Number To Text Webpage

Description of Function

- This software makes use of dictionaries to denote pronunciations. Another option would have been to use a list and access strings based on their index; however, this would have been more complicated to organize for the "teen" list as more operations or empty entries would have been required. This includes dictionaries to specify the pronunciation of:
    - single digits
    - teen numbers
    - double digits (e.g. twenty, thirty, forty etc)
    - orders of magnitude (e.g. thousands, millions etc)
- I thought there were a few ways to process the input number to determine the string. To begin with, as the documentation provided specifies "dollars" and "cents" in the example string, I decided to limit the user input to this format. Ie, always round to two decimal places. I also decided to limit the input maximum to the maximum value of a decimal as this value type seemed to be the most consistent when parsing the value to the input variable. The size of this number is also significant. In terms of processing the input variable, I toyed with a few potential methods:
    - Read the input number from left to right one at a time. I determined I should not take this approach as it would have too many conditionals and special cases and would not be a simple easy to read solution.
    - Read input number from right to left which would feel more natural with code as you can extract the last number easily with % 10. However, I also determined that this would have too many conditionals for the solution to be simple
    - Parse the input to a string and append 0s to the left side to ensure the total character count is divisible by 3 (much like the segments used in the English number speech pattern). Then split this string into 3-digit segments and read from most significant digit to least significant, much in the same fashion as the provided solution. However, I determined that my current solution would be better as less operations on the data would be required to "translate" the number to a string
- The logic behind the software is built up from four functions, from "top to bottom" they are:
    - "DollarsToText" takes a decimal input and essentially splits this into two longs (dollars and cents) which are then each passed to "NumberToText". The two strings returned by this function will be appended with "DOLLARS" and "CENTS" as required. This function returns a full string denoting the user input as text. Note that this is the only accessible function from this class.
    - "NumberToText" takes a long input and splits it into three-digit segments which mimic the English speech pattern. This is done from least significant to most (takes 3 least significant digits, saves to list, then divides long by 1000). The list is then reversed to ensure that the string magnitudes are calculated correctly. Each segment

in the list will then be passed to “TripleDigitToString”. The string returned by this function will be appended to the output string along with a “magnitude” string (eg thousand, million etc) as required, based on the number of segments remaining in the list. This function will return the number entered as text.

  - “TripleDigitToString” takes a long input (this should ONLY be a three or fewer digit number. Note that there is no input validation, however as this is a private function it is not accessible from outside the class. Hence this is taken care of by the other functions that are accessible). It returns a string based on the following logic:
    - If the long is > 99, finds x = input/100 and appends x HUNDRED to the string (the single digit string for x is determined using the SingleDigitText dict)
    - If the long % 100 is greater than 0 and the input is > 100 (or in the unique case where this is the final segment eg 1,000,023 is said like “one million and twenty three”), an “AND” is appended
    - If the input % 100 is < 10 then the SingleDigitText dict is used to append the required string
    - Else the input % 100 is passed to “DoubleDigitToString”. The string returned from this is appended to the output string
  - “DoubleDigitToString” takes a long input (this should ONLY be a two-digit number. Note that there is no input validation, however as this is a private function it is not accessible from outside the class. Hence this is taken care of by the other functions that are accessible). It returns a string based on the following logic:
    - If the input is < 20, used the TeenDigitText dict to return the required string
    - Else finds the input / 10 and used the DoubleDigitText dict to append the required string
      - If the input % 10 is not 0, then uses the SingleDigitText to append the required string (along with an “AND”)
- For the graphical interface of the solution, I decided to go with something very simple as there will be less chance of confusion when a client opens a simple interface. I have included the following:
  - A large text display of what will be the translated text of the number provided.
  - A text box to input the desired value
    - I have limited the input of this to have a maximum value the same as the maximum value of a decimal
    - I have also limited this to only allow numerical input
  - A button to translate to text
  - A string under the input box that will display the “pretty” version of the inputted number
  - I also chose to make the colours more “soothing” to the eyes than the default white.

Test Plan

To test the software, it will need to be installed using the instructions outlined in readme.md Once this is done, the software can be accessed by navigating to the webpage.

Objective

Determine if the software gives the expected text translation of some inputted number.

Expected Results

- The software should not accept negative numbers
- Any inputted number over the decimal maximum should be automatically set to the decimal maximum
- The output will be written as dollars and cents in all caps

Strategy

- Software can be tested by inputting numbers in to text box on screen and determining if the given output is expected
- The user should input many variations of numbers to compare
- The source code also includes a unit test which can be performed, extra tests can be added as required to confirm operation