FibLog

CSCI 112, Lab 1

File names: Names of files, functions, and variables, when specified, must be EXACTLY as specified. This includes simple mistakes such as capitalization.

Individual work: All work must be your own. Do not share code with anyone other than the instructor and teaching assistants. This includes looking over shoulders at screens with the code open. You may discuss ideas, algorithms, approaches, *etc.* with other students but NEVER actual code. Do not use code written by anyone else, in the class or from the internet.

Documentation: Each file should begin with a docstring that includes your name, the class number and name, the lab number, and a short description of the lab, as well as documentation pertinent to that particular file.

Matrix: Write a `Matrix` class, in a module `matrix.py`. The initialization should take the number of rows and columns, and then an iterable of length `rows * cols`. It should implement at least the following methods:

- `__init__`
- `__str__`
- `__mul__`
- `__pow__`

Which should enable the following interactions:

```
>>> from matrix import Matrix
>>> m1 = Matrix(3, 4, range(3*4))
>>> print(m1)
 0 1 2 3
 4 5 6 7
 8 9 10 11
>>> print(m1 * Matrix(4,3,range(4*3)))
 42 48 54
 114 136 158
 186 224 262
>>> m2 = Matrix(2, 2, [0,1,1,1])
>>> print(m2)
 0 1
 1 1
>>> print(m2 * m2)
 1 1
 1 2
>>> print(m2 ** 2)
 1 1
 1 2
>>> m3 = Matrix(2, 2, 'abcd')
>>> print(m3)
 a b
 c d
```

Unit test: Implement a unit test module for your matrix class. Call the unit test module `matrix_test.py`.

Fibonacci: In a module named `fibonacci.py`, implement the Fibonacci function three different ways. One will execute in exponential time, $O(2^n)$, one in linear time, $O(n)$, and one in log time $O(\log n)$. Mathematically, the fibonacci function is defined as follows. defined as

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{otherwise} \end{cases}$$

The log time algorithm should be based on calculating the powers of the matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Unit test: Implement a unit test module for your Fibonacci functions. Call the unit test module `fibonacci_test.py`.

Timing: Finally, implement a module called `fibonacci_timing.py` to test the runtime of your three fibonacci functions. Use the `time.process_time_ns` function to get more accurate timings, in nanoseconds. Design tests to show the different times of the different functions and illustrate the differences between $O(2^n)$, $O(n)$, and $O(\log n)$.

Writeup: Write a short paper discussing your findings from the timing experiments, and why they support (or don't!) our order of magnitude estimates of their runtimes.

This writeup must be in either a plain text or a LaTeX document. LaTeXis optional, but I highly recommend that you learn to use it as soon as possible. There are many online tutorials, and *all* of my lab notes and lectures are written in LaTeX! A good place to get started in LaTeX is `https://www.overleaf.com/`.

Put this writeup in a file called either `lab01writeup.txt` or `lab01writeup.tex`

If you use LaTeX you may leave in place the miscellaneous files produced on compilation to pdf format.

Turn in: Put the following files into a folder called `lab01csci112`, zip it into a single compressed file and submit to canvas:

- `matrix.py`
- `matrix_test.py`
- `fibonacci.py`
- `fibonacci_test.py`
- `fibonacci_timing.py`
- Either:
  - `lab01writeup.txt`
  - `lab01writeup.tex`