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
```

   Note: This code uses while loops instead of for loops because the list is being modified in the inner loop.

   ANS:

   Need to start the inner loop with `j = i+1` so that we don't find that a node is equal to itself.

   Need to **not** increment `j` after a deletion, so that we can check the next value (which has been shifted left)

   Need to reset `count` after a deletion.

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)
```

ANS:

We are reading the values in as strings and then sorting them. We were expecting to sort them in terms of integers, but unfortunately, we forgot to convert the strings to integers.

Python sorts lexicographically - that is, it looks at each character in turn, and will sort two strings based on the first character that is different. Thus, even though 2 is less than 11, the string `"2"` is greater than the string `"11"`.

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(8)
1
>>> 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:

ANS:
This fixes the infinite loop:

```
def log10_broken2(val):
    count = 0
    while val > 1:
        val = val // 10
        count += 1
    return count
```

Is there another bug? Remember this code should be rounding up. Make sure that your solution is correct. Verify that `log10(8)` is `1`. Put your new version here:

ANS:
Yes. The problem is due to using integer division instead of real division "/", since something like 8//10 is 0. Using "/" instead fixes the problem.

```
def log10(val):
    count = 0
    while val > 1:
        val = val / 10
        count += 1
    return count
```

Note: Problems 4 and 5 have been moved to ICA 38