
s="OOAOAOAOAOAOA"  
print(s.replace('A','B'))

# to check whether the given string is alphanumeric or not  
  
s="abc123"  
#string should be with alpha / numeric / alpha numeric any of these will return true  
print(s," ia aphanumeric string : IS IT TRUE BHAVANI ? :",s.isalnum())  
s="abcdef"  
print(s," ia APLHA string only characters : IS IT TRUE BHAVANI ? :",s.isalpha())  
s="1234@#$qwee"  
print(s," ia ascii string : IS IT TRUE BHAVANI ? :",s.isascii())  
s="abc123"  
print(s," ia digit string : IS IT TRUE BHAVANI ? :",s.isdigit())  
s="123456"  
print(s," ia digit string : IS IT TRUE BHAVANI ? :",s.isdigit())  
s="HELLO MURUGAN NAANU SYNDICATE MEMBER PUSHPA PESIRE "  
print(s)  
s=" "  
print(s," is string with only space :: IS IT TRUE BHAVANI ? ",s.isspace())  
s="HELLO MURUGAN NAANU SYNDICATE MEMBER PUSHPA PESIRE "  
print(s," is string with only UPPERCASE LETTERS :: IS IT TRUE BHAVANI ? ",s.isupper())  
print(s," is string with only LOWERCASE LETTERS :: IS IT TRUE BHAVANI ? ",s.islower())  
print(s.lower())  
s=s.lower()  
print(s," is string with only LOWERCASE LETTERS :: IS IT TRUE BHAVANI ? ",s.islower())  
print(s.upper())  
s=s.lower()

print(s.capitalize())  
print(s.title())  
print(s.ljust(50,"\*"))  
print(s.rjust(50,"\*"))

s="hello man"  
s=s.title()  
print(s.ljust(20,"\*"))  
  
print(s.rjust(20,"\*"))  
  
print(s.center(20,"\*"))

# removing spaces in strings  
s=" PUSHPA PESIRE "  
print(s.strip())  
s=" PUSHPA PESIRE "  
print(s.lstrip())  
s=" PUSHPA PESIRE "  
print(s.rstrip())  
  
s=" PUSHPA PESIRE "  
print(s.replace("P","K"))  
print(s)  
s=" PUSHPA PUSHPA PUSHPA PUSHPA PUSHPA "  
print(s)  
print(s.replace("PUSHPA","PULKA"))  
print(s)  
  
#replacing the string only 2 times with the new string  
print(s.replace("PUSHPA","PULKA",2))  
  
print(s.split(" "))  
s=" PUSHPA\*PUSHPA\*PUSHPA\*PUSHPA\*PUSHPA "  
  
print(s.split("\*"))  
s=" PUSHPA PUSHPA PUSHPA PUSHPA PUSHPA "  
  
print(s.zfill(55))  
s=" PushPA PushPA PushPA PushPA PushPA "  
  
print(s.swapcase())

#TUPLES

#tuples  
  
weeks=("sunday","monday","tuesday","wednesday","thursday","friday","saturday")  
print(weeks)  
print(type(weeks))  
  
weeks="sunday","monday","tuesday","wednesday","thursday","friday","saturday"  
print(weeks)  
print(type(weeks))  
  
for i in range(len(weeks)):  
 print(weeks[i])  
  
  
weeks[2]="SURYADAY"  
  
try:  
 weeks[2]="SURYADAY"  
except:  
 print("SOMETHING WENT WRONG")  
else:  
 print("NO EXCEPTION")  
finally:  
 print("BYE BYE ")

#STRING METHODS

# Python3 program to show the  
# working of upper() function  
text = 'geeKs For geEkS'  
  
# upper() function to convert  
# string to upper case  
print("\nConverted String:")  
print(text.upper())  
  
# lower() function to convert  
# string to lower case  
print("\nConverted String:")  
print(text.lower())  
  
# converts the first character to  
# upper case and rest to lower case  
print("\nConverted String:")  
print(text.title())  
  
# original string never changes  
print("\nOriginal String")  
print(text)  
  
# Python program to demonstrate the  
# use of capitalize() function  
  
# capitalize() first letter of string  
# and make other letters lowercase  
name = "geeks FOR geeks"  
  
print(name.capitalize())  
  
# demonstration of individual words  
# capitalization to generate camel case  
name1 = "geeks"  
name2 = "for"  
name3 = "geeks"  
print(name1.capitalize() + name2.capitalize() + name3.capitalize())  
  
# First character in each word is  
# uppercase and remaining lowercases  
s = 'Geeks For Geeks'  
print(s.istitle())  
  
# First character in first  
# word is lowercase  
s = 'geeks For Geeks'  
print(s.istitle())  
  
# Third word has uppercase  
# characters at middle  
s = 'Geeks For GEEKs'  
print(s.istitle())  
  
s = '6041 Is My Number'  
print(s.istitle())  
  
# word has uppercase  
# characters at middle  
s = 'GEEKS'  
print(s.istitle())  
  
# Python code to demonstrate the working of  
# index()  
  
# initializing target string  
ch = "geeksforgeeks"  
  
# initializing argument string  
ch1 = "geeks"  
  
# using index() to find position of "geeks"  
# starting from 2nd index  
# prints 8  
pos = ch.index(ch1, 2)  
  
print("The first position of geeks after 2nd index : ", end="")  
print(pos)  
  
# Python3 code to demonstrate  
# working of isupper()  
  
# initializing string  
isupp\_str = "GEEKSFORGEEKS"  
not\_isupp = "Geeksforgeeks"  
  
# Checking which string is  
# completely uppercase  
print("Is GEEKSFORGEEKS full uppercase ? : " + str(isupp\_str.isupper()))  
print("Is Geeksforgeeks full uppercase ? : " + str(not\_isupp.isupper()))  
  
# Python code for implementation of isdigit()  
  
# checking for digit  
string = '15460'  
print(string.isdigit())  
  
string = '154ayush60'  
print(string.isdigit())  
  
# Python3 code to print  
# all encodings available  
  
from encodings.aliases import aliases  
  
# Printing list available  
print("The available encodings are : ")  
print(aliases.keys())  
  
# Python code for implementation of isspace()  
  
# checking for whitespace characters  
string = 'Geeksforgeeks'  
  
print(string.isspace())  
  
# checking if \n is a whitespace character  
string = '\n \n \n'  
  
print(string.isspace())  
  
string = 'Geeks\nfor\ngeeks'  
print(string.isspace())  
  
# Python code for implementation of isnumeric()  
  
# checking for numeric characters  
string = '123ayu456'  
print(string.isnumeric())  
  
string = '123456'  
print(string.isnumeric())  
  
# Python code for implementation of isdigit()  
  
# checking for digit  
string = '15460'  
print(string.isdigit())  
  
string = '154ayush60'  
print(string.isdigit())  
  
# Python code for implementation of isalpha()  
  
# checking for alphabets  
string = 'Ayush'  
print(string.isalpha())  
  
string = 'Ayush0212'  
print(string.isalpha())  
  
# checking if space is an alphabet  
string = 'Ayush Saxena'  
print(string.isalpha())  
  
# Python program to demonstrate the use of  
# isalnum() method  
  
# here a,b and c are characters and 1,2 and 3  
# are numbers  
string = "abc123"  
print(string.isalnum())  
  
# here a,b and c are characters and 1,2 and 3  
# are numbers but space is not a alphanumeric  
# character  
string = "abc 123"  
print(string.isalnum())  
  
word = 'geeks for geeks'  
  
# returns first occurrence of Substring  
result = word.find('geeks')  
print("Substring 'geeks' found at index:", result)  
  
result = word.find('for')  
print("Substring 'for ' found at index:", result)  
  
# How to use find()  
if (word.find('pawan') != -1):  
 print("Contains given substring ")  
else:  
 print("Doesn't contains given substring")  
  
# Python code shows the working of  
# .endswith() function  
  
text = "geeks for geeks."  
  
# returns False  
result = text.endswith('for geeks')  
print(result)  
  
# returns True  
result = text.endswith('geeks.')  
print(result)  
  
# returns True  
result = text.endswith('for geeks.')  
print(result)  
  
# returns True  
result = text.endswith('geeks for geeks.')  
print(result)  
  
# Python program to illustrate  
# string center() in python  
string = "geeks for geeks"  
  
new\_string = string.center(24)  
  
# here filchar not provided so takes space by default.  
print("After padding String is: ", new\_string)  
  
#SETS ADD FUNCTION

*"""*

*>>> s = set('HackerRank')  
>>> s.add('H')  
>>> print s  
set(['a', 'c', 'e', 'H', 'k', 'n', 'r', 'R'])  
>>> print s.add('HackerRank')  
None  
>>> print s  
set(['a', 'c', 'e', 'HackerRank', 'H', 'k', 'n', 'r', 'R'])*

*"""*

#sets  
s={}  
print(type(s))  
  
s=set()  
print(type(s))  
  
s={1,3,2,4,5,7,6,9,8}  
print(s)  
print(s)  
print(s)  
print(s)  
s.add(1)  
s.add(2)  
s.add(10)  
print(" doesnt allow duplicates.... and also insertion order not preserved")  
print(s)  
  
s.remove(1)  
print(s)  
  
s.discard(10)  
s.remove(5)  
s.discard(100)  
#s.remove(100)  
print(s)  
  
print(s.pop())  
  
print(" CLEAR METHOD CLEARS ALL THE SET ")  
s.clear()  
print(s)  
  
#print(del s)  
s={1,3,2,4,5,7,6,9,8}  
print(s)  
print(" LENGTH OF THE SET IS : ",len(s))  
print(" ELEMENTS IN THE SET ")  
for ele in s:  
 print(ele)  
  
#set objects are not subscriptable , cannot be accessed with the index since the set is unordered  
  
t=set()  
for i in range(20):  
 t.add(i)  
  
print(s," IS SUBSET OF ",t," IS IT TRUE BHAVANI ",s.issubset(t))  
  
print(t," IS SUPERSET OF ",s," IS IT TRUE BHAVANI ",t.issuperset(s))  
  
print(s.union(t))  
s={1,3,5,6,7,9}  
t={1,2,3,4,5,6}  
print(s.intersection(t))  
print(t.difference(s))  
print(s.symmetric\_difference(t))  
  
print(s.copy())  
print(s.intersection\_update(t))  
s={1,3,5,6,7,9}  
t={1,2,3,4,5,6}  
  
print(s.difference\_update(t))  
s={1,3,5,6,7,9}  
t={1,2,3,4,5,6}  
  
print(s.symmetric\_difference\_update(t))  
s={1,3,5,6,7,9}  
t={1,2,3,4,5,6}  
print(s)  
print(t)  
t=s.copy()  
print(s)  
print(t)

#dictionaries  
  
d={}  
print(type(d))  
d=dict()  
print(type(d))  
d={"1":"SURYA","2":"MAHESH","3":"GOPI","4":"PEMMA","5":"SUBBBU"}  
print(d)  
d={1:"SURYA",2:"MAHESH",3:"GOPI",4:"PEMMA",5:"SUBBBU"}  
  
i=1  
while i<len(d):  
 print(d[i])  
 i=i+1  
  
del d[2]  
print(d)  
  
#del d  
  
  
d1={6:"SATHI"}  
d2={7:"PITHI"}  
  
d.update(d1)  
print(d)  
  
d.update(d2)  
print(d)  
  
print(" DICTIONARIES USING COMPREHENSIONS")  
d={i:i\*\*2 for i in range(10)}  
print(d)  
  
import math  
d={i:math.pow(i,3) for i in range(10)}  
print(d)  
  
import math as m  
d={i:m.pow(i,4) for i in range(5)}  
print(d)  
  
d={i:i\*\*2 for i in range(10) if i%2!=0}  
print(d)  
  
d={i:i\*\*2 for i in range(10) if i%2==0}  
print(d)  
  
d={"NAME":{"FNAAME":"SURYA","LNAME":"MAHESH"},"REGNO":124003143}  
print(d["NAME"])  
  
print(d["REGNO"])  
  
d1=dict()  
d1=d.copy()  
print(d1)  
  
print(d.items())  
print(d.keys())  
print(d.values())  
  
d1={i:i\*\*2 for i in range(10) if i%2!=0}  
print(d1)  
d.update(d1)  
print(d)

#DICTIONARIES  
  
salaries = { "surya":10000000 ,"MAHESH":1000000," gopi":100100 ,"pemma": 100000}  
print(type(salaries))  
print(salaries)  
  
#UPDATE

# update() only works for iterable objects  
  
myset=set()  
myset.update([1, 2, 3, 4])  
print(myset)  
  
myset.update({1, 7, 8})  
print(myset)  
  
myset.update({1, 6}, [5, 13])  
print(myset)

# REMOVING ITEMS  
  
myset.discard(10)  
print(myset)  
myset.remove(13)  
print(myset)

#COMMON SET OPERATIONS  
  
a = {2, 4, 5, 9}  
b = {2, 4, 11, 12}  
  
print(a.union(b)) # Values which exist in a or b  
print(a.intersection(b) )# Values which exist in a and b  
  
#The union() and intersection() functions are symmetric methods:  
  
print(a.union(b) == b.union(a))  
print(a.intersection(b) == b.intersection(a))  
print(a.difference(b) == b.difference(a))

#SETS AND DICTIONARIES ARE NOT ACCESSED VIA INDEXES , CANT POSSIBLE FOR INDEXING  
try:  
 for i in range(len(salaries)):  
 print(salaries[i])  
except Exception:  
 print(Exception)  
 print("oopss something went wrong")  
 print("SETS AND DICTIONARIES ARE NOT ACCESSED VIA INDEXES , CANT POSSIBLE FOR INDEXING")  
  
  
# AND DICTIONARIES ARE ACCESSED VIA keys  
print(salaries["surya"])  
print(salaries["pemma"])  
  
print(" lets check what happens if we try to access any pair with non existing key of the dictionary")  
  
try:  
 print(salaries["mahesh"])  
except Exception as e:  
 print(e)  
 print("oopss something went wrong")  
 print("MAY BE REQUESTED KEY NOT FOUND IN THE DICTIONARY ")  
 print("oopss something went wrong")  
 print("TO AVOID THIS ERROR OR EXCEPTION U MAY USE , GET METHOD FOR ACCESSING PARTICULAR PAIR OF DICTIONARY")  
  
print(salaries.get("mahesh"))  
  
print(salaries.get("surya"))  
print(salaries.get("MAHESH"))  
print(salaries.get("pemma"))  
  
print(salaries.get("surya",1))  
print(salaries.get("MUMMA",4))  
print(salaries.get("DUMMA",20))  
print(salaries.get("KAMMA",100))  
  
print(salaries)

#functions

def display(str):  
 print("SYNDICATE MEMBER ",str," PESIRE")  
 print("YEVVA THAGGEDHELE")  
  
def show(str):  
 print("SYNDICATE MEMBER ", str, " PESIRE")  
 print("YODIKI BAYAPADEDHELE")  
a=int(input("ENTER A VALUE :"))  
s=input("ENTER UR NAME ")  
  
if a%2==0:  
 display(s)  
else:  
 show(s)  
  
def showdetails(s):  
 age=int(input("ENTER AGE :"))  
 ht=int(input("ENTER HEIGHT :"))  
 wt = int(input("ENTER WEIGHT :"))  
 det=" \nNAME :"+s+" \nAGE :"+str(age)+" \nHEIGHT :"+str(ht)+" \nWEIGHT :"+str(wt)+" \nVALUE :"+str(a)  
 return det  
  
if s=="pushpa":  
 st=showdetails(s)  
print(st)

#function with return statement and arguments

a=int(input("enter value a: "))  
b=int(input("enter value b: "))  
def add(a,b):  
 c=a+b  
 return c  
result=add(a,b)  
print("RESULT OF SUM OF ",a,b," is :",result)

a=int(input("enter value a: "))  
b=int(input("enter value b: "))  
def add(a,b):  
 c=a+b  
 return c  
result=add(a,b)  
print("RESULT OF SUM OF ",a,b," is :",result)  
  
def display(a,b):  
 print("a,b VALUES :",a,b)  
display(a,b)  
  
  
#default arguments  
c=30  
d=40  
def display(a,b,c,d=100):  
 print("a,b VALUES :",a,b,c,d)  
  
display(10,20,30)  
display(1,2,3,4)  
display(a,b,c,d)  
  
#keyword arguments  
def display(a,b,c):  
 print(a,b,c)  
display(c=300,a=100,b=200)  
  
  
#arbitary function  
def display(\*data):  
 print(data)  
display(11,12,13,14,15,16,17,18,19,20)  
  
#global declarations  
  
def display(a,b):  
 a+=1  
 b+=1  
 print(a,b)  
display(11,12)  
  
z=100  
def show():  
 z=10  
 print("Z = ",z)  
show()  
print("Z = ",z)  
  
z=100  
def show():  
 global z  
 z+=1  
 print("Z = ",z)  
show()  
print("Z = ",z)  
  
  
#lambda functions  
add=lambda a,b:a+b  
print("SUM OF ",a,",",b,"IS : ",add(1,2))

#LEAP YEAR CHECK

def is\_leap(year):  
 leap = False  
 if year%400==0:  
 leap=True  
 elif year%100==0:  
 leap=False  
 elif year%4==0:  
 leap=True  
 return leap  
  
year = int(input())  
print(is\_leap(year))

