Function Name: `perfectARecipe`

Inputs:

1. (*double*) A number representing how many issues your recipe has

Outputs:

1. (*double*) The number of times you had to tweak your recipe in order to perfect it

Topics: (*iteration*), (*conditionals*)

Background:

After years of working your way up from professional dishwasher status, you've finally achieved your dreams and became the head chef of your very own bistro, Le Petit MATLAB! Although it's only been a few weeks since you first opened, you already have such an amazing reputation that Andre Michelin himself has informed you that he will be visiting your restaurant in 3 hours! Upon hearing this news, you take a look through your pantry and realize that you've completely run out of the ingredients that you use to make your normal menu! You know that you can't let such a good opportunity go to waste though, so you grab all of the ingredients you can find and mix them together in hopes of creating a new dish that will blow Andre off his feet. Alas, even after 2 hours of trying, your recipe still has a worrying number of flaws. With only an hour left before the fate of your bistro changes forever, you turn towards the salt to your pepper, MATLAB, in hopes of getting your number of errors down to just one, since you know that no matter how hard you try or how many spices you add, peanut butter and flour tortillas just don't go well together.

Function Description:

You are given the number of errors present in your recipe. Perform the following mathematical operations until the number of errors becomes equal to 1:

- If the number of errors is even, divide it by 2.
- If the number of errors is odd, multiply it by 3 and add 1.

Count how many attempts it takes you to get the number of errors to be equal to 1, and output this value as your final answer.

Example:

```
errors = 13
[count] = perfectARecipe(errors)
count = 9
```

Notes:

- The input will always be a positive integer.

Function Name: `pastrySales`

Inputs:

1. (*double*) A vector of doubles representing the bakery's profits every other day

Outputs:

1. (*double*) A vector of doubles representing the bakery's profits every day

Banned Functions:

```
interp1()
```

Topics: (*interpolation*), (*iteration*)

Background:

You work for Grandma's Pastry Shop. As much as you love her, her organization skills (or lack thereof) make you a bit crazy. For instance, she doesn't even know what half of the daily profits from last week are because she smudged icing on the paper! You set off on a mission to find the numbers and help Grandma get organized with your handy friend Matlab.

Function Description:

Given a vector of every other days' profits, estimate the profits for each missing day by averaging the profit from the day before and after it. The output vector should contain the values present in the original vector, as well as each average value placed between the two numbers that were used to find it.

Example:

```
knownProfits = [50 60 44]
outVec = pastrySales(knownProfits)
outVec -> [50 55 60 52 44]
```

Notes:

- You will not have to worry about rounding. (All interpolated values will be integers.)
- The input vector is guaranteed to be at least length 2.

Function Name: `goGetAC00kie`

Inputs:

1. (*char*) A vector of chars containing words separated by spaces

Outputs:

1. (*double*) Length of the longest word
2. (*double*) Length of the shortest word
3. (*double*) Average length of all the words, rounded to 2 decimal places
4. (*double*) Mode length of all the words

Banned Functions:

```
max, min, mean, mode, sort, maxk, mink
```

Topics: (*iteration*), (*string tokenization)*

Background:

Dang n00b, tough day. You just t00k a test and boy was it a d00zy. Now, you feel so f00lish because you realized s00n after that you g00fed and misunderst00d the long coding question like your parent or guardian misunderst00d your middle sch00l phase 😔✋. To reb00t your m00d, you open your h00tenanny with a c00l background of a timberd00dle and navigate to Yah00!™ to l00kup a c00kie tyc00n near you. You are going to go get a c00kie, but what flavor to ch00se?

Function Description:

Given the input vector of chars, determine the length of the longest c00kie name (output 1), the length of the shortest c00kie name (output 2), the average c00kie name length rounded to 2 decimal places (output 3), and the mode c00kie name length (output 4). Assign each of these values to their respective output, and b00yah! You have c00kies out the waz00. Clear eyes. Full heart. Big Brain. Can't Lose.

Example:

```
c00kies = 'snickerd00dle raisin mint sugar oreo chocolate'
[long, short, avg, mode] = goGetAC00kie(c00kies)
long -> 13
short -> 4
avg -> 6.83
mode -> 4
```

Notes:

- All c00kie flavors will be one word.
- The longest possible c00kie flavor is 15 characters.
- A single space will separate each word, with no trailing spaces.

- Mode cookie length refers to the most common length of cookie name, not how often that length occurs.
- No special characters or punctuation will be present.
- There is guaranteed to be one mode c00kie flavor length.

Hints:

- Don't forget to round the average c00kie name length to 2 decimal places.
- Make sure your function name matches the description (goGetAC**00**kie with double zeros instead of o's).
- The zeros function may be useful!

