- 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.
```

In [1]:
```
!pip install pyinputplus
```

```
Requirement already satisfied: pyinputplus in c:\users\rosha\anaconda3\lib\site-packages (0.2.12)
Requirement already satisfied: pysimplevalidate>=0.2.7 in c:\users\rosha\anaconda3\lib\site-packages (from pyinputplus) (0.2.12)
Requirement already satisfied: stdiomask>=0.0.3 in c:\users\rosha\anaconda3\lib\site-packages (from pyinputplus) (0.0.6)

WARNING: Ignoring invalid distribution -ython (c:\users\rosha\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ython (c:\users\rosha\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ython (c:\users\rosha\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ython (c:\users\rosha\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ython (c:\users\rosha\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -ython (c:\users\rosha\anaconda3\lib\site-packages)
```

In [4]:
```
import pyinputplus as pyip

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

```
Enter a positive number : -5

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_492/3943296391.py in <module>
      2
      3 spam = pyip.inputNum("Enter a positive number : ")
----> 4 assert spam > 0
      5 print(spam, 'is a positive number')

AssertionError:
```

- 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 [5]:
```
eggs='Hello'
bacon ='hello'

assert eggs.lower() != bacon.lower() or eggs.upper() != bacon.upper()
print('The eggs and bacon variables are not the same!')
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_492/2374175613.py in <module>
      2 bacon ='hello'
      3
----> 4 assert eggs.lower() != bacon.lower() or eggs.upper() != bacon.upper()
      5 print('The eggs and bacon variables are not the same!')

AssertionError:
```

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

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

In [6]:
```
assert False
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_492/2103537015.py in <module>
----> 1 assert False

AssertionError:
```

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

In [ ]:
```
import logging as lg
lg.basicConfig(level=lg.DEBUG, format=' %(asctime)s - %(levelname)s - %(message)s')
```

- 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 [ ]:
```
import logging as lg
lg.basicConfig(filename='programLog.txt',level=lg.DEBUG, format=' %(asctime)s - %(levelname)s - %(message)s')
```

- What are the five levels of logging?

```
Five level of loggins are DEBUG, INFO, WARNING, ERROR, and CRITICAL.

logging.debug() - variable's state and small details

logging.info() - general events, confirm a program is working

logging.warning() - potiental problem to work on in the future

logging.error() - record an error that caused program to fail to do something

logging.critical() - fatal error that has caused
```

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

In [ ]:
```
import logging as lg
lg.disable(lg.CRITICAL)
```

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

```
You can disable logging messages without removing the logging function calls.

You can selectively disable lower-level logging messages.

You can create logging messages. Logging messages provides a timestamp.
```

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

```
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.
```

- After you click Continue, when will the debugger stop

```
It will stops at next breakpoint, if there are no further breakpoints program will be fully executed.
```

- What is the concept of a breakpoint?

```
A breakpoint is an intentional stopping point or pause put into a program for debugging purposes.
```

In [ ]:
```

```