Function Name: `f`

Inputs:

1. *(double)* An x value at which to evaluate the function

Outputs:

1. *(double)* The resulting y value

Function Description:

Write a function in MATLAB that will evaluate the function, $f(x)$, for any given x value.

$$f(x) = \frac{1.17\sqrt{1 + x^{0.9}} + sin(x^2 - 4)}{1.5e^{(x^2 - g(x))}}$$

$g(x)$ is another function defined as:

$$g(x) = x * |x|$$

Notes:

- Round your final output to 4 decimal places.
- There are built-in MATLAB functions for square root, sine, exponentials and absolute value (`sqrt()`, `sin()`, `exp()`, and `abs()`, respectively).
- You can use the `help()` function to look up these or any other built-in functions you may be confused about.

Function Name: `applesAndOranges`

Inputs:

1. *(double)* The number of apples
2. *(double)* The number of oranges
3. *(double)* The number of good apples
4. *(double)* The number of good oranges

Outputs:

1. *(double)* The probability of randomly picking a bad apple
2. *(double)* The probability of randomly picking a bad orange

Function Description:

This function should calculate the probability of randomly picking a bad apple or bad orange from a bag. The total number of apples and oranges will be given in the first two inputs and the number of good apples and good oranges will be given as the third and fourth inputs. The output value should be expressed as a percentage, not a decimal. So if there was a 25% chance of drawing a bad apple, this would be expressed as the integer 25 rather than the decimal 0.25. Round all percentages to the nearest hundredth of a percent.

Notes:

- The number of good apples/oranges will always be less than or equal to the number of apples/oranges.

Function Name: `fib`

Inputs:

1. *(double)* Which Fibonacci number to calculate

Outputs:

1. *(double)* The specified Fibonacci number

Function Description:

You are probably familiar with the Fibonacci sequence: the sequence of numbers (starting with 0, 1) where every number is the sum of the previous 2 numbers. From this definition, you might think in order to calculate the n^{th} Fibonacci number, you would have to first calculate all the numbers that came before it. However, this is not the case! The formula for any Fibonacci number is:

$$F_n = \frac{\varphi^n - (-\varphi)^{-n}}{\sqrt{5}}$$

φ is the golden ratio:

$$\varphi = \frac{1+\sqrt{5}}{2}$$

To calculate any Fibonacci number, all you have to do is plug 'n' into this equation. This formula is mathematically exact (no rounding would be necessary in theory).

Notes:

- Due to MATLAB's internal rounding of large numbers, you should round your answer to the nearest integer.

Function Name: `sepDigits`

Inputs:

1. *(double)* A three-digit integer

Outputs:

1. *(double)* The hundreds-place digit of the input
2. *(double)* The tens-place digit of the input
3. *(double)* The ones-place digit of the input

Function Description:

This function will take in a three-digit integer (integer between `100` and `999`, inclusive) and should return each digit as a separate number. For example, if the input is `472`, the first output would be `4`, the second would be `7` and the third would be `2`.

Notes:

- You only have to deal with three-digit positive integers.
- The `mod()` and `floor()` functions will do all the heavy lifting.

Function Name: `clockHands`

Inputs:

1. *(double)* The current position of the hour hand
2. *(double)* The current position of the minute hand
3. *(double)* A positive or negative number of minutes

Outputs:

1. *(double)* The position of the hour hand after the specified time
2. *(double)* The position of the minute hand after the specified time

Background:

It is not always immediately obvious where the hands of a clock will be after a certain amount of time. It is even harder to visualize where the hands *were* some amount of time in the past. Luckily, this is not a very difficult problem for a computer to solve, so you will use that to your advantage.

Function Description:

This function will take in the current position of the hour hand, as an integer between `0` and `11` (`0` for noon/midnight), the current position of the minute hand, as an integer between `0` and `59` (`0` for "on-the-hour") and a positive or negative number of minutes elapsed.

Given this information, determine the new position of the clock hands. You should assume that the hour hand does not move until the next hour has begun. For example, the hour hand stays on "2" from 2:00 until 2:59 and only at 3:00 does the hour hand move to "3".

Notes:

- The `mod()` and `floor()` functions will be useful.
- As you do this problem notice the behavior of `mod()` for negative inputs. This is a very important function in programming and will come up again and again in the class!

Hints:

- One way of solving this problem involves calculating the total number of minutes after noon/midnight before and after the given minutes have elapsed.
- Another way of solving it involves splitting the given number of minutes into a number of hours and a number of minutes.
- Pick whichever method makes more sense to you (or come up with your own method).