Function Name: `cookieCounter`

Inputs:

1. (*double*) 1XN vector of cookie calories

Outputs:

1. (*double*) 1X3 vector of the three calorie values summing to 1371

Banned Functions:

```
nchoosek, combntns
```

Topics: (*nested loops*), (*masking*)

Background:

You have started working for the CS 1371 Cookie Company™ and are helping them create their newest product: the Triple Cookie Box! In order to make printing the nutritional information easier, the company has decided every box will have exactly 1371 calories! Your job, then, is to decide which cookies go in each box so that the three calorie values from the three cookies add up to exactly 1371!

Function Description:

Given a vector of unique values, with each index representing the calorie value for a different cookie, find the three unique values that add up to exactly 1371. Each value in the input vector is guaranteed to be a nonnegative integer (0 inclusive) and unique (no repeated values). Once you find the combination of three values that equal 1371, output the values as a vector sorted in descending order.

Example:

```
calories = [1008 67 89 0 1221 789 343 83 1300];

>> [values] = cookieCounter(calories)

values →
      [1221, 83, 67]
```

Notes:

- Each input is guaranteed to have at least three values.
- There will always be only three values that add up to exactly 1371 in the input vector.

Hints:

- Remember there is only going to be one unique combination, so two possible solutions include finding all possible combinations first, then selecting the correct one, and checking each combination, stopping once you find the correct combination
- Think about how you can use nested for loops to create different combinations of three numbers each.
- Masking might be useful as well

Function Name: `holyCannoli`

Inputs:
1. (*char*) MxN map of cannoli strewn around your city

Outputs:
1. (*char*) MxN pathed map
2. (*char*) a string describing your cannoli condition

Topics: (*iteration: pathing*), (*conditionals*), (*arrays*)

Background:

You absolutely LOVE cannoli. Can't get enough. In fact, you love them so much you're gonna drive around town with your cannoliTracker9000™ to hunt down all the cannoli in the city and eat them. Better start now before the cannoli cravings get too strong!

Function Description:

You are given an array that contains a map of the cannoli around your city. Write a function that paths around the array, eats cannoli, and leaves a trail of where you've been. There are a few special tiles (characters) on the map to keep in mind:

- `'>', '<', '^', and 'v'` indicate a direction change (direction-changing tiles)
  - `'>'` directs you right
  - `'<'` directs you left
  - `'^'` directs you up
  - `'v'` directs you down (lower-case v)
- `'c'` indicates a cannoli at that location
- `'f'` indicates your stopping position
- `'.'` is an empty tile with no special attributes

Your starting position is always `map(1,1)`, or row one, column one of the map. There will be a direction-changing tile at that position giving you your initial direction. Proceed by going in that direction.

As you iterate through the map:
- Change directions **when you land on** a direction-changing tile to the indicated direction.
- Keep going in the same direction as you were previously going if you do not land on a direction-changing tile.
- For each `'c'` you path over, add 1 to the total amount of cannoli you have eaten.
- At every location you've been, replace the character at that location in the map with a `'#'`. This should represent the path you will take on your journey.
- If you eat over 30 cannoli (**on the 31st cannoli**), stop iterating. You have eaten too many cannoli! You should eat the 31st cannoli and change the location to a `'#'`.
- If you reach the character `'f'`, stop iterating. You have reached the end of your journey!

Once you finish iterating through the map, output the pathed map (output 1), and count how many cannoli you missed. Depending on how you finished, output one of the following strings (output 2):

- If you ate too many cannoli, output:

```
'Darn, I ate too many cannoli. I wasn't able to eat
      <number of cannoli missed> of them :('
```

- If you reached the stopping position, output:

```
'I ate <number of cannoli eaten> cannoli, and missed
     <number of cannoli missed> along the way!'
```

Example:

```
map = ['v..c.>v';
       '..c..c.';
       '>.cc.^.';
       '..c...c';
       'c.f.c.<']

>> [pathedMap, cannoliDesc] = holyCannoli(map)

mapFinal →
     ['#..c.##';
      '#.c..##';
      '#######';
      '..c...#';
      'c.#####']

outstr →
     'I ate 5 cannoli, and missed 4 along the way!'
```

Notes:

- You should replace the character **at every location you path over**, **including the initial and final positions**, any `'c'`, any `'#'`, and any `'.'`.
- Paths are not guaranteed to be a set length across test cases.
- You might cross over previously travelled paths, but you will not be told to go out of bounds of the array.
- Eating **over** 30 cannoli and reaching the `'f'` cannot occur simultaneously.
- The only characters present on the board at any time are `>`, `<`, `^`, `v`, `c`, `f`, `.`, and `#`.
- The starting position will **always** be `>`, `<`, `^`, or `v`