#FUNCTIONS  
#BUILT IN FUNCTIONS  
  
print(-1)  
print(abs(-1))  
  
  
print(" returns true if all the elements in the iteratable object are true")  
l=[1,0]  
print(all(l))  
  
l=[1,2,3,4,5]  
print(all(l))  
  
print(" returns true if any of the elements in the iteratable object are true")  
l=[1,0]  
print(any(l))  
  
l=[0,0,0,0]  
print(any(l))  
  
print(ascii("a"))  
  
print(bin(10))  
print(oct(10))  
print(hex(10))  
  
print(bool(1))  
print(bool(0))  
print(bool("HELLO"))  
print(bool(""))  
  
#USERDEFINED - FUNCTIONS  
def downloadfile(url):  
 print("URL : ",url)  
 print("DOWNLOADED SUCCESSFULLY")  
  
downloadfile("www.google.com")  
  
#FUNCTIONS V/S LAMBDA FUNCTIONS  
  
def multiply(x,y):  
 print(x\*y)  
  
multiply(20,10)  
  
  
l=lambda x,y: x\*y  
  
l(10,20)  
  
  
#FUNCTIONS AND LAMBDA FUNCTIONS  
  
def cratemultiplier(x):  
 return lambda y:x\*y  
  
multiply=cratemultiplier(10)  
  
print(multiply(555))  
print(multiply(222))

#recursive functions  
#factorial  
def fact(n):  
 if n==0 or n==1:  
 return 1  
 else:  
 return n\*fact(n-1)  
print("FACTORIAL OF 5 IS :",fact(5))  
  
import math  
print("FACTORIAL OF 10 IS :",math.factorial(10))  
  
#gcd  
def gcd(a,b):  
 if a==0:  
 return b  
 elif b==0:  
 return a  
 else:  
 return gcd(b,a%b)  
   
print("GCD OF 10,100 IS :",gcd(10,100))

#TUPLES are immutable  
t=(1,2,3,4,5,6,7,8,9,10)  
print(type(t))  
print("SIZE of the tuple : ",len(t))  
print(t[0])  
  
print(" elements in tuple as follows : ")  
for i in range(len(t)):  
 print(t[i])  
  
print(" tuples are immutable ..... cant change the elements .. but can fetch the elements using index")  
#t[0]=100  
  
print(" DOESNT SUPPORT ITEM ASSIGNMENT ")  
# no replacements / changing the item values  
# iterations in tuple are fast than the list  
# when u are no way needed to change the values of the list then we can go with the tuples  
  
print(" TO ENCHANCE THE SPEED OF THE EXECUTION WE CAN USE TUPLE INSTEAD OF LISTS ")  
  
  
#SETS  
  
s={1,5,2,4,3,8,9,10}  
print(type(s))  
  
print(" SET SIZE : ",len(s))  
  
# insertion order not preserved in sets  
print(s)  
  
print(" SET ELEMENTS AS FOLLOWS : ")  
#always prints in random order  
  
s={100,123,100,100,26,1000,254,100}  
  
#set doesnt allows duplicatess  
  
print(" SET USES HASH , WE IMPROVE THE PERFORMANCE AND FETCH THE ELEMENTS AS FAST AS POSSIBLE")  
print(" as sequence or insertion order not preserved then indexing slicing not possible here")  
  
print(s)  
s.discard(1000)  
s.remove(100)  
s2=s.copy()  
print(s2)  
s3={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}  
s.update(s3)  
print(s)

#sets  
#CONCEPT

*"""  
>>> print set()  
set([])  
  
>>> print set('HackerRank')  
set(['a', 'c', 'e', 'H', 'k', 'n', 'r', 'R'])  
  
>>> print set([1,2,1,2,3,4,5,6,0,9,12,22,3])  
set([0, 1, 2, 3, 4, 5, 6, 9, 12, 22])  
  
>>> print set((1,2,3,4,5,5))  
set([1, 2, 3, 4, 5])  
  
>>> print set(set(['H','a','c','k','e','r','r','a','n','k']))  
set(['a', 'c', 'r', 'e', 'H', 'k', 'n'])  
  
>>> print set({'Hacker' : 'DOSHI', 'Rank' : 616 })  
set(['Hacker', 'Rank'])  
  
>>> print set(enumerate(['H','a','c','k','e','r','r','a','n','k']))  
set([(6, 'r'), (7, 'a'), (3, 'k'), (4, 'e'), (5, 'r'), (9, 'k'), (2, 'c'), (0, 'H'), (1, 'a'), (8, 'n')])  
  
"""*

#CREATING SETS  
  
myset = {"a","b"} # Directly assigning values to a set  
  
print((myset))  
  
myset = set("ab") # Initializing a set  
  
print((myset))  
  
myset = set(("a","b")) # Initializing a set  
  
print((myset))  
  
myset = set(['a', 'b']) # Creating a set from a list  
  
print((myset))

#SETS

s=set()  
  
print(type(s))  
  
s={1,2,3,4}  
  
print(s)

#add function

myset=set()  
  
myset.add('c')  
print(myset)  
myset.add('a') # As 'a' already exists in the set, nothing happens  
print(myset)  
myset.add((5, 4))  
print(myset)

s.add(1)  
s.add(10)  
s.add(100)  
s.add(11001)  
s.add(1200)  
  
print(s)  
  
print(1 in s)  
print( 12 in s)  
print(s.pop())  
  
print(s)

#VARIABLES IN PYTHON  
num=10  
print("ADDRESS OF ", num ," is : ", id(num))  
name="SURYA"  
print("ADDRESS OF ", name , " is : ", id(name))  
a=10  
b=a  
print("ADDRESS OF A,B", id(a),id(b))  
print(" VARIABLES WITH SAME CONTENT POSSESS SAME ADRESS OOPS...NICCCEEE effient data usage")  
print(" ADDRESS NOT DEPENDS ON THE VARIABLE .. IT DEPENDS ON THE DATA OR CONTENT IN IT ")  
print(" if we change value then address of the variable changess naiceee : )")  
  
print(" garbage collection mechanism is implicit in python ")  
print(" PYTHON DONT HAVE IMMUTABLE VARIABLES OR CONSTANTS ")

#data types  
  
#None  
#NUMERIC  
num=10  
print( num ," is OF : ",type(num)," TYPE")  
  
num=3.14242  
print( num ," is OF : ",type(num)," TYPE")  
  
num="HELLO"  
print( num ," is OF : ",type(num)," TYPE")  
  
num='SYNDCATE MEMBER PUSHPA PESIRE'  
print( num ," is OF : ",type(num)," TYPE")  
  
num= 1+ 2j  
print( num ," is OF : ",type(num)," TYPE")  
  
#convesrions  
num=2.22  
  
print(int(num))  
  
num=100  
print(float(num))  
  
num=100  
print(complex(num))  
  
  
num=100  
print(str(num))  
  
  
num=100  
print(bool(num))  
  
"""  
  
some conversions are not possible   
num='SYNDCATE MEMBER PUSHPA PESIRE'  
print(float(num))  
  
"""  
l=[1,2,3,4,5]  
print(l, " IS OF : ",type(l)," TYPE")  
  
l=(1,2,3,4,5)  
print(l, " IS OF : ",type(l)," TYPE")  
  
l={1,2,3,4,5}  
print(l, " IS OF : ",type(l)," TYPE")  
  
l="12345"  
print(l, " IS OF : ",type(l)," TYPE")  
  
print(range(10))  
  
print(" LIST OF NUMBERS < 10")  
print(list(range(10)))  
  
print(" LIST OF EVEN NUMBERS < 10")  
print(list(range(0,10,2)))  
print(" LIST OF ODD NUMBERS < 10")  
print(list(range(1,10,2)))  
  
print(type(range(10)))  
  
#dictionaries  
d={1:"a",2:"b",3:"c",4:"d",5:"e"}  
print(d)  
print(d.keys())  
print(d.values())  
d2=d.copy()  
print(d2)

#operators  
#arithmetic operators  
a=10  
b=20  
print(a,b)  
print(a+b)  
print(a-b)  
print(a\*b)  
print(a/b)  
  
#assignment operators  
a+=b  
b+=2  
print(a,b)  
a-=b  
b-=2  
print(a,b)  
  
a\*=2  
b\*=2  
print(a,b)  
  
a/=2  
b/=2  
print(a,b)  
  
#unary operators  
  
a=-a  
b=-b  
print(a,b)  
print(-a,-b)  
  
print(a>b)  
print(a<b)  
print(a==b)  
  
#logical operators  
a=10  
b=11  
  
#and  
if a>0 and b>0:  
 print(a,b," BOTH ARE POSITIVE ")  
elif a>0 and b<0:  
 print(a," is positive ",b," is negative")  
elif a<0 and b>0:  
 print(a," is negative ",b," is positive")  
else:  
 print(a, b, " BOTH ARE NEGATIVE ")  
  
#or  
if a>0 or b>0:  
 print(" OKAY")  
else:  
 print(" SORRY ")  
  
#not  
print(bool(a))  
print(bool(not a))

#conversions  
a=10  
print(a)  
print(bin(a))  
print(hex(a))  
print(oct(a))  
  
print(hex(0b1010))

#bitwise operators  
print(12)  
#compliment  
print(" NEGATION OF 12--->",-12)  
print(" COMPLIMENT OF 12--->",~12)  
#bitwise and &operator  
print(1&0)  
print(1&1)  
print(11&10)  
print(21&5)  
  
#bitwise or | operator  
print(1|0)  
print(1|1)  
print(11|10)  
print(21|5)  
  
#bitwise xor ^ operator  
print(1^0)  
print(1^1)  
print(11^10)  
print(21^5)  
  
#bitwise left shift and right shift operator  
print(10<<2)  
print(26<<3)  
print(100>>2)  
print(1>>2)

#literal  
  
a=None  
print(type(a))

#operators  
  
x=10  
y=15  
z=20  
if(x==y and x==z):  
 print("ALL ARE SAME")  
if x<=y and x<=z:  
 if y<=z:  
 print(" x is smallest")  
if x!=y or y!=z or x!=z:  
 print("ALL ARE NOT IDENTICAL")  
  
#identity operators  
print(id(1))  
print(id(2))  
print(id(1) is id(2))  
print(id(1) is id(1))  
print(id(1) is not id(2))  
print(id(1) is id("1"))  
print(id(1) is not id(1))  
print(id(1) is id(1.0))  
print(id(1) is id(1+0j))  
  
print(type("sastra") is type(100))  
  
# membership operator  
  
print(1 in range(10))  
print(20 not in range(10))

#math functions  
  
import math  
  
print(math.sqrt(25))  
print(math.ceil(1.1))  
print(math.floor(1.9))  
print(math.pow(2,3))  
print(math.log(8,2))  
print(math.factorial(5))  
print(math.gcd(1,5))  
print(math.pi)  
print(math.e)  
  
import math as mahesh  
print(mahesh.factorial(5))

#sum of list of integers < 10  
print(sum(list(range(10))))

#USER INPUT  
  
a=int(input("ENTER VALUE FOR a : "))  
b=int(input("ENTER VALUE FOR a : "))  
print(a,b)  
  
a=float(input("ENTER VALUE FOR a : "))  
b=float(input("ENTER VALUE FOR b : "))  
print(a,b)

# IF , ELIF , ELSE  
  
a=int(input(" enter value a "))  
b=int(input(" enter value b "))  
if a%2==0:  
 print(" a IS EVEN ")  
else:  
 print(" a IS ODD ")  
if b%2==0:  
 print(" b IS EVEN ")  
else:  
 print(" b IS ODD ")  
   
   
if a%2==0 and b%2==0:  
 print(" both are even")  
elif a%2!=0 and b%2!=0:  
 print(" both are odd")  
elif a%2!=0 and b%2==0:  
 print(" a odd b even ")  
elif a%2==0 and b%2!=0:  
 print(" a even b odd")

# IF , ELIF , ELSE WE CAN EVEN USE THE CLOSE BRACKETS   
  
a=int(input(" enter value a "))  
b=int(input(" enter value b "))  
if (a%2==0):  
 print(" a IS EVEN ")  
else:  
 print(" a IS ODD ")  
if (b%2==0):  
 print(" b IS EVEN ")  
else:  
 print(" b IS ODD ")  
  
  
if (a%2==0 and b%2==0):  
 print(" both are even")  
elif (a%2!=0 and b%2!=0):  
 print(" both are odd")  
elif (a%2!=0 and b%2==0):  
 print(" a odd b even ")  
elif (a%2==0 and b%2!=0):  
 print(" a even b odd")

#LOOPS  
#LOOPS  
  
l=[]  
l=[x for x in range(10)]  
print(l)  
  
for x in l:  
 if(x%2==0):  
 if(x%4==0):  
 print(x\*(x+1))  
 elif(x%6==0):  
 print(x\*\*3)  
 else:  
 print(x\*x\*x)  
  
  
a=10  
  
while a!=0:  
 print(a)  
 a=a-1

#FOR LOOP  
i=0  
for i in range(10):  
 print("PYTHON PROGRAMMING")  
  
  
s="PUSHPA"  
  
  
for x in list(range(1,20,2)):  
 print(x,end=" ")  
  
print(" ")  
  
  
for i in s:  
 print(i,end=" ")  
print(" ")  
  
for x in tuple(range(1,20,2)):  
 print(x,end=" ")  
  
print(" ")  
  
  
for x in set(range(1,20,2)):  
 print(x,end=" ")  
  
print(" ")

#BREAK   
  
x=int(input(" HOW MANY CANDIES U WANT: "))  
limit=50  
i=0  
while i<x:  
 if i>limit:  
 print(" LIMIT REACHED ")  
 print(" OUT OF STOCK ")  
 break  
 else:  
 print("CANDY : ", i)  
 i += 1

#CONTINUE   
i=0  
a=int(input("ENTER LIMIT VALUE "))  
for i in range(a):  
 if i%3==0:  
 print(i," is divisible by 3")  
 continue  
 else:  
 print(i," is NOT divisible by 3")

#CONTINUE  
#USING WHILE LOOP  
  
a=int(input("ENTER LIMIT VALUE "))  
i=0  
while i<a:  
 if i%3==0:  
 i = i + 1  
 continue  
   
 print(i, " is NOT divisible by 3")  
 i=i+1

#PASS  
#USING WHILE LOOP  
  
a=int(input("ENTER LIMIT VALUE "))  
i=0  
while i<a:  
 if i%3==0:  
 i = i + 1  
 pass  
 print(i, " ")  
 i=i+1  
  
print("BYE")

#OOPS  
class human:  
  
 height=5.6  
 color="BLACK"  
 weight=60  
 haircolor = "WHITE"  
  
 def \_\_init\_\_(self):  
 print("INIT METHOD CALLED ")  
  
  
 def walk(self):  
 print("WALKING ")  
  
 def talk(self):  
 print("TALKING ")  
  
 def eat(self):  
 print("EATING ")  
  
#print(color)  
  
hobj=human()  
print(hobj.color)  
print(hobj.haircolor)  
hobj.talk()  
hobj.walk()  
hobj.eat()

class computer:  
 #class method  
 def configuaration(self):  
 print(" 16 GB , INTEL PROCESSOR ,1TB STORAGE ")  
  
 def message(self):  
 print(" hello murugan ,syndicate ,member ","pushpa"," pesire")  
  
  
 # class attributes  
  
  
  
comp=computer()  
a=10  
b="pushpa"  
print(type(a))  
print(type(b))  
print(type(comp))  
  
  
comp.configuaration()  
comp.message()  
  
computer.configuaration(comp)  
computer.message(comp)

#OOPS  
  
class computer:  
 #class method  
 def configuaration(self):  
 print(" 16 GB , INTEL PROCESSOR ,1TB STORAGE ")  
  
 def message(self):  
 print(" hello murugan ,syndicate ,member ","pushpa"," pesire")  
  
 def \_\_init\_\_(self):  
 print(" IN INIT METHOD , SIMILAR TO CONSTRUCTOR")  
  
  
 # class attributes  
  
  
  
comp=computer()  
comp2=computer()  
a=10  
b="pushpa"  
print(type(a))  
print(type(b))  
print(type(comp))  
  
computer.configuaration(comp)  
computer.message(comp)  
  
  
comp.configuaration()  
comp.message()  
  
  
comp2.configuaration()  
comp2.message()

#INIT METHOD  
class computer:  
 #class method  
 def configuaration(self):  
 print(self.a," GB ram ",self.b," TB storage")  
  
 def message(self):  
 print(" hello murugan ,syndicate ,member ",self.s," pesire")  
  
 def \_\_init\_\_(self,a,b,s):  
 self.a=a  
 self.b = b  
 self.s = s  
 print(" IN INIT METHOD , SIMILAR TO CONSTRUCTOR")  
  
  
 # class attributes  
  
comp=computer(1,2,"pushpa")  
comp2=computer(10,20,"PUSHPA")  
  
comp.configuaration()  
comp.message()  
  
comp2.configuaration()  
comp2.message()

#NON PARAMETERISED CONSTRUCTOR

class human:  
  
 def \_\_init\_\_(self):  
 self.color= "YELLOW"  
 self.height= 5  
 self.weight= 5  
 self.hair= " ORANGE "  
 print("INIT METHOD WITH OUT PARAMETERS CALLED ")  
  
   
  
 def display(self):  
 print(self.color)  
 print(self.height)  
 print(self.hair)  
 print(self.weight)  
  
  
 def walk(self):  
 print("WALKING ")  
  
  
 def talk(self):  
 print("TALKING ")  
  
  
 def eat(self):  
 print("EATING ")  
  
  
  
hob = human()  
  
print(hob.color)  
print(hob.hair)  
print(hob.height)  
print(hob.weight)  
  
hob.talk()  
hob.walk()  
hob.eat()  
hob.display()

