Lab 4 – PART 1

Topics Covered:

- Debugging strategies
- Problem solving

Pre-Lab Questions

There are 4 types of programming errors:

1. **Compile-time errors** are typos in the syntax that prevent the code from compiling
2. **Runtime errors** occur while the program is running and cause it to crash
3. **Infinite loops** are loops that never finish executing which cause the program to continue running until the user interrupts/ends it manually
4. **Logic errors** are mistakes in the code logic that cause unexpected results but do not crash the program

Question 1: Take a look at the following code segment. What is the compile-time error?

```
for k in range(10,-1,3):
    if k % 2 = 0:
        print("even")
    else:
        print("odd")
```

a) `for k in range(10,-1,3):`
b) `if k % 2 = 0:`
c) `else:`

Question 2: Take a look at the following code segment. What is the compile-time error?

```
for x in range(10):
    if x % 4 > 0:
        print("hi")
    else:
        print(hello)
```

a) `for x in range(10):`
b) `if x % 4 > 0:`
c) `print("hi")`
d) `print(hello)`

Question 3: Take a look at the following code segment. What type of error will occur?

```
v = 10
num = -5
while num <= 5:
      print(v / num)
      num = num + 1
```

a) Compile-time error
b) Runtime error
c) Infinite loop
d) Logic error
e) None, there are no errors of any kind.

Question 4: Take a look at the following code segment. What type of error will occur?

```
v = 10
num = -5
while num <= 5:
      print(v / num)
num = num + 1
```

a) Compile-time error
b) Runtime error
c) Infinite loop
d) Logic error
e) None, there are no errors of any kind.

Question 5: Take a look at the following code segment. When prompted for input, assume we type in **15**, **25**, and then **35**. What is the final output (hint: there is a logic error in the code)?

```
total = ""
for x in range(3):
    n = input("Enter a number: ")
    total = n + total
print(total)
```

a) 15
b) 35
c) 75
d) 152535
e) 352515

Question 6: Take a look at the following code segment. What is the output?

```
d = 0
for p in range(5,1):
    if p % 2 == 0:
        d = d + 1
    else:
        d = d + 2
res = 10 / d
print(res)
```

a) 2.0
b) 6.0
c) This will produce an infinite loop
d) This will produce a compile-time error
e) This will produce a runtime error

Question 7: Take a look at the following code segment. When prompted for input, assume we type in **85**. What do you think would be printed out?

```
grade = float(input("Enter your grade: "))
if grade > 50:
    print("D")
elif grade > 60:
    print("C")
elif grade > 70:
    print("B")
elif grade > 80:
    print("A")
elif grade > 90:
    print("A+")
else:
    print("F")
```

a) A+
b) A
c) B
d) D
e) F

Lab 4 – PART 2

We will now check the correctness of the code from Question 7 in the Pre-Lab questions.

1. **Copy the following code exactly as it is shown here:**

```
grade = float(input("Enter your grade: "))
if grade > 50:
    print("D")
elif grade > 60:
    print("C")
elif grade > 70:
    print("B")
elif grade > 80:
    print("A")
elif grade > 90:
    print("A+")
else:
    print("F")
```

Run the code several times, each time typing in one of the following input values. Write down the output for each input grade.

45 F
55 D
65 D
75 D
85 D
95 D

Are these the letter grades you expect to see for each of these input values? Most of them are incorrect. It's important to know that the order of conditionals matters, especially when dealing with operators such as <, <=, >, or >=. Remember that the first conditional that is True will be executes and no elif conditionals will even be checked after that. Fix this code by re-arranging the set of if and elif conditionals so that it works correctly. Test it with each of the input values from above and make sure it gives the proper output now.

45 F
55 D
65 C
75 B
85 A
95 A+

The process of programming is not just writing code but also finding errors in your code and fixing them. One of the most effective strategies for debugging or troubleshooting a program is through print lines in the code that show output from the program as it executes. It helps a lot because we get to see the values of variables that are being printed out but it also helps by what is NOT printed out. If we expect a specific word to be printed out and it does not print out when we run the program, that tells us that the code is not executing as expected. That helps us to narrow down the source of the problem.

2. Copy the following code exactly as it is shown here:

```
for n in range(11):
    div_2 = 0
    div_3 = 0
    div_4 = 0

    if n % 2 == 0:
        div_2 += 1
    elif n % 3 == 0:
        div_3 += 1
    elif n % 4 == 0:
        div_4 += 1

print(str(div_2) + " numbers divisible by 2")
print(str(div_3) + " numbers divisible by 3")
print(str(div_4) + " numbers divisible by 4")
```

This program is supposed to consider numbers 1 through 10 and count how many of these numbers are divisible by 2, by 3, and by 4. These three counts should then be printed out.

Run the code above to see the values for the three counts printed out at the end. Do these look correct? What do you think the correct values should be? Count on your fingers or on paper to determine what values you expect for each of them (i.e. for the first one, it should be 5 because the numbers 2, 4, 6, 8, and 10 are divisible by 2).

Since the printed values are incorrect, we need to determine whether the error is in the loop or in the conditionals or somewhere else (and there may be more than one error!).

