CS100M: Section Exercises for Sep. 13-14

1. Write a MATLAB script to print the first n Fibonacci numbers. Remember that the Fibonacci numbers are defined as $F_n = F_{n-1} + F_{n-2}$ with $F_1 = 1$ and $F_2 = 1$. Notice that to calculate any F_n, you know need to know the two previous Fibonacci numbers—you do not need to keep track of the entire sequence at any time. Use *scalar* variables only. A scalar is a variable that stores a single value at one time. Your script will begin with the following statements:

```
n= input('Input n: ');
value1= 1;
value2= 1;
```

2. Write a MATLAB script to print the numbers $F_n, F_n+1, F_n+2, \ldots, F_{n+1}-1, F_{n+1}$. For example, if $n = 5$, then your script prints 5, 6, 7, 8 since $F_5 = 5$ and $F_6 = 8$. Your script begins with the following statements:

```
n= input('Input n: ');
value1= 1;
value2= 1;
% Add the necessary code here
```

3. Write a MATLAB script to print all the numbers between 1 and n, exclusive, that divide n (without a remainder using integer division). n is a user input value. (Hint: you already know how to check whether or not a number divides another number. Think back to the first lab.)

Optional Challenge Question: Refer to Question 3 and write a MATLAB script to print the *prime numbers* that divide n.

Reminder: This is a reminder about certain nice properties of *if*-statements and how to cut down on superfluous code. Suppose you have a *nonnegative* ray angle A in degrees. The following code determines in which quadrant A lies:

```
A = input('Input ray angle: ');
A = mod(A, 360);  %Given nonnegative A, result will be in the interval [0,360)

if (A < 90)
  quadrant= 1;
elseif (A < 180)
  quadrant= 2;
elseif (A < 270)
  quadrant= 3;
else
  quadrant= 4;
end

fprintf('Ray angle %f lies in quadrant %d\n', A, quadrant);
```

Notice that in the second condition, it is **not** necessary to check for `A>=90` in addition to `A<180` because the second condition, in the *elseif* branch, is executed **only if** the first condition evaluates to *false*. That means that by the time the computer gets to the second condition, it already knows that `A` is ≥ 90 so writing `A>=90 && A<180` as the second condition would be redundant. Similarly, the concise way to write the third condition is to write only `A<270` as above—unnecessary to write the compound condition `A>=180 && A<270`. This is the nice (efficient) feature of "cascading" and "nesting." If I do not cascade or nest, but instead use independent `if`-statements, then I *must* use compound conditions in some cases, as shown in the fragment below:

```
A= mod(A, 360);  %Given nonnegative A, result will be in the interval [0,360)
if (A < 90)
  quadrant= 1;
end
if (A >=90 && A < 180)
  quadrant= 2;
end
if (A >=180 && A < 270)
  quadrant= 3;
end
if (A >=270)
  quadrant= 4;
end
```