We've done a few scattered programming exercises in previous homework assignments, but today we start our cumulative project work in earnest. As the course web site explains, the project is due at the end of the term. But do not put it off. Do today's project work today, so that you better understand upcoming classes and exams.

Your code will be tested against various unit tests and integration tests. Therefore your code must exactly conform to the specifications. For example, if you put the `cnot` gate into qUtilities.py instead of qConstants.py, then the tests fail. If you rename the function `first` to `measureFirst`, then the tests fail. If you return 0 where a specification asks you to return $|0\rangle$, then the tests fail. Sorry, but there is no room for creativity in the *interfaces.*

On the bright side, there is room for creativity in the *implementations.* For example, in later assignments you might want to add helper functions, that I haven't requested. Do that whenever you want, as long as it doesn't interfere with my specifications.

Your code should be clean. Ideally it would explain itself through the use of meaningful abstractions and identifiers. For example, if you want to return $|0\rangle$, then use `qc.ket0` rather than manually typing `numpy.array([1 + 0j, 0 + 0j])`. If you want to apply a gate to a state, then use `qg.application` rather than manually typing NumPy code. Insert comments to explain code that isn't self-explanatory.

A. Download qUtilities.py again from the course web site. I have added an `equal` function. It consists of an interface, a "doc string" that specifies the function's intended behavior, and several lines of implementation. Read the specification. Pay attention to the warnings.

B. To qConstants.py, add constant 4×4 matrices `swap` and `cnot`. As we will learn shortly in class, `cnot` is

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

C. Download qGates.py from the course web site. In it, find the `application` function. Currently the implementation is simply the keyword `pass`, which does nothing. Implement the function by replacing `pass` with a single line of NumPy code. (By the way, in the future we will generalize many of these functions to handle states and gates of any number of qbits.)

D. In qGates.py, implement the `tensor` function. By the way, `numpy.kron` does most or all of this task for you. You are welcome to test your code against `numpy.kron`, but your final code should not use `numpy.kron` (because the goal of this exercise is to help you learn how to compute tensor products).

E. In qGates.py, there is a `main` function that runs simple tests on your code. Run qGates.py to activate the tests. If your code fails the tests, then fix your code.

F. Download qMeasurement.py from the course web site. Implement the `first` function. This task is not trivial. You need to understand the math of partial measurement, including the special cases where $\sigma_0 = 0$ or $\sigma_1 = 0$, and translate it into Python.

G. In qMeasurement.py, implement the `last` function. Probably you will need to first work on paper, mimicking the math of how to measure the first qbit, to figure out the math of measuring the second qbit. So this task is definitely not trivial.

H. In qMeasurement.py, there is `main` function that runs tests on your code. Run qMeasurement.py to activate the tests. If your code fails the tests, then fix your code.