CSSE 120

Session 5: Debugging and Tracing Code by Hand

Name:_______ **SOLUTION** ________ CM: _________ Section: _____ Grade: ________ of 10

1. What are some good practices for writing code so that you avoid bugs? Check all that apply:

   _____ Do iterative enhancement. ***[Answers shown in RED should be checked.]***

   _____ Keep patterns, like the accumulator pattern, in mind.

   _____ Break a problem into subproblems and solve each part.

   _____ Use descriptive names in your programs.

   _____ Avoid using *print* statements in your code, since specifications often call for no side effects.

2. What is *iterative enhancement*? Select one:

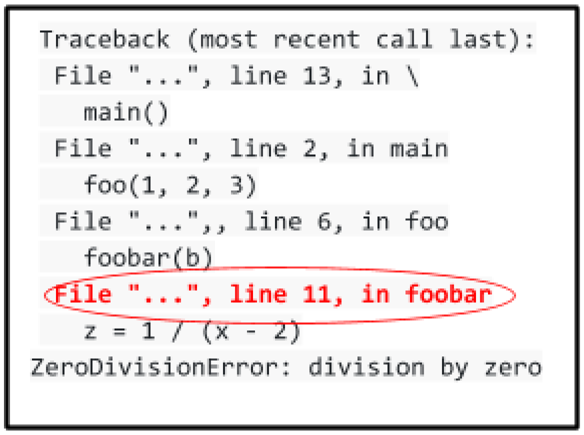
   _____ Writing loops, adding a little to a total each time through the loop, as in
   `total = total + k**2.`

   _____ Breaking a program into small parts and using helper functions for those parts.

   _____ Writing your program, adding complexity a little at a time and testing as you go.

3. True / [✓ ***False***] (circle one): When I get a red error message when I run my code, I find my error by clicking on the *lowest* link, right above the error, even if it goes into library code that I didn't write.

4. In the stack trace and error message shown to the right:



```
Traceback (most recent call last):
 File "...", line 13, in \
   main()
 File "...", line 2, in main
   foo(1, 2, 3)
 File "...",, line 6, in foo
   foobar(b)
 File "...", line 11, in foobar
   z = 1 / (x - 2)
ZeroDivisionError: division by zero
```

   - The "File" lines in the stack trace are links to code. (We put … instead of the actual links to simplify this question.) Circle the link (that is, the line that begins with *File:*) that you should click on to go straight to the line that initiated the "crash" (more precisely, the "generation of an *exception*").
   - True / [✓ ***False***] (circle your choice). The error in the code is at that line to which your circled link takes you. (Also, ***explain your answer briefly***.)
     Explanation: That line is just the line that GENERATED the exception, not necessarily the actual SOURCE of the error.
   - What was the value of ***x*** when the code generated the stack trace? **2**

5. [✓ ***True***] / False (circle one): When my program runs but a test fails, I should re-work the failed test by hand and add *print* statements to my code at each step to see if the computer is doing what I thought it should do.

6. When is Exam 1 -- what day, and what time, in what room? (Look at the course calendar on the Home Page now for the answer. Follow the link for answers to this and the next question.)

   Monday, December 16. 8:30 to 11 p.m. Rooms O257, O259, O267, O269 for the 4 sections.

7. What 3 things must I complete as my Admission Ticket for taking Exam 1?
   - The 05a-Debugging project.
   - The 05b-Exam1Practice project.
   - The Paper-and-Pencil practice problems.

8. With your instructor's guidance, do the following problem (taken from a previous term's exam). As you do so, **pay attention to how you can keep track of things as you trace the code by marking up your answer in helpful ways.**

   Consider the code below. It is a contrived example with poor style but will run without errors. What does it print when it runs? Write your answer in the box to the right. Make notations in the code as desired to show your work.

```
def main():
    a = blue(7)
    b = red(6, 4)
    print('Main:', a, b)


def blue(x):
    print('Blue:', x)
    x = 2 * x
    print('Green:', x)
    return x + 3
    print('Yellow:', x * 100)


def red(r, s):
    print('Red:', r, s)
    print('OK', 3 * blue(s))
    return blue(r + s)
    print('Black', r + s)


print(blue(1))
main()
```

Output:

I have put extra spaces to make the answer easier to read.

```
Blue:   1
Green:  2
5
Blue:   7
Green:  14
Red:    6  4
Blue:   4
Green:  8
OK   33
Blue:   10
Green:  20
Main:   17   23
```

9. With your instructor's guidance, do the following problem (taken from a previous term's exam). As you do so, **pay attention to how you can keep track of things as you trace the code by marking up your answer in helpful ways.**

   Consider the code below. It is a contrived example with poor style but will run without errors. In this problem, you will trace the execution of the code. As each location is encountered during the run:

   1. ***CIRCLE* each variable** that is ***defined*** at that location.
   2. ***WRITE t****he* ***VALUE*** of each variable that you ***circled*** directly ***BELOW*** the circle.

```
def main():
    w = 1
    x = 2
    y = 3
    z = 4
    #### Location 1

    z = cat(w, x, y, y)
    #### Location 2

    w = 999
    a = 44
    a = dog(a)
    #### Location 3


def dog(a):
    #### Location 4
    w = 100
    a = a + w
    w = w + 25
    #### Location 5
    return a


def cat(w, z, y, x):
    #### Location 6
    w = 50
    x = 101
    b = w
    b = b + 45
    #### Location 7
    return w

#### Location 8
main()
#### Location 9
```

Location 1	a	b	w 1	x 2	y 3	z 4
Location 2	a	b	w 1	x 2	y 3	z 50
Location 3	a 144	b	w 999	x 2	y 3	z 50
Location 4	a 44	b	w	x	y	z
Location 5	a 144	b	w 125	x	y	z
Location 6	a	b	w 1	x 3	y 3	z 2
Location 7	a	b 95	w 50	x 101	y 3	z 2
Location 8	a	b	w	x	y	z
Location 9	a	b	w	x	y	z

Make notations in the code as desired to show your work.

Ask for help if you do not understand the instructions for this problem.

10. With your instructor's guidance, do the following problem (taken from a previous term's exam). As you do so, **pay attention to how you can keep track of things as you trace the code by marking up your answer in helpful ways.**

    Consider the code below. It is a contrived example with poor style but will run without errors.

    What does it print when it runs? Write your answer in the box to the right. Use the empty space to keep track of variables as you work.

```
a = 0
b = 3
c = 15
for k in range(4):
    a = a + (10 * k)
    b = b + (a + 1)
    print(k, a, b, c)
    c = c + 1

print('done')
print(a, b, c)
```

Output:

I have put extra spaces to make the answer easier to read.

```
0     0     4    15

1    10    15    16

2    30    46    17

3    60   107    18

done

60    107    19
```