COMPSCI 260, Introduction to Computational Genomics **27 August 2019**

Problem Set 0

Due: 5pm on 6 September 2019

Introduction

The purpose of Problem Set 0 is two-fold: first, to have you reflect carefully on why you are taking this class and what you are hoping to get out of it, and second, to familiarize yourself with the general layout of COMPSCI 260 problem sets and the submission procedures we will follow this semester.

In this "toy" problem set, you will write a small piece of Python code and then answer some introductory questions in a `README` file. These files will then be submitted to Gradescope, where you will be able to see what our TAs and UTAs see when grading your work, and you will also get to see some of the output of your code. Note that when you submit, we will generally *not* tell you if your output is actually correct: that is for you to reason about and ascertain for yourself.

In later problem sets, problems are often complex and multifaceted. It will be helpful to outline your high-level solution clearly on paper before you start to develop your programs and write code. Once your strategy is clear, develop your code piece-by-piece and step-by-step. You can check that each piece is working correctly by printing some intermediate results to the screen. Be sure to test your programs thoroughly before submitting to convince yourself they are working as desired.

We want you to think through new and difficult concepts and experience the deep satisfaction of figuring them out for yourself, but our goal is not for you to feel frustrated for hours. Please post questions to Piazza, or come to office hours if you need more hands-on help.

Organization of problems

Many problems throughout the semester will be organized in a **plan** → **develop** → **check** → **reflect** cycle. This cycle is meant to match the steps you would take if you were to approach a similar problem in the real world. One immediate inference you can make from this cycle is that the course is not just about developing code; it's about using code to solve a real problem and then reflecting on the consequences and implications.

During the **plan** step, you will generally be given some background information about the problem and answer some guiding questions intended to clarify your thinking or inform your decisions in later parts of the problem.

The **develop** step is where you will develop algorithms and implement them as functions in Python. Please be sure to comment your code clearly to explain your logic: It is hard for us to give partial credit if we cannot determine your thought process. If a part of the problem only requires you to develop code *and* your code is well-documented, you do not need to provide any additional description of your code in your `README`.

The **check** step will generally require you to run your code on some input and examine its output. Naturally, you may choose to print out some raw diagnostic info as you develop your code, but when it is complete and ready to submit, your code will be expected to *report* its results clearly, in a human-readable format that facilitates interpretation. The details and importance of interpretable reporting are addressed in greater detail below.

The **reflect** step will contain some final questions asking you to think about your results in a larger context. This section will require some critical thinking about what your results mean and why they are what they are. Reflection questions are often open-ended, and we are *not* looking for specific elements in your response; instead, we are looking for clarity, depth, insight, and mastery of concepts. How we (subjectively) grade these responses is addressed in greater detail in the next section.

As an aside, you will occasionally encounter a problem or task labeled as an **extra challenge**. These are included for the intellectually curious and are not graded. Feel free to spend time on them if you've finished the assigned problems and are looking for, well, an extra challenge.

Subjective problems and grading

Within **plan** and **reflect** steps, you will often encounter open-ended questions (typically italicized). Your responses to these questions will be graded subjectively, unlike certain other questions where your answers may be deemed either correct or incorrect.

When grading subjective problems, we will generally use the following labels for your responses:

- Excellent: demonstrates clarity, depth, insight, and conceptual mastery (among the best in the class)
- Good: completely correct and thoughtfully engaged (likely in the top half of the class)
- Acceptable: completely correct but not as engaged, or mostly correct but with some small errors
- Incorrect or Missing: incorrect, trivial, or missing entirely; these will be assigned 0 points

Your responses to open-ended questions should be clear, well-organized, and easy to follow, yet engaged with the problem in a deep way that indicates you have reflected on its consequences and implications. Note: conceptual mastery is often reflected in an ability to say the most important things precisely yet succinctly, meaning that an excessively long response is often not an excellent response.

The *report* keyword

We will use **report** as a keyword to denote places your code should print the requested output to the screen in a way that is friendly to and useful for a human reader. This is an important practice to develop any time you expect your code to be used by human beings (including yourself and your human graders in this course). Friendly and useful reporting of output requires more than just dumping out a value. For instance, consider the relative usefulness of these two outputs:

```
$ 643981

$ The assembled bacterial genome sequence is 643981 nucleotides long.
```

Similarly, if an output of your code is the aforementioned genome sequence, it might be useful to print the entire sequence to a file in a specified interchange format, but it would not be a good idea to print it to the console. However, it might be useful to print out some sort of simple summary like this (of course, it's up to you to decide what kind of output would useful in the specific context of the problem):

```
$ Assembled bacterial genome sequence (643981 nucleotides):

  5' -- 1 AGACTACGTCGATATCGATATCGCGGCATATCGC ...  ACGCGCGATCAGAAA 643981 -- 3'
```

Any time you see the **report** keyword, you should include your human-interpretable output in your `README`. Please include the entire output unless specifically noted in the problem to do otherwise.

