Programming For Cybersecurity
Lab Topic 10- errors, testing and logging

Introduction.

In this lab we are going to make a function called Fibonacci in a module called myFunctions.
The Fibonacci function should take in a number and return a list containing a Fibonacci sequence of that many numbers.
A Fibonacci sequence is a series of numbers that starts off with 0 and 1 and each subsequence number is the sum of the previous two.
E.g. if we called the function with the number 7, it would generate a list of a Fibonacci sequence of 7 numbers.
[0, 1 ,1, 2, 3, 5, 8]
If the user enters in a number less than 0 we should raise a ValueError.
If the user enters 0 it should return an empty list.

We will write the test cases before we write the function body.

I would suggest that you create another folder in labs called **Topic09-errors**.

Set up and tests

1. Let's say that you have been asked to a function called Fibonacci that takes a number and returns a list containing a Fibonacci sequence of that many numbers. The function will be in a module called myfunctions.py
2. Create a file called myFunctions.py and define the function fibonacci that takes in a number, for the moment let it just return an empty list

```
def fibonacci(number):
    return []
```

3. Create the section at the bottom of the file that will contain the testing code. (check the code by running it, it should output all good)

```
if __name__ == '__main__':
    # tests will go here
    print("all good")
```

4. Write some test-cases that test that the function called with the parameters 7, 11, 0 and 1, all return the correct values

```
return7 = [0,1,1,2,3,5,8]
return11 = [0,1,1,2,3,5,8,13,21,34,55]
assert fibonacci(7) == return7, 'incorrect return for 7'
assert fibonacci(11) == return11, 'incorrect return for 11'
assert fibonacci(0) == [], 'incorrect return value for 0'
assert fibonacci(1) == [0], 'incorrect return value for 1'
```

5. This program should return an assertion error if it is run because we have not written the body of the function yet.

```
Traceback (most recent call last):
  File ".\myfunctions.py", line 27, in <module>
    assert fibonacci(7) == return7, 'incorrect return value for 7'
```

6. While we are here let's write the code to test that the function will return an ValueError if a number less than 0 is passed in.

```
try:
   fibonacci(-1)
except ValueError:
    # we want this exception to be thrown
    # so this is an example where we want to do nothing
    pass
else:
    # if the exception was not thrown we should throw
    # Assertion error
    assert False, 'fibonacci missing ValueError'
    # or
    #raise AssertionError("fibonacci no valueError ")
```

We can now start writing the code for the fibonacci function

7. First let's write the "happy path" code, when we run this it will throw an assert error for 0. (I got an error so I added some debug logging)

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

```
def fibonacci(number):
    a = 0
    b = 1
    fibonacciSequence = [0]
    # we have one in the list already so number - 1 times
    # this is not the most efficient code
    # could have used yield
    for i in range(1,number):
        fibonacciSequence.append(b)
        # this is funky code make a = b and b = a + b
        a , b = b, a + b
    logging.debug("%d: %s",number, fibonacciSequence)
```

8. Add the code in case the user enters 0, (this will throw an assert error for not throwing the valueError, when run)

```
if number == 0:
        return []
```

9. Add the code to throw an ValueError if the number is less than 0 (it should all work now)

```
if number < 0:
        raise ValueError('number must be > 0')
```

Using this module

10. Write a program called **useFib.py** that uses this module to prompt a user for a number and will return the list of the Fibonacci sequence, (remember to turn off the debug level in the module by commenting out the line logging.basicConfig)

In myFunctions.py

```
#logging.basicConfig(level=logging.DEBUG)
```

in useFib.py

```
# This program prompts the user for a number and
# Prints out the fibonacci sequence of that many numbers
import myFunctions

nTimes = int(input('how many:'))
print(myFunctions.fibonacci(nTimes))
```

Output

```
$ python .\useFib.py
how many:34
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657,
46368, 75025, 121393, 196418, 317811, 514229, 832040,
1346269, 2178309, 3524578]
```