You only have to do one of following: 2F (this assignment) or 3M.

PIC 16, Winter 2018 – Assignment 2F

Assigned 1/19/2018. Code (a single `.py` file) due by the end of class 1/24/2018 on CCLE. Hand in a printout of this document with the "Questions" portion filled out by the end of class on 1/24/2018.

In this assignment, you will run an experiment to determine the relative efficiency of `map`, list comprehension, and `for` loops while performing an identical task with each strategy. Based on the results, you will develop a set of guidelines on when to use each. You may find the Quick Guide to Python Performance helpful.

Task

Consider the following code to generate an array of the squares of N integers:

```
def f(x):
    return x**2
x = range(N)
y = []
for i in x:
    y.append(f(i))
```

First, for a large integer `N`, measure the execution time. Be sure to measure only the execution time of the `for` loop – the part that is actually doing the work.

Next, write a lambda function `g` that accomplishes the same task as `f`.

Then, write additional code to time the execution of `for` loops that do the same job, except in one case use `g` and in the other case do not use `g` or `f` (just write out the `**2` operation inside the loop).

You should find that the code without a function (or lambda function) call is significantly faster. This is because it takes time to call a function in addition to executing its contents. (This is called "function call overhead.")

Test three more cases:

- Rather than appending to the list `y`, first initialize `y` to `range(N)` then square the elements in-place. That is, is it more efficient to re-use an existing list (if possible)?
- Use list comprehension to generate the list of squares
- Use `map` to generate the list of squares

Making modifications as needed to test hypotheses, answer the questions below. Base your answers only on consistent findings, not one-time tests. *These questions require some thought.* For instance, it doesn't make sense to answer NO to question 1 on the basis that your `for` loop that appends `i**2` to an empty array is faster than the `for` loop that uses `f(i)` to modify an existing array. If both approaches are possible, then there is no reason not to combine the best of both, that is, write a `for` loop that uses `i**2` to modify an existing array. You may briefly justify your answers on the back if desired.

Questions:

1. When it is possible to re-use an existing list in a `for` loop, is this faster than appending to an empty one?
2. Is it faster to use list comprehension than a `for` loop when the operation you need to perform requires a single function call?
3. Under what circumstances might it be faster to use a `for` loop than list comprehension?
4. Is it faster to use `map` than list comprehension when the operation you need to perform requires a single function call?
5. Under what circumstances might it be faster to use list comprehension than `map`?
6. Is a lambda function faster than a regular function?