Submitting your work to Gradescope

To organize our grading, you will submit each individual problem as a separate 'Programming Assignment' on Gradescope. For example, Problem Set 1 will require three separate submissions for: `PS1 Problem 1`, `PS1 Problem 2`, and `PS1 Problem 3`. Each problem will often be a discrete, independent assignment, but occasionally we will ask you to use code and build upon work from a previous problem.

When submitting to Gradescope, you can either upload individual files or a single `.zip` file containing them. Unless we specify otherwise, we expect you to submit only your Python code and the `README` file for the problem (plus any additional files like images, diagrams, or other attachments you explicitly reference in your `README`: we won't know to look for them unless you mention it). This means you should omit some of the files we provide in the problem set that remain unmodified, such as the assignment PDF, `compsci260lib.py`, .fasta sequence files, and other reference material.

After you are done with the entire problem set, you will complete a special `FLAG.txt` file which you must include as part of your submission of whatever is the *last* problem. In `FLAG.txt`, you will provide summary information about the whole problem set: your name, NetID, the estimated number of hours you spent on the full problem set (must be a single integer, but can be your best guess), and an explicit statement that you have abided by the Duke Community Standard. Specifically, you will be required to assert that you neither received nor gave help on this problem set, or otherwise to cite any help that you received or gave.

Gradescope autograding

After submission, most of the code you develop will be run through an autograder. For this reason, make sure to carefully follow all instructions regarding file and function names found in the PDF or skeleton code we provide. Upon submission, you will receive some simple validation feedback from the autograder, but as mentioned above, this basic validation will not represent the full extent of the autograding.

The initial validation will check that: you have submitted all of the required files, your `FLAG.txt` (for the last problem) is properly formatted, and the return values of functions we ask you to implement are properly formatted.

Additional autograder tests that evaluate code correctness will not be visible to you until after the grades have been released.

Problem 1: In the beginning... (25 points)

This "toy" problem set has only one problem: fill out a pre-course survey, in the form of a short Python programming assignment and a follow-up reflection. This problem is structured to give you an initial exposure to the **plan** → **develop** → **check** → **reflect** cycle, as well as to familiarize you with submitting to Gradescope and the autograder.

When you are done, you should submit the following files to Gradescope before the submission deadline:

1. `survey.py`
2. `README.problem1.txt`
3. `FLAG.txt`

Successful on-time completion of PS0 will amount to 3% of your final course grade. Future problem sets are graded on a scale of 0–100, and are each worth 12% of your final course grade, so you can think of this problem as being worth 25 points. Unlike later problem sets, PS0 is graded all-or-nothing.

Plan

You will ultimately be answering the following four questions within `README.problem1.txt`. The answers to the first three questions must be **reported** by your code, meaning that you will copy the (human-readable) output of your code into your `README` in order to answer them. When that's done, you will add your response to the fourth question to your `README` as a reflection.

- What is the highest Duke COMPSCI course number you have taken? This should be an integer, like 101, 201, 230, 250, etc. If this is your first Duke COMPSCI course, please answer 0.
- On a scale of 0-3, how much programming experience have you had in Python?

  0 = I have not used Python before, but can confidently program in another language.

  1 = I have used Python in one context (like COMPSCI 101) but have little other experience.

  2 = I have programmed in Python in multiple contexts.

  3 = I'm a Python ninja.
- What Python editor will you be using this semester? (Most likely answer will be PyCharm.)
- What are you hoping to gain by taking this class? How will you achieve it? Why?

Develop

a) Within `survey.py`, write a function called `make_survey_response_dict` that takes as input three parameters and returns a dictionary with those values respectively assigned to three keys: `course`, `python`, and `editor`. If you call this function with your answers to the first three questions above (an integer, an integer in the set $\{0, 1, 2, 3\}$, and a string), the returned dictionary should look something like this:

```
{
    'course': 101,
    'python': 1,
    'editor': "PyCharm"
}
```

Check

b) **Report** your answers to the first three survey questions.

[Just to be clear, this means: write code that first calls the `make_survey_response_dict` function with your answers, and then takes the resulting dictionary and uses it to print human-readable sentences to the console. As always, you should copy this console output into your `README`. This is just a toy example, but reflects what we mean when we ask you to **report** something throughout the semester.]

Reflect

c) *What are you hoping to gain by taking this class? How will you achieve it? Why?*

[Just to be clear, this means: write a response to this open-ended question into your `README`.]