Expectations for Submitted Assignments

In this short notebook, we'll discuss expectations for submitted assignments, including homework and in-class assignments. The same expectations will apply to the midterm and final exam.

Submission Format

We recommend that you complete most of your homework assignments in Jupyter Notebooks. Most of you will find this to be both easier and more fun. It's also easy for us to grade. There may be occasions in which we encourage you to use other submission formats instead; however, these will be rare.

As a backup, you may instead choose to do your homework in `.py` files (i.e. Python script files). This is discouraged unless (a) you are having issues running Jupyter or (b) the Jupyter interface is not accessible to you. **You must notify me in advance** if you intend to submit `.py` files for your homework assignments or exams. You do not need to state your reason for doing so, but you do need to tell me ahead of time.

Most of the expectations described below will apply equally to both notebook-based assignments and `.py`-based assignments.

Do Not Submit `.ipynb` Files

A Jupyter Notebook is natively a file of format `.ipynb`. **Do not submit `.ipynb` files. We will not grade `.ipynb` files.**

Instead, you must convert your notebook into a `.pdf`. There are two ways:

1. In the Jupyter `File` menu, choose `File > Download As > PDF via LaTeX (.pdf)`. This method may not work for you, depending on whether you have LaTeX installed and available to Jupyter.
2. In your browser's `File` menu, choose `Print`. Then, choose `Save as PDF`.

In either case, submit the resulting `.pdf` on CCLE.

Header Material

The top of your assignment should include the homework assignment number (e.g. "HW4"), your name, and the date. It should also include the honor statement shown below.

Expository Text

Your notebook should include expository text that (a) states the problem you are solving and (b) describes the operation of each code chunk. We will provide you with an outline of each homework assignment that includes the problem statements, so if you use the outline then that will satisfy (a). An example of (b) is shown below.

Comments and Docstrings

Your code should include comments and docstrings.

Comments (marked in Python with a `#`) explain the intention and operation of small lines or blocks of code.

Docstrings give a detailed explanation of the inputs and outputs of a function. Docstrings are typically placed directly below the first line of a function definition. They are placed between three pairs of quotation marks `"""like this"""`. You won't need to worry about docstrings until we discuss defining functions.

HW5: Example

- *By Michael Perlmutter*
- Jan 4th, 2021

> I affirm that I personally wrote the text, code, and comments in this homework assignment.

Problem 1

Write a function called `say_hello()` that repeatedly prints the phrase "Hello Python!". The function should take as an argument the number of times to print this phrase. It is not necessary for the function to return anything. For example:

```
say_hello(5)

Hello Python!     # output
Hello Python!
Hello Python!
Hello Python!
Hello Python!
```

In [3]:
```
def say_hello(k):
    """
    print the phrase "Hello Python!" a number of times specified by the user.

    Parameters
    ----------
    k: int, the number of times to print the phrase "to boldly go"

    Return
    ----------
    No return value
    """

    # I'll use a for loop to print repeatedly
    for i in range(k):
        print("Hello Python!")
```

In [4]:
```
say_hello(5)
# ---
```

```
Hello Python!
Hello Python!
Hello Python!
Hello Python!
Hello Python!
```

Problem 2

Write a function that returns a Collatz sequence (https://en.wikipedia.org/wiki/Collatz_conjecture) starting with a user-specified integer. The sequence should be returned as an `np.array()`. The first entry should always be the user input, while the last entry should always be 1. For example:

```
collatz(17)

# output
array([17., 52., 26., 13., 40., 20., 10.,  5., 16.,  8.,  4.,  2.,  1.])
```

I'll approach this problem by:

1. Initializing an empty `array()` to hold the sequence.
2. Looping and updating the sequence according to the rules described here (https://en.wikipedia.org/wiki/Collatz_conjecture) until the sequence equals one. In each iteration, I'll add the current term to the sequence.
3. Then, I'll return the sequence.

In [3]:
```
import numpy as np

def collatz(k0):
    """
    Parameters
    ----------
    k: int, the initial integer at which to initiate the sequence

    Return
    ----------
    np.array(), an array of ints
    """

    c = np.array([])        # initialize an empty array
    k = k0
    c = np.append(c, k)     # not forgetting the initial value!

    while k > 1:            # loop until the sequence reaches 1
        if k % 2 == 0:      # even case
            k = k/2
        else:               # odd case
            k = 3*k+1
        c = np.append(c, k) # add result to c
    return c
```

In [4]:
```
collatz(17) # test, looks ok!
```

Out[4]:
```
array([17., 52., 26., 13., 40., 20., 10.,  5., 16.,  8.,  4.,  2.,  1.])
```