In [1]:
```
#stack
#Traditional implementation
class Stack:
  def __init__(self):
    self.arr=[]
    self.size=3
    self.f=0
  def push(self,element):
    if len(self.arr)==self.size:
      print('STACK IS OVERFLOW')
    else:
      self.arr.append(element)
  def pop(self):
    if len(self.arr)==0:
      print('STACK IS UNDERFLOW')
    else:
      self.arr.pop()
  def peek(self):
    if len(self.arr)==0:
      print('STACK IS UNDERFLOW')
    else:
      return self.arr[-1]
  def isEmpty(self):
    if len(self.arr)==0:
      return True
    else:
      return False
  def display(self):
    print(self.arr)
s=Stack()
s.push(1)
s.push(2)
s.push(3)
s.push(4)
s.display()
print(s.peek())
```

```
STACK IS OVERFLOW
[1, 2, 3]
3
```

In [2]:
```
#Reversing stack usign temp stack
class Stack1:
  def __init__(self):
    self.arr=[]
    self.p=[]
    self.size=3
    self.f=0
  def push(self,element):
    if len(self.arr)==self.size:
      print('STACK IS OVERFLOW')
    else:
      self.arr.append(element)
  def pop(self):
    if len(self.arr)==0:
      print('STACK IS UNDERFLOW')
    else:
      self.p.append(self.arr[-1])
      self.arr.pop()
  def peek(self):
    if len(self.arr)==0:
      print('STACK IS UNDERFLOW')
    else:
      return self.arr[-1]
  def isEmpty(self):
    if len(self.arr)==0:
      return True
    else:
      return False
  def display(self):
    print(self.arr)
s=Stack1()
s.push(1)
s.push(2)
s.push(3)
s.pop()
s.pop()
s.pop()
print(s.p[::-1])
```

```
[1, 2, 3]
```

In [3]:
```
#Queue
#QUEUE
class Queue:
  def __init__(self):
    self.arr=[]
    self.size=3
    self.f=0
  def push(self,element):
    if len(self.arr)==self.size:
      print('STACK IS OVERFLOW')
    else:
      self.arr.append(element)
  def pop(self):
    if len(self.arr)==0:
      print('STACK IS UNDERFLOW')
    else:
      self.arr.pop(0)
  def peek(self):
    if len(self.arr)==0:
      print('STACK IS UNDERFLOW')
    else:
      return self.arr[0]
  def isEmpty(self):
    if len(self.arr)==0:
      return True
    else:
      return False
  def display(self):
    print(self.arr)
s=Queue()
s.push(1)
s.push(2)
s.push(3)
s.pop()
print(s.arr)
```

```
[2, 3]
```

In [4]:
```
#stack using QUEUEs
class Queue:
  def __init__(self):
    self.arr=[]
    self.a1=[]
    self.size=3
    self.f=0
  def push(self,element):
    if len(self.arr)==self.size:
      print('QUEUE IS OVERFLOW')
    else:
      self.arr.append(element)
  def pop(self):
    if len(self.arr)==0:
      print('QUEUE IS UNDERFLOW')
    else:
      a=self.arr.pop(0)
      self.a1.append(a)
  def peek(self):
    if len(self.arr)==0:
      print('QUEUE IS UNDERFLOW')
    else:
      return self.arr[0]
  def isEmpty(self):
    if len(self.arr)==0:
      return True
    else:
      return False
  def display(self):
    print(self.arr)
s=Queue()
s.push(1)
s.push(2)
s.push(3)
s.pop()
s.pop()
s.pop()
print(s.arr)
print(s.a1[::-1])
```

```
[]
[3, 2, 1]
```

In [5]:
```
#implement linear search on stack
s=[1,2,3,4,5]
a=3
l=[]
f=0
while len(s)!=0:
  b=s.pop()
  if a==b:
    print('Element found')
    f=1
  l.append(b)
if f==0:
  print('Element not found')
```

```
Element found
```

In [ ]:
```
#Guessing game
from random import randint
num=randint(1,50)
chance=5
while chance!=0:
  n=int(input('Guess:'))
  if n==num:
    print('You won')
    break
  elif n<num:
    print(n,'is smaller than the actual number')
  else:
    print(n,'is greater than the actual number')
  chance-=1
print(num)
```

```
Guess:5
5 is smaller than the actual number
Guess:9
9 is smaller than the actual number
```

In [1]:
```
#Binary search using normal method
l=[1,2,3,4,5]
a=5
high=len(l)-1
low=0
b=0
while low<=high:
  mid=(low+high)//2
  if l[mid]==a:
    print('Element found at',mid)
    b=1
    break
  elif l[mid]>a:
    high=mid-1
  elif l[mid]<a:
    low=mid+1
if b==0:
  print('Element not found')
```

```
Element found at 4
```

In [2]:
```
#binary search using recursion
a=0
def binarySearch(l,low,high,key):
  mid=(low+high)//2
  if l[mid]==key:
    print('Element found at',mid)
    a=1
    return a
  elif l[mid]>key:
    high=mid-1
  elif l[mid]<key:
    low=mid+1
  return binarySearch(l,low,high,key)
l=[1,2,3,4,5]
key=5
high=len(l)-1
low=0
a=binarySearch(l,low,high,key)
if a==0:
  print('Element not found')
```

```
Element found at 4
```

In [ ]:
```

```