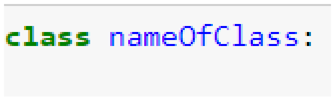
LAB 4

CLO:07

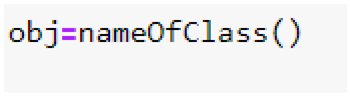
CLO	0	1	2	3
CLO2	0 for no problem solved	3.0 Marks for Question 1	3.0 Marks for Question 1	.0 Marks for Question 1

Create classes in python:



```
class nameOfClass:
```

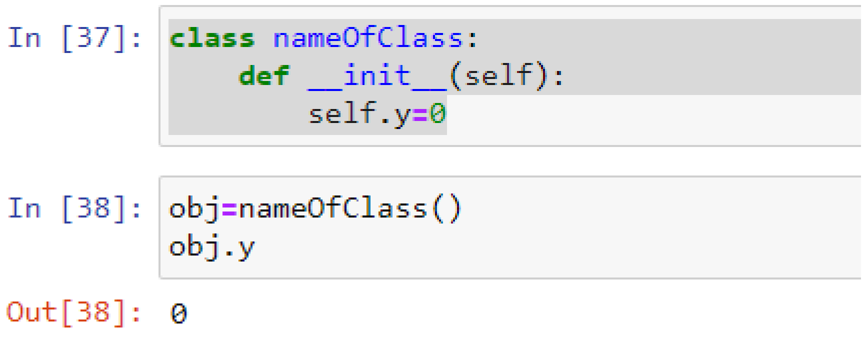
Create object:



```
obj=nameOfClass()
```

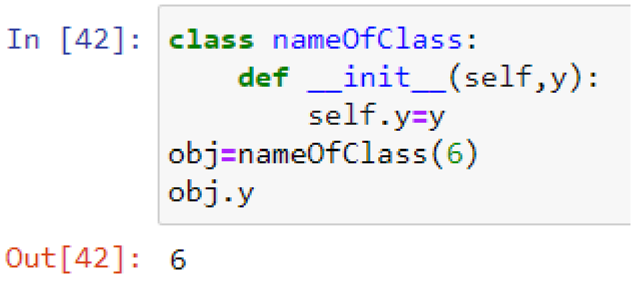
The init () Function:

To understand the meaning of classes we have to understand the built-in init () function. All classes have a function called init (), which is always executed when the class is being initiated. Use the init () function to assign values to object properties, or other operations that are necessary to do when the object is being created. The __init__() function is called automatically every time the class is being used to create a new object.



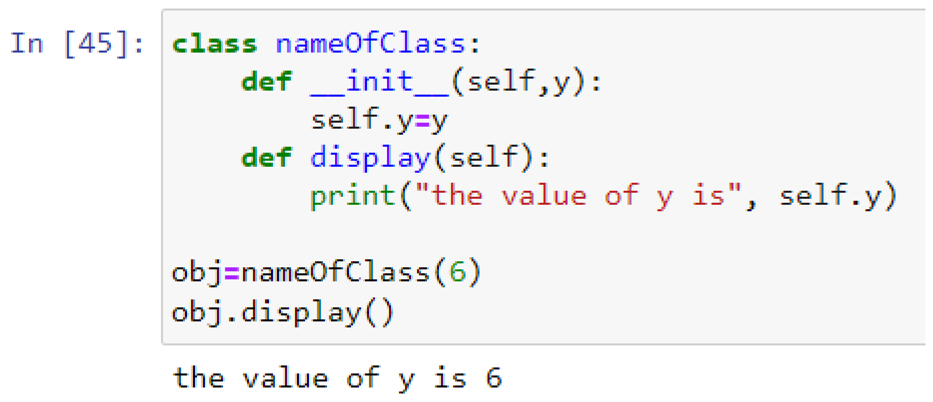
```
In [37]: class nameOfClass:
             def __init__(self):
                 self.y=0

In [38]: obj=nameOfClass()
         obj.y

Out[38]: 0
```

Pass the values:



```
In [42]: class nameOfClass:
             def __init__(self,y):
                 self.y=y
         obj=nameOfClass(6)
         obj.y

Out[42]: 6
```