#PARAMETERISED CONSTRUCTOR

class human:  
  
 def \_\_init\_\_(self, a, b, c, d):  
 self.color = a  
 self.height = b  
 self.weight = c  
 self.hair = d  
 print("INIT METHOD WITH PARAMETERS CALLED ")  
  
  
  
 def display(self):  
 print(self.color)  
 print(self.height)  
 print(self.hair)  
 print(self.weight)  
  
  
 def walk(self):  
 print("WALKING ")  
  
  
 def talk(self):  
 print("TALKING ")  
  
  
 def eat(self):  
 print("EATING ")  
  
  
hobj = human("BLACK",10,10,"WHITE")  
  
print(hobj.color)  
print(hobj.hair)  
print(hobj.height)  
print(hobj.weight)  
  
hobj.talk()  
hobj.walk()  
hobj.eat()  
hobj.display()

#CONSTRUCTOR METHOD  
  
class computer:  
 def \_\_init\_\_(self,a,b,c):  
 print(" ATTRIBUTES OF THE CLASS : ")  
 self.a = a  
 self.b = b  
 self.c = c  
 print("\n a :",self.a, "\n b :",self.b, "\n c :", self.c)  
  
 def display(self):  
 print(" ATTRIBUTES OF THE CLASS : ")  
 print("\n a :",self.a, "\n b :",self.b, "\n c :", self.c)  
  
 def getdetails(self):  
 print(" THEN ")  
 self.display()  
 print(" PROVIDE VALUES FOR THE ATTRIBUTES ")  
 self.a= input(" ENTER VALUE OF A ")  
 self.b = input(" ENTER VALUE OF B ")  
 self.c = input(" ENTER VALUE OF C ")  
 print(" NOW ")  
 self.display()  
  
  
  
comp1=computer(1,2,3)  
comp1.display()  
comp1.getdetails()  
print(id(comp1))  
  
comp2=computer(4,5,6)  
comp2.display()  
comp2.getdetails()  
print(id(comp2))

#class and objects  
  
class person:  
  
 def \_\_init\_\_(self,name,age,height,weight):  
  
 self.name=name  
 self.age=age  
 self.height=height  
 self.weight=weight  
 self.display()  
  
  
 def display(self):  
 print("DETAILS OF : ", self.name)  
 print("NAME : ", self.name)  
 print("AGE : ", self.age)  
 print("HEIGHT : ", self.height)  
 print("WEIGHT : ", self.weight)  
  
 def getdetails(self):  
 print(" provides details for the attributes of the class")  
  
 self.name=input(" ENTER NAME OF THE PERSON")  
 self.age=input(" ENTER AGE OF THE PERSON")  
 self.height=input(" ENTER HEIGHT OF THE PERSON")  
 self.weight=input(" ENTER WEIGHT OF THE PERSON")  
  
 def update(self,n,a,h,w):  
 self.name=n  
 self.age=a  
 self.height=h  
 self.weight=w  
  
  
 def check(self,other):  
  
 if self.name==other.name:  
 print(" BOTH ",self.name,other.name," WITH SAME NAME")  
 else:  
 print(" BOTH ",self.name,other.name, " WITH DIFFERENT NAME")  
  
 if self.age==other.age:  
 print(" BOTH ",self.name,other.name," WITH SAME AGE")  
 else:  
 print(" BOTH ",self.name,other.name," WITH DIFFERENT AGE")  
  
 if self.height==other.height:  
 print(" BOTH ",self.name,other.name," WITH SAME HEIGHT")  
  
 else:  
 print(" BOTH ",self.name,other.name, " WITH DIFFERENT HEIGHT")  
  
 if self.weight==other.weight:  
 print(" BOTH ",self.name,other.name," WITH SAME WEIGHT")  
  
 else:  
 print(" BOTH ",self.name,other.name," WITH DIFFERENT WEIGHT")  
  
  
p1=person("SURYA",19,5.6,60)  
  
#p1.getdetails()  
p1.display()  
p1.update("MAHESH",16,6,60)  
p1.display()  
  
p2=person("SUKU",16,6,66)  
p1.check(p2)  
  
p1.name="SANKU"  
p2.name="SHANKINI"  
  
p1.display()  
p2.display()  
  
print(" AFTER UPDATING NAMES ")  
print(p1.name)  
print(p2.name)

#instance variables , CLASS VARIABLES (STATIC VARIABLES  
print("CLASS VARIABLES ALSO CALLED AS STATIC VARIABLES ")  
class car:  
  
 #class variables  
 #inside the class outside the init  
 wheels=4  
 color="BLACK"  
  
 def \_\_init\_\_(self):  
 # INSTANCE variables  
 # inside the class AND INSIDE the init  
 self.milage=60  
 self.modelname="bmw"  
 self.cost=100000  
  
 def display(self):  
 print(" MILLAGE : ",self.milage," MODEL NAME :",self.modelname," COST :",self.cost)  
  
c1=car()  
c2=car()  
c1.display()  
c2.display()  
  
c1.modelname="HOHOHO"  
c1.cost=20000000  
c1.milage=1000  
  
c2.modelname="BENZ"  
c2.cost=5000000  
c2.milage=100  
  
c1.display()  
print(c1.wheels," WHEELS")  
print(c1.color," COLOR")  
  
  
c2.display()  
print(c2.wheels," WHEELS")  
print(c2.color," COLOR")

# METHODS IN PYTHON  
#INSTANCE METHODS  
#CLASS METHODS  
#STATIC METHODS  
  
#getters , setters  
  
class student:  
  
 school="NARAYANA SCHOOL"  
  
 def \_\_init\_\_(self,m1,m2,m3):  
 self.m1 = m1  
 self.m2 = m2  
 self.m3 = m3  
  
 def average(self):  
 avg=(self.m1+self.m2+self.m3)/3  
 print("AVERAGE : ",avg)  
  
 def display(self):  
 print(" M1 : ",self.m1," M2: ",self.m2," M3 : ",self.m3)  
  
 def setm1(self,m1):  
 self.m1=m1  
  
 def setm2(self,m2):  
 self.m2=m2  
  
 def setm3(self,m3):  
 self.m3=m3  
  
 def getm1(self):  
 return self.m1  
  
 def getm2(self):  
 return self.m2  
  
 def getm3(self):  
 return self.m3  
  
 @classmethod  
 def schoolinfo(cls):  
 print(cls.school)  
  
  
s1=student(100,100,100)  
s2=student(50,50,50)  
s3=student(35,35,35)  
  
s1.average()  
s1.display()  
print(s1.school)  
  
s2.average()  
s2.display()  
print(s2.school)  
  
s3.average()  
s3.display()  
print(s3.school)  
  
s1.setm1(70)  
s1.setm2(80)  
s1.setm3(75)  
  
print(s1.getm3())  
print(s1.getm2())  
print(s1.getm1())  
  
s1.schoolinfo()  
s2.schoolinfo()

# INNER CLASS  
#INNER CLASS OBJECT CREATING INSIDE THE INIT METHOD OF THE OUTER CLASS

class student:  
  
 def \_\_init\_\_(self,name,rollnum):  
 self.name=name  
 self.rollnum=rollnum  
 self.mylap=self.laptop()  
  
  
 def show(self):  
 print("NAME : ",self.name," ROLLNUM : ",self.rollnum)  
 self.mylap.showin()  
  
 class laptop:  
  
 def \_\_int\_\_(self):  
 self.brand="LENOVO"  
 self.processor=" AMD RYZEN 5 "  
 self.cost=45000  
  
  
 def showin(self):  
 print("BRAND NAME : ",self.brand," PROCESSOR : ",self.processor," COST : ",self.cost)  
  
  
  
s1=student("MAHESH",3143)  
s2=student("SURYA ",3144)  
  
s1.mylap.brand="HP"  
s1.mylap.cost=100000  
s1.mylap.processor="INTEL"  
  
  
s2.mylap.brand="HP"  
s2.mylap.cost=100000  
s2.mylap.processor="INTEL"  
  
  
  
s1.show()  
s2.show()

#INHERITANCE

#SINGLE LEVEL INHERITANCE  
class baseclass:  
 a=100  
 b=200  
  
 def \_\_init\_\_(self):  
 print(" BASE CLASS INIT METHOD ")  
 self.a=1000  
 self.b=2000  
  
 def display(self):  
 print("BASE CLASS DISPLAY METHOD")  
 print(self.a)  
 print(self.b)  
  
class derivedclass(baseclass):  
 c=300  
 d=400  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" DERIVED CLASS INIT METHOD ")  
 self.a=3333  
 self.b=4444  
 self.c=3000  
 self.d=4000  
  
 def display(self):  
 super().display()  
 print("DERIVED CLASS DISPLAY METHOD ")  
 print(self.c)  
 print(self.d)  
  
  
b=baseclass()  
d=derivedclass()  
b.display()  
d.display()  
  
print(b.a," - a")  
print(b.b," - b")  
  
print(d.a," - a")  
print(d.b," - b")  
print(d.c," - c")  
print(d.d," - d")

#SINGLE LEVEL INHERITANCE  
class parent:  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class child(parent):  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
print("PARENT CLASS")  
p=parent()  
p.feature1()  
p.feature2()  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()

#MULTI LEVEL INHERITANCE  
  
class parent:  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class child(parent):  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
class grandchild(child):  
 def feature5(self):  
 print("FEATURE 5 IS WORKING")  
  
 def feature6(self):  
 print("FEATURE 6 IS WORKING")  
  
  
print("PARENT CLASS")  
p=parent()  
p.feature1()  
p.feature2()  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()  
  
  
print("GRAND CHILD CLASS")  
g=grandchild()  
g.feature1()  
g.feature2()  
g.feature3()  
g.feature4()  
g.feature5()  
g.feature6()

#HEIRARCHIAL INHERITANCE  
class baseclass:  
 a=100  
 b=200  
  
 def \_\_init\_\_(self):  
 print(" BASE CLASS INIT METHOD ")  
 self.a=1000  
 self.b=2000  
  
 def display(self):  
 print("BASE CLASS DISPLAY METHOD")  
 print(self.a)  
 print(self.b)  
  
class derivedclass(baseclass):  
 c=300  
 d=400  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" DERIVED CLASS INIT METHOD ")  
 self.a=3333  
 self.b=4444  
 self.c=3000  
 self.d=4000  
  
 def display(self):  
 super().display()  
 print("DERIVED CLASS DISPLAY METHOD ")  
 print(self.c)  
 print(self.d)  
  
  
class derivedclass2(baseclass):  
 c=300  
 d=400  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" DERIVED2 CLASS INIT METHOD ")  
 self.a=3333  
 self.b=4444  
 self.c=3000  
 self.d=4000  
  
 def display(self):  
 super().display()  
 print("DERIVED2 CLASS DISPLAY METHOD ")  
 print(self.c)  
 print(self.d)  
  
  
b=baseclass()  
b.display()  
print(b.a," - a")  
print(b.b," - b")  
  
  
d=derivedclass()  
d.display()  
print(d.a," - a")  
print(d.b," - b")  
print(d.c," - c")  
print(d.d," - d")  
  
  
d2=derivedclass2()  
b.display()  
print(d2.a," - a")  
print(d2.b," - b")  
print(d2.c," - c")  
print(d2.d," - d")

#HEIRARCHIAL INHERITANCE

#HEIRARCHIAL INHERITANCE  
class baseclass:  
 a=100  
 b=200  
  
 def \_\_init\_\_(self):  
 print(" BASE CLASS INIT METHOD ")  
 self.a=1000  
 self.b=2000  
  
 def display(self):  
 print("BASE CLASS DISPLAY METHOD")  
 print(self.a)  
 print(self.b)  
  
class derivedclass(baseclass):  
 c=300  
 d=400  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" DERIVED CLASS INIT METHOD ")  
 self.a=3333  
 self.b=4444  
 self.c=3000  
 self.d=4000  
  
 def display(self):  
 super().display()  
 print("DERIVED CLASS DISPLAY METHOD ")  
 print(self.c)  
 print(self.d)  
  
  
class derivedclass2(baseclass):  
 c=300  
 d=400  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" DERIVED2 CLASS INIT METHOD ")  
 self.a=3333  
 self.b=4444  
 self.c=3000  
 self.d=4000  
  
 def display(self):  
 super().display()  
 print("DERIVED2 CLASS DISPLAY METHOD ")  
 print(self.c)  
 print(self.d)  
  
  
b=baseclass()  
b.display()  
print(b.a," - a")  
print(b.b," - b")  
  
  
d=derivedclass()  
d.display()  
print(d.a," - a")  
print(d.b," - b")  
print(d.c," - c")  
print(d.d," - d")  
  
  
d2=derivedclass2()  
b.display()  
print(d2.a," - a")  
print(d2.b," - b")  
print(d2.c," - c")  
print(d2.d," - d")

#MULTIPLE INHERITANCE  
  
class parent1:  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class parent2:  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
class child(parent1,parent2):  
 def feature5(self):  
 print("FEATURE 5 IS WORKING")  
  
 def feature6(self):  
 print("FEATURE 6 IS WORKING")  
  
  
print("PARENT1 CLASS")  
p1=parent1()  
p1.feature1()  
p1.feature2()  
  
print("PARENT2 CLASS")  
p2=parent2()  
p2.feature3()  
p2.feature4()  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()

#CONSTRUCTOR IN INHERITANCE  
  
  
#SINGLE LEVEL INHERITANCE  
class parent:  
 def \_\_init\_\_(self):  
 print(" PARENT CLASS INIT CALLED......")  
  
  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class child(parent):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" CHILD CLASS INIT CALLED.........")  
  
  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
print("PARENT CLASS")  
p=parent()  
p.feature1()  
p.feature2()  
  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()

#CONSTRUCTOR IN INHERITANCE  
  
#MULTIPLE INHERITANCE  
  
class parent1:  
 def \_\_init\_\_(self):  
 print(" PARENT 1 CLASS INIT CALLED")  
  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class parent2:  
 def \_\_init\_\_(self):  
 print(" PARENT 2 CLASS INIT CALLED")  
  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
class child(parent1,parent2):  
   
 def \_\_init\_\_(self):  
 print(" CHILD CLASS INIT CALLED")  
  
 def feature5(self):  
 print("FEATURE 5 IS WORKING")  
  
 def feature6(self):  
 print("FEATURE 6 IS WORKING")  
  
  
print("PARENT1 CLASS")  
p1=parent1()  
p1.feature1()  
p1.feature2()  
  
print("PARENT2 CLASS")  
p2=parent2()  
p2.feature3()  
p2.feature4()  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()

#CONSTRUCTOR IN INHERITANCE  
#MULTIPLE INHERITANCE  
  
class parent1:  
 def \_\_init\_\_(self):  
 print(" PARENT 1 CLASS INIT CALLED")  
  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class parent2:  
 def \_\_init\_\_(self):  
 print(" PARENT 2 CLASS INIT CALLED")  
  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
class child(parent1,parent2):  
  
 def feature5(self):  
 print("FEATURE 5 IS WORKING")  
  
 def feature6(self):  
 print("FEATURE 6 IS WORKING")  
  
  
print("PARENT1 CLASS")  
p1=parent1()  
p1.feature1()  
p1.feature2()  
  
print("PARENT2 CLASS")  
p2=parent2()  
p2.feature3()  
p2.feature4()  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()

#CONSTRUCTOR IN INHERITANCE  
  
#MULTIPLE INHERITANCE

#BIASED AND UNFAIR HERE WHEN CHILD CLASS CREATED AND SUPER().INIT

#CALLED THEN WE ARE AMBIGIOUS THAT PARENT 1 OR PARENT2 CLASS INIT CALLED ..

