Python Programming

Chapter 5 Exam – “Finding and Fixing Problems”

1. **Which of the following is NOT a general category of error?**
    a. Sneaky Error
    b. Syntax error
    c. Logical error
    d. Runtime exception

2. **What general term to software engineers use to describe errors in programs?**
    a. Bugs
    b. Breakpoints
    c. Tracepoints
    d. Flops

3. **Which of the following is NOT displayed in a syntax error message?**
    a. The suggested best practice for avoiding the error in the future
    b. The file name where the error was found
    c. The line of code where the error was found
    d. A carat (^) pointing at the likely location of the error within a statement

4. **When you get a syntax error, what should you do if you can't find the problem at the line of code where the carat is pointing?**
    a. Look backwards (upwards) at earlier statements to see where the problem starts
    b. Look downwards (forwards) at later statements to see where the problem ends
    c. Engage the debugger and set a breakpoint
    d. Call the Python support hotline

5. **If a program runs without any error messages, but doesn't do what you want it to do, what kind of error do you likely have in your code?**
    a. Logical error
    b. Syntax error
    c. Runtime exception
    d. Any of these is possible

6. **When do you normally see the effects of a logical error?**
    a. When the program is running
    b. Before the program starts
    c. After the program ends
    d. Any of these times is possible

7. **What happens when a runtime exception is encountered?**
    a. Your program halts completely with an error message
    b. Your program hangs forever without any more input or output
    c. Your program continues to run but may not behave the way you wanted
    d. An automatic error report is sent to the Python Software Foundation

8. **Why are logical errors sometimes harder to find and fix?**
    a. You do not see any error messages that give you hints about the problem
    b. The logical error messages are more complicated than other error messages
    c. You have to keep re-starting your program after it crashes
    d. You don't have any tools to help you find and fix those kinds of problems

9. **When conducting a code review, in what order should you normally examine your statements?**
    a. From top to bottom, in the order they normally execute
    b. From bottom to top, working backwards
    c. Start with the longest and most complex statements first
    d. Start with the shortest and easiest statements first

10. **When Python runs a particular statement, which of the following will influence the behavior and success of that statement?**
    a. Earlier statements that have already run to create or update variable values
    b. The comments you have left in the code for that statement
    c. The statements that will run afterwards to provide more information
    d. All of these are true

11. **Which of the following statements is true about a code review?**
    a. You may be able to skip to a suspicious area of code if you think you know where the problem lies
    b. You should always start at the very beginning, no matter how many statements you have or where you think the problem lies
    c. Code review is by far the slowest way to find and fix problems
    d. Code reviews will not work if you don't import the debugger firsts

12. **What is wrong with the following code?**

    age = int(input (How old are you?))

    a. It is missing double quotes around the input() string parameter
    b. It will throw an exception at runtime due to the nested function calls
    c. It does not correctly assign the result value to the variable
    d. Nothing is wrong; the statement will run successfully

13. **If you want to print "Password verified" when the user guesses the matching password, what is wrong with the following code?**

```
password = "SECRET"

guess = input("What's the password? ")

if (guess != password):

        print("Password verified")

else:

        print("Access denied")
```

a. It will print the opposite of what you want due to a logical error
b. It will show a syntax error
c. It will throw a runtime exception
d. Nothing is wrong; it will do exactly what you want

14. **Where might you want to place program trace statements to help you understand your program?**
    a. All of these are good places
    b. Inside an "if" body to show when that logical path is taken
    c. Before a long calculation to show all of the initial variable values
    d. After a long calculation to show the results of the mathematical expression

15. **Which function do we suggest using to create your program trace statements?**
    a. print()
    b. trace()
    c. breakpoint()
    d. display()

16. **In what state is your program if it is executing at full speed and you can't observe your variables and lines of code in the debugger?**
    a. Running state
    b. Break state
    c. Hyper state
    d. Unknown state

17. **What do you need to do to enable the Python debugger for your program?**
    a. import the "pdb" library
    b. Nothing; the debugger is always available by default
    c. Call pdb.load() to load the "pdb" library
    d. Call pdb.set_trace() to load the "pdb" library

18. **Which of the following statements successfully sets a breakpoint for the Python debugger?**
    a. pdb.set_trace()
    b. pdb.breakpoint()
    c. pdb.stop()
    d. print("Breaking here")

19. **Which Python debugger command would you use to run the next line of code and then return to the break state?**
    a. step (s)
    b. continue (c)
    c. list (l)
    d. prompt (p)

20. **What Python debugger command will show you several lines of code around the line that is about to be executed?**
    a. list (l)
    b. display (d)
    c. print (p)
    d. status (s)