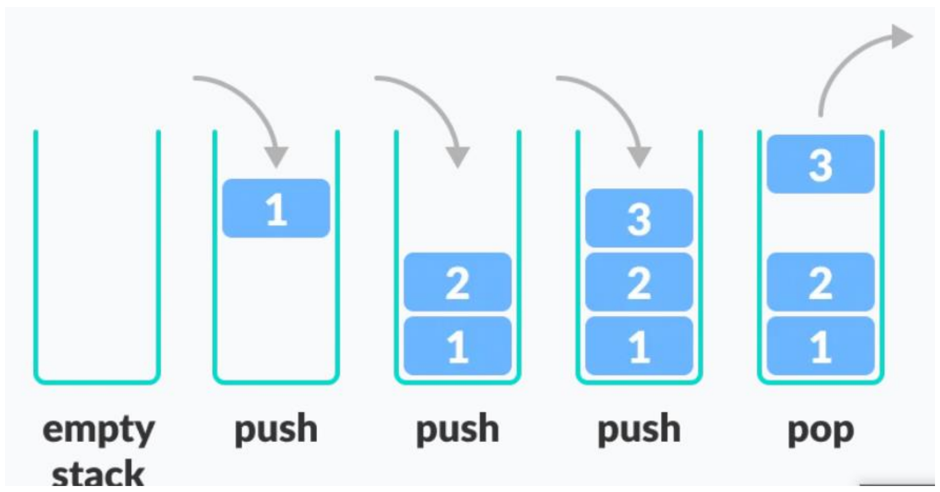
Create and call other functions:

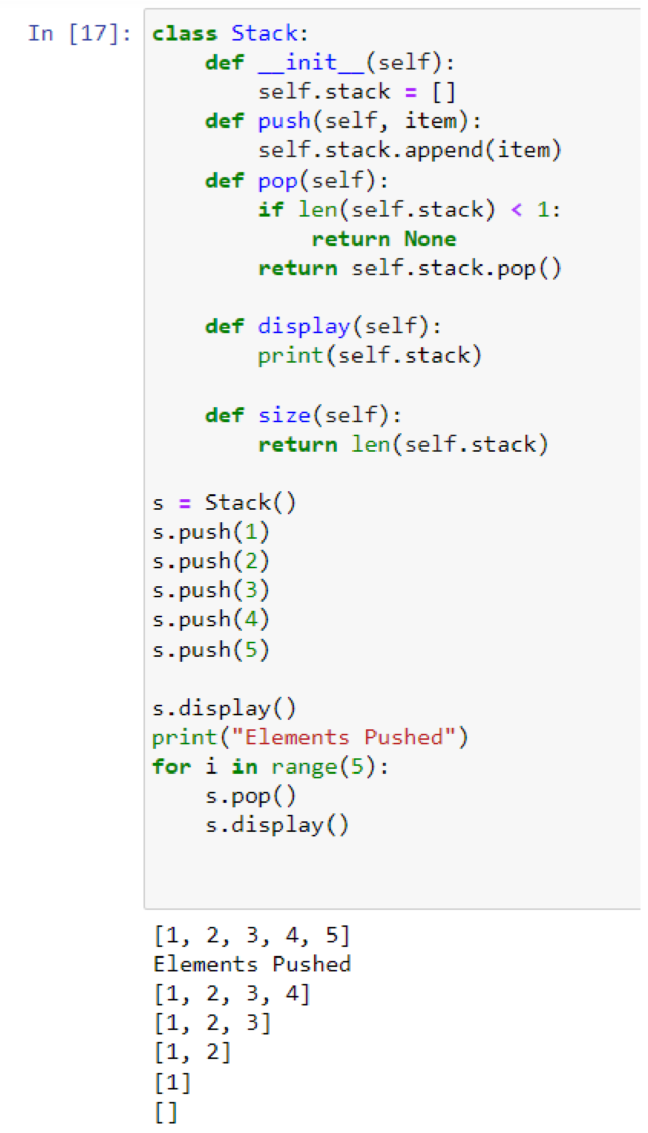


```
In [45]: class nameOfClass:
             def __init__(self,y):
                 self.y=y
             def display(self):
                 print("the value of y is", self.y)

         obj=nameOfClass(6)
         obj.display()

         the value of y is 6
```

<u>Question:1</u>

Implement Stack using Python.



