cs4_section2

February 8, 2019

Debugging is a very important (arguably **THE MOST IMPORTANT**) part of the programming design process. Debugging can be done effectively in IDE, but that can be time consuming and requires knowledge of an IDE. We'll teach this later on in the course when you're dealing with more and more complicated functions.

Debugging should not just be done when you finish writing a program; it should be an ongoing process while writing your code. You should debug your code every few lines- this makes it much more manageable to fix bugs. The easiest and fastest method to debug is to use print statements. Print statements allow the programmer to see the current state of the program without the use of an IDE. Let's look at an example.

```
In [50]: def search(num, list):
             '''
             Given a list, return true if a number is found in the list. Otherwise, return false

             '''
             found = False
             length = len(list) - 1
             for i in range(length):
                 if num == i:
                     found = True
                 else:
                     found = False
             return found
```

There are *3 bugs* in the following code. We'll debug these one at a time. Before we debug, let's write some test cases to check if our answers are correct.

```
In [51]: def test_search():
             '''
             Write some test cases here. Think of all the possible cases that we can get.
             '''
             #print(search(1,[0,1,2,3,4]))
             print(search(10,[0,5,10]))

         test_search()
```

```
0 False
5 False
```

```
10 False
True
```

Let's try to identify our first bug. Use **print** statements to identify which variables are incorrectly set or run during iteration. Write your code below with the statements.

```
In [ ]: def search(num, list):
            '''
            Given a list, return true if a number is found in the list. Otherwise, return false.
            '''
            found = False
            length = len(list) - 1
            for i in range(length):
                if num == i:
                    found = True
                else:
                    found = False
            return found
```

Let's try to identify our second bug. Use **print** statements to identify which variables are incorrectly set or run during iteration. Write your code below with the statements. Remember, you can add more test cases if you think these examples need to be set.

```
In [ ]: def search(num, list):
            '''
            Given a list, return true if a number is found in the list. Otherwise, return false.
            '''
            found = False
            length = len(list) - 1
            for i in range(length):
                if num == i:
                    found = True
                else:
                    found = False
            return found
```

Let's try to identify the final bug. Use **print** statements to identify which variables are incorrectly set or run during iteration. Write your code below with the statements.

```
In [ ]: def search(num, list):
            '''
            Given a list, return true if a number is found in the list. Otherwise, return false.
            '''
            found = False
            length = len(list)
            for i in list:
                print(i, found)
                if num == i:
```

```
                found = True
                return found
            else:
                found = False
        return found
```

```
In [38]: def reverse(s):
             """
             This function takes in a string and reverses it WITHOUT using list splicing. Try to
             instead.
             """
             reverse=""
             for char in s:
                 reverse = char+reverse
             return reverse
```

Testing is a very important process of the design and implementation process. Write some test cases below using assert.

```
In [39]: def test_reverse():
             """
             Add your test cases below
             """
             assert(reverse('hello')=='olleh'), "Normal case failed"
             assert(reverse('a')=='a'), "Single character case failed"
             assert(reverse('aba')=='aba'), "Palindrome case failed"
             assert(reverse('')==''), "Blank case failed"
             print("we're done!")

         test_reverse()
```

```
we're done!
```

The next bit of code that we will work with is manipulating images. You can think of images as 2D arrays that are objects.

So quickly: what is an object?

An object, in coding, has features (attributes) and can do things (have methods or functions).

For instance, a puppy might have atrributes such as age, breed, and fur color, while having the methods bark, eat, and sleep. We'll talk about this further in the semester, but it is good to understand the basics behind objects- we'll be using them on HW02.

The object that we will be using on HW2 is image. The image object has many attributes- the important ones that we want is the height and width (measured in pixels) and the RGB (red-green-blue) value of each pixel. The scale that we will use is from 0-255 for each RGB. Thus, (0,0,0) is black and (255,255,255) is white. You can test more colors here. Let's look at some code that inverts the colors in image- i.e. we get the complementary color, like red to cyan.

Trace the code and try to figure out what each line does:

```
def invert(filename):
    img = load_image(filename)
    height = img.get_height()
    width = img.get_width()
    for r in range(height):
        for c in range(width):
            rgb = img.get_pixel(r, c)
            red = rgb[0]
            green = rgb[1]
            blue = rgb[2]
            new_rgb = [255 - red, 255 - green, 255 - blue]
            img.set_pixel(r, c, new_rgb)
    new_filename = 'invert_' + filename
    img.save(new_filename)
```

How would we modify the code to work only on the top half of the image? Bottom half of the image? Checkerboard manager?

```
In [ ]: True
```