In [1]:

*#1a) Write a python program to read 2 numbers from the keyboard and perform the basic arithmetic operations based on the choice.*  
*#(1-Add, 2-Subtract, 3-Multiply, 4-Divide)*  
  
num1 **=** int(input("Enter First Number: "))  
num2 **=** int(input("Enter Second Number: "))  
  
print("Enter which operation would you like to perform?")  
ch **=** int(input("Enter any of these char for specific operation 1-Add, 2-Subtract, 3-Multiply, 4-Divide : "))  
  
result **=** 0  
**if** ch **==**1:  
 result **=** num1 **+** num2  
**elif** ch **==**2:  
 result **=** num1 **-** num2  
**elif** ch **==**3:  
 result **=** num1 **\*** num2  
**elif** ch **==**4:  
 result **=** num1 **/** num2  
**else**:  
 print("Input character is not recognized!")  
  
print("The result is : ", result)

Enter First Number: 10  
Enter Second Number: 30  
Enter which operation would you like to perform?  
Enter any of these char for specific operation 1-Add, 2-Subtract, 3-Multiply, 4-Divide : 3  
The result is : 300

In [2]:

*# 1b) Write a python program to find the factorial of number using while loop.*  
  
**def** factorial(n):  
 num **=** 1  
 **while** n **>=** 1:  
 num **=** num **\*** n  
 n **=** n **-** 1  
 **return** num  
  
n**=**int(input("Enter the number for finding it's factorial "))  
f **=** factorial(n)  
print(f)

Enter the number for finding it's factorial 7  
5040

In [3]:

*# 1c) Write a python program to add 10 numbers by inputting each from the keyboard using for loop*  
  
Sum **=** 0  
print("Please Enter 10 Numbers \n")  
**for** i **in** range(1, 11):  
 *#explain the syntax*  
 num **=** int(input("Number %d = " **%i**))  
 Sum **=** Sum **+** num  
  
print("The Sum of 10 Numbers = ", Sum)

Please Enter 10 Numbers   
  
Number 1 = 10  
Number 2 = 20  
Number 3 = 30  
Number 4 = 40  
Number 5 = 50  
Number 6 = 60  
Number 7 = 70  
Number 8 = 80  
Number 9 = 90  
Number 10 = 100  
The Sum of 10 Numbers = 550

In [6]:

*#2a) Write a python function linearSearch() to read an array and search for the key element.*   
*# Display the appropriate messages. Use the recursive function*  
  
*# function linearSearch definition*  
*# parameters for linearSearch: array list, the index value and the item to be searched*  
*# the order of sending these parameters need not be the same*  
**def** linearSearch(arr,index,searchItem):  
*# 1. CHECK if the index value is greater than the length of array*  
*# its just a good practice.*  
 **if** index**>=**len(arr):  
 **return** **-**1  
*# here we will check for the value at the index position*  
*#IS EQUAL to the search item value we have given*  
 **if** arr[index]**==**searchItem:  
 **return** index  
*#in the return the index+1 means incrementation of index*  
*#or in simple terms go to the next index value*  
 **return** linearSearch(arr,index**+**1,searchItem)  
  
*#input to search the item*  
searchItem**=**int(input("Please provide input search item:"))  
  
*#input array*  
arr**=**[30,40,20,50,10]  
  
*#calling the LinearSearchRecursive function and passing 3 parameters*  
*#Storing the same in result variable*  
*#explain why 0*  
result**=**linearSearch(arr,0,searchItem)  
  
*#if condition to print the value*  
*#if the value is not present in the list it will return not found*  
**if** result**<**0:  
 print(searchItem," is not found")  
**else**:  
*#if the value is present in the list it will give the value + its index value*  
 print(searchItem," is found at index:",result)

Please provide input search item:50  
50 is found at index: 3

In [7]:

*# 2b) Write a python program to define a function max\_of\_three() that takes three numbers as arguments*   
*# and returns the largest of them using default arguments.*  
  
**def** max\_of\_three(a, b, c):  
 **if** (a **>=** b) **and** (a **>=** c):  
 largest **=** a  
  
 **elif** (b **>=** a) **and** (b **>=** c):  
 largest **=** b  
 **else**:  
 largest **=** c  
  
 **return** largest  
  
  
res **=** max\_of\_three(2, 4, 7)  
print("Largest Number: ", res)