#BUT SYSTEM IS BIASED AND UNFAIR   
  
class parent1:  
 def \_\_init\_\_(self):  
 print(" PARENT 1 CLASS INIT CALLED")  
  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class parent2:  
 def \_\_init\_\_(self):  
 print(" PARENT 2 CLASS INIT CALLED")  
  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
class child(parent1,parent2):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" CHILD CLASS INIT CALLED")  
  
 def feature5(self):  
 print("FEATURE 5 IS WORKING")  
  
 def feature6(self):  
 print("FEATURE 6 IS WORKING")  
  
  
print("PARENT1 CLASS")  
p1=parent1()  
p1.feature1()  
p1.feature2()  
  
print("PARENT2 CLASS")  
p2=parent2()  
p2.feature3()  
p2.feature4()  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()

#CONSTRUCTOR IN INHERITANCE  
  
#MULTIPLE INHERITANCE  
#METHOD RESOLUTION ORDER  
  
class parent1:  
 def \_\_init\_\_(self):  
 print(" PARENT 1 CLASS INIT CALLED")  
   
  
 def feature1(self):  
 print("FEATURE 1 IS WORKING")  
  
 def feature2(self):  
 print("FEATURE 2 IS WORKING")  
  
class parent2:  
 def \_\_init\_\_(self):  
 print(" PARENT 2 CLASS INIT CALLED")  
  
 def feature3(self):  
 print("FEATURE 3 IS WORKING")  
  
 def feature4(self):  
 print("FEATURE 4 IS WORKING")  
  
class child(parent1,parent2):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 print(" CHILD CLASS INIT CALLED")  
  
 def feature5(self):  
 print("FEATURE 5 IS WORKING")  
  
 def feature6(self):  
 print("FEATURE 6 IS WORKING")  
  
  
print("PARENT1 CLASS")  
p1=parent1()  
p1.feature1()  
p1.feature2()  
  
print("PARENT2 CLASS")  
p2=parent2()  
p2.feature3()  
p2.feature4()  
  
print("CHILD CLASS")  
c=child()  
c.feature1()  
c.feature2()  
c.feature3()  
c.feature4()

#DUCK TYPING  
  
class pycharm:  
  
 def execute(self):  
 print("COMPILING ")  
 print("RUNNING ")  
  
class mycharm:  
  
 def execute(self):  
 print("SPELLCHECK")  
 print("CONVENTION CHECK")  
 print("COMPILING ")  
 print("RUNNING ")  
  
  
class laptop:  
  
 def code(self,myide):  
 myide.execute()  
  
  
myide=pycharm()  
lap1=laptop()  
lap1.code(myide)  
  
myide=mycharm()  
lap1=laptop()  
lap1.code(myide)

# POLYMORPHISM IN INHERITANCE  
#DUCK TYPING  
#PROVIDING THE TYPE OF THE COMPILER DYNAMICALLY AT RUN TIME

class pycharm:  
  
 def execute(self):  
 print("COMPILING IN PYCHARM")  
 print("RUNNING IN PYCHARM ")  
  
class mycharm:  
  
 def execute(self):  
 print("SPELLCHECK IN MYCHARM")  
 print("CONVENTION CHECK IN MYCHARM")  
 print("COMPILING IN MYCHARM")  
 print("RUNNING IN MYCHARM ")  
  
  
class laptop:  
  
 def code(self,myide):  
 myide.execute()  
  
  
a=int(input("\n ENTER 1 FOR PYCHARM IDE \n ENTER 2 FOR MYCHARM IDE "))  
if(a==1):  
 myide = pycharm()  
 lap1 = laptop()  
 lap1.code(myide)  
  
elif(a==2):  
 myide = mycharm()  
 lap1 = laptop()  
 lap1.code(myide)  
  
else:  
 print(" INVALID INPUT ")  
 print(" DEFAULT PROVIDING MYCHARM COMPILER ")  
 myide = pycharm()  
 lap1 = laptop()  
 lap1.code(myide)

# OPERATOR OVERLOADING

print(" INTEGERS ")  
a=10  
b=5  
  
print(a+b)  
print(int.\_\_add\_\_(a,b))  
print(a-b)  
print(int.\_\_sub\_\_(a,b))  
print(a\*b)  
print(int.\_\_mul\_\_(a,b))  
print(a/b)  
print(int.\_\_divmod\_\_(a,b))  
  
print(" FLOATING POINT VARIABLES ")  
a=1.5  
b=2.5  
  
print(a+b)  
print(float.\_\_add\_\_(a,b))  
print(a-b)  
print(float.\_\_sub\_\_(a,b))  
print(a\*b)  
print(float.\_\_mul\_\_(a,b))  
print(a/b)  
print(float.\_\_divmod\_\_(a,b))  
  
  
print(" STRING DATA TYPES ")  
a="HELLO"  
b="WORLD"

print(a+b)  
print(str.\_\_add\_\_(a,b))  
  
print(str.\_\_mul\_\_(a,5))  
print(str.\_\_mul\_\_(b,5))  
  
print(a\*5)  
print(b\*5)

#CONVRTING TO STRING TYPE

a=10  
  
print(a)  
print(a.\_\_str\_\_())

# OPERATOR OVERLOADING  
  
class student:  
  
 def \_\_init\_\_(self):  
 self.m1=0  
 self.m2=0  
  
 def \_\_init\_\_(self,m1,m2):  
 self.m1=m1  
 self.m2=m2  
  
 def \_\_add\_\_(self, other):  
 m1 = self.m1+other.m1  
 m2 = self.m2 + other.m2  
 s3=student(m1,m2)  
  
 return s3  
  
 def \_\_sub\_\_(self, other):  
 m1 = self.m1 - other.m1  
 m2 = self.m2 - other.m2  
 s3 = student(m1, m2)  
  
 return s3  
  
 def \_\_mul\_\_(self, other):  
 m1 = self.m1 \* other.m1  
 m2 = self.m2 \* other.m2  
 s3 = student(m1, m2)  
  
 return s3  
  
 def \_\_eq\_\_(self, other):  
 if(self.m1==other.m1 and self.m2==other.m2 ):  
 return True  
 else:  
 return False  
  
 def \_\_gt\_\_(self, other):  
 if (self.m1 > other.m1 and self.m2 > other.m2):  
 return True  
 else:  
 return False  
  
 def \_\_ge\_\_(self, other):  
 if (self.m1 >= other.m1 and self.m2 >= other.m2):  
 return True  
 else:  
 return False  
  
 def \_\_lt\_\_(self, other):  
 if (self.m1 < other.m1 and self.m2 < other.m2):  
 return True  
 else:  
 return False  
  
 def \_\_le\_\_(self, other):  
 if (self.m1 <= other.m1 and self.m2 <= other.m2):  
 return True  
 else:  
 return False  
  
 def \_\_str\_\_(self):  
 return '{} {}'.format(self.m1,self.m2)  
  
  
  
 def display(self):  
 print("M1 : ",self.m1)  
 print("M2 : ",self.m2)  
  
  
s1=student(100,100)  
s2=student(50,50)  
s1.display()  
s2.display()  
  
s3=s1+s2  
s3.display()  
  
s3=s1-s2  
s3.display()  
  
s3=s1\*s2  
s3.display()  
  
if(s1==s2):  
 print("both the students are with SAME marks ")  
else:  
 print("both the students are with DIFFERENT marks ")  
  
if(s1>s2):  
 print("STUDENT IS GREATER THAN STUDENT 2 ")  
else:  
 print("STUDENT IS NOT GREATER THAN STUDENT 2")  
  
if(s1>=s2):  
 print("STUDENT IS GREATER THAN EQUAL TO STUDENT 2 ")  
else:  
 print("STUDENT IS LESS THAN STUDENT 2")  
  
if(s1<s2):  
 print("STUDENT IS LESSER THAN STUDENT 2 ")  
else:  
 print("STUDENT IS NOT LESSER THAN STUDENT 2")  
  
if(s1<=s2):  
 print("STUDENT IS LESSER THAN EQUAL TO STUDENT 2 ")  
else:  
 print("STUDENT IS GREATER THAN STUDENT 2")  
  
print(s1.\_\_str\_\_())  
print(s2.\_\_str\_\_())

# METHOD OVERLOADING   
  
class student:  
  
 def \_\_init\_\_(self,m1,m2):  
 self.m1=m1  
 self.m2=m2  
  
 def display(self):  
 print(self.m1,self.m2)  
  
  
 def sum(self,a=None,b=None,c=None):  
  
 if a!=None and b!=None and c!=None:  
 sum = a + b + c  
 elif a!=None and b!=None:  
 sum = a + b  
 else:  
 sum=a  
 return sum  
  
  
s1=student(100,100)  
s1.display()  
  
print(s1.sum(10,20,30))  
print(s1.sum(200,300))  
print(s1.sum(100,200))  
print(s1.sum(100))

#METHOD OVERLOADING USING DEFAULT PARAMETERS  
#COMPILE TIME POLYMORPHISM  
  
class demo:  
  
 def add(self,a=0,b=0,c=0,d=0):  
 add=a+b+c+d  
 print("ADDITION =",add)  
  
 def sum(self, a=None, b=None, c=None):  
 if a != None and b != None and c != None:  
 sum = a + b + c  
 elif a != None and b != None:  
 sum = a + b  
 else:  
 sum = a  
 print("SUM =", sum)  
  
  
 def display(self, a=None, b=None, c=None):  
 if a != None and b != None and c != None:  
 print(a,b,c)  
 elif a != None and b != None:  
 print(a,b)  
 elif a != None:  
 print(a)  
 else:  
 print("NO PARAMETER FOUND ")  
  
  
 def show(self,a=None,b=None,c=None,d=None):  
 print("A=",a,"B=",b,"C=",c,"D=",d)  
  
  
d=demo()  
d.add()  
d.add(1)  
d.add(1,2)  
d.add(1,2,3)  
d.add(1,2,3,4)  
  
d.sum()  
d.sum(1)  
d.sum(1,2)  
d.sum(1,2,3)  
  
d.display()  
d.display("MAHESH")  
d.display("MAHESH","SURYA")  
d.display("MAHESH","SURYA","SANKU")  
  
  
d.show()  
d.show(1)  
d.show(1,2)  
d.show(1,2,3)  
d.show(1,2,3,4)

#METHOD OVERLOADING

class human:  
  
 def \_\_init\_\_(self, a, b, c, d):  
 self.color = a  
 self.height = b  
 self.weight = c  
 self.hair = d  
 print("INIT METHOD WITH PARAMETERS CALLED ")  
  
  
  
 def display(self):  
 print(self.color)  
 print(self.height)  
 print(self.hair)  
 print(self.weight)  
  
 def walk(self,name=None):  
 print(str(name)+" WALKING ")  
  
 def talk(self,name=None):  
 print(str(name)+" TALKING ")  
  
  
 def eat(self,name=None):  
 print(str(name)+ " EATING ")  
  
  
hobj = human("BLACK",10,10,"WHITE")  
  
print(hobj.color)  
print(hobj.hair)  
print(hobj.height)  
print(hobj.weight)  
  
hobj.talk()  
hobj.walk("MAHESH")  
hobj.walk()  
hobj.eat("SURYA")  
hobj.eat()  
hobj.display()

# METHOD OVERRIDING  
  
class parent:  
 def show(self):  
 print(" IN PARENT SHOW ")  
  
class child(parent):  
 pass  
  
  
  
  
  
p1=parent()  
p1.show()  
  
c1=child()  
c1.show()

#RUNTIME POLYMORPHISM  
#METHOD OVERRIDING  
  
class parent:  
 def \_\_init\_\_(self):  
 print("PARENT CLASS INIT METHOD")  
  
 def transport(self):  
 print("PARENT CLASS TRANSPORT METHOD")  
 print("CYCLE")  
 print("RIKSHAW")  
  
class child(parent):  
 def \_\_int\_\_(self):  
 super().\_\_init\_\_()  
 print("CHILD CLASS INIT METHOD")  
  
 def transport(self):  
 super().transport()  
 print("CHILD CLASS TRANSPORT METHOD")  
 print("MOTOR - CYLE")  
 print("AUTO - RIKSHAW")  
  
  
p=parent()  
p.transport()  
  
c=child()  
c.transport()

# METHOD OVERRIDING  
  
class parent:  
 def show(self):  
 print(" IN PARENT SHOW ")  
  
class child(parent):  
 def show(self):  
 print(" IN CHILD SHOW ")  
  
  
p1=parent()  
p1.show()  
  
c1=child()  
c1.show()

#ABSTRACTION

# HAVE TO IMPORT ABC,abstractmethod from abc module

#decorator @abstractmethod should be provided at each and every abstract

#methods of the abstract class

#ABSTRCT CLASS CANNOT BE INSTANTIATED

#CONCRETE CLASSES CAN BE INSTANTIATED

#CONCRETE CLASS IS CLASS THAT EXTENDS THE ABSTRACT CLASS AND OVERRIDING

#ALL THE METHODS OF THE CLASS AND PROVIDING THE IMPELEMENTATION FOR THE

#ALL THE ABSTRACT METHODS … NOW THIS CONCRETE CLASS CAN BE INSTANTIATED

# OBJECTS CAN BE CREATED FOR THIS CNCRETE METHOD

IMP:

IF U FOR GOT TO OVER RIDE PARTICULAR ABSTRACT METHOD FROM THE ABSTRACT CLASS

THEN THE EXTENDING CLASS NOT PERFECTLY CONCRETE , IT IS STILL ABSTRACT

So WHEN EXTENDING A ABSTRACT CLASS INHERIT AND OVEERRIDE ALL THE METHODS OF THE CLASS AND PROVIDE IMPLEMENTATION OF ALL THE ABSTRAT METHODS

#from abc import ABC,abstractmethod

# @abstractmethod

MUST AND SHOULDUU

from abc import ABC,abstractmethod  
  
class abstractdemo(ABC):  
 @abstractmethod  
 def houseintrest(self):  
 None  
  
 @abstractmethod  
 def vehicleintrest(self):  
 None  
  
class sbi(abstractdemo):  
 def \_\_init\_\_(self):  
 print("STATE BANK OF INDIA PVT.LTD")  
  
 def houseintrest(self):  
 print("HOUSING INTREST 10%")  
  
  
 def vehicleintrest(self):  
 print("VEHICLE INTREST 20%")  
  
class lvb(abstractdemo):  
 def \_\_init\_\_(self):  
 print("LAXMI VILAS BANK PVT.LTD")  
  
 def houseintrest(self):  
 print("HOUSING INTREST 22%")  
  
  
 def vehicleintrest(self):  
 print("VEHICLE INTREST 33%")  
  
  
#obj=abstractdemo() cannot instantiate the abstract class directly  
#OBJECTS CAN BE CREATED AND A CLASS CAN BE INSTANTIATED ONLY FROM THE CONCRETE CLASS  
  
s=sbi()  
s.vehicleintrest()  
s.houseintrest()  
  
l=lvb()  
l.vehicleintrest()  
l.houseintrest()

#DATA HIDING  
  
class demo:  
 a=10  
 b=100  
 \_\_c=1000  
  
 def display(self):  
 print("DISPLAY METHOD IN demo CLASS")  
  
 def \_\_display(self):  
 print("PRIVATE SHOW METHOD IN demo CLASS")  
 print("PRIVATE MEMBERS OF DEMO CLASS ARE : ")  
 print("c ",self.\_\_c)  
  
  
  
obj=demo()  
obj.display()  
#obj.show()  
#print(obj.c)  
print(obj.b)  
print(obj.a)

#EXCEPTION HANDLING

print("HELLO MURUGAN NAANU SYNDICATE MEMBER PUSHPA PESIRE...EXCEPTION HANDLING")  
# EXCEPTION HANDLING  
#TRY EXCEPT BLOCKS  
try:  
 a = int(input(" ENTER A VALUE FOR NUMERATOR "))  
 b = int(input(" ENTER A VALUE FOR DENOMINATOR "))  
 print("WORKING STARTED")  
 print(a / b)  
 print("WORKING ENDED ")  
   
except ZeroDivisionError as e:  
 print("YOU CANNOT DIVIDE /0",e)  
 print("WORKING ENDED WITH AN EXCEPTION")  
  
except ValueError as e:  
 print("VALUE ERROR ", e)  
 print("WORKING ENDED WITH AN EXCEPTION")  
  
except Exception as e:  
 print("SOMETHING WENT WRONG OOPS ", e)  
 print("WORKING ENDED WITH AN EXCEPTION")  
  
finally:  
 print("BYE BYE :)")  
  
""" SAME SAME  
except Exception as e:  
 print(e,"YOU CANNOT DIVIDE /0")  
   
except Exception :  
 print(e,"YOU CANNOT DIVIDE /0")  
"""

try:  
 print(10+"surya")  
except:  
 print("OOPS....Exception")  
else:  
 print("NO EXCEPTION")  
  
try:  
 print(10+"surya")  
except Exception:  
 print(Exception)  
else:  
 print("NO EXCEPTION")  
  
try:  
 print(10+"surya")  
except Exception as e:  
 print(e)  
else:  
 print("NO EXCEPTION")  
finally:  
 print("BYE BYE")

#ZeroDivisionError  
  
try:  
 print(1/0)  
except Exception:  
 print("OOPS , EXCEPTION RAISED : ",Exception)  
  
try:  
 print(1/0)  
except Exception as e:  
 print("OOPS , EXCEPTION RAISED : ", e)  
try:  
 print(1/0)  
except ZeroDivisionError as e:  
 print("OOPS , EXCEPTION RAISED : ", e)  
  
#CUSTOMISED EXCEPTION MESSAGE  
try:  
 print(1/0)  
except:  
 print(" ZERO SHOULD NOT BE IN DENOMINATOR ")  
  
#ValueError  
  
print(" IF U GIVE INPUT AS INTEGER ITS FINE ELSE , IT RAISES VALUE ERROR ")  
try:  
 a=int(input("ENTER A VALUE : "))  
except ValueError as e:  
 print("OOPS , EXCEPTION RAISED : ", e)  
  
#if we know about the type of error, then name\_of\_error as e else u can use  
# Exception as e ...  
  
print(" IF U GIVE INPUT AS INTEGER ITS FINE ELSE , IT RAISES VALUE ERROR ")  
try:  
 a=int(input("ENTER A VALUE : "))  
except Exception as e:  
 print("OOPS , EXCEPTION RAISED : ", e)  
  
print(" IF U GIVE INPUT AS FLOAT ITS FINE ELSE , IT RAISES VALUE ERROR ")  
try:  
 a=float(input("ENTER A VALUE : "))  
except ValueError as e:  
 print("OOPS , EXCEPTION RAISED : ",e)  
  
#IndexError  
  
try:  
 l=[1,2,3,4]  
 print(l[10])  
  
except IndexError as e:  
 print("OOPS , EXCEPTION RAISED : ", e)  
  
  
class demo:  
 a=10  
  
obj=demo()  
#a exist , b not exists raises exception  
try:  
 print(obj.a)  
 print(obj.b)  
except Exception as e:  
 print("OOPS , EXCEPTION RAISED : ", e)  
  
#AttributeError  
  
try:  
 print(obj.a)  
 print(obj.c)  
except AttributeError as e:  
 print("OOPS , EXCEPTION RAISED : ", e)  
  
#FileNotFoundError  
  
#for providing customised statements for exceptions  
try:  
 fo=open("surya.txt","r")  
 print(fo.read())  
 fo.close()  
 print("FILES MOWA")  
except:  
 print(" FILE NOT FOUND ERROR ")  
  
try:  
 fo=open("surya.txt","r")  
 print(fo.read())  
 fo.close()  
 print("FILES MOWA")  
except FileNotFoundError as e :  
 print(" FILE NOT FOUND ERROR EXCEPTION RAISED ",e )  
  
#NameError  
  
try:  
 a=10  
 print(z)  
except NameError as e :  
 print("EXCEPTION RAISED ",e )  
  
try:  
 a=10  
 print("a value = ",a)  
except Exception as e:  
 print(" EXCEPTION RAISED ",e)  
else:  
 print(" HOHO NO EXCEPTIONS RAISED HERE ...")  
  
