1.Write a Python program to find a target values in a list using linear search with following steps:

a. Initialize the list to store the input elements.

b. Initialize found-False.

C. Enter the item to be searched (match_item).

d. For each element in the list

    1. if match_item = value

        a. return match_item's position.

e. If the match_item is not in the list, display an error message that the item is not found in the list.

In [11]:
```
lst=[]
found=False
pos=0
n=int(input("Enter the number of elements in the list"))
for i in range(0,n):
    x=int(input("Enter the # element in the list %d:"%(i+1)))
    lst.append(x)
match_item=int(input("Enter the number that your looking for :"))
while pos<n and not found:
    if lst[pos]==match_item:
        found=True
    else:
        pos+=1
if found:
    print("The element was found in the position %d"%(pos+1))

else:

    raise Exception("The element is not in the provided list")
```

```
Enter the number of elements in the list4
Enter the # element in the list 1:5454
Enter the # element in the list 2:5456
Enter the # element in the list 3:7878
Enter the # element in the list 4:9898
Enter the number that your looking for :5555

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[11], line 19
     15     print("The element was found in the position %d"%(pos+1))
     17 else:
---> 19     raise Exception ("The element is not in the provided list")

Exception: The element is not in the provided list
```

2.Write a Python program to implement binary search to find the target values from the list:

a. Create a separate function to do binary search.

b. Get the number of inputs from the user.

In [20]:
```
def binary_search(x):
    first=0
    last=n-1
    found=False
    lst=[]

    for i in range(0,n):
        m=int(input("Enter the Element %d : "%(i+1)))
        lst.append(m)


    while (first<last) and not found:
        mid=(first+last)//2
        if lst[mid]==x:
            found=True
            print("The element was found at position %d : "%(mid+1))
        elif(lst[mid]>x):
            first=mid+1
        else:
            last=mid-1

n=int(input("Enter the number of elements present in the list: "))
binary_search(20)
```

```
Enter the number of elements present in the list: 3
Enter the Element 1 : 10
Enter the Element 2 : 20
Enter the Element 3 : 30
The element was found at position 2 :
```

6.Write a Python script to perform the following operations on a singly linked list

1. Create a list
2.find the smallest element from the list
3.Insert an element if it is not a duplicate
4.Display the elements in reverse order

In [21]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    # Function to add a new node to the end of the list
    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    # Function to find the smallest element in the list
    def find_smallest(self):
        if not self.head:
            return None
        current = self.head
        smallest = current.data
        while current.next:
            current = current.next
            if current.data < smallest:
                smallest = current.data
        return smallest

    # Function to insert an element if it is not a duplicate
    def insert_unique(self, data):
        if not self.head:
            self.head = Node(data)
            return
        current = self.head
        while current:
            if current.data == data:
                return
            if not current.next:
                current.next = Node(data)
                return
            current = current.next

    # Function to display elements in reverse order
    def display_reverse(self):
        stack = []
        current = self.head
        while current:
            stack.append(current.data)
            current = current.next
        while stack:
            print(stack.pop(), end=" ")
        print()
```

```
# Main program
if __name__ == "__main__":
    linked_list = LinkedList()

    # Create a list
    elements = [55, 45, 4224,7745, 452, 4562, 74563]
    for element in elements:
        linked_list.append(element)

    # Find the smallest element
    smallest = linked_list.find_smallest()
    print(f"Smallest element in the list: {smallest}")

    # Insert an element if it is not a duplicate
    new_element = 4
    linked_list.insert_unique(new_element)

    # Display elements in reverse order
    print("Elements in reverse order:")
    linked_list.display_reverse()
```

```
Smallest element in the list: 45
Elements in reverse order:
4 74563 4562 452 7745 4224 45 55
```

7.write a python script to implement various operations on stack

1.push
2.pop
3.display

In [41]:
```
class Stack:

    def __init__(self):
        self.__stack=[]

    def is_empty(self):
        if len(self.__stack)==0:
            return True
        else:
            return False
    def push(self,item):
        self.__stack.append(item)

    def pop(self):
        if len(self.__stack)==0:
            raise Exception('Stack is empty!!- no need to call pop punction ')

        else:
            return self.__stack.pop()

    def peek(self):
        print(self.__stack)
        if (self.is_empty()):
            raise Exception('Empty stack')
        else:
            return self.__stack[len(self.__stack)-1]

s=Stack()
for i in range(1,11):
    s.push(i)
```

In [42]:
```
while not s.is_empty():
    print("Pop:\t ",s.pop())
    print()
    print("display:\t ",s.peek())
```