Largest Number: 7

In [11]:

*#2c) Write a python program to define a function generate\_n\_chars() that takes an integer n and a character c*   
*#and returns a string, n characters long.*  
*# For example, generate\_n\_chars(5,"x") should return the string "xxxxx“ using keyword only parameters*  
  
*#define the fuction generate\_n\_chars()*   
*# first parameter is an integer*   
*# second parameter is a string*  
**def** generate\_n\_chars(n, str):  
 *#declare an empty variable result*  
 result **=** ""  
 **for** x **in** range(n):  
 result **+=** str  
 **return** result  
  
*# as per question using the same input parameters.*  
print("The string input x will be printed 5 time",generate\_n\_chars(5, "x"))  
  
*# you can also test the same with*  
*#print(generate\_n\_chars(10, "Hello"))*  
*#print(generate\_n\_chars(10, "RIT"))*

The string input x will be printed 5 time xxxxx

In [20]:

*# 3a) Write a python program to implement a stack and queue using lists*  
  
*#Stack using list*  
stack **=** []   
  
*#Function to print top element of stack*  
**def** top(stack):  
 **if** stack **!=** []:  
 print(stack[**-**1] **+** " is top element")  
 **else**:  
 print("Stack Empty!!!")  
  
*#Function to print size of stack*  
**def** size(stack):  
 print("Size of stack is " **+** str(len(stack)))  
  
*#Function to check if a stack is empty*  
**def** empty(stack):  
 **if** stack **==** []:  
 print("True")  
 **else**:  
 print("False")  
  
*# append() function is used to push element in the stack*   
stack**.**append('a')   
stack**.**append('b')   
stack**.**append('c')  
  
size(stack)  
  
print(stack)   
  
top(stack)  
  
*# pop() function to pop element from stack in LIFO order*   
print(stack**.**pop() **+** " is popped")  
  
print(stack)   
  
empty(stack)  
  
print(stack**.**pop() **+** " is popped")  
print(stack**.**pop() **+** " is popped")  
  
print(stack)   
  
empty(stack)

Size of stack is 3  
['a', 'b', 'c']  
c is top element  
c is popped  
['a', 'b']  
False  
b is popped  
a is popped  
[]  
True

In [21]:

*# 3a) Write a python program to implement a stack and queue using lists*  
**class** Node:  
 **def** \_\_init\_\_(self, data):  
 self**.**data **=** data  
 self**.**next **=** **None**  
   
**class** Queue:  
 **def** \_\_init\_\_(self):  
 self**.**head **=** **None**  
 self**.**last **=** **None**  
   
 **def** enqueue(self, data):  
 **if** self**.**last **is** **None**:  
 self**.**head **=** Node(data)  
 self**.**last **=** self**.**head  
 **else**:  
 self**.**last**.**next **=** Node(data)  
 self**.**last **=** self**.**last**.**next  
   
 **def** dequeue(self):  
 **if** self**.**head **is** **None**:  
 **return** **None**  
 **else**:  
 to\_return **=** self**.**head**.**data  
 self**.**head **=** self**.**head**.**next  
 **return** to\_return  
   
a\_queue **=** Queue()  
**while** **True**:  
 print('enqueue <value>')  
 print('dequeue')  
 print('quit')  
 do **=** input('What would you like to do? ')**.**split()  
   
 operation **=** do[0]**.**strip()**.**lower()  
 **if** operation **==** 'enqueue':  
 a\_queue**.**enqueue(int(do[1]))  
 **elif** operation **==** 'dequeue':  
 dequeued **=** a\_queue**.**dequeue()  
 **if** dequeued **is** **None**:  
 print('Queue is empty.')  
 **else**:  
 print('Dequeued element: ', int(dequeued))  
 **elif** operation **==** 'quit':  
 **break**

enqueue <value>  
dequeue  
quit  
What would you like to do? enqueue 10  
enqueue <value>  
dequeue  
quit  
What would you like to do? enqueue 20  
enqueue <value>  
dequeue  
quit  
What would you like to do? enqueue 30  
enqueue <value>  
dequeue  
quit  
What would you like to do? dequeue  
Dequeued element: 10  
enqueue <value>  
dequeue  
quit  
What would you like to do? quit

