Drill Problem #1

Function Name: batterUp

Inputs:

1. (*double*) a 1xN vector of outcomes of a round of batting practice in baseball

Outputs:

1. (*double*) a 1x5 vector of the totals of each outcome from the round of batting practice
2. (*double*) the average score of the round of batting practice
3. (*string*) a brief sentence of how well the round of batting practice was

Function Description:

PLAY BALL!! Baseball is considered "America's pastime" because the professionals were established by 1876 in America. In order to practice, baseball players oftentimes take what is referred to as a "round" of batting practice. Now it's your turn!

Write a MATLAB function to iterate through a vector of outcomes from your round of batting practice. The following outcomes and their associated point values are listed below:

Outcome	*Point Value*
Home Run	10
Regular Hit	5
Foul Ball	1
Out	0
Swing and a Miss	NaN

You will need to keep track of how many times each outcome occurs and store them as your first output in a 1x5 vector like so: [#homeRuns, #regularHits, #foulBalls, #outs, #swingsAndMisses]. Your next output should be the average score of your entire round of batting practice. For instance, if you had the input [5, NaN, 0, 1, 10], your average score would be: 16/5 = 3.2000.

Your final output should be a string of a sentence of how well your round of batting practice was. If your average score was greater than or equal to 5, your sentence should read:

`'You had a great round!'`

If your average score was greater than or equal to 2, your sentence should read:

`'You had an ok round.'`

If your average score was below 2, your sentence should read:

`'Not your best round…'`

Notes:

- The vector of outcomes must follow the order listed above. (i.e. [#homeRuns, #regularHits, #foulBalls, #outs, #swingsAndMisses])
- The punctuation in each output must match the solution file exactly in order to receive credit.

Hints:

- The `isnan()` function may be useful, as well as other conditional statements.

Drill Problem #2

Function Name: findSS

Inputs:

1. (*double*) A 1xN vector of numbers representing experimental data of a system that reaches steady state.

Outputs:

1. (*double*) The steady state value reached by the system, rounded to the nearest hundredths decimal place.
2. (*double*) An estimate of the percentage of time that the system was at steady state.

Function Description:

A steady state system is one that reaches equilibrium after a given amount of time. In the world of engineering there are many systems that will reach a steady state. For example, when you let a hot cup of coffee sit outside for a while, it will incrementally cool down until it is the same temperature as the outside air. This temperature is the so-called "steady state value": when the temperature of the system has stopped changing. If you're an engineer using temperature sensors to study a cooling cup of coffee, there will be random errors in the data taken. For example, when the coffee is actually at an unchanging temperature of 100 °F, the same temperature sensor might tell you 100.174 °F one second and then change to 99.846 °F the next. It is rather easy to visually observe that the temperature is probably at 100 °F, but telling a computer to do this can be difficult, which is where MATLAB comes in!

Write a MATLAB function to iteratively find the steady state value of experimental data. To do this, find the initial mean of the input vector, then remove 10 values off the front of that vector; find a second mean of the shortened vector, then calculate the relative change between those two mean values. Then treat the shortened vector as the original, and repeat those steps until the relative change is less than or equal to 0.0005. The final mean value you calculate will be the steady state value of the system. The formula for relative change used for this problem is given by:

$$Relative\ Change = \frac{abs(avg2 - avg1)}{avg1}$$

Round the steady state value you find to the hundredths decimal place, and that is your first output.

The second output will be the percentage of time the system is in steady state (expressed as a double), rounded to the nearest 10% (the tens place of this percentage expression). To find this, take the final length of the iterated vector over the original length of the vector, multiplied by 100 and rounded appropriately.

Notes:

- You MUST use iteration to solve this problem.

Drill Problem #3

Function Name: nFib

Inputs:

1. (*double*) A number to begin the sequence
2. (*double*) An positive integer (N) denoting the number of terms to return

Outputs:

1. (*double*) A 1xN vector of the resulting Fibonacci sequence

Function Description:

The Fibonacci sequence is very important in mathematics, physics, nature, life, etc. Each number in the sequence is the sum of the previous two values, where the first two numbers in the sequence are always 0 and 1, respectively.

Write a MATLAB function that puts a twist on the classic Fibonacci sequence. This function will input a number to begin the sequence and the number of terms of the Fibonacci sequence to evaluate, and it will output a vector of the corresponding sequence. If the initial term is a 0 or 1, the second term will be a 1; if the initial term is any other number, the second term will be that initial number, repeated.

Therefore, one can find the sequence out to 6 terms, beginning at the number 2, with the following:

$$Fibonacci_1 = 2$$
$$Fibonacci_2 = 2$$
$$Fibonacci_3 = Fibonacci_1 + Fibonacci_2 = 4$$
$$Fibonacci_4 = Fibonacci_2 + Fibonacci_3 = 6$$
$$Fibonacci_5 = Fibonacci_3 + Fibonacci_4 = 10$$
$$Fibonacci_6 = Fibonacci_4 + Fibonacci_5 = 16$$

Meaning the output 1xN vector of the entire sequence to 6 terms would be:

```
out =  [2 2 4 6 10 16]
```

Notes:

- You will not have any negative input values
- You MUST use iteration to receive credit for this problem.

