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

Sol.

```
spam=int(input("enter an integer ")
If spam>=0:
    Print(f"the no. is {spam}")
else:
    raise AssertionError("this is assertion error")
```

2. Create an assert statement that throws an AssertionError if the variables eggs and bacon contain strings that are the same, even if their cases are different (for example, 'hello' and 'hello' are considered the same, as are 'goodbye' and 'GOODbye').

Sol.

```
bacon=input("enter the bacon value")
eggs=input("enter the egg value")
if bacon.lower() != eggs.lower():
        print(f" bacon is {bacon} and eggs is {eggs}")
elif bacon.lower() == eggs.lower():
     raise AssertionError("this is assertion error")
```

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

Sol.

```
bacon=input("enter the bacon value")
eggs=input("enter the egg value")
if bacon.lower() != eggs.lower():
        raise AssertionError("this is assertion error")
elif bacon.lower() == eggs.lower():
     raise AssertionError("this is assertion error")
print("this is important")
```

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

Sol. import logging

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

5. What are the two lines that must be present in your software in order for it to log?

Is it possible for logging.debug() to log a message to a file called programLog.txt?

Sol. import logging

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG,

format=' %(asctime)s - %(levelname)s - %(message)s')

6. What are the five stages of logging?

Sol. DEBUG, INFO, WARNING, ERROR and CRITICAL

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

Sol. logging.disable(logging.CRITICAL)

8. Why is it easier to use logging messages instead of print() to show the same message?

Sol. The Python logging library lets you:

°Control what's emitted

°Define what types of information you want to include in your logs

°Configure how it looks when it's emitted

°Most importantly, set the destination for your logs

9. What is the difference between the debugger's Step Over, Step In, and Step Out buttons?

Sol. The Step in button will move the debugger into a function call.

The Over button will quickly execute the function call without stepping into it.

The Out button will quickly execute the rest of the code until it steps out of the function it currently is in.

10.When will the debugger stop after you press Continue?

Sol. Continue starts your program again, and it will then run until it hits a break point or finishes.

11. What is the concept of a breakpoint?

Sol. Using breakpoint() function: In this method, we simply introduce the breakpoint where you have doubt or somewhere you want to check for bugs or errors.

12. In Mu, how do you place a breakpoint on a line of code?

Sol. right-click the line and select Set Breakpoint from the context menu.