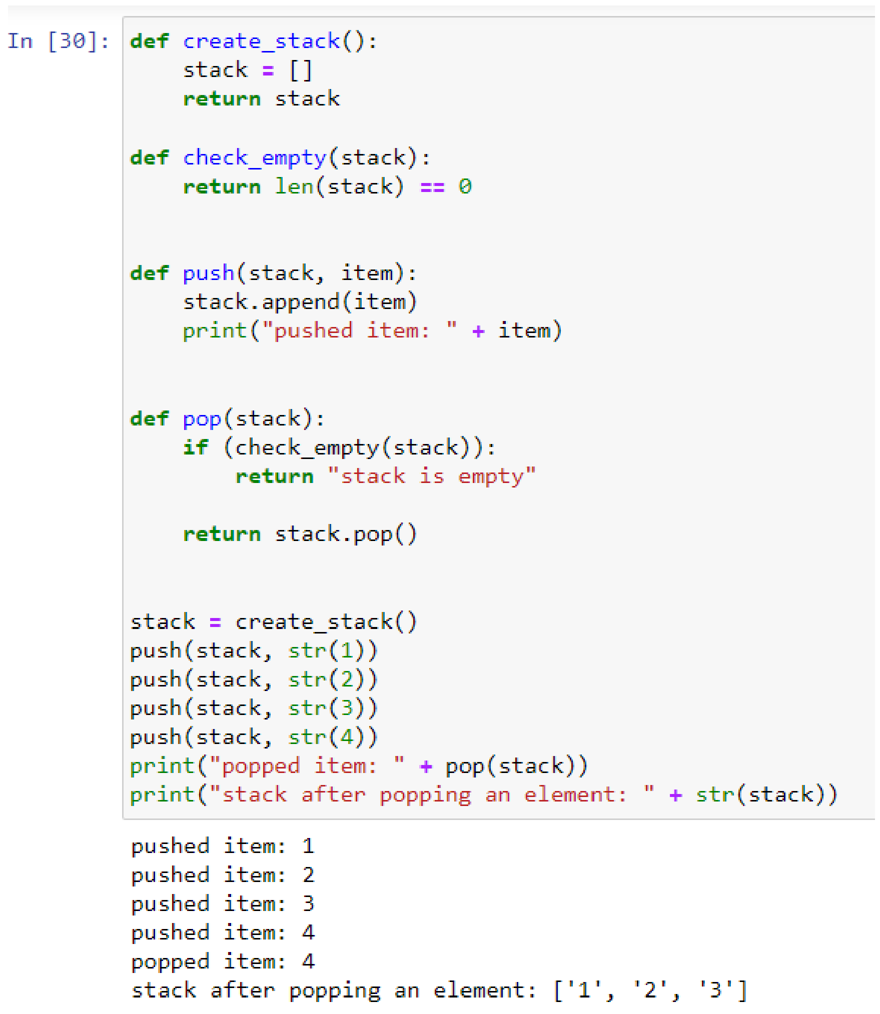


In [17]:

```python
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, item):
        self.stack.append(item)
    def pop(self):
        if len(self.stack) < 1:
            return None
        return self.stack.pop()

    def display(self):
        print(self.stack)

    def size(self):
        return len(self.stack)

s = Stack()
s.push(1)
s.push(2)
s.push(3)
s.push(4)
s.push(5)

s.display()
print("Elements Pushed")
for i in range(5):
    s.pop()
    s.display()
```

```
[1, 2, 3, 4, 5]
Elements Pushed
[1, 2, 3, 4]
[1, 2, 3]
[1, 2]
[1]
[]
```

Simple implementation:

