1. Building Linked Lists

Build_linked_list.py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

n = Node(1)
n.next = Node(2)
n.next.net = Node(3)
```

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    def append(self, data):
        new_node = Node(data)
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node
```

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    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

    def print(self):
        output = []
        current = self.head
        while current:
            output.append(str(current.data))
            current = current.next
        print('->'.join(output))

ll = LL()
ll.append(1)
ll.append(2)
ll.append(3)

ll.print()
```

2. Searching

Search.py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    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
```

```
    def search(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False

    def print(self):
        output = []
        current = self.head
        while current:
            output.append(str(current.data))
            current = current.next
        print('->'.join(output))

elements = [0,1,2,3,4,5,6,7,8,9]
random.shuffle(elements)
ll = LL()
for e in elements:
    ll.append(e)

ll.print()

print(ll.search(5))
```

- There's only one little problem.
- Python's default recursion limit, its maximum stack size, is a thousand. That means that if your list is longer than 1000 elements long, this approach isn't going to work.
- Okay, great. All of our problems are solved because we made the recursion limit 2000.

```
import random
import sys

sys.setrecursionlimit(2000)


class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def search(self, data):
        if self.data == data:
            return True
```

```
        if self.next:
            return self.next.search(data)

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

    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

    def search(self, data):
        return self.head.search(data)

    def search(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False

    def print(self):
        output = []
        current = self.head
        while current:
            output.append(str(current.data))
            current = current.next
        print('->'.join(output))

elements = [0,1,2,3,4,5,6,7,8,9]
random.shuffle(elements)
ll = LL()
for e in elements:
    ll.append(e)

ll.print()
…
```

- Go to the powerpoint
- No, this is not a robust extensible enterprise solution. In addition, each time you recurs the stack itself takes time and memory to create and then pop back out again

```
import random
import sys

sys.setrecursionlimit(2000)


class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def search(self, data):
        if self.data == data:
            return True
        if self.next:
            return self.next.search(data)

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

    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

    # def search(self, data):
    #     return self.head.search(data)

    def search(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False
```

```
…
ll = LL()
for e in elements:
    ll.append(e)

ll.print()

print(ll.search(5))

print(ll.search(10))
```

3. Deleting Nodes

Deleting_finished.py

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    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

    def search(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False

    def delete(self, data):
        if not self.head:
```

```
            return

        if self.head.data == data:
            self.head = self.head.next
            return

        current = self.head
        while current.next:
            if current.next.data == data:
                current.next = current.next.next
                return
            current = current.next

    def print(self):
        output = []
        current = self.head
        while current:
            output.append(str(current.data))
            current = current.next
        print('->'.join(output))

elements = [0,1,2,3,4,5,6,7,8,9]

ll = LL()
for e in elements:
    ll.append(e)

ll.print()

ll.delete(0)
ll.delete(5)
ll.delete(9)

ll.print()
```

4. Inserting Nodes

Inserting_nodes.py

```
import random

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

```
class LL:
    def __init__(self):
        self.head = None

    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

    def search(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False

    def delete(self, data):
        if not self.head:
            return

        if self.head.data == data:
            self.head = self.head.next
            return

        current = self.head
        while current.next:
            if current.next.data == data:
                current.next = current.next.next
                return
            current = current.next

    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
```

```
        if self.head.data > data:
            self.head = new_node
            return

        current = self.head
        while current.next and current.next.data < data:
            current = current.next
        new_node.next = current.next
        current.next = new_node

    def print(self):
        output = []
        current = self.head
        while current:
            output.append(str(current.data))
            current = current.next
        print('->'.join(output))

elements = [0,1,2,3,4,5,6,7,8,9]
random.shuffle(elements)

ll = LL()
for e in elements:
    ll.insert(e)
    ll.print()
```

challenge1_solution.py

```
def remove_duplicates(self):
        current = self.head
        while current and current.next:
            if current.next.data == current.data:
                current.next = current.next.next
            else:
                current = current.next
```