In [12]:

*#3b) Write a python program to create a list of tuples having first element as the strings*   
*#and the second element as the length of the string.*   
*#Output the list of tuples sorted based on the length of the string*  
  
*# Creating and Print tuple list : first element string and second element is lenght of the string*  
tupList **=**[("KTM",3), ("Enfield", 7), ("Ducati", 6), ("Honda", 5)]  
*# why there is a type convesion*  
print("List of Tuple before sorting : " **+** str(tupList))  
  
*# Sorting List of Tuples in ascending order*   
tupList**.**sort(key **=** **lambda** x: x[**-**1])  
  
*#Printing Sorted List*   
print("List of Tuple after sorting : " **+** str(tupList))

List of Tuple before sorting : [('KTM', 3), ('Enfield', 7), ('Ducati', 6), ('Honda', 5)]  
List of Tuple after sorting : [('KTM', 3), ('Honda', 5), ('Ducati', 6), ('Enfield', 7)]

In [13]:

*#3c) Write a python program to create a list and perform the following operations :*  
  
*#CREATION OF LIST*  
myList **=** [6.3,2.2,12,32]  
  
*# a) Inserting an element*  
myList**.**insert(3, 4)  
print("After inserting at index 3, the value 4 ",myList)  
  
*#• Removing an element*  
myList**.**remove(4)  
print("After removing the value 4 from the list ",myList)  
  
*#• Appending an element*  
myList**.**append(4)  
myList**.**append(5)  
myList**.**append(6)  
  
print("After appending the values 4,5,6 ",(myList))  
  
*#• Displaying the length of the list*  
print("The length of the list is ",len(myList))  
  
*#• Popping an element*   
myList**.**pop()  
print("After poping the last element 6 ",myList)  
  
*#• Clearing the list*  
myList**.**clear()  
print("After clearing the list",myList) *#just for confirmation that there are no elements*

After inserting at index 3, the value 4 [6.3, 2.2, 12, 4, 32]  
After removing the value 4 from the list [6.3, 2.2, 12, 32]  
After appending the values 4,5,6 [6.3, 2.2, 12, 32, 4, 5, 6]  
The length of the list is 7  
After poping the last element 6 [6.3, 2.2, 12, 32, 4, 5]  
After clearing the list []

In [14]:

*#4a) Write a tiny Python program numDict.py that makes a dictionary whose keys are the words ‘one’, ‘two’, ‘three’,*  
*#and ‘four’, and whose corresponding values are the numerical equivalents, 1, 2, 3, and 4 (ints, not strings).*  
  
*# creating the dictionary*  
numDict **=** {"one":1, "two":2,"three":3,"four":4}  
  
*# printing the dictionary*  
print("Dictionary \'numDict\' is...")  
print(numDict)  
  
*# printing the keys & values*  
*#print("Dictionary \'numDict\' keys & values...")*  
*#for x, y in numDict.items():*  
*# print(x, ':', y)*

Dictionary 'numDict' is...  
{'one': 1, 'two': 2, 'three': 3, 'four': 4}

In [15]:

*#4b) Write a Python program to store PROFILE\_DATA(user\_id, name, DOB, qualification, work experience) in a dictionary*   
*#and pretty print the dictionary contents. (import pprint)*  
  
**import** pprint  
  
  
PROFILE\_DATA **=** {"user\_id":1, "name":"abc","DOB": "11/01/2023","qualification" :"BE","work experience" : 3}  
  
print("Printing the normal way \n",PROFILE\_DATA)  
  
pp **=** pprint**.**PrettyPrinter()  
print("\n After using pretty printer")  
pp**.**pprint(PROFILE\_DATA)

Printing the normal way   
 {'user\_id': 1, 'name': 'abc', 'DOB': '11/01/2023', 'qualification': 'BE', 'work experience': 3}  
  
 After using pretty printer  
{'DOB': '11/01/2023',  
 'name': 'abc',  
 'qualification': 'BE',  
 'user\_id': 1,  
 'work experience': 3}

In [ ]: