Work with your neighbor. (This will be graded for participation only.)

1. The following code is supposed to iterate through a list of values, and remove duplicates; the values are not sorted, and so the duplicate values (if any) can be widely separated. If any duplicates are found, it keeps the first version and discards the rest; the surviving values must be in the same order as they began. (Also, this function is required to modify the list in place; it cannot simply return a new list.) There are many ways to do this. The code below uses nested loops; it looks for duplicate values by comparing every value to every other and removes the second one if it finds a duplicate.

   But it has several noteworthy bugs; see if you can find them. The first couple bugs are easy to find; give it almost any list, with more than a couple values (including some duplicates), and you will find them. But the next bug is more subtle - you will only be able to find it with specific inputs. To find that one, use the input [-50, 66, 80, 58, -50, 86, -19, -35, 45, 80, 80, -6, 34].

   Write a description of the bugs. Also, describe the techniques that you used to debug this code. Did you add `print()` statements? Where? What did you print out? Did you write additional testcases?

```
def remove_dups(alist):
    count = len(alist)
    i = 0
    while i < count:
        j = i
        while j < count:
            if alist[j] == alist[i]:
                alist.pop(j)
            j += 1

        i += 1
```

2. Why does the following function sometimes print the values out of order?

```
def sort_input():
    data = []
    while True:
        val = input("Enter a number (blank line to end) ")
        if val == "":
            break
        data.append(val)
    for v in sorted(data):
        print(v)
```

3. Recall that if $y = 10^x$, then $\log_{10}(y) = x$. Below is a simple function to calculate the log (base 10) of a number, *rounding up*. So $\log_{10}(100) = 2$, and $\log_{10}(108) = 3$.

```
# calculates the base-10 logarithm of a number,
# rounding up.
def log10(val):
    count = 0
    while val != 1:
        val = val // 10
        count += 1
    return count
```

Usage:

```
>>> log10(10)
1
>>> log10(18)
2
>>> log10(100)
2
>>> log10(108)
3
```

```
>>> log10(1000)
3
>>> log10(1008)
4
```

Why is this an infinite loop for most arguments? Fix the bug. Write your code here:

Is there another bug? Remember this code is rounding up. Make sure that your solution is correct. Verify that `log10(108)` is `3`. Put your new version here:

4. The following code to remove the last element in a linked list does not work for all linked lists:

```
def remove_last(self):
    if self._head == None:    #the list is empty
        return None

    curr = self._head
    while curr._next != None:
        prev = curr
        curr = curr._next
    prev._next = None
    return curr
```

a) Give an example linked list that causes this code to crash with an exception.

b) Write a correct version of the code.

5. This function attempts to verify that a tree is a Binary Search Tree (BST). Unfortunately, it has a few bugs. (Note: an empty tree and a leaf are both BSTs.)

```
def is_BST(root):
    if root is None:
        return True
    if root._left is None and root._right is None:
        return True
    if root._left._val > root._val or
                root._right._val < root._val:
        return False
    return is_BST(root._left) and is_BST(root._right)
```

a) The function errors out on many trees. Give an example of a tree that makes this function crash. Then fix that bug and write the fixed version below:

b) Does your fixed code work for all BSTs? If not, give an example of a tree for which your improved function will return an incorrect answer. (I.e., it returns `True` even if the tree is not a BST).

c) Explain why your function does not work for your example tree.

d) How would you fix this problem? You don't need to write the code, just explain in a couple of sentences what you need to do for a correct solution.