Lab 15

Learning Objectives:

I. **What characters do I type next?** *How do I utilize Cython and C code to speed up my code?* This lab will give you the opportunity to practice profiling your code and then optimizing the specific parts that will give you the best speedup.

C. **Computing time is cheap—use it.** *Except when it's not.* This lab will give you the opportunity to practice optimizing code that really does need it.

Part 1 asks you to understand and profile a piece of code.
Part 2 asks you to rewrite it in Cython.
Part 3, if you have time, asks you to rewrite it in C, interfaced through Cython.

Part 1: Profiling a Python-only Game of Life

We've written a Python-only version of the Game of Life, and it's pretty slow. However, some parts are slower than others, and there's no part optimizing the fast parts. Hence, **your first task is to profile the Game of Life** written in pure Python code.

Take a look at the file `life_py.py`. Compare this code against the algorithm from lecture and convince yourself that it is doing the right thing. **Note that the for loop over `range(1, N-1)` omits the boundaries.** Try running and plotting it using e.g.

```
life.life(N=40, plotting=True, version="python")
```.

Now try profiling the Python version using cProfile! Read the documentation for `life` to figure out how to use it. Try different choices of N and see if the speed scales as expected. Try also using `%timeit` and see how much the profiler slows down the code. **You should probably set plotting to `False`** because the plotting over the ssh X-server takes a long time and outweighs the actual execution time.

Part 2: Life in Cython

In this part, you'll rewrite the `update` function in Cython and observe the speedup that comes with this simple change in the code. In order to keep the old Python version, write this new function in the file `life_cy.pyx`. All function calls and definitions have already been written; all you have to do is fill in the body for `update_cy` and `getNeighbors`. Make use of all of the tricks that you learned about in lecture to maximize the performance of those two functions. They won't look very different from the old Python versions, but you should be able to get at least a factor of 5 speedup with a few modifications.

Once you have written the two functions, first run the simulation in `"cython"` mode with plotting set to True. Does it behave correctly? If so, compare the performance when version is set to `"python"` to when it is set to `"cython"`. What speedup do you get?

Part 3: If You Have Time, Life in C

Now it is time to maximize the performance of the *Life* simulation. **Your task it to implement Life in C.**

Notice that the Cython file `life_cy.pyx` is set up to call a pure C function, called `update_pure_c`, through the wrapper function `update_c`. Go to the file `life.c` and implement the `update_pure_c` and `getNeighbor` functions in pure C.

Once you have written the two functions, first run the simulation in "c" mode with plotting. Does it behave correctly? If so, go ahead and compare the performance of all three versions. By writing straightforward C code, you should be able to get a relative speedup of over 100 over the pure Python version.