Chapter-11

Debugging

Practice Questions

In [2]:
```
import logging, traceback
```

1. Write an assert statement that triggers an AssertionError if the variable spam is an integer less than 10.

In [14]:
```
spam =  int(input())

assert spam >= 10
```

```
10
```

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).

In [16]:
```
eggs = 'hello'

bacon = 'Hell0'

assert(eggs.lower() != bacon.lower())

eggs = 'goodbye'

bacon = 'GOODbye'

assert(eggs.upper() == bacon.upper())
```

3. Write an assert statement that always triggers an AssertionError.

In [26]:
```
n = int(input())
for i in range(n):
    assert i > i+1
```

```
5

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-26-846fbbef198e> in <module>
      1 n = int(input())
      2 for i in range(n):
----> 3     assert i > i+1

AssertionError:
```

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

In [20]:
```
import logging

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

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?

In [22]:
```
import logging

logging.basicConfig(filename='programLog.txt', level=logging.DEBUG, format='%(asc
```

6. What are the five logging levels?

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

7. What line of code can you add to disable all logging messages in your program?

In [23]:
```
logging.disable()
```

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

> The main thing about log messages is that you're free to fill your program with as many as you like, and you can always disable them later by adding a single logging.disable()

> Unlike print(), the logging module makes it easy to switch between showing and hiding log messages.

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

> ***step In*** : Clicking the Step In button will cause the debugger to execute the next line of code and then pause again

> ***step out***: Clicking the Step Out button will cause the debugger to execute lines of code at full speed until it returns from the current function

> ***Step Over***: Clicking the step Over button to execute the first print() call

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

> After clicking continue the program execute normally until it reaches the break point

11. What is a breakpoint?

> A breakpoint is a specific line of code and forces the debugger to stop at that point when the debuggrer arrivess at that point

12. How do you set a breakpoint on a line of code in Mu?

> To set a breakpoint, click the line number in the file editor to cause a red dot to appear, marking the breakpoint.