**FSDS MAY BATCH 2022(Python Basics 11)**

**Submitted by: Shubham Tiwari**

Q1. Create an assert statement that throws an AssertionError if the variable spam is a negative integer.

Ans: **assert spam >= 0, "Error: spam is a negative integer"**

Q2. 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).

Ans:

**assert eggs.lower() != bacon.lower(), "Error: eggs and bacon contain the same string"**

Q3. Create an assert statement that throws an AssertionError every time.

Ans**: assert False, "This is an intentionally triggered AssertionError"**

Q4. What are the two lines that must be present in your software in order to call logging.debug()?

Ans: **import logging**

**logging.basicConfig(level=logging.DEBUG)**

These two lines need to be present in your code to enable **logging.debug()** calls. The **import logging** line imports the Python logging module and the **logging.basicConfig(level=logging.DEBUG)** line sets the logging level to **DEBUG**. The logging module will only log messages with a severity level equal to or higher than the level specified in **basicConfig**. With the level set to **DEBUG**, all debug messages will be logged.

Q5. 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?

Ans :To send **logging.debug()** messages to a file named **programLog.txt**, these two lines are necessary:

**import logging**

**logging.basicConfig(filename='programLog.txt',level=logging.DEBUG, filemode='w')**

The **import logging** line imports the Python logging module. The **logging.basicConfig(filename='programLog.txt',level=logging.DEBUG, filemode='w')** line sets the logging configuration to write debug messages to the specified file. The **filemode='w'** argument tells **basicConfig** to open the file in write mode, which means that the file will be overwritten each time the program is run.

Q6. What are the five levels of logging?

Ans: The five levels of logging in Python's logging module, in increasing order of severity, are:

1. DEBUG
2. INFO
3. WARNING
4. ERROR
5. CRITICAL

Each logging level represents a different severity of an event that has occurred in the program. The logging module can be configured to log messages at specific levels. For example, if the logging level is set to **WARNING**, messages with level **WARNING**, **ERROR**, and **CRITICAL** will be logged, but messages with level **DEBUG** and **INFO** will be ignored.

Q7. What line of code would you add to your software to disable all logging messages?

Ans: To disable all logging messages in Python, you can set the logging level to **logging.CRITICAL** with the following line of code:

**logging.basicConfig(level=logging.CRITICAL)**

By setting the logging level to **CRITICAL**, only log messages with a severity level of **CRITICAL** will be processed, effectively disabling all other log messages.

Q8.Why is using logging messages better than using print() to display the same message?

Ans: Using logging messages is better than using print() statements for several reasons:

1. **Flexibility**: Logging messages can be easily configured to show different levels of verbosity and can be filtered based on severity, making it easier to debug issues in a production environment.
2. **Persistence**: Log messages are usually written to a file, making it easier to analyze logs over a longer period of time, whereas print statements disappear when the program ends.
3. **Structured data:** Log messages typically include additional information such as a timestamp and log level, making it easier to understand and analyze the log data.
4. **Separation of concerns:** Logging messages allow you to separate the process of generating messages from the process of consuming them, making it easier to change how log messages are handled without affecting the code that generates them.
5. **Centralized control:** Centralizing log messages in a single place makes it easier to control and manage log messages, as opposed to having print statements scattered throughout the codebase.

Q9. What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?

Ans: In a Python debugger, the "Step Over", "Step In", and "Step Out" buttons are used to control the execution flow of your code while debugging.

1. Step Over (F10): This button allows you to execute the next line of code without entering any function calls. If the next line of code is a function call, the debugger will skip over it and move on to the next line.
2. Step In (F11): This button allows you to enter a function call and step through its code line by line. The debugger will pause execution at the first line of the function and allow you to step through its code.
3. Step Out (Shift + F11): This button allows you to step out of the current function and return to the calling function. The debugger will execute the remaining lines of code in the current function and then pause execution at the next line in the calling function.

These buttons can be used in combination to help you understand the flow of execution in your code and identify the source of any bugs.

Q10. After you click Continue, when will the debugger stop ?

Ans: The debugger will stop after clicking the "Continue" button in Python when it encounters one of the following conditions:

1. A breakpoint: If a breakpoint has been set in our code, the debugger will stop when it reaches the line of code with the breakpoint.
2. An exception: If an exception occurs, the debugger will stop at the line of code where the exception was raised and allow us to inspect the state of your program.
3. The end of the program: If there are no breakpoints or exceptions, the debugger will stop when it reaches the end of the program and the program will exit.

In all cases, the debugger provides you with a view into the state of our program and allows us to inspect variables, call stack, and other information that can help us identify and fix bugs in our code.

Q11. What is the concept of a breakpoint?

Ans: A breakpoint is a mechanism in a debugger that allows you to pause the execution of your program at a specific line of code. When the debugger reaches a breakpoint, it will stop the execution of the program and allow us to inspect the state of our program, including the values of variables, the call stack, and other information that can help us identify and fix bugs in our code.

In Python, we can set a breakpoint by clicking on the line number in the editor or by adding the line **import pdb; pdb.set\_trace()** in our code. When the debugger reaches the line with the breakpoint, it will enter the Python debugger (pdb) and allow you to inspect and debug your code.

Breakpoints are a powerful tool in the debugging process, as they allow us to isolate a specific section of code and focus on understanding and fixing issues in that section. Additionally, breakpoints can be disabled or removed, which makes it easy to switch between different debugging scenarios and saves time in the debugging process.