Drill Problem #4

Function Name: ugaFunc

Inputs:
1. (*char*) A string representation of a math function
2. (*double*) A value at which to evaluate the function

Outputs:
1. (*double*) The value of the function at the given value

Function Description:

Given a math function and a value, you want to write a MATLAB function that evaluates the math function at that value. Except your function must use the math skills taught at u[sic]ga. That is, your function completely ignores order of operations and parentheses, and it evaluates all operators strictly left-to-right. You are guaranteed to only have +, -, *, /, and ^ as operators in the function. The left-hand side of the input function will always follow the form:

```
functionName(varName) =
```

There will always be one and only one operator between each number and/or variable in the input function. There will also not be any leading operators. The value that you evaluate the function at will always be greater than or equal to zero.

Notes:
- Extraneous spaces in the input string should not mess up your function
- The variable name can be multiple characters long
- There will be no square brackets in the input string, but it may have parentheses (which you will ignore)

Hints:
- You might find the `strrep()` function helpful

Drill Problem #5

Function Name: datHookshotDoe

Inputs:

1. (*double*) A 1x2 vector representing a row and column
2. (*char*) An MxN string array representing cardinal directions
3. (*double*) An MxN array representing distance to travel
4. (*char*) An MxN string array of jumbled letters
5. (*double*) An MxN array of rupee values

Outputs:

1. (*char*) A string found after following the directions through the array of letters
2. (*double*) A total number of rupees collected after travelling through the array

Function Description:

In the Legend of Zelda games, it is usual to be challenged with a series of puzzles (or evil creatures) in order to advance between rooms in dungeons, find keys, and unveil treasure chests with the next dungeon weapon (or that oh so useful compass…). While in the most recent temple, Link has found a scroll infinitely more useful than the mere map other dungeons provide.

Examining the scroll, he sees a vector of two numbers; an array of the letters `'N'`, `'S'`, `'E'`, and `'W'`; an array of seemingly random integers; another array of jumbled letters, this time spanning the entire alphabet; and an array of rupee values (rupees being the form of currency in the game). Navi, Link's fairy, gives an essential hint ("this scroll is probably useful for this dungeon"), and Link realizes that the scroll is a puzzle! It is a set of instructions to find what the dungeon's weapon is and where it is located.

The vector of two numbers are Link's starting coordinates in the arrays. The first array of the letters `'N'`, `'S'`, `'E'`, and `'W'` are cardinal directions through which to traverse the arrays. The first set of numbers is the distance to travel in any given turn. The second char array is a set of jumbled letters that—when formed in the correct combination—spells the weapon that may be found in Link's current dungeon. Finally, the last array contains values for rupees that Link picks up along the way, because there is money laying around everywhere in this dungeon.

Write a function called `datHookshotDoe` that follows the pattern given by the clues in the input cardinal direction array and travel distance array to output a string indicating the dungeon's weapon and a double of the total number of rupees found in the dungeon. The first input is a 1x2 vector representing the starting index. This is the location of the first letter in the string array (fourth input) and where to begin in all four input arrays.

Next, using the second input (the direction array) and the third input (the distance travelled array), grab the values in each array corresponding to the current index location. The direction will be designated in the second input by `'N'`, `'S'`, `'E'`, or `'W'` for north (up), south (down), east (right), or west (left). Find the number of steps to be taken in the third input, and travel that many spaces in the direction given. The number of steps to be taken will always be positive.

Trace through all the arrays simultaneously, using the directions and distances as a guide. Save the letter found in each step (from the array in the fourth input) into a string for the final dungeon weapon answer. Further, keep track of the total number of rupees collected along the trip—the number of rupees being an array given in the fifth input. Include the letter and rupee value found in the starting location. Stop when the direction is `‘D’` (for destination) and the number of spaces is 0.

For example:

If the inputs were the following:

```
[1,3] , [‘NSEW;    ,  [4 1 1  2;  , [‘ZiSh;  , [ 1 5 50 1;
          DEWW’]       0 1 2 18]       del ‘]     10 5  5 1]
```

The starting index would be the first row and third column (1,3) in the forth input (`‘S’`); the fifth input would give 50 rupees to start with. Then look at index (1,3) in the second input (`‘E’`) and in the third input (1). This gives directions to go to the index 1 to the right of the current position; this makes the updated position now (1,4), which holds `‘h’` in the fourth input and 1 rupee. Now the stored string should read `‘Sh’` and the total rupees be 51. To find the next index, look at the position (1,4) in the second and third inputs. Follow the pattern until the letter `‘D’` is found in the direction array and 0 is in the travel distance array. The output string of this example is `‘Shield’` and the total rupee amount is 76.

Notes:

- The string array holding the password may include non-letter characters (i.e., `‘ ’`, `‘.’`, `‘?’`, etc.), but an apostrophe will not appear.
- The correct path will not take you out of bounds, but a wrong one might.