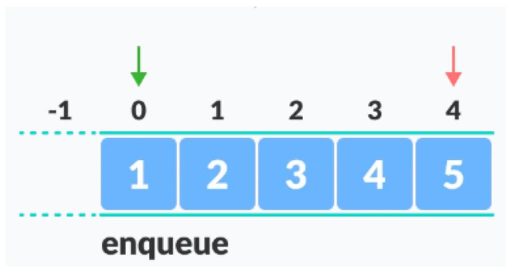


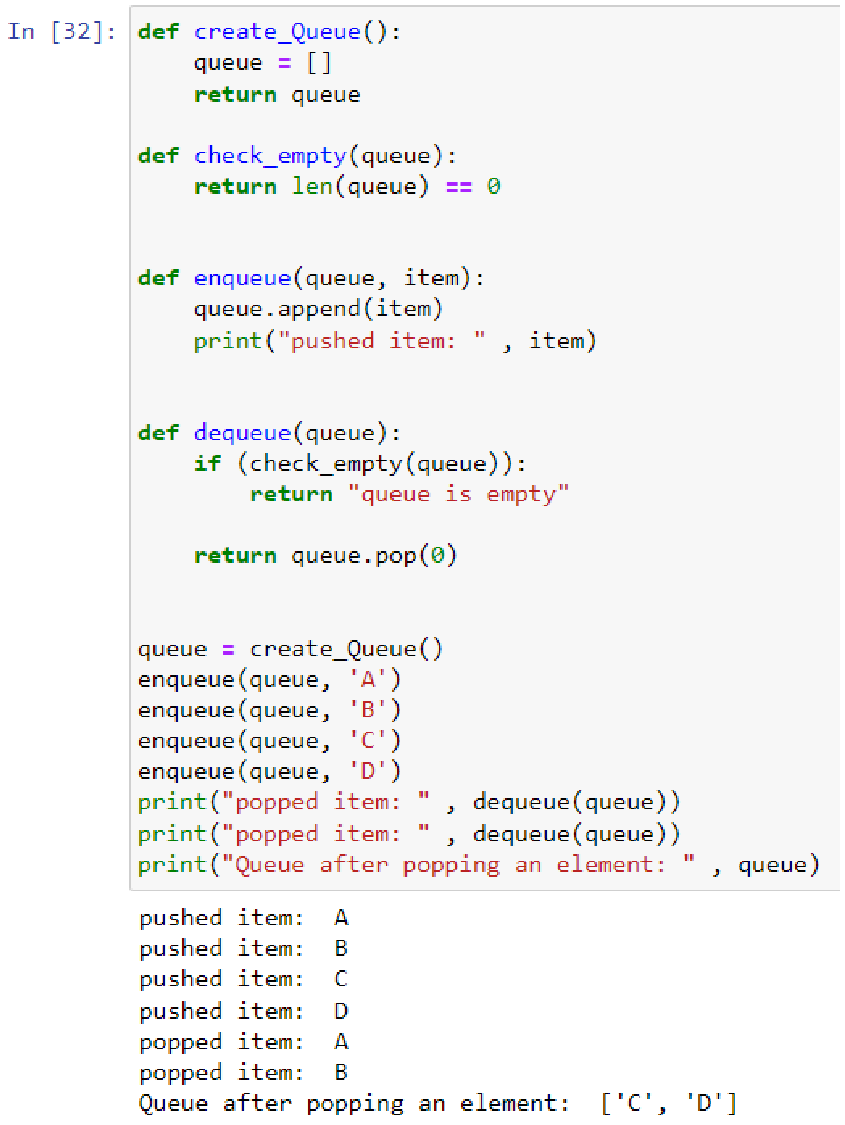
```
In [30]: def create_stack():
             stack = []
             return stack

         def check_empty(stack):
             return len(stack) == 0


         def push(stack, item):
             stack.append(item)
             print("pushed item: " + item)


         def pop(stack):
             if (check_empty(stack)):
                 return "stack is empty"

             return stack.pop()


         stack = create_stack()
         push(stack, str(1))
         push(stack, str(2))
         push(stack, str(3))
         push(stack, str(4))
         print("popped item: " + pop(stack))
         print("stack after popping an element: " + str(stack))
```

```
pushed item: 1
pushed item: 2
pushed item: 3
pushed item: 4
popped item: 4
stack after popping an element: ['1', '2', '3']
```

Question:2

Implement Queue using Python.