#ELSE BLOCK  
#finally block  
finally:  
 print("\n\n BYE BYE BAAABU BYE BYE BYE BYE , BYE BYE BAABU...\n\n")

#EXCEPTION HANDLING

weeks=("sunday","monday","tuesday","wednesday","thursday","friday","saturday")  
print(weeks)  
print(type(weeks))  
  
try:  
 weeks[2]="SURYADAY"  
except:  
 print("SOMETHING WENT WRONG")  
else:  
 print("NO EXCEPTION")  
finally:  
 print("BYE BYE ")

class hello():  
 def run(self):  
 for i in range(5):  
 print("HELLO")  
  
  
class hii:  
 def run(self):  
 for i in range(5):  
 print("HII")  
  
  
t1=hello()  
t2=hii()  
  
t1.run()  
t2.run()

#MULTI THREADING  
  
from threading import \*  
from time import sleep  
  
class hello(Thread):  
 def run(self):  
 for i in range(50):  
 print("HELLO")  
 sleep(2)  
  
  
class hii(Thread):  
 def run(self):  
 for i in range(10):  
 print("HII")  
 sleep(2)  
  
  
t1 = hello()  
t2 = hii()  
  
t1.start()  
sleep(0.2)  
t2.start()  
  
t1.join()  
t2.join()  
print("BYE")

#COMMENTS IN PYTHON  
  
# for single line comments  
  
*"""  
 for multiline comments  
  
 """*

USERDEFINED EXCEPTIONS

age=int(input("ENTER AGE : "))  
  
try:  
 if age < 18:  
 raise Exception(" ONLY ADULTS ARE ALLOWED TO VOTE ")  
 else:  
 print(" U CAN CAST YOUR VOTE ! ")  
except Exception as e:  
 print("OOPS !",e)  
 print(" SORRY ! ")

#IMPORT MATH LIBRARY

# Import math Library  
import math  
  
# Print the value of Euler e  
print(math.e)  
  
# Print the value of pi  
print(math.pi)  
  
# Import math Library  
import math  
  
# radius of the circle  
r = 4  
  
# value of pie  
pie = math.pi  
  
# area of the circle  
print(pie \* r \* r)  
  
# Import math Library  
import math  
  
# Print the value of tau  
print(math.tau)  
  
# Import math Library  
import math  
  
# Print the positive infinity  
print(math.inf)  
  
# Print the negative infinity  
print(-math.inf)  
  
a = 2.3  
  
# returning the ceil of 2.3  
print("The ceil of 2.3 is : ", end="")  
print(math.ceil(a))  
  
# returning the floor of 2.3  
print("The floor of 2.3 is : ", end="")  
print(math.floor(a))  
  
a = 5  
  
# returning the factorial of 5  
print("The factorial of 5 is : ", end="")  
print(math.factorial(a))  
  
a = 15  
b = 5  
  
# returning the gcd of 15 and 5  
print("The gcd of 5 and 15 is : ", end="")  
print(math.gcd(b, a))  
  
a = -10  
  
# returning the absolute value.  
print("The absolute value of -10 is : ", end="")  
print(math.fabs(a))  
  
# initializing the value  
test\_int = 4  
test\_neg\_int = -3  
test\_float = 0.00  
  
# checking exp() values  
# with different numbers  
print(math.exp(test\_int))  
print(math.exp(test\_neg\_int))  
print(math.exp(test\_float))  
  
print("The value of 3\*\*4 is : ", end="")  
  
# Returns 81  
print(pow(3, 4))  
  
# returning the log of 2,3  
print("The value of log 2 with base 3 is : ", end="")  
print(math.log(2, 3))  
  
# returning the log2 of 16  
print("The value of log2 of 16 is : ", end="")  
print(math.log2(16))  
  
# returning the log10 of 10000  
print("The value of log10 of 10000 is : ", end="")  
print(math.log10(10000))  
  
# print the square root of 0  
print(math.sqrt(0))  
  
# print the square root of 4  
print(math.sqrt(4))  
  
# print the square root of 3.5  
print(math.sqrt(3.5))  
  
a = math.pi / 6  
  
# returning the value of sine of pi/6  
print("The value of sine of pi/6 is : ", end="")  
print(math.sin(a))  
  
# returning the value of cosine of pi/6  
print("The value of cosine of pi/6 is : ", end="")  
print(math.cos(a))  
  
# returning the value of tangent of pi/6  
print("The value of tangent of pi/6 is : ", end="")  
print(math.tan(a))  
  
a = math.pi / 6  
b = 30  
  
# returning the converted value from radians to degrees  
print("The converted value from radians to degrees is : ", end="")  
print(math.degrees(a))  
  
# returning the converted value from degrees to radians  
print("The converted value from degrees to radians is : ", end="")  
print(math.radians(b))  
  
# initializing argument  
gamma\_var = 6  
  
# Printing the gamma value.  
print("The gamma value of the given argument is : "  
 + str(math.gamma(gamma\_var)))

* CALENDAR MODULE

# Python program to display calendar of  
# given month of the year  
  
# import module  
import calendar  
  
yy = 2017  
mm = 11  
  
# display the calendar  
print(calendar.month(yy, mm))  
  
# using calendar to print calendar of year  
# prints calendar of 2018  
print("The calendar of year 2018 is : ")  
print(calendar.calendar(2018, 2, 1, 6))  
  
# Python code to demonstrate the working of  
# calendar() and firstweeksday()  
  
# importing calendar module for calendar operations  
import calendar  
  
# using calendar to print calendar of year  
# prints calendar of 2012  
print("The calendar of year 2012 is : ")  
print(calendar.calendar(2012, 2, 1, 6))  
  
# using firstweekday() to print starting day number  
print("The starting day number in calendar is : ", end="")  
print(calendar.firstweekday())  
  
# Python code to demonstrate the working of  
# isleap() and leapdays()  
  
# importing calendar module for calendar operations  
import calendar  
  
# using isleap() to check if year is leap or not  
if (calendar.isleap(2008)):  
 print("The year is leap")  
else:  
 print("The year is not leap")  
  
# using leapdays() to print leap days between years  
print("The leap days between 1950 and 2000 are : ", end="")  
print(calendar.leapdays(1950, 2000))  
  
# using month() to display month of specific year  
print("The month 5th of 2016 is :")  
print(calendar.month(2016, 5, 2, 1))  
  
# using month() to display month of specific year  
print("The month 5th of 2016 is :")  
print(calendar.month(2016, 5, 2, 1))  
  
# using prmonth() to print calendar of 1997  
print("The 4th month of 1997 is : ")  
calendar.prmonth(1997, 4, 2, 1)  
  
# using setfirstweekday() to set first week day number  
calendar.setfirstweekday(4)  
  
print("\r")  
  
# using firstweekday() to check the changed day  
print("The new week day number is : ", end="")  
print(calendar.firstweekday())  
  
# using weekday() to print day number of date  
print("The day number of 25 April 1997 is : ", end="")  
print(calendar.weekday(1997, 4, 25))

GETTING WEEK DAY NAME FROM THE GIVEN DATE

# MONTH DAY YEAR  
  
import calendar  
n1,n2,n3=map(int,input().split())  
print((calendar.day\_name[calendar.weekday(n3,n1,n2)]).upper())

RANDOM MODULE

# import random  
import random  
  
# prints a random value from the list  
list1 = [1, 2, 3, 4, 5, 6]  
print(random.choice(list1))  
  
random.seed(5)  
  
print(random.random())  
print(random.random())  
  
# Generates a random number between  
# a given positive range  
r1 = random.randint(5, 15)  
print("Random number between 5 and 15 is % s" % (r1))  
  
# Generates a random number between  
# two given negative range  
r2 = random.randint(-10, -2)  
print("Random number between -10 and -2 is % d" % (r2))  
  
# Python3 program to demonstrate  
# the use of random() function .  
  
# import random  
from random import random  
  
# Prints random item  
print(random())  
  
# Python3 program to demonstrate the use of  
# choice() method  
  
# import random  
import random  
  
# prints a random value from the list  
list1 = [1, 2, 3, 4, 5, 6]  
print(random.choice(list1))  
  
# prints a random item from the string  
string = "geeks"  
print(random.choice(string))  
  
# prints a random item from the tuple  
tuple1 = (1, 2, 3, 4, 5)  
print(random.choice(tuple1))  
  
# import the random module  
import random  
  
# declare a list  
sample\_list = [1, 2, 3, 4, 5]  
  
print("Original list : ")  
print(sample\_list)  
  
# first shuffle  
random.shuffle(sample\_list)  
print("\nAfter the first shuffle : ")  
print(sample\_list)  
  
# second shuffle  
random.shuffle(sample\_list)  
print("\nAfter the second shuffle : ")  
print(sample\_list)

PATTERNS

for i in range(5):  
 for j in range(i + 1):  
 print(" \* ", end=" ")  
 print(" ")  
  
print(" ")  
  
for i in range(0, 5):  
 # inner loop to handle number of columns  
 # values changing acc. to outer loop  
 for j in range(0, i + 1):  
 # printing stars  
 print(" \* ", end=" ")  
 print(" ")  
  
for i in range(5):  
 for j in range(i + 1):  
 print((j + 1), " ", end=" ")  
 print(" ")  
  
print(" ")  
print(" ")  
  
count = 0  
for i in range(5):  
 for j in range(i + 1):  
 count += 1  
 print(count, " ", end=" ")  
 print(" ")  
  
print(" ")  
  
start = 64  
for i in range(5):  
 for j in range(i + 1):  
 start += 1  
 print(chr(start), " ", end=" ")  
 print(" ")  
  
print(" ")  
print(" ")  
  
start = 64  
for i in range(5):  
 start += 1  
 for j in range(i + 1):  
 print(chr(start), " ", end=" ")  
 print(" ")  
  
print(" ")  
print(" ")  
  
start = 64  
for i in range(5):  
 start += 1  
 for j in range(i + 1):  
 print(chr(start), " ", end=" ")  
 print(" ")  
  
print(" ")  
print(" ")  
  
for i in range(5):  
 for j in range(5):  
 if (i + j) >= 4:  
 print("\* ", end=" ")  
 else:  
 print(" ", end=" ")  
  
 print(" ")  
  
print(" ")  
print(" ")  
  
for i in range(5):  
 for j in range(5):  
 if (i + j) >= 4:  
 print(j, end=" ")  
 else:  
 print(" ", end=" ")  
  
 print(" ")  
  
print(" ")  
print(" ")  
  
for i in range(5):  
 for j in range(5):  
 if (i + j) >= 4:  
 print("# ", end=" ")  
 else:  
 print(" ", end=" ")  
  
 print(" ")  
  
print(" ")  
print(" ")  
  
  
# Python 3.x code to demonstrate star pattern  
  
# Function to demonstrate printing pattern triangle  
def triangle(n):  
 # number of spaces  
 k = n - 1  
  
 # outer loop to handle number of rows  
 for i in range(0, n):  
  
 # inner loop to handle number spaces  
 # values changing acc. to requirement  
 for j in range(0, k):  
 print(end=" ")  
  
 # decrementing k after each loop  
 k = k - 1  
  
 # inner loop to handle number of columns  
 # values changing acc. to outer loop  
 for j in range(0, i + 1):  
 # printing stars  
 print("\* ", end="")  
  
 # ending line after each row  
 print("\r")  
  
  
# Driver Code  
n = 5  
triangle(n)  
  
rows = 6  
  
for num in range(rows):  
  
 for i in range(num):  
 print(num, end=" ") # print number  
  
 # line after each row to display pattern correctly  
  
 print(" ")  
  
rows = 6  
x = 0  
for num in range(rows):  
 for i in range(num):  
 x += 1  
 print(x, end=" ") # print number  
  
 # line after each row to display pattern correctly  
  
 print(" ")  
  
print(" \n")  
  
rows = 6  
x = -1  
for num in range(rows):  
 x += 1  
 for i in range(num):  
 print(x, end=" ") # print number  
  
 # line after each row to display pattern correctly  
  
 print(" ")  
  
print(" \n")  
  
print(" \n")  
rows = 5  
for num in range(rows, 0, -1):  
  
 for i in range(num):  
 print(num, end=" ") # print number  
  
 # line after each row to display pattern correctly  
  
 print(" ")  
  
print(" \n")  
rows = 5  
x = 0  
for num in range(rows, 0, -1):  
 x += 1  
 for i in range(num):  
 print(x, end=" ") # print number  
  
 # line after each row to display pattern correctly  
  
 print(" ")  
  
rows = 6  
start = 64  
for num in range(rows):  
  
 for i in range(num):  
 print(chr(num + 64), end=" ") # print number  
  
 # line after each row to display pattern correctly  
  
 print(" ")  
  
rows = 6  
for num in range(rows):  
  
 for i in range(num):  
 print(chr(i + 65), end=" ") # print number  
  
 # line after each row to display pattern correctly  
  
 print(" ")

rows = 5

for i in range(rows, 0, -1):

num = i

for j in range(0, i):

print(num, end=" ")

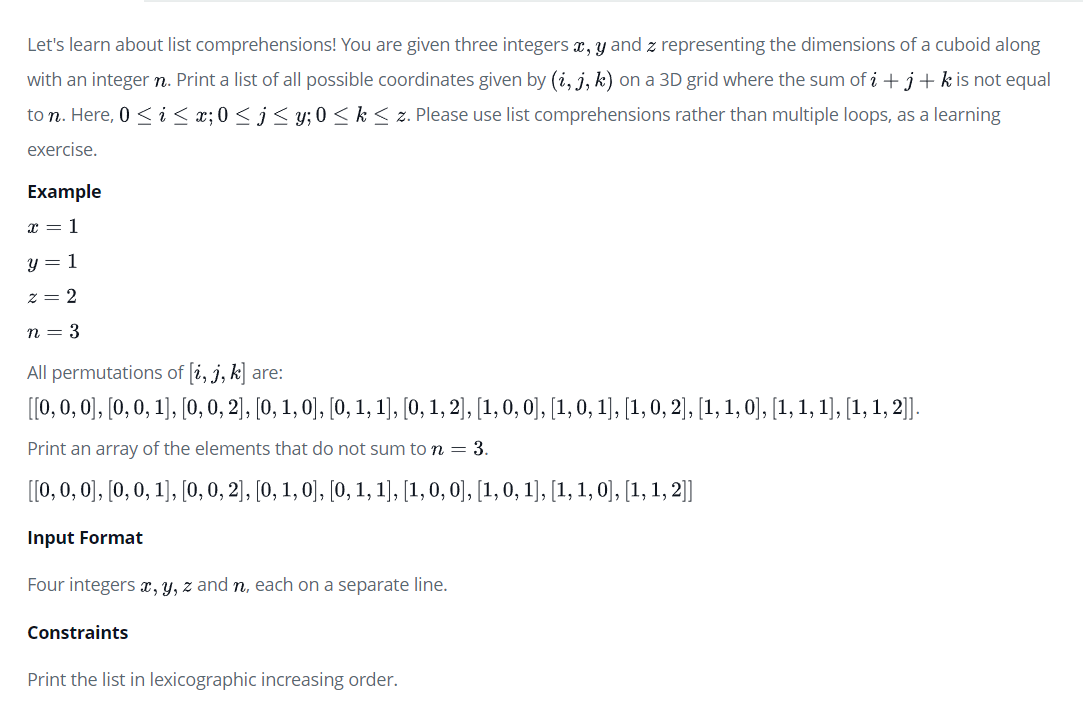
print("\n")

#PATTERNS  
  
a =int(input(" ENTER THE SIZE OF THE PATTERN: "))  
  
i=0  
while i<a:  
 j=0  
 while j<a:  
 print("# ",end=" ")  
 j+=1  
 print(" ")  
 i+=1  
  
print(" ")  
i=0  
while i<a:  
 print("\* " \* a)  
 i+=1  
  
  
print(" ")  
i=0  
for i in range(a):  
 j=0  
 for j in range(a):  
 print("% ",end=" ")  
 j+=1  
 print(" ")  
 i+=1  
  
print(" ")  
i=0  
for i in range(a):  
 j=0  
 while j<a:  
 print("& ",end=" ")  
 j+=1  
 print(" ")  
 i+=1  
  
print(" ")  
i=0  
for i in range(a+1):  
 j=0  
 for j in range(i):  
 print("% ",end=" ")  
 j+=1  
 print(" ")  
 i+=1  
  
  
print(" ")  
i=0  
for i in range(a):  
 j=0  
 for j in range(a):  
 if(i+j)>=a-1:  
 print("% ", end=" ")  
 else:  
 print(" ", end=" ")  
 j += 1  
 print(" ")  
 i += 1  
  
print(" ")  
i=0  
for i in range(a):  
 j=0  
 for j in range(a):  
 if(i+j)>=a:  
 print(" ", end=" ")  
 else:  
 print("% ", end=" ")  
 j += 1  
 print(" ")  
 i += 1  
  
  
print(" ")  
i=0  
for i in range(a+1):  
  
 j=0  
 for j in range(a + 1):  
 if j <= i:  
 print(" ", end=" ")  
 else:  
 print("% ", end=" ")  
 j += 1  
 print(" ")  
 i+=1