Immediately after the for-loop declaration, add a line to print out n, and leave the rest of the code the same as it is.

```
for n in range(11):
    print(n)
    div_2 = 0
    div_3 = 0
    div_4 = 0

    ...
```

Now run the program again. You should see all the values of n printed out from 0 to 10. You might notice one problem is that we wanted to start at 1, not 0, so we need to modify the range slightly:

```
for n in range(1,11):
```

Run the program again. As you can see, this did not change the output, but it did fix one small part of the problem even if it doesn't look like it now. Including 0 would have caused some of the count values to be incorrect. But the main problem hasn't been solved so let's add more print lines to try to figure out where the errors are coming from.

```
for n in range(1,11):
    print(n)
    div_2 = 0
    div_3 = 0
    div_4 = 0

    if n % 2 == 0:
        print("div by 2")
        div_2 += 1
    elif n % 3 == 0:
        print("div by 3")
        div_3 += 1
    elif n % 4 == 0:
        print("div by 4")
        div_4 += 1

print(str(div_2) + " numbers divisible by 2")
print(str(div_3) + " numbers divisible by 3")
print(str(div_4) + " numbers divisible by 4")
```

We have now added print lines inside each of the if/elif statements so we should see more clearly where the error is coming from depending on how many of these lines get printed out. Run the program again and count how many times you see "div by 2", "div by 3", and "div by 4" printed out in the console. Do these numbers match the counts being computed and printed at the end of the program?

At this point, you should see "div by 2" printed out 5 times and "div by 3" twice. Yet at the bottom we are seeing "1 numbers divisible by 2" and "0 numbers divisible by 3". We do not see any "div by 4" which does match the 0 count printed out at the bottom; however it is incorrect because 4 and 8 are divisible by 4, but this is a different error and we'll come back to it in a bit.

We noticed that the new print lines are being printed out more times than the count values at the bottom. This means something is going on with these counts.

Maybe you've already noticed the problem here. If not, one strategy to help you see it better is to add one more print line in the loop that prints out the value of div_2 every iteration of the loop (you may want to comment out the other print lines to reduce the amount of output) and run the program again. You will see that this values alternates between 1 and 0 many times.

The problem is that the div_2, div_3, and div_4 variables are being declared and set to 0 inside the loop which means that at every iteration of the loop, they are reset to 0. This should be done **BEFORE** the loop so they begin at 0 but do not get reset during the loop. Move these 3 lines to the top of the program as shown below.

```
div_2 = 0
div_3 = 0
div_4 = 0

for n in range(1,11):
    print(n)
    ...
```

Run the program again. We're getting closer! However, there is still another error. It says 2 numbers are divisible by 3 but we know there are actually 3 such numbers: 3, 6, and 9. Likewise, we earlier mentioned that 2 numbers are divisible by 4 (4 and 8), but it says 0 numbers are divisible by 4.

If you commented out the print lines from before, un-comment them. When you run it, you should see each number n and between some of them, a message "div by 2", "div by 3", or "div by 4". For which values of n does it say "divisible by 3"? We know it should say it for 3, 6, and 9. Which one of those numbers does not show that message? You should see that 3 and 9 both have the "div by 3" message and 6 does not. However, since 6 is even, it does have the "div by 2" message. This should give you a hint as to the source of the error.

The conditionals are set up as if and elif statements but this means at most one conditional can execute since they are grouped together. The number 6 is divisible by both 2 and 3 but in this code, it will only execute the divisible by 2 conditional and never get to the divisible by 3 conditional. Similarly, 4 and 8 are divisible by 4 but also by 2. So we need to fix this by having 3 separate if-statements rather than elif statements. The final code should look like this:

```
div_2 = 0
div_3 = 0
div_4 = 0

for n in range(1,11):
    print(n)

    if n % 2 == 0:
        print("div by 2")
        div_2 += 1
    if n % 3 == 0:
        print("div by 3")
        div_3 += 1
    if n % 4 == 0:
        print("div by 4")
        div_4 += 1

print(str(div_2) + " numbers divisible by 2")
print(str(div_3) + " numbers divisible by 3")
print(str(div_4) + " numbers divisible by 4")
```

Now when you run it again, you should see the correct results! You can see how adding print lines helped you to debug your code and find the errors.

3. Identify errors and fix the code.

The following code segment is a short program that prompts the user for an initial number and then prompts for 3 additional numbers. It checks if all 3 of those additional numbers are smaller than the first number. If so, it should print out True, but if at least one number is not smaller than the first number, it should print out False. There are 2 errors in this program. Find the errors and correct the code. (Hint: it must work with negative numbers.) Note: you can use the PyCharm Debugger or any other strategy to help debug this program.

Test cases

Inputs: 12, 7, 5, 9	Expected Output: True
Inputs: 5, -3, 1, 4	Expected Output: True
Inputs: 3, 1, 7, 2	Expected Output: False

```
1.  n = int(input("Enter a number: "))
2.  all_smaller = True
3.
4.  for n in range(3):
5.      next_num = int(input("Enter another number: "))
6.      if next_num < n:
7.          all_smaller = False
8.
9.  print(all_smaller)
```