# Debugging Mastery Problems

## 1

### Problem description:

Write a function that returns the larger of two numbers.

### Buggy code:

def larger\_number(a, b)  
 if a > b  
 return a  
 else:  
 return b

### Current AI solution:

def larger\_number(a, b)  
 if a > b  
 return a  
 else:  
 return b

### ChatGPT right solution:

def larger\_number(a, b):  
 if a > b:  
 return a  
 else:  
 return b

## 2

### Problem description:

Write a function that returns the sum of all elements in a list.

### Buggy code:

def sum\_of\_list(lst):  
 return sum(lst

### Current AI solution:

def larger\_number(a, b)  
 if a > b  
 return a  
 else:  
 return b

### ChatGPT right solution:

def sum\_of\_list(lst):  
 return sum(lst)

## 3

### Problem description:

Write a function that finds the second largest number in a list.

### Buggy code:

def second\_largest(lst):  
 lst.sort()  
 return lst[1]

### Current AI solution:

def larger\_number(a, b)  
 if a > b  
 return a  
 else:  
 return b

### ChatGPT right solution:

def second\_largest(lst):  
 lst.sort()  
 return lst[-2]

## 4

### Problem description:

Implement a basic Stack class with push and pop methods.

### Buggy code:

class Stack:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def push(self, item):  
 self.items.append(item)  
  
 def pop(self):  
 return self.items.pop(0)

### Current AI solution:

def larger\_number(a, b)  
 if a > b  
 return a  
 else:  
 return b

### ChatGPT right solution:

class Stack:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def push(self, item):  
 self.items.append(item)  
  
 def pop(self):  
 return self.items.pop()

## 5

### Problem description:

Implement a basic linked list with methods to add and remove nodes.

### Buggy code:

class Node:  
 def \_\_init\_\_(self, data):  
 self.data = data  
 self.next = None  
  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def add(self, data):  
 new\_node = Node(data)  
 if not self.head:  
 self.head = new\_node  
 else:  
 current = self.head  
 while current.next:  
 current = current.next  
 current = new\_node  
  
 def remove(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

### Current AI solution:

def larger\_number(a, b)  
 if a > b  
 return a  
 else:  
 return b

### ChatGPT right solution:

class Node:  
 def \_\_init\_\_(self, data):  
 self.data = data  
 self.next = None  
  
class LinkedList:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def add(self, data):  
 new\_node = Node(data)  
 if not self.head:  
 self.head = new\_node  
 else:  
 current = self.head  
 while current.next:  
 current = current.next  
 current.next = new\_node  
  
 def remove(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