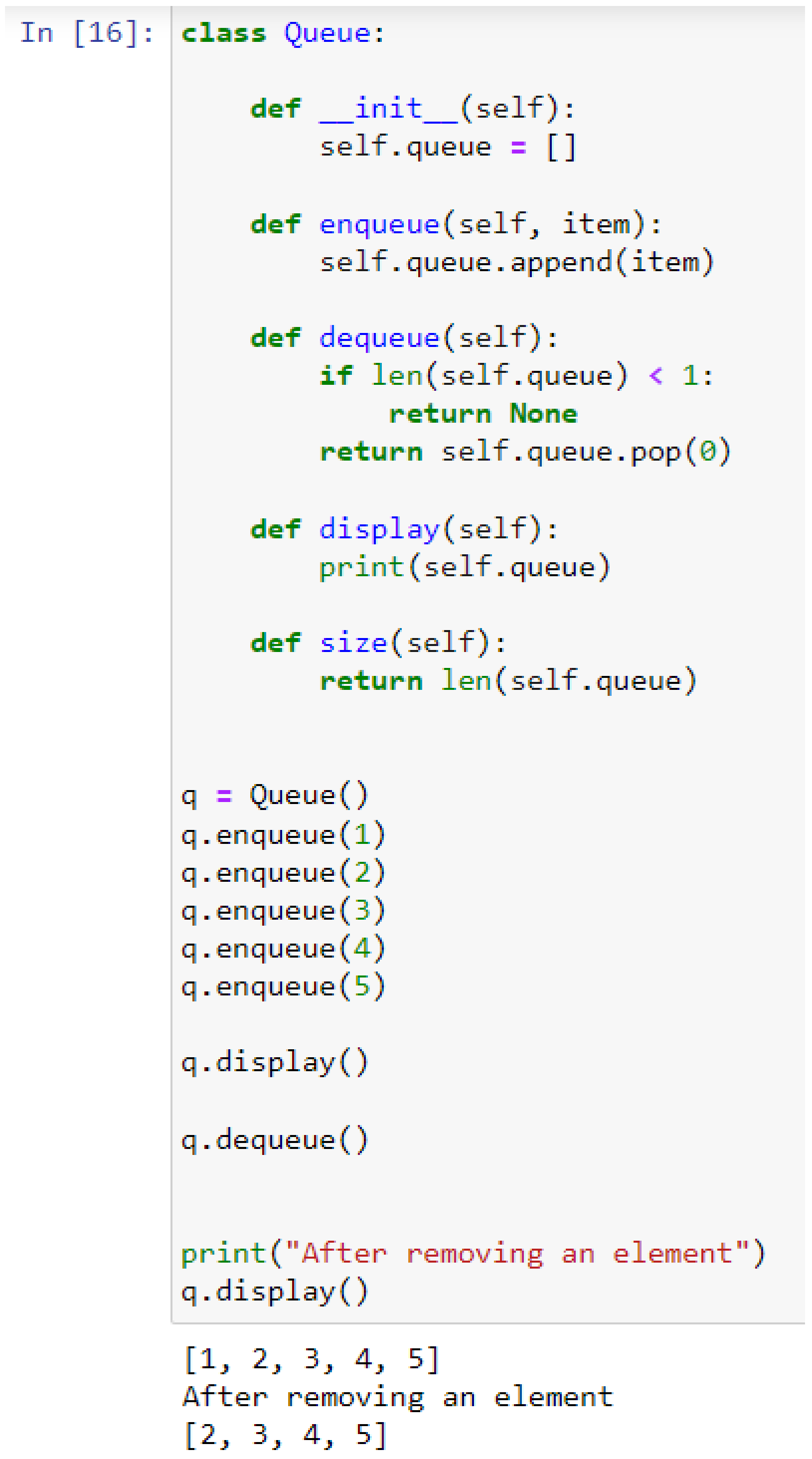




```
In [32]: def create_Queue():
             queue = []
             return queue

         def check_empty(queue):
             return len(queue) == 0


         def enqueue(queue, item):
             queue.append(item)
             print("pushed item: " , item)


         def dequeue(queue):
             if (check_empty(queue)):
                 return "queue is empty"

             return queue.pop(0)


         queue = create_Queue()
         enqueue(queue, 'A')
         enqueue(queue, 'B')
         enqueue(queue, 'C')
         enqueue(queue, 'D')
         print("popped item: " , dequeue(queue))
         print("popped item: " , dequeue(queue))
         print("Queue after popping an element: " , queue)
```

```
pushed item:  A
pushed item:  B
pushed item:  C
pushed item:  D
popped item:  A
popped item:  B
Queue after popping an element:  ['C', 'D']
```

With Class:



