CS 120: Intro to Computer Programming II

In-Class Activity - 07 Debugging - Day 1

Some of these examples were taken, or adapted, from sources on the web. The sources are available upon request, but I'm not providing them to you right now, because I want you to solve these yourself.

Activity 1 - Turn in this one

Leave space for all of your group members to find the bugs! Some people in the group will be able to glance through, and find some of the bugs (although there were a couple that weren't obvious to me until I ran the code). But remember: **our goal is for everybody to learn,** not to just "finish quick." So if you see the bugs, don't point them out - instead, let the whole group work together, and stumble across the bugs together.

Maybe we could even rotate bug-fixing? When we find a new bug in the code, ask somebody who hasn't talked yet to help the group debug this one.

(Follow this policy for **all** of the debugging exercises we do, over the next few days.)

The following code has a ton of errors, packed into a very little space: syntax errors, runtime errors that crash the program, and even some logical errors which cause silent bugs. How many can your group find?

```
import Random
a = random.randint(1,12)
b = random.randint(1,12)
for i in range(l0):
    question = "What is "+a+" x "+b+"? "
    answer = input(question)
    if answer = a*b
        print (Well done!)
    else:
        print("No.")
```

Turn in a list of all of the bugs that your group found.

Solution:

- `Random` should not be capitalized
- Need to cast `a` and `b` to `str` in the question
- Need to convert the answer to an integer for the comparison
- Need quotes around `Well done!`
- Need to either break out of the loop when the user succeeds - or else generate new a,b ever time. (The purpose of the program isn't 100% clear.)

(activity continues on next page)

Activity 2 - Turn in this one

The following code is supposed to iterate through an array 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 always 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 array in place; it cannot simply return a new array.)

There are a lot of ways to do this (this algorithm is actually **very inefficient,** there are lots of better options), but this code chooses to use nested loops; it looks for duplicate values by comparing every value to every other, and removes the second one if it finds a problem.

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

Turn in a description of both 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(arr):
    count = len(arr)
    i = 0
    while i < count:
        j = i
        while j < count:
            if arr[j] == arr[i]:
                arr.pop(j)
            j += 1
        i += 1
```

NOTE: This code uses `while` loops, instead of `for` loops, because Python doesn't allow you to modify an array while you're doing a `for` loop over it.

Solution:

- 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)

(activity continues on next page)

Activity 3 - Optional

OPTIONAL. Complete this if you have time, and turn it in. If you don't have time, you may report to your TA that you ran out of time.

I've heard about Bezier curves for years, but I finally decided to sit down and learn how they work. They're really cool, and surprisingly simple!

I've written a demo program, which shows how Bezier curves work - but unfortunately, it has some bugs. Your job is to make it work the right way. To see what it's **supposed** to look like, download and run `bezier_demo.mp4`, from the class website or D2L. But once you've seen what it's supposed to do, download `bezier_broken.py` and get to work!

Blank Lines, Please?

The first thing you'll notice is that I've removed all of the blank lines in the program (blame my son, it was his idea!). See how hard it is to read? Try to break the code up into logical chunks; add some blank lines to make it easier to read.

Operator Overloading

When you run the program, the first thing that you will see is that Python doesn't know how to subtract two `Vector` objects. That's perfectly reasonable - `Vector` isn't a standard class! But, you can use Python's operator overloading to fix that.

I haven't shown you operator overloading yet, but you should be able to get the general idea by looking at my `Vector` class - I've already implemented addition and multiplication. What do you guess might be the method name to implement subtraction?

Additional Errors

I have two additional bugs in the code (that I know of), which you will need to debug once you start running the code. These will be a little more challenging, as it will be necessary for the group to **analyze the code** and figure out how it works. What do the various variables do? What are these loops for? What is that `t` variable all about?

As you tear apart the program, remember to **keep everybody involved!** Take turns, figuring out the program piece by piece; let everybody talk. By the end, we hope that everybody will have at least a rough understanding of how the code works.

HINT: There are a number of features to this program. Sometimes, the best way to understand a program is to strip it down to its minimal parts. Try commenting out **almost everything,** and then adding pieces back, one at a time, to see how everything works together.

Solution: Download my fixed program: `bezier_fixed.py`