#PATTERNS  
#WHILE LOOP  
i=0  
while(i<10):  
 print("SURYA MAHESH")  
 i+=1  
  
#PATTERN  
"""  
\* \* \* \* \*   
\* \* \* \* \*   
\* \* \* \* \*   
\* \* \* \* \*   
\* \* \* \* \*   
  
"""  
  
i=0  
while(i<5):  
 j = 0  
 while(j<5):  
 print("\* ",end=" ")  
 j+=1  
 print(" ")  
 i+=1  
  
print(" ")  
  
"""  
$ $ $ $ $   
$ $ $ $ $   
$ $ $ $ $   
$ $ $ $ $   
$ $ $ $ $   
  
"""  
  
i=0  
while(i<5):  
 j = 0  
 while(j<5):  
 print("$ ",end=" ")  
 j+=1  
 print(" ")  
 i+=1

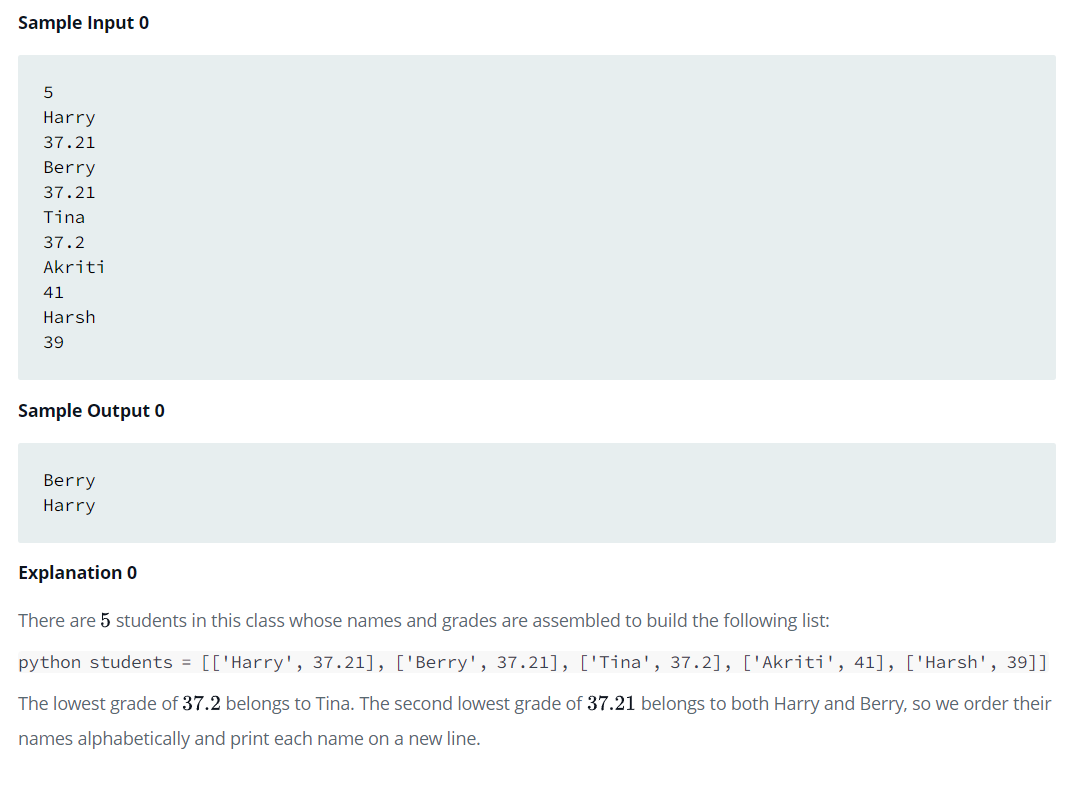
IMPORTANT THING FOR HACKER RANK PROBLEMS

a="1 2 3 4 5"  
l1=list(a.split())  
print(l1)  
  
l2=list(map(int,l1))  
print(l2)

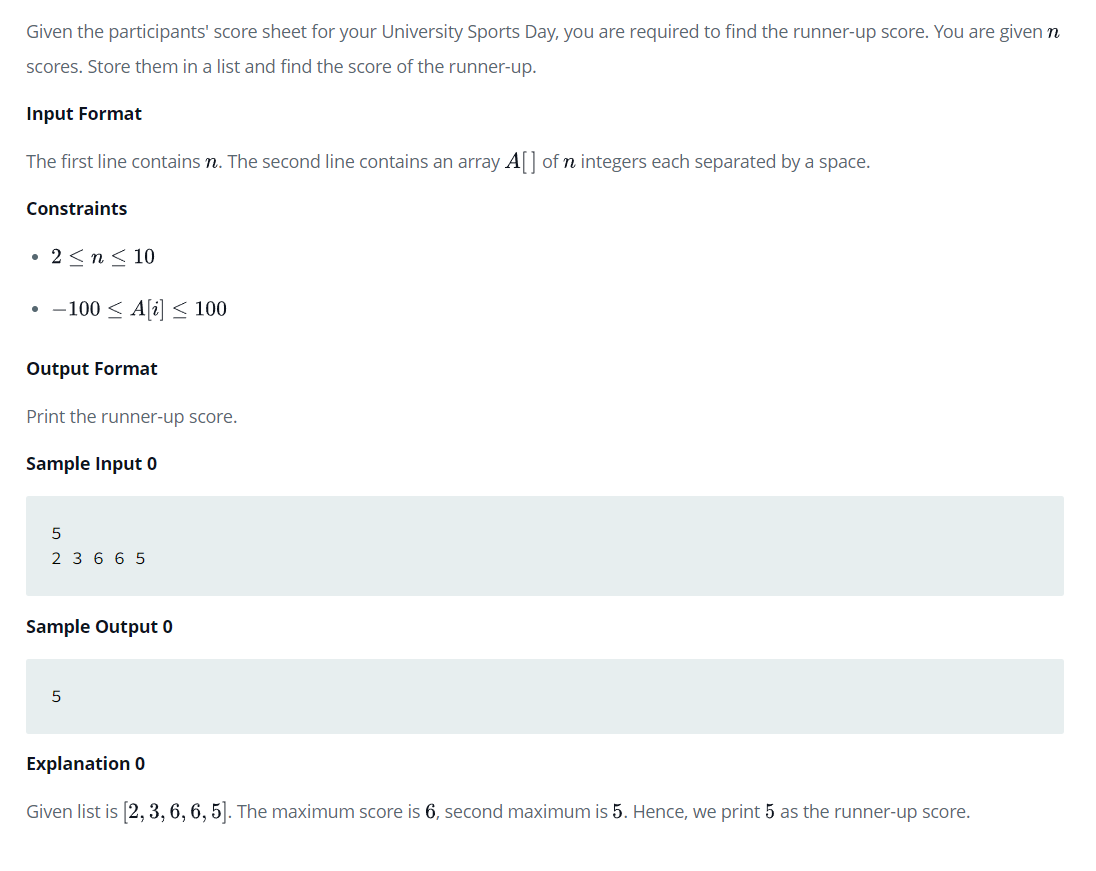


LIST COMPREHENSIONS

x = int(input())  
y = int(input())  
z = int(input())  
n = int(input())  
  
  
print([[a, b, c] for a in range(x + 1) for b in range(y + 1) for c in range(z + 1) if a + b + c != n])

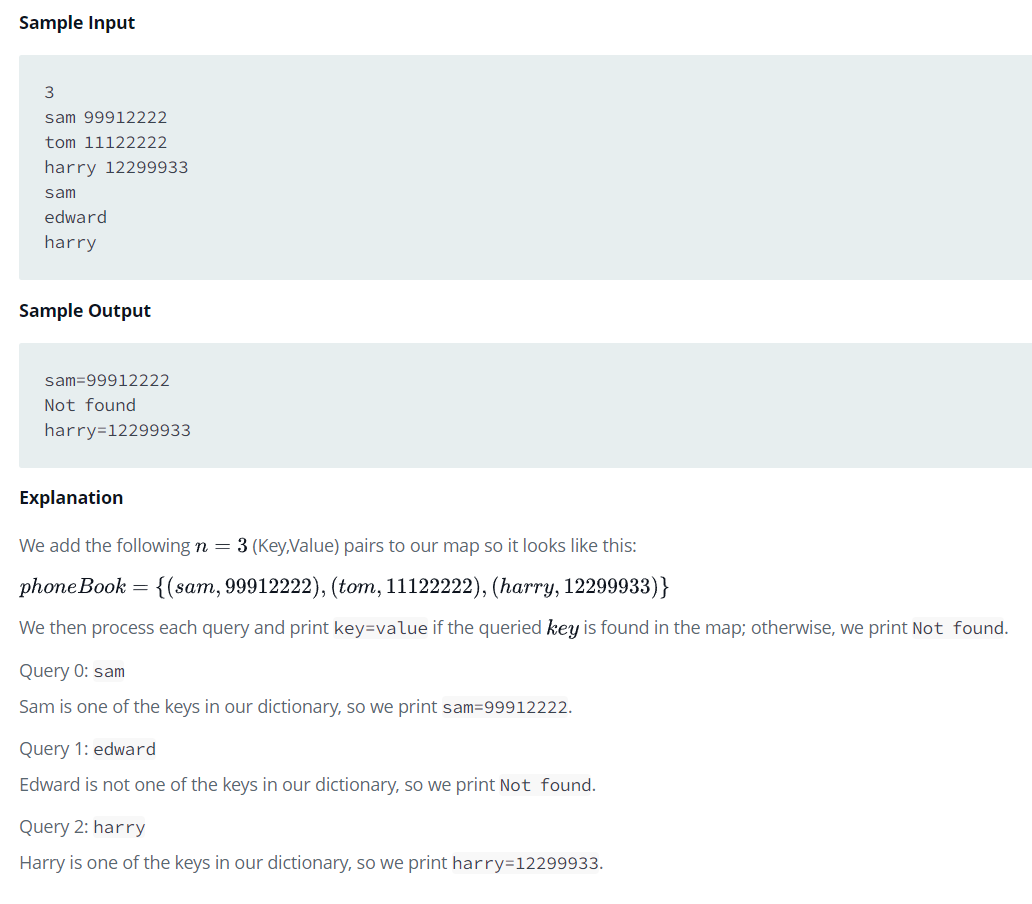


*'''  
  
  
l=[['ha', 37.21], ['be', 37.21], ['t', 37.2], ['a', 41.0], ['harsh', 39.0]]  
  
  
  
t=int(input())  
l=[]  
for i in range(t):  
 l1=[]  
 l1.append(input())  
 l1.append(float(input()))  
 l.append(l1)  
  
  
'''*t=int(input())  
l=[]  
for i in range(t):  
 l1=[]  
 l1.append(input())  
 l1.append(float(input()))  
 l.append(l1)  
  
#print(l)  
  
d=dict()  
for item in l:  
 d[item[1]]=item[0]  
  
s=set(d.keys())  
#print(s)  
l2=list(s)  
l2=sorted(l2)  
l2.remove(min(l2))  
  
req=l2[0]  
#print(l2)  
  
l3=[]  
for item in l:  
 if item[1]==req:  
 l3.append(item[0])  
 #print(item[0])  
  
  
l3=sorted(l3)  
  
for i in l3:  
 print(i)

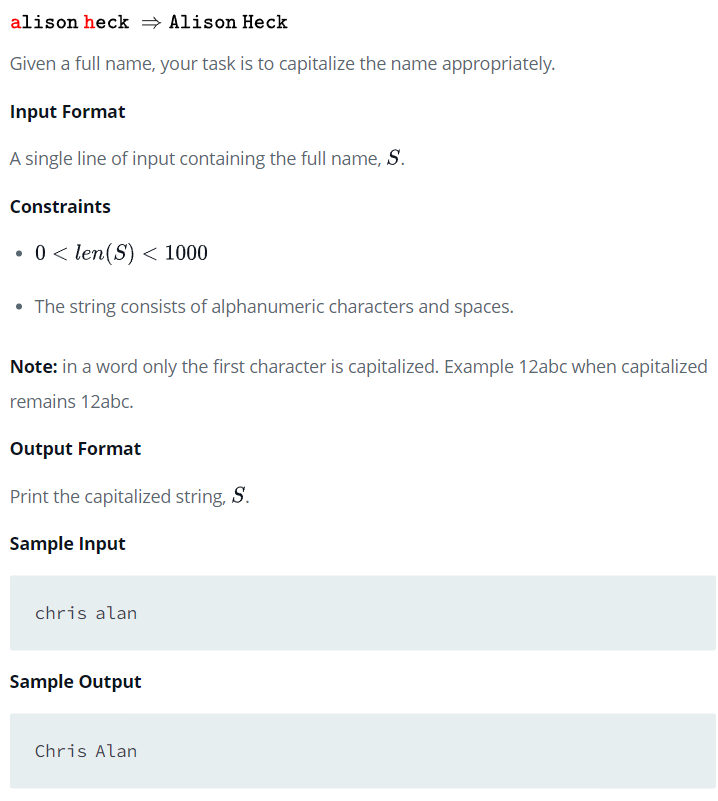


#SOLUTION -1

n = int(input())  
arr = map(int, input().split())  
print (sorted(set(arr))[-2])  
  
  
#SOLUTION -2  
  
n= int(input())  
l = []  
  
for i in range(n):  
 x=int(input())  
 l.append(x)  
  
s = set(l)  
s.remove(max(s))  
l = list(s)  
print(max(l))



#SOLUTION 1  
  
t=int(input())  
l=list()  
d=dict()  
for i in range(t):  
 l=list(input().split())  
 d[l[0]]=l[1]  
  
while True:  
 try:  
 name=input()  
 if name in d:  
 print("%s=%s"%(name,d[name]))  
 else:  
 print('Not found')  
 except:  
 break  
  
  
#SOLUTION 2  
  
n = int(input())  
name\_numbers = [input().split() for \_ in range(n)]  
phone\_book = {k: v for k, v in name\_numbers}  
while True:  
 try:  
 name = input()  
 if name in phone\_book:  
 print('%s=%s' % (name, phone\_book[name]))  
 else:  
 print('Not found')  
 except:  
 break



s1="surya mahesh"  
s2=s1.split()  
print(" ".join(i.capitalize() for i in s2))  
  
s1="surya mahesh"  
s2=s1.split()  
print("@".join(i.capitalize() for i in s2))  
  
s1="surya mahesh"  
s2=s1.split()  
print("%".join(i.capitalize() for i in s2))  
  
s1="surya mahesh"  
s2=s1.split()  
print("&".join(i.capitalize() for i in s2))

#SUBSTRING COUNT IN MAIN STRING( REPETITIONS)

def count\_substring(string, sub\_string):  
 count = 0  
 for i in range(len(string)):  
 temp = string[i:(i + len(sub\_string))]  
 if temp == sub\_string:  
 count += 1  
  
 return count

#TREES , DATA STRUCTURES , TREE TRAVERSALS

class node:  
 def \_\_init\_\_(self,data):  
 self.leftnode=None  
 self.rightnode=None  
 self.nodedata=data  
  
  
root=node(1)  
root.leftnode=node(2)  
root.rightnode=node(3)  
  
root.leftnode.leftnode=node(4)  
root.leftnode.rightnode=node(5)  
  
root.rightnode.leftnode=node(6)  
root.rightnode.rightnode=node(7)  
  
def inorder(root):  
 if root:  
 inorder(root.leftnode)  
 print(root.nodedata,end=" ")  
 inorder(root.rightnode)  
  
print("INORDER TRAVERSAL")  
inorder(root)  
print(" ")  
def preorder(root):  
 if root:  
 print(root.nodedata,end=" ")  
 preorder(root.leftnode)  
 preorder(root.rightnode)  
  
  
print("PREORDER TRAVERSAL")  
preorder(root)  
print(" ")  
  
def postorder(root):  
 if root:  
 postorder(root.leftnode)  
 postorder(root.rightnode)  
 print(root.nodedata,end=" ")  
  
print("POSTORDER TRAVERSAL")  
postorder(root)  
print(" ")

#ARRAYS ( DATA STRUCTURES)

import array as arr  
a=arr.array("i",[1,2,3,4,5,6,7,8,9])  
print(a)  
  
for item in a:  
 print(item,end=" ")  
print(" ")  
  
a.append(10)  
for item in a:  
 print(item,end=" ")  
  
print(" ")  
print(a.pop())  
  
for item in a:  
 print(item,end=" ")  
print(" ")  
  
a.extend([10,11,12,13,14,15])  
for item in a:  
 print(item,end=" ")  
print(" ")  
  
print(a.count(1))  
a.remove(5)  
for item in a:  
 print(item,end=" ")  
print(" ")  
  
a.insert(2,100)  
  
for item in a:  
 print(item,end=" ")  
print(" ")  
  
a.reverse()  
for item in a:  
 print(item,end=" ")  
print(" ")  
  
print(a.index(100))  
  
l=a.tolist()  
print(l)  
  
  
a.tounicode()  
for item in a:  
 print(item,end=" ")  
print(" ")

#STACK USING LISTS

#STACKS USING LISTS  
  
stack=[]  
def empty(stack):  
 if len(stack)==0:  
 print("STACK IS EMPTY")  
 else:  
 print("STACK IS NOT EMPTY")  
  
stack.append(1)  
stack.append(2)  
stack.append(3)  
stack.append(4)  
  
print(stack)  
empty(stack)  
try:  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
except:  
 print("STACK UNDERFLOW")  
  
empty(stack)  
print(stack)