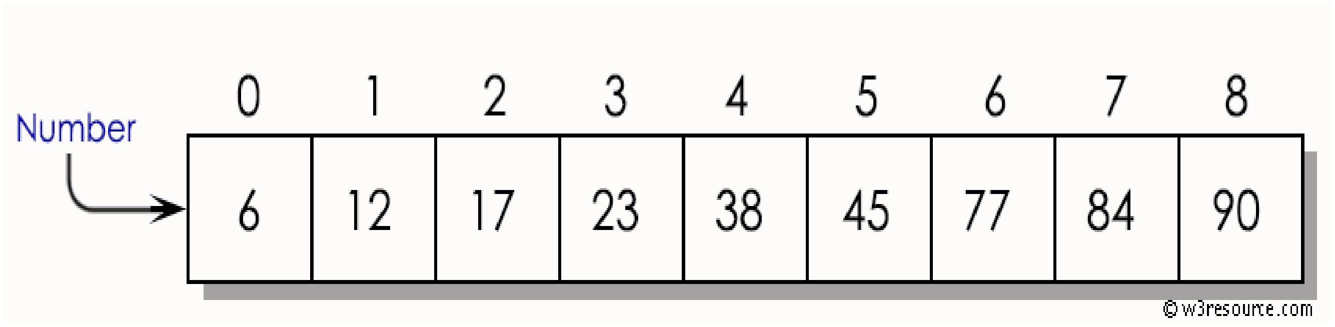
```
In [16]: class Queue:

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

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

             def dequeue(self):
                 if len(self.queue) < 1:
                     return None
                 return self.queue.pop(0)

             def display(self):
                 print(self.queue)

             def size(self):
                 return len(self.queue)


         q = Queue()
         q.enqueue(1)
         q.enqueue(2)
         q.enqueue(3)
         q.enqueue(4)
         q.enqueue(5)

         q.display()

         q.dequeue()


         print("After removing an element")
         q.display()
```

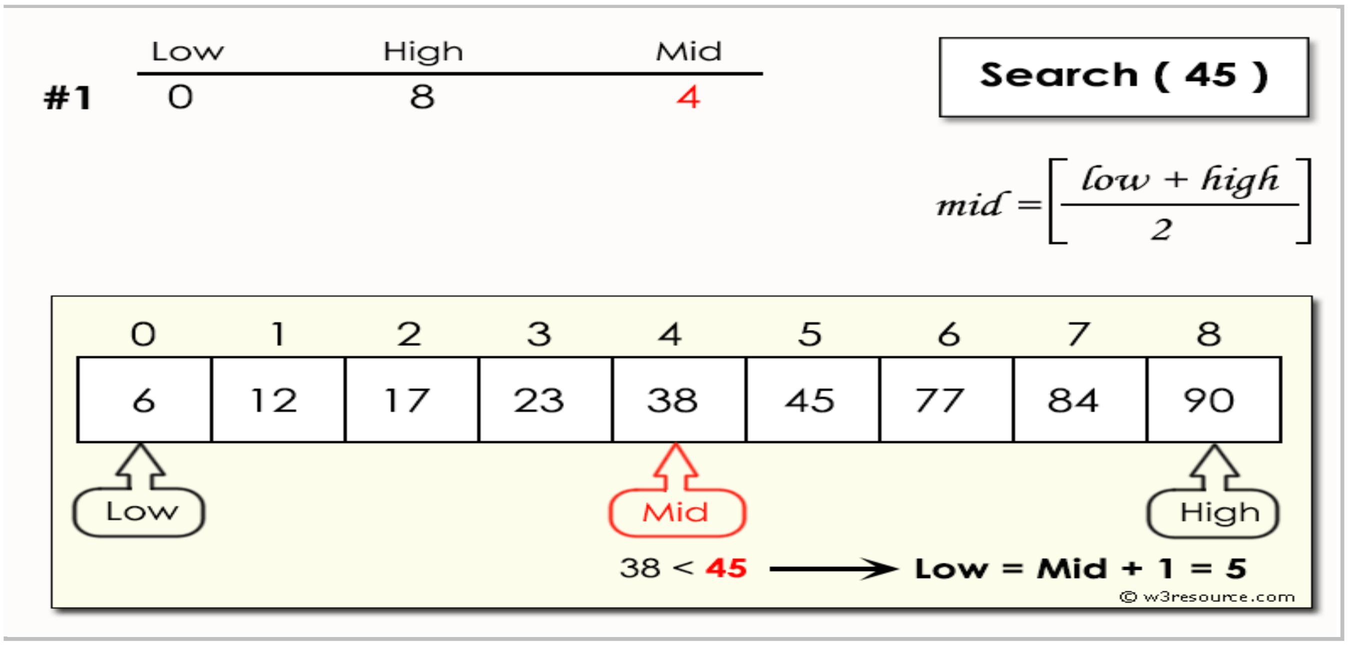
```
[1, 2, 3, 4, 5]
After removing an element
[2, 3, 4, 5]
```

