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:

\[
\begin{align*}
&\text{n= input('Input n: ');} \\
&\text{value1= 1;} \\
&\text{value2= 1;}
\end{align*}
\]

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 \( n = 6 \), then your script prints 8, 9, 10, 11, 12, 13 since \( F_6 = 8 \) and \( F_7 = 13 \). Your script begins with the following statements:

\[
\begin{align*}
&\text{n= input('Input n: ');} \\
&\text{value1= 1;} \\
&\text{value2= 1;}
\end{align*}
\]

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.)

4. Refer to Question 3. Use a while-loop instead of a for-loop.

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

\[
A = \text{input('Input ray angle: ')}; \\
A = \text{mod}(A, 360); \quad \%\text{Given nonnegative } A, \text{ result will be in the interval } [0, 360)
\]

\[
\begin{align*}
\text{if } (A < 90) \\
\quad \text{quadrant}= 1; \\
\text{elseif } (A < 180) \\
\quad \text{quadrant}= 2; \\
\text{elseif } (A < 270) \\
\quad \text{quadrant}= 3; \\
\text{else} \\
\quad \text{quadrant}= 4;
\end{align*}
\]

\[
\text{fprintf('Ray angle }%f\text{ lies in quadrant }%d\text{n', }A, \text{ quadrant});
\]

Notice that in the second condition, it is not necessary to check for \( A \geq 90 \) 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 \( \geq 90 \) so writing \( A \geq 90 \land A < 180 \) 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 \( A \geq 180 \land A < 270 \). 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:

\[
A = \text{mod}(A, 360); \quad \%\text{Given nonnegative } A, \text{ result will be in the interval } [0, 360)
\]

\[
\begin{align*}
\text{if } (A < 90) \\
\quad \text{quadrant}= 1; \\
\text{end} \\
\text{if } (A \geq 90 \land A < 180) \\
\quad \text{quadrant}= 2; \\
\text{end} \\
\text{if } (A \geq 180 \land A < 270) \\
\quad \text{quadrant}= 3; \\
\text{end} \\
\text{if } (A \geq 270) \\
\quad \text{quadrant}= 4;
\end{align*}
\]