SEARCHING CODE

```
class SearchableList:
    def __init__(self, items):
        self.items = items

    def linear_search(self, target):
        for index, value in enumerate(self.items):
            if value == target:
                return index
        return -1

    def contains(self, target):
        return self.linear_search(target) != -1

# Example usage
if __name__ == "__main__":
    numbers = [5, 3, 8, 1, 2]
    searchable_list = SearchableList(numbers

    target = 8
    if searchable_list.contains(target):
        print(f"Target {target} is in the list.")
    else:
        print(f"Target {target} is not in the list."
```

INSERTION CODE

```
class DynamicArray:
    def __init__(self):
        self.size = 0  # Number of elements in the array
        self.capacity = 1  # Initial capacity of the array
        self.array = [None] * self.capacity  # Initialize the array
       def insert(self, value):
        if self.size == self.capacity:  # Check if we need to resize
            self.resize(self.capacity * 2)  # Double the capacity
        self.array[self.size] = value  # Insert the value
        self.size += 1  # Increment the size
      def resize(self, new_capacity):
        new_array = [None] * new_capacity  # Create a new array with the new capacity
        for i in range(self.size):
            new_array[i] = self.array[i]  # Copy old elements to new array
        self.array = new_array  # Update the array reference
        self.capacity = new_capacity  # Update capacity
     def __str__(self):
        return str(self.array[:self.size])  # Return a string representation of the array


    dynamic_array.insert(1)
    dynamic_array.insert(2)


    print("Dynamic Array:", dynamic_array)
```

DELETION CODE

```
my_list = [1, 2, 3, 4, 5]
my_list.remove(3)
print("List after deletion:", my_list)
del my_list[1]  # Deletes the element at index 1 (which is 2)
print("List after index deletion:", my_list)
my_dict = {'a': 1, 'b': 2, 'c': 3}
del my_dict['b']
print("Dictionary after deletion:", my_dict
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
 def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def delete_node(self, key):
        curr_node = self.head
```

```
            if curr_node and curr_node.data == key:
            self.head = curr_node.next
            curr_node = None
            return
        prev_node = None
        while curr_node and curr_node.data != key:
            prev_node = curr_node
            curr_node = curr_node.next

            if not curr_node:
            return
    prev_node.next = curr_node.next
        curr_node = None

ll = LinkedList()
ll.append(1)
ll.append(2)
ll.append(3)ll.delete_nodecurr = ll.head
linked_list_values = []
while curr:
    linked_list_values.append(curr.data)
    curr = curr.next
print("Linked List after deletion:", linked_list_values)
```