PYTHON ASSIGNMENT 11

1. **Create an assert statement that throws an AssertionError if the variable spam is a negative integer.**

   The assert statement is used to continue the execute if the given condition evaluates to True. If the assert condition evaluates to False, then it raises the AssertionError exception with the specified error message.

   ```
   import pyinputplus as pyip
   spam = pyip.inputNum(" Enter a positive number :")
   assert spam > 0
   print(spam,'is a positive number')
   ```

2. **Write an assert statement that triggers an AssertionError if the variables eggs and bacon contain strings that are the same as each other, even if their cases are different (that is, 'hello' and 'hello' are considered the same, and 'goodbye' and 'GOODbye' are also considered the same).**

   ```
   assert eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!'
   ```

   or

   ```
   assert eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!'
   ```

3. **Create an assert statement that throws an AssertionError every time.**

   assert False - this always triggers an exception

4. **What are the two lines that must be present in your software in order to call logging.debug()?**

   To be able to call logging.debug(), you must have these two lines at the start of your program:

   ```
   import logging
   logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -%(levelname)s - %(message)s')
   ```

5. **What are the two lines that your program must have in order to have logging.debug() send a logging message to a file named programLog.txt?**

   To be able to send logging messages to a file named programLog.txt with logging.debug(), you must have these two lines at the start of your program:

   ```
   import logging
   ```

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG,format=' %(asctime)s - %(levelname)s - %(message)s')

6. **What are the five levels of logging?**

The standard logging levels in Python (in increasing order of severity) and their applicability are:

- DEBUG - Detailed information, typically of interest when diagnosing problems.
- INFO - Confirmation of things working as expected.
- WARNING - Indication of something unexpected or a problem in the near future e.g. 'disk space low'.
- ERROR - A more serious problem due to which the program was unable to perform a function.
- CRITICAL - A serious error, indicating that the program itself may not be able to continue executing.

The default log level is WARNING, which means that only events of this level and above are logged by default.

7. **What line of code would you add to your software to disable all logging messages?**

```
import logging as lg
lg.disable(lg.CRITICAL)
```

8. **Why is using logging messages better than using print() to display the same message?**

As a best practice for logging in Python, we should use the built-in logging module. With the logging module, we can log different levels of messages to different destinations. We can also configure the log messages to be sent to some channels so that we can get notified when some errors occur. You can disable logging messages without removing the logging function calls. You can selectively disable lower-level logging messages and can create logging messages. Logging messages provides a timestamp.

9. **What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?**

- Step in: means that if there is a function call, it goes inside the function and you can see how the function is executing line by line till it returns and you go back to the next line right after the function call.
- Step over: means that if there is a function call, it just executes it like a black box and returns the result, but you cannot see how the function was executed.

- Step out: means that if you have stepped in a function and now you want to skip seeing how the rest of the function is going to execute, you Step out and the function returns. Then, you go back to the next line that is the line right after the function call.

10. After you click Continue, when will the debugger stop?

Continue execution and only stop when a breakpoint is encountered. Continue execution until the line with a number greater than the current one is reached. With a line number argument, continue execution until a line with a number greater or equal to that is reached.

11. What is the concept of a breakpoint?

Python breakpoint() is a new built-in function introduced in Python 3.7. Python code debugging has always been a painful process because of the tight coupling between the actual code and the debugging module code. For example, if you are using pdb debugger, then you will have to call pdb.set_trace() in your program code. If you want to use any other debugger, let's say web-pdb then you will have to remove all the code related to PDB and add web_pdb.set_trace() method. This causes a huge overhead in using python debugger and makes the python code hard to debug and maintain. That's why Python 3.7 has introduced breakpoint() method that allows us to write loosely coupled debugging code.

Python breakpoint() function calls sys.breakpointhook() function. By default, sys.breakpointhook() calls pdb.set_trace() function. So at the very least, using breakpoint() provide convenience in using a debugger because we don't have to explicitly import pdb module.