STACK USING ARRAYS

#STACKS USING ARRAYS  
import array as arr  
  
stack=arr.array("i")  
  
def empty(stack):  
 if len(stack)==0:  
 print("STACK IS EMPTY")  
 else:  
 print("STACK IS NOT EMPTY")  
  
stack.append(1)  
stack.append(2)  
stack.append(3)  
stack.append(4)  
  
print(stack)  
empty(stack)  
try:  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
 print(stack.pop(), "POPPED OUT")  
except:  
 print("STACK UNDERFLOW")  
  
empty(stack)  
print(stack)

STACK IMPLEMENTATION USING LISTS AND CLASSES

class stack:  
 def \_\_init\_\_(self):  
 self.items=[]  
 def isempty(self):  
 if len(self.items)==0:  
 print("STACK IS EMPTY")  
 else:  
 print("STACK IS NOT EMPTY")  
 #or  
 if self.items==[]:  
 return True  
 else:  
 return False  
 def push(self,data):  
 self.items.append(data)  
 def pop(self):  
 return self.items.pop()  
  
s=stack()  
s.push(10)  
s.push(20)  
s.push(30)  
s.push(40)  
s.push(50)  
  
print(s.items)  
print(s.isempty())  
  
print(s.pop(),"POPPED OUT")  
print(s.pop(),"POPPED OUT")  
print(s.pop(),"POPPED OUT")  
  
print(s.items)  
print(s.isempty())  
print(s.pop(),"POPPED OUT")  
print(s.pop(),"POPPED OUT")  
  
print(s.items)  
print(s.isempty())

QUEUE USING (from collecions import deque )

from collections import deque  
  
q=deque()  
  
q.append(1)  
q.append("hello")  
q.append(3.14)  
  
print(q)  
  
q.appendleft(2)  
q.appendleft("hii")  
q.appendleft(2.78)  
  
print(q)  
print(q.pop())  
print(q.pop())  
print(q.pop())  
  
print(q)  
  
print(q.popleft())  
print(q.popleft())  
print(q.popleft())  
  
print(q)

QUEUE’S FROM COLLETIONS

from collections import deque  
  
queue=deque()  
  
print(queue)  
for i in range(1,6):  
 queue.append(i)  
 print(queue)  
  
for i in range(6,11):  
 queue.appendleft(i)  
 print(queue)  
  
for i in range(1,6):  
 queue.pop()  
 print(queue)  
  
for i in range(6,11):  
 queue.popleft()  
 print(queue)

LINKED LISTS IN PYTHON

# A simple Python program for traversal of a linked list  
  
# Node class  
class Node:  
  
 # Function to initialise the node object  
 def \_\_init\_\_(self, data):  
 self.data = data # Assign data  
 self.next = None # Initialize next as null  
  
  
# Linked List class contains a Node object  
class LinkedList:  
  
 # Function to initialize head  
 def \_\_init\_\_(self):  
 self.head = None  
  
 # This function prints contents of linked list  
 # starting from head  
 def printList(self):  
 temp = self.head  
 while (temp):  
 print (temp.data)  
 temp = temp.next  
  
  
# Code execution starts here  
if \_\_name\_\_=='\_\_main\_\_':  
  
 # Start with the empty list  
 llist = LinkedList()  
  
 llist.head = Node(1)  
 second = Node(2)  
 third = Node(3)  
  
 llist.head.next = second; # Link first node with second  
 second.next = third; # Link second node with the third node  
  
 llist.printList()

CUSTOM LINKED LISTS (USERDEFINED)

#CUSTOM LINKED LISTS  
  
class node:  
 def \_\_init\_\_(self,data):  
 self.data=data  
 self.next=None  
  
class linkedlist:  
 def \_\_init\_\_(self):  
 self.head=None  
  
 def show(self):  
 node=self.head  
 while node is not None:  
 print(node.data,"-->",end=" ")  
 node=node.next  
 print("None")  
 print(" ")  
 def insert(self,data):  
 newnode=node(data)  
 newnode.next=self.head  
 self.head=newnode  
  
  
  
l=linkedlist()  
node1=node(5)  
l.head=node1  
l.show()  
l.insert(4)  
l.show()  
l.insert(3)  
l.show()  
l.insert(2)  
l.show()  
l.insert(1)  
l.show()

REVERSE AN INTEGER

n=int(input("ENTER INTEGER : "))  
temp=n  
rev=0  
while n>0:  
 d=n%10  
 rev=rev\*10+d  
 n=n//10  
  
print("REVERSED INTEGER : ",rev)

TUPLE OF VALUES , LIST OF VALUES INPUTED BY USER

a=()  
l=[]  
n= int(input("ENTER LIMIT : "))  
for i in range(n):  
 item=int(input("ENTER ELEMENT VALUE: "))  
 l.append(item)  
  
  
print(l)  
a=tuple(l)  
#or a=a(l)  
#or a= tuple(l)  
print(a)

PRIME OR NOT

num=int(input("ENTER A NUMBER TO CHECK ITS PRIME OR NOT : "))  
limit=int(num/2)+1  
prime=True  
for i in range(2,limit):  
 rem=num%i  
 if rem==0:  
 prime = False  
 print(num," IS NOT PRIME NUMBER ")  
 break  
  
if prime==True:  
 print(num," IS PRIME NUMBER ")

PRIME OR NOT METHOD TWO

num=int(input("ENTER A NUMBER TO CHECK ITS PRIME OR NOT : "))  
# "//" is INTEGER DIVISION  
limit=num//2+1  
for i in range(2,limit):  
 rem=num%i  
 if rem==0:  
 print(num," IS NOT PRIME NUMBER ")  
 break  
  
else:  
 print(num," IS PRIME NUMBER ")

PYTHON TO CREATE A PHONE DICTIONARY

limit=int(input("ENTER LIMIT : "))  
phbook={ }  
  
# m=dict()  
phnum=None  
name=None  
  
for i in range(limit):  
 phnum=int(input("ENTER PHONE NUMBER : "))  
 name=input("ENTER NAME : ")  
 #TO ADD KEY:VALUE PAIRS INTO DICTIONARY  
 #METHOD -1  
 phbook[phnum]=name  
 #METHOD -2  
 """   
   
 contact=dict({phnum:name})  
 phbook.update(contact)  
  
 """  
  
  
print(phbook)  
  
try:  
 num = int(input("ENTER PH.NUMBER TO SEARCH IN PHONE BOOK "))  
 print("NAME OF THE PERSON OF THAT NUMGER : ", phbook[num])  
except Exception as e:  
 print("RECORDS NOT FOUND IN PHONE BOOK ",e)

SORTING LISTS

#SORTING IN ASCENDING ORDER USING BUBBLE SORT  
  
l=[4,1,3,2,5]  
print("INITIALLY LIST : ",l)  
for i in range(len(l)):  
 for j in range(len(l)-i-1):  
 if l[j]>l[j+1]:  
 temp=l[j]  
 l[j]=l[j+1]  
 l[j+1]=temp  
  
  
print("ASCENDING LIST : ",l)  
  
  
#SORTING IN ASCENDING ORDER USING SORT() BUILTIN FUNCTION  
l=[4,1,3,2,5]  
print(" INITIALLY LIST : ",l)  
l.sort()  
print("ASCENDING LIST : ",l)  
  
#SORTING IN ASCENDING ORDER USING SORTED(list) BUILTIN FUNCTION  
l=[4,1,3,2,5]  
print(" INITIALLY LIST : ",l)  
print("ASCENDING LIST : ",sorted(l))  
  
#SORTING IN DESCENDING ORDER USING BUBBLE SORT  
  
l=[4,1,3,2,5]  
print(" INITIALLY LIST : ",l)  
for i in range(len(l)):  
 for j in range(len(l)-i-1):  
 if l[j]<l[j+1]:  
 temp=l[j]  
 l[j]=l[j+1]  
 l[j+1]=temp  
  
  
print("DESCENDING LIST : ",l)  
  
  
#SORTING IN DESCENDING ORDER USING SORT() BUILTIN FUNCTION  
l=[4,1,3,2,5]  
print(" INITIALLY LIST : ",l)  
l.sort(reverse=True)  
print("DESCENDING LIST : ",l)  
  
#SORTING IN DESCENDING ORDER USING SORTED(list) BUILTIN FUNCTION  
l=[4,1,3,2,5]  
print(" INITIALLY LIST : ",l)  
print("ASCENDING LIST : ",sorted(l,reverse=True))

PATTERN

PRINT PATTERN OF PASCAL – TRAINGLE

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

n=int(input("ENTER ROWS : "))  
  
for i in range(0,n):  
 for j in range(0,n-i-1):  
 print("",end=" ")  
 for j in range(0,i+1):  
 print("\*",end=" ")  
 print()

PALINDROME CHECK

n=int(input("ENTER INTEGER : "))  
temp=n  
rev=0  
while n>0:  
 d=n%10  
 rev=rev\*10+d  
 n=n//10  
  
n=temp  
if rev==n:  
 print(n," IS PALINDROME NUMBER ")  
else:  
 print(n, " IS NOT A PALINDROME NUMBER ")

PALINDROME CHECK USING QUEUE AND STACKS CONCEPT

( from collections import deque )

append(), pop() , popleft()

import sys  
from collections import deque  
  
class Solution:  
 def \_\_init\_\_(self):  
 self.q = deque()  
  
 def pushCharacter(self, element):  
 self.q.append(element)  
  
 def enqueueCharacter(self, element):  
 self.q.append(element)  
  
 def popCharacter(self):  
 return self.q.pop()  
  
 def dequeueCharacter(self):  
 return self.q.popleft()  
  
  
# read the string s  
s = input("ENTER A STRING : ( TO CHECK IT'S PALINDROME OR NOT ) : ")  
# Create the Solution class object  
obj = Solution()  
  
l = len(s)  
# push/enqueue all the characters of string s to stack  
for i in range(l):  
 obj.pushCharacter(s[i])  
 obj.enqueueCharacter(s[i])  
  
isPalindrome = True  
'''  
pop the top character from stack  
dequeue the first character from queue  
compare both the characters  
'''  
for i in range(l // 2):  
 if obj.popCharacter() != obj.dequeueCharacter():  
 isPalindrome = False  
 break  
# finally print whether string s is palindrome or not.  
if isPalindrome:  
 print("The word, " + s + ", is a palindrome.")  
else:  
 print("The word, " + s + ", is not a palindrome.")

POWER OF X as N

x=int(input("ENTER BASE : "))  
n=int(input("ENTER POWER : "))  
print(x\*\*n)  
  
import math  
print(math.pow(x,n))  
  
  
result=1  
for i in range(n):  
 result=result\*x  
print(result)

PALINDROME CHECK , ( BY CONSIDERING INPUTED NUMBER AS STRING VIA SLICING)

num=input("ENTER ANY NUMBER : ")  
if num==num[::-1]:  
 print("PALINDROME")  
else:  
 print("NOT A PALINDROME")

PATTERN

ENTER THE TERMINATING CHARACTER: E

A

A B

A B C

A B C D

A B C D E

ch=input("ENTER THE TERMINATING CHARACTER: ")[0]

a=ord(ch)  
  
for x in range(65,a+1):  
 for y in range(65,x+1):  
 print(chr(y),end=" ")  
 print(" ")

FIBANOCCI SERIES USING FOR LOOP

print("TYPE-1")  
number=int(input("ENTER THE RANGE OF SERIES : "))  
first =0  
second=1  
  
for num in range(number):  
 if num<=1:  
 next=num  
 else:  
 next=first+second  
 first=second  
 second=next  
  
 print(next,end=" ")  
  
print(" ")  
  
print("TYPE-2")  
first =1  
second=1  
  
for num in range(number):  
 if num<=1:  
 next=num  
 else:  
 next=first+second  
 first=second  
 second=next  
  
 print(next,end=" ")

PATTERN

ENTER NUMBER OF ROWS : 5

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

n=int(input("ENTER NUMBER OF ROWS : "))  
  
for i in range(1,n+1):  
 for j in range(1,i+1):  
 print(j,end=" ")  
 print(" ")

### [any()](https://docs.python.org/2/library/functions.html#any)

This expression returns True if **any** element of the iterable is true.  
If the iterable is empty, it will return False.

**Code**

>>> any([1>0,1==0,1<0])

True

>>> any([1<0,2<1,3<2])

False

[**all()**](https://docs.python.org/2/library/functions.html#all)

This expression returns True if **all** of the elements of the iterable are true. If the iterable is empty, it will return True.

**Code**

>>> all(['a'<'b','b'<'c'])

True

>>> all(['a'<'b','c'<'b'])

False

NUMPY .

import numpy as np

print(np.\_\_version\_\_)

a=np.array([1,2,3,4,5])

print(a)

print(a[0])

a=np.array([[1,2,3,4,5],[1,2,3,4,5],[1,2,3,4,5]])

print(a)

print(a[0][0])

a=np.array([[[1,2,3,4,5],[1,2,3,4,5],[1,2,3,4,5]],[[1,2,3,4,5],[1,2,3,4,5],[1,2,3,4,5]]])

print(a)

print(a[0][0][0])

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

print(a)

b=np.asarray(a,dtype=float)

print(b)

b=np.asarray(a,dtype=int)

print(b)

b=np.asarray(a,dtype=str)

print(b)

a=[[1,2,3,4,5],[1,2,3,4,5],[1,2,3,4,5]]

b=np.asarray(a,dtype=str)

print(b)

a=[[1,2,3,4,5],[4,5,6,7,8],[9,10,11,12,13]]

b=np.asarray(a,dtype=int,order="C")

print(b)

for i in np.nditer(b):

print(i)

a=[[1,2,3,4,5],[4,5,6,7,8],[9,10,11,12,13]]

b=np.asarray(a,dtype=int,order="F")

print(b)

for i in np.nditer(b):

print(i)

#INTIALIZING array

a=np.zeros(3)

print(a)

print()

a=np.zeros([5,5])

print(a)

print()

a=np.zeros([2,3,3])

print(a)

print()

print(np.full([5,5],10))

print()

print(np.full([5,5],4))

print(np.random.rand(5,5))

print()

a=np.ones([3,3])

print(a)

print()

a=np.eye(4)

print(a)

print()

a=np.eye(3)

print(a)

print()

a=np.arange(1,11,1)

print(a)

print()

a=np.arange(10,110,10)

print(a)

print()

a=np.arange(10,110,10,dtype=float)

print(a)

print()

a=np.arange(10,70,10)

print(a)

a=a.reshape(2,3)

print(a)

print()

a=a.reshape(3,2)

print(a)

a=a.reshape(6,1)

print(a)

print()

a=a.reshape(1,6)

print(a)

print()

a=np.linspace(10,100,10)

print(a)

a=np.linspace(10,100,10,endpoint=False)

print(a)

a=np.linspace(10,100,10,endpoint=False,retstep=True)

print(a)

print()

a=np.linspace(10,100,10,endpoint=False,retstep=True,dtype=int)

print(a)

print()

a=np.logspace(10,100,10,endpoint=False,base=2)

print(a)

print()

# PROPERTIES

a=np.arange(1,10,1).reshape(3,3)

print()

print(np.size(a))

print(np.shape(a))

print(a.dtype)

#ARRAY OPERATIONS

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

print(a)

print(a.dtype)

print(np.size(a))

print(np.shape(a))

print(len(a))

#INDEXING

print(" ")

for i in range(len(a)):

for j in range(len(a[i])):

print(a[i][j],end=" ")

print("")

print(" ")

print(" ")

for i in range(len(a)):

print(a[i])

print(" ")

#SLICING

a=np.arange(10,110,10)

print(a)

print(a[2:])

print(a[:])

print(a[:5])

print(a[3:7])

#COPY

b=np.copy(a)

print(a)

print(b)

#VIEW

c=b.view()

print(c)

b[2]=0

print(a)

print(b)

print(c)

#sort

a=np.array([5,1,3,2,4])

print(np.sort(a))

a=np.array([[20,10,30],[5,4,6]])

print(np.sort(a))

a=np.array([[20,10,30],[5,4,6]])

print(np.sort(a,axis=1))

a=np.array([[20,10,30],[5,4,6]])

print(np.sort(a,axis=0))

d=np.dtype([("name","S1"),("percent","<f8")])

marks=np.array([("MAHESH",90),("SURYA",92),("SANKU",80)])

print(marks)

print(np.sort(marks))

a=np.array([1,2,3])

b=np.array([100,200,300])

print(a)

print(b)

print(np.append(a,b))

print(a)

print(b)

print(np.insert(a,2,1000))

print(np.insert(a,1,[5,55,555]))

b=np.insert(a,1,[5,55,555])

print(b)

print(np.delete(b,1))

c=np.concatenate((a,b))

print(c)

print(np.delete(c,1))

c=np.delete(c,1)

print(c)

print(c.reshape([2,4]))

a=np.array([1,2,3,4])

b=np.array([5,6,7,8])

print(np.concatenate((a,b)))

res=np.stack(a)

print(res[0])

res=np.stack((a,b))

print(res)

print(res[0])

print(np.vstack((a,b)))

print(np.hstack((a,b)))

print(np.dstack((a,b)))

x=np.hstack((a,b))

print(np.split(x,4))

y=np.arange(10,130,10)

print(y)

print(np.split(y,(2,6)))

print(y.reshape(4,3))

b=np.arange(10,130,10)

print(b)

print(np.split(b,(2,6)))

print(b.reshape(4,3))

print(np.where(b==80))

print(np.where(b==50))

print(np.where(b%20==0))

print(np.where(b%30==0))

a=np.array([1,2,3,4,5,6,7,8,9],dtype=int)

print(a)

a=np.array([1,2,3,4,5,6,7,8,9],dtype=float)

print(a)

