CS1112 Exercise 3

When you have completed the exercise, show this sheet and any associated programs to your discussion instructor, who will record that you have completed the work. If you do not finish this exercise in class, you have until Sunday, 2/9, at 9pm to get your exercise checked off during consulting hours or during TAs’ office hours.

1 Multiples of $k$

The following program reads an integer $k$ and outputs all positive multiples of $k$ up to 1000. Fill in the blank.

```matlab
k = input('Please enter a positive integer smaller than 1000: ');

for j = ________________
    fprintf(’%d ', j)
end
fprintf(’\n’)
```

2 Approximate square root (again!)

The square root of a positive value $A$ can be computed by building “increasingly square” rectangles with area $A$. Write a script to solicit a positive value $A$ and an integer $N$. Then compute $\sqrt{A}$ by building $N$ increasingly square rectangles. Let the first rectangle have length $A$ and width 1. The final square root value is the average of the length and width of the $N$th rectangle.

Do not use arrays, i.e., you will use scalar variables $L$ and $W$ for the length and width of a rectangle, respectively.

3 Approximate $\pi$

[Modified from Insight Exercise P2.1.5] For large $n$,

\[
T_n = 1 + \frac{1}{2^2} + \cdots + \frac{1}{n^2} = \sum_{k=1}^{n} \frac{1}{k^2} \approx \frac{\pi^2}{6}
\]

\[
R_n = 1 - \frac{1}{3} + \cdots + \frac{(-1)^{n+1}}{2n-1} = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{2k-1} \approx \frac{\pi}{4}
\]

giving two different ways to estimate $\pi$:

\[
\tau_n = \sqrt{6T_n} \quad \rho_n = 4R_n
\]

Write a script that displays the value of $|\pi - \rho_n|$ and $|\pi - \tau_n|$ for $n = 100, 200, \ldots, 1000$ in one table. Do not use arrays.
Review last week’s exercise (Read this at home, not for discussion section)

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

```matlab
A = input('Input ray angle: ');  
A = rem(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 (boolean expression) above, 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 execution gets to the second condition, it already knows that $A$ is $\geq 90$ so writing the compound conditional $A\geq 90 && A<180$ 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 $A\geq 180 && A<270$. 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:

```matlab
A= rem(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
```