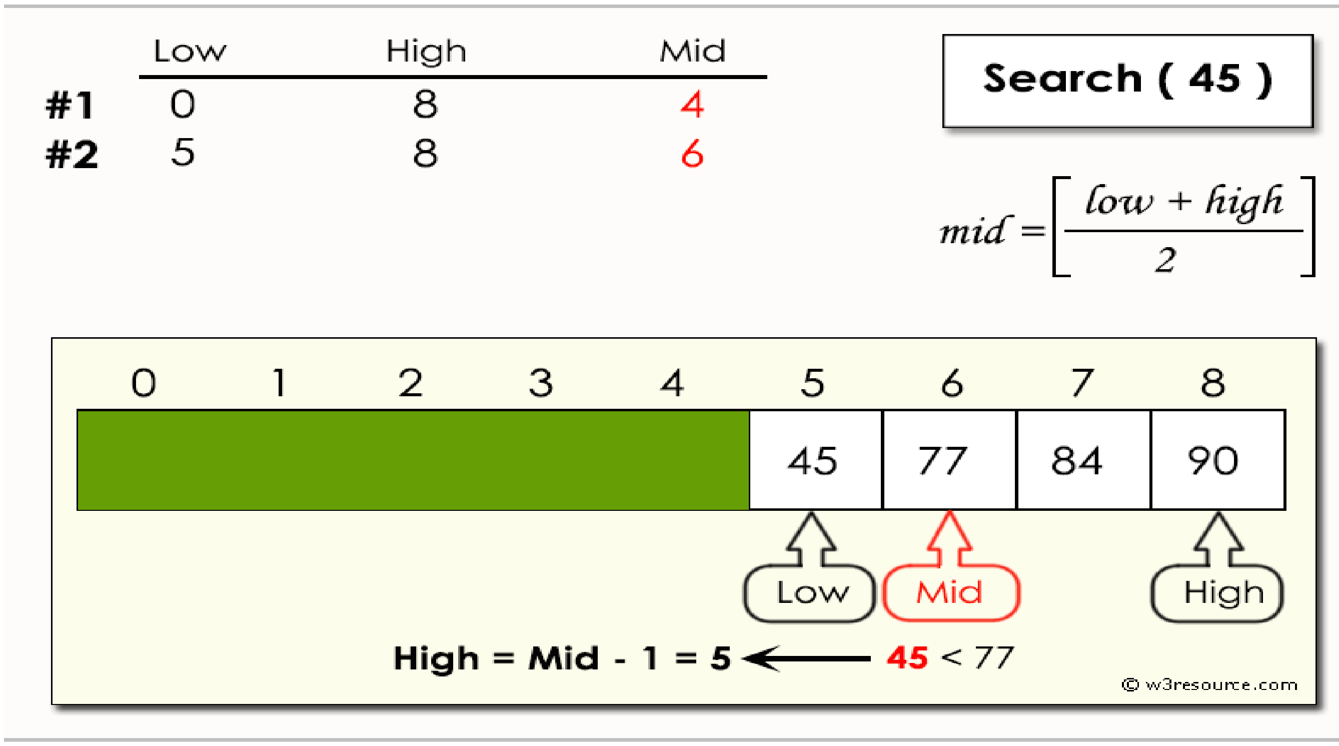
Question:3

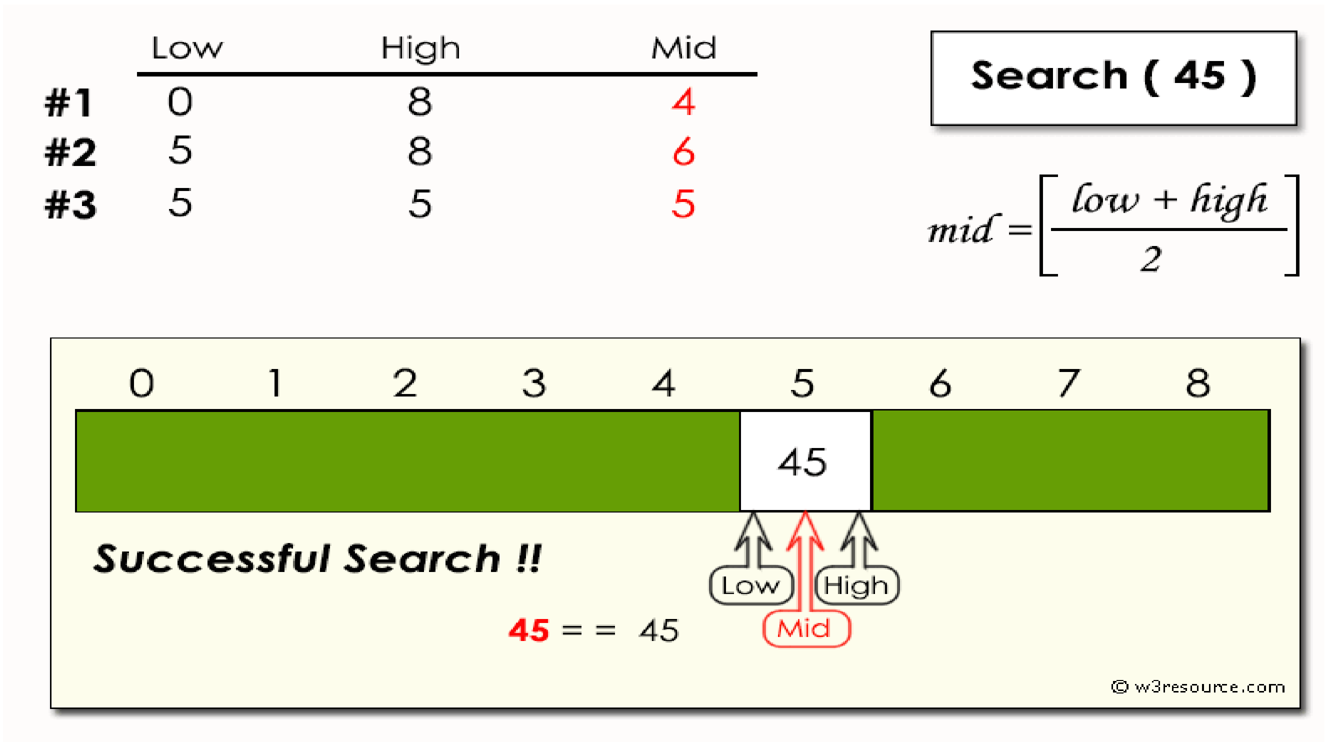
In computer science, a binary search or half-interval search algorithm finds the position of a target value within a sorted array. The binary search algorithm can be classified as a dichotomies divide-and-conquer search algorithm and executes in logarithmic time.

Step by step Example:









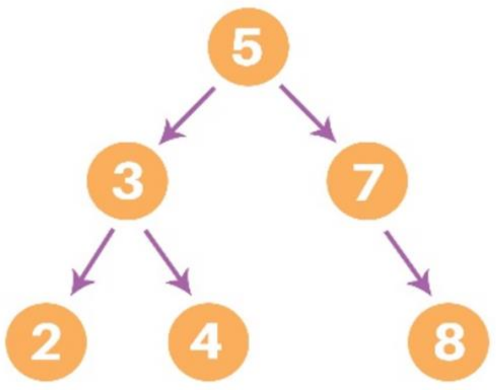
Breadth First Search:
Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores all of the neighbor nodes at the present depth

prior to moving on to the nodes at the next depth level. It uses a queue (First In First Out) instead of a stack and it checks whether a vertex has been discovered before enqueueing the vertex.
BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbour nodes (nodes which are directly
Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. Breadth-First Search in tree and graph is almost the same.

Algorithm:

```
BFS (visited ,G, s)          //Where G is the graph and s is the source
      node let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbor
vertices are marked.

      mark s as visited.
      while (Q is not
      empty)
       //Removing that vertex from queue, whose neighbor will be visited
           now v = Q.dequeue( )

          //processing all the neighbours
          of v for all neighbours w of v
          in Graph G
               if w is not visited
                 Q.enqueue( w )//Stores w in Q to further visit its
                        neighbor mark w as visited.
```

Bfs:



```python
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : [],
  '8' : []
}
```

```python
visited = []
queue = []
```

```python
print("Breadth-First Search")
bfs(visited, graph, '5')
```