# ARITHMETIC OPERATIONS

a=np.array([1,2,3,4,5,6,7,8,9],dtype=int)

print(a)

b=np.array([1,2,3,4,5,6,7,8,9],dtype=int)

print(b)

print(np.add(a,b))

print(np.subtract(a,b))

print(np.multiply(a,b))

print(np.divide(a,b))

print(np.exp(a))

a=np.array([1,4,9,16,25])

print(a)

print(np.sqrt(a))

print(np.array\_equal(a,b))

a=np.array([1,2,3,4,5,6,7,8,9],dtype=int)

print(a)

b=np.array([1,2,3,4,5,6,7,8,9],dtype=int)

print(b)

print(np.array\_equal(a,b))

print(np.sum(a))

print(np.min(a))

print(np.max(a))

print(np.mean(a))

print(np.median(a))

print(np.var(a))

print(np.std(a))

BUILT INS

EVAL

>>> eval("9 + 5")

14

>>> x = 2

>>> eval("x + 3")

5

>>> type(eval("len"))

<type 'builtin\_function\_or\_method'>

TKINTER - GUI

Empty frame

from tkinter import \*  
  
main=Tk()  
main.mainloop()

FRAME BASIC WINDOW

from tkinter import \*  
  
#obj is the instance of Tk class  
  
obj=Tk()  
  
#GUI LOGIC  
obj.mainloop()

geometry(WxH) , maxsize(W,H) , minsize(W,H)

Label(text=”HHHSHHSAJHD”)

GUI - LABEL

from tkinter import \*  
  
# surya is instance for Tk class  
surya=Tk()  
  
  
# geometry( WIDTH X HEIGHT )  
surya.geometry("500x500")  
  
# minsize ( width , height )  
surya.minsize(100,100)  
  
# maxsize ( width , height )  
surya.maxsize(700,700)  
  
# Label(text="PUSHPARAJ") - NO USER INTERACTION  
l1 = Label(text="WELCOME TO GUI")  
l1.pack()  
  
# GUI LOGIC  
surya.mainloop()

PNG IMAGES

from tkinter import \*  
  
myframe= Tk()  
myframe.title("IMAGE - FRAME")  
myframe.geometry("500x500")  
  
myframe.maxsize(1000,1000)  
myframe.minsize(100,100)  
  
mylabel1=Label(text=" WELCOME TO GUI ")  
mylabel1.pack()  
  
mypic=PhotoImage(file="MyLogo.png")  
mylabel2=Label(image=mypic)  
mylabel2.pack()  
  
myframe.mainloop()

JPG IMAGES

from tkinter import \*  
from PIL import Image,ImageTk  
  
myframe= Tk()  
myframe.title("IMAGE - FRAME")  
myframe.geometry("500x500")  
  
myframe.maxsize(1000,1000)  
myframe.minsize(100,100)  
  
mylabel1=Label(text=" WELCOME TO GUI ")  
mylabel1.pack()  
  
image1=Image.open("Mypic.jpg")  
photo1=ImageTk.PhotoImage(image1)  
  
mylabel2=Label(image=photo1)  
mylabel2.pack()  
  
myframe.mainloop()

LABEL , PACK ATTRIBUTES

# LABEL ATTRIBUTES , PACK ATTRIBUTES   
  
from tkinter import \*  
  
frame1 = Tk()  
frame1.title(" PUSHPA GUI ")  
frame1.geometry("750x750")  
frame1.minsize(100,100)  
frame1.maxsize(1000,1000)  
  
label1=Label(text=" WELCOME TO PUSHPA WORLD ",font=15)  
label1.pack()  
  
# LABEL OPTIONS  
"""  
text = adds text  
bg = background  
fg = foreground  
font   
1) font=("comicsansns",20,"bold")  
2) font="comicsansns 20 bold"  
3) font = 10  
 we can simply add what ever label options / label attributes we need   
  
, padx ( padding in x direction ),pady - ( Padding in y direction ) ((, relief  
  
 """  
  
label2=Label(text=" PUSHPA , PUSHPA RAJ YEVVA THAGGEDHELE ",bg="red",fg="yellow",padx=20,pady=20,font=("comicsansns",20,"bold"))  
label2.pack()  
  
  
label3=Label(text=" PUSHPA ANTE FLOWER ANUKINTIVA................",bg="pink",fg="green",font=15)  
label3.pack()  
  
label4=Label(text=" FIRE UU........ FIRE EHHHH ....................... ",bg="orange",fg="blue",padx=10,pady=10,font="comicsansns 15 bold")  
label4.pack()  
  
label5=Label(text=" PUSHPA ANTEY YODIKI BAYAPADDU....",bg="red",fg="yellow",font=15,borderwidth=30,relief=SUNKEN)  
label5.pack()  
  
label6=Label(text=" SHIKAWAT SIR NA KODAKA.. ENDHEY BULLETKI BAYAPADE BRAND AAA NADHIII....",bg="magenta",fg="blue",font=15,borderwidth=30)  
label6.pack()  
  
# pack attributes ,  
# anchor = "ne" , "se" , " nw "," sw "  
# side=top, bottom , left , right  
  
  
label1.pack(anchor="nw")  
label2.pack(anchor="sw")  
label3.pack(anchor="ne")  
label4.pack(anchor="se")  
"""  
label5.pack(anchor="nw",side=BOTTOM)  
label6.pack(anchor="sw",side=TOP)  
label3.pack(anchor="ne",side=LEFT)  
label4.pack(anchor="se",side=RIGHT)  
  
"""  
  
frame1.mainloop()

FRAMES IN GUI

from tkinter import \*  
  
obj=Tk()  
obj.title("FRAME")  
obj.geometry("750x750")  
#obj.maxsize(1000,1000)  
obj.minsize(500,500)  
  
f1=Frame(obj,bg="grey",borderwidth=6)  
f1.pack(side=TOP,fill="x",padx=50,pady=30)  
l1=Label(f1,text="WELCOME TO GUI",fg="red",bg="black",font="comicsansns 20 bold")  
l1.pack()  
  
f2=Frame(obj,bg="grey",borderwidth=6)  
f2.pack(side=BOTTOM,fill="x",pady=50,padx=30)  
l2=Label(f2,text="THANK YOU !",fg="red",bg="black",font="Helvetica 20 bold")  
l2.pack()  
  
l3=Label(text="HELLO ! SURYA MAHESH.. NICE TO SEE YOU.. HOPE WE WILL MEET AGAIN ",fg="BLUE",font="comicsansns 15 bold")  
l3.pack()  
  
  
# frames are inside obj's labels inside the frame or labels should be inside obj  
obj.mainloop()

BUTTONS IN GUI

from tkinter import \*  
  
obj=Tk()  
obj.title("BUTTON WINDOW")  
obj.geometry("750x750")  
#obj.maxsize(1000,1000)  
obj.minsize(100,100)  
  
l=Label(text="WELCOME TO GUI",font="Helvetica 15 bold")  
l.pack()  
  
f1=Frame(obj,borderwidth=5)  
f1.pack(side=LEFT, anchor="nw")  
  
def hello():  
 print("HELLO MURUGAN SYNDICATE MEMBER PUSHPA PESIRE..")  
 l1=Label(text="HELLO MURUGAN SYNDICATE MEMBER PUSHPA PESIRE..",fg="blue",font=20)  
 l1.pack()  
  
def diologue():  
 print("PUSHPA NAAM SUNKE FLOWER SAMJE KYA ? FIRE EH MEIII...")  
 l3 = Label(text="PUSHPA NAAM SUNKE FLOWER SAMJE KYA ? FIRE EH MEIII...", fg="blue", font=20)  
 l3.pack()  
  
  
def byebye():  
 print("FIRE UUU FIRE EHHH ")  
 l2 = Label(text="FIRE UUU FIRE EHHH ", fg="blue", font=20)  
 l2.pack()  
  
  
b1=Button(f1,bg="red",text="CLICK HERE",command=hello)  
b1.pack(side=LEFT)  
  
b2=Button(f1,bg="red",text="SUBMIT NOW",command=byebye)  
b2.pack(side=LEFT)  
  
b3=Button(bg="red",text="PUSHPA",command=diologue)  
b3.pack()  
  
"""  
  
b3=Button(f1,bg="red",text="CLICK HERE")  
b3.pack()  
  
b4=Button(f1,bg="red",text="SUBMIT NOW")  
b4.pack()  
  
"""  
  
  
obj.mainloop()

BUTTONS AND COMMAND

from tkinter import \*  
  
obj=Tk()  
obj.title("BUTTON WINDOW")  
obj.geometry("750x750")  
#obj.maxsize(1000,1000)  
obj.minsize(100,100)  
  
l=Label(text="WELCOME TO GUI",font="Helvetica 15 bold")  
l.pack()  
  
f1=Frame(obj,borderwidth=5)  
f1.pack(side=LEFT, anchor="nw")  
  
def hello():  
 print("HELLO MURUGAN SYNDICATE MEMBER PUSHPA PESIRE..")  
 l1=Label(text="HELLO MURUGAN SYNDICATE MEMBER PUSHPA PESIRE..",fg="magenta",font=20)  
 l1.pack()  
  
def diologue():  
 print("PUSHPA NAAM SUNKE FLOWER SAMJE KYA ? FIRE EH MEIII...")  
 l3 = Label(text="PUSHPA NAAM SUNKE FLOWER SAMJE KYA ? FIRE EH MEIII...", fg="blue", font=20)  
 l3.pack()  
  
  
def byebye():  
 print("FIRE UUU FIRE EHHH ")  
 l2 = Label(text="FIRE UUU FIRE EHHH ", fg="orange", font=20)  
 l2.pack()  
  
  
b1=Button(f1,bg="red",text="CLICK HERE",command=hello)  
b1.pack(side=BOTTOM)  
  
b2=Button(f1,bg="red",text="SUBMIT NOW",command=byebye)  
b2.pack(side=BOTTOM)  
  
  
b3=Button(f1,bg="red",text="PUSHPA",command=diologue)  
b3.pack(side=BOTTOM)  
  
  
"""  
  
b3=Button(f1,bg="red",text="CLICK HERE")  
b3.pack()  
  
b4=Button(f1,bg="red",text="SUBMIT NOW")  
b4.pack()  
  
"""  
  
  
obj.mainloop()

GRID ,TEXTFIELDS, LOGIN FORM

from tkinter import \*  
  
obj=Tk()  
obj.title("BUTTON WINDOW")  
obj.geometry("750x750")  
#obj.maxsize(1000,1000)  
obj.minsize(100,100)  
  
  
"""  
l=Label(obj,text="WELCOME TO GUI",font="Helvetica 15 bold")  
l.pack()  
"""  
  
  
l1=Label(obj,text="USERNAME",font="Helvetica 15 bold")  
#l1.pack()  
l1.grid()  
  
l2=Label(obj,text="PASSWORD",font="Helvetica 15 bold")  
#l2.pack()  
l2.grid(row=1)  
  
# variable classes in tkinter  
# BooleanVar , DoubleVar , IntVar , StringVar  
  
  
t1= StringVar()  
t2=StringVar()  
  
userentry = Entry(obj,textvariable=t1)  
passentry = Entry(obj,textvariable=t2)  
  
userentry.grid(row=0,column=1)  
passentry.grid(row=1,column=1)  
  
  
def done():  
 print("USERNAME : ",t1.get()," PASSWORD : ",t2.get())  
 l1=Label(text=t1.get(),bg="black",fg="red",font=20)  
 l1.grid()  
 l2 = Label(text=t2.get(),bg="black",fg="red",font=20)  
 l2.grid()  
 obj2 = Tk()  
 obj2.title("BUTTON WINDOW")  
 obj2.geometry("750x750")  
 # obj.maxsize(1000,1000)  
 obj2.minsize(100, 100)  
 l1 = Label(obj2,text="HELLO "+str(t1.get()), bg="black", fg="red", font=20)  
 l1.pack()  
 l2 = Label(obj2,text="WELCOME TO PUSHPA WORLD ", bg="black", fg="red", font=20)  
 l2.pack()  
 obj2.mainloop()  
  
  
b1=Button(text="SUBMIT",command=done,bg="red")  
b1.grid()  
obj.mainloop()

SUBMIT FORM , INTERACTIVE BUTTONS

from tkinter import \*  
  
obj=Tk()  
obj.title("BUTTON WINDOW")  
obj.geometry("750x750")  
#obj.maxsize(1000,1000)  
obj.minsize(100,100)  
  
l=Label(obj,text=" WELOCOME TO GUI ",fg="magenta",bg="black",font="Heveltica 20 bold")  
l.grid(row=0,column=3)  
  
l1=Label(obj,text=" NAME ",font="Heveltica 15 bold")  
l1.grid(row=1,column=2)  
t1=StringVar()  
t1entry=Entry(obj,textvariable=t1)  
t1entry.grid(row=1,column=3)  
  
l2=Label(obj,text=" PHONE ",font="Heveltica 15 bold")  
l2.grid(row=2,column=2)  
t2=StringVar()  
t2entry=Entry(obj,textvariable=t2)  
t2entry.grid(row=2,column=3)  
  
l3=Label(obj,text=" GENDER ",font="Heveltica 15 bold")  
l3.grid(row=3,column=2)  
t3=StringVar()  
t3entry=Entry(obj,textvariable=t3)  
t3entry.grid(row=3,column=3)  
  
l4=Label(obj,text=" EMERGENCY CONTACT ",font="Heveltica 15 bold")  
l4.grid(row=4,column=2)  
t4=StringVar()  
t4entry=Entry(obj,textvariable=t4)  
t4entry.grid(row=4,column=3)  
  
l5=Label(obj,text=" PAYMENT ",font="Heveltica 15 bold")  
l5.grid(row=5,column=2)  
t5=IntVar()  
t5entry=Entry(obj,textvariable=t5)  
t5entry.grid(row=5,column=3)  
  
def submitclicked():  
 s=""  
 s+="\n NAME : "+t1.get()+"\n PHONE : "+t2.get()+"\n GENDER : "+t3.get()+"\n EMERGENCY CONTACT: "+t4.get()+"\n PAYMENT : "+str(t5.get())  
 print(s)  
 obj2 = Tk()  
 obj2.title("BUTTON WINDOW")  
 obj2.geometry("750x750")  
 # obj.maxsize(1000,1000)  
 obj2.minsize(100, 100)  
  
 l=Label(obj2,text=s,font="Helvetica 15 bold",bg="yellow",fg="red")  
 l.grid(row=3,column=3)  
  
  
 obj2.mainloop()  
  
  
b1=Button(obj,text=" SUBMIT ",font="Heveltica 17 bold",bg="grey",command=submitclicked)  
b1.grid(row=7,column=3)  
  
  
obj.mainloop()

Checkbutton

from tkinter import \*  
  
obj=Tk()  
obj.title("BUTTON WINDOW")  
obj.geometry("750x750")  
#obj.maxsize(1000,1000)  
obj.minsize(100,100)  
  
l=Label(obj,text=" WELOCOME TO GUI ",fg="magenta",bg="black",font="Heveltica 20 bold")  
l.grid(row=0,column=3)  
  
l1=Label(obj,text=" NAME ",font="Heveltica 15 bold")  
l1.grid(row=1,column=2)  
t1=StringVar()  
t1entry=Entry(obj,textvariable=t1)  
t1entry.grid(row=1,column=3)  
  
l2=Label(obj,text=" PHONE ",font="Heveltica 15 bold")  
l2.grid(row=2,column=2)  
t2=StringVar()  
t2entry=Entry(obj,textvariable=t2)  
t2entry.grid(row=2,column=3)  
  
l3=Label(obj,text=" GENDER ",font="Heveltica 15 bold")  
l3.grid(row=3,column=2)  
t3=StringVar()  
t3entry=Entry(obj,textvariable=t3)  
t3entry.grid(row=3,column=3)  
  
l4=Label(obj,text=" EMERGENCY CONTACT ",font="Heveltica 15 bold")  
l4.grid(row=4,column=2)  
t4=StringVar()  
t4entry=Entry(obj,textvariable=t4)  
t4entry.grid(row=4,column=3)  
  
l5=Label(obj,text=" PAYMENT ",font="Heveltica 15 bold")  
l5.grid(row=5,column=2)  
t5=IntVar()  
t5entry=Entry(obj,textvariable=t5)  
t5entry.grid(row=5,column=3)  
  
s=""  
  
def submitclicked():  
 s=""  
 s+="\n NAME : "+t1.get()+"\n PHONE : "+t2.get()+"\n GENDER : "+t3.get()+"\n EMERGENCY CONTACT: "+t4.get()+"\n PAYMENT : "+str(t5.get())  
 print(s)  
 obj2 = Tk()  
 obj2.title("BUTTON WINDOW")  
 obj2.geometry("750x750")  
 # obj.maxsize(1000,1000)  
 obj2.minsize(100, 100)  
  
 l=Label(obj2,text=s,font="Helvetica 15 bold",bg="yellow",fg="red")  
 l.grid(row=3,column=3)  
  
  
 obj2.mainloop()  
  
  
b1=Button(obj,text=" SUBMIT ",font="Heveltica 13 bold",bg="grey",command=submitclicked)  
b1.grid(row=9,column=3)  
  
def order():  
 print("FULL MEALS ORDERED ")   
  
  
cb1=Checkbutton(obj,text="MEALS ORDER ",font="Heveltica 17 bold",bg="grey",command=order)  
cb1.grid(row=7,column=2)  
  
obj.mainloop()