```
Pop:      10

[1, 2, 3, 4, 5, 6, 7, 8, 9]
display:          9
Pop:      9

[1, 2, 3, 4, 5, 6, 7, 8]
display:          8
Pop:      8

[1, 2, 3, 4, 5, 6, 7]
display:          7
Pop:      7

[1, 2, 3, 4, 5, 6]
display:          6
Pop:      6

[1, 2, 3, 4, 5]
display:          5
Pop:      5

[1, 2, 3, 4]
display:          4
Pop:      4

[1, 2, 3]
display:          3
Pop:      3

[1, 2]
display:          2
Pop:      2

[1]
display:          1
Pop:      1

[]
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[42], line 4
      2 print("Pop:\t ",s.pop())
      3 print()
----> 4 print("display:\t ",s.peek())

Cell In[41], line 28, in Stack.peek(self)
     26 print(self.__stack)
     27 if (self.is_empty()):
---> 28     raise Exception('Empty stack')
     29 else:
     30     return self.__stack[len(self.__stack)-1]

Exception: Empty stack
```

8..write a python script to implement various operations on queue

1.insert
2.delete
3.display

In [59]:
```
class queue:
    def __init__(self):
        self.__queue = []

    def is_empty(self):
        if len(self.__queue)==0:
            return True
        else:
            return False

    def peek(self):
        print(self.__queue)
        if (self.is_empty()):
            raise Exception('Empty queue')
        else:
            return self.__queue[len(self.__queue)-1]

    def enqueue(self,item):
        self.__queue.append(item)

    def dequeue(self):
        if len(self.__queue)==0:
            raise Exception('Stack is empty!!- no need to call pop punction ')

        else:
            return self.__queue.pop(0)

q=queue()
q.enqueue(30)
for i in range(100,110):
    q.enqueue(i)
```

In [60]:
```
while not q.is_empty():

    print("Delete:\t ",q.dequeue())
    print()
    print("display:\t ",q.peek())
```

```
Delete:   30

[100, 101, 102, 103, 104, 105, 106, 107, 108, 109]
display:          109
Delete:   100

[101, 102, 103, 104, 105, 106, 107, 108, 109]
display:          109
Delete:   101

[102, 103, 104, 105, 106, 107, 108, 109]
display:          109
Delete:   102

[103, 104, 105, 106, 107, 108, 109]
display:          109
Delete:   103

[104, 105, 106, 107, 108, 109]
display:          109
Delete:   104

[105, 106, 107, 108, 109]
display:          109
Delete:   105

[106, 107, 108, 109]
display:          109
Delete:   106

[107, 108, 109]
display:          109
Delete:   107

[108, 109]
display:          109
Delete:   108

[109]
display:          109
Delete:   109

[]
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[60], line 5
      3 print("Delete:\t ",q.dequeue())
      4 print()
----> 5 print("display:\t ",q.peek())

Cell In[59], line 14, in queue.peek(self)
     12 print(self.__queue)
     13 if (self.is_empty()):
---> 14     raise Exception('Empty queue')
     15 else:
     16     return self.__queue[len(self.__queue)-1]

Exception: Empty queue
```

In [45]:
```
print(q)
```

```
<__main__.queue object at 0x0000018A22916210>
```

9.Write a program in python to convert the following infix expression to its postfix form using push and pop operations of a Stack

a. A/B^C+D*E-F*G

b. (B^2-4*A*C)^(1/2) (100)

In [61]:
```
def infix_to_postfix(expression):
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}

    def has_higher_precedence(op1, op2):
        return precedence[op1] >= precedence[op2]

    def is_operator(char):
        return char in '+-*/^'

    def infix_to_postfix_internal(expression):
        stack = []
        postfix = []
        for char in expression:
            if char.isalnum():  # Operand
                postfix.append(char)
            elif char == '(':  # Left parenthesis
                stack.append(char)
            elif char == ')':  # Right parenthesis
                while stack and stack[-1] != '(':
                    postfix.append(stack.pop())
                stack.pop()  # Remove the '(' from the stack
            elif is_operator(char):  # Operator
                while stack and stack[-1] != '(' and has_higher_precedence(stac
                    postfix.append(stack.pop())
                stack.append(char)

        while stack:
            postfix.append(stack.pop())

        return ''.join(postfix)

    return infix_to_postfix_internal(expression)


infix_expression1 = "A/B^C+D*E*-F+G"
infix_expression2 = "(B^2-4*A*C)^(1/2)*(100)"

postfix_expression1 = infix_to_postfix(infix_expression1)
postfix_expression2 = infix_to_postfix(infix_expression2)

print("Infix expression 1:", infix_expression1)
print("Postfix expression 1:", postfix_expression1)

print("Infix expression 2:", infix_expression2)
print("Postfix expression 2:", postfix_expression2)
```

```
Infix expression 1: A/B^C+D*E*-F+G
Postfix expression 1: ABC^/DE**+F-G+
Infix expression 2: (B^2-4*A*C)^(1/2)*(100)
Postfix expression 2: B2^4A*C*-12/^100*
```

In [ ]: