

For this assignment you are to create and submit a single Python module; that is, a file with a “.py” extension containing nothing but Python code (which, of course, includes comments, such as the standard six-line header). The file must include four (4) function definitions with precisely these names (including case):

1. CollatzSequence
2. CollatzSearch
3. IsAlphabetic
4. GetTokens

Your file, which must be named “hw2.py,” may also contain other functions and global variables, but you will be graded exclusively on the correctness and clarity of the four functions named above (as well as those that they may call). Here is what you need to do:

1. Read chapters 7, 8, and 10 of *How to Think Like a Computer Scientist*. It is recommended that you skim all the chapters up through chapter 10, but the ones cited above are particularly important. Also, it is recommended that you read chapters 2 and 3 in *Dive into Python*. Be sure you are familiar with lists and dictionaries, in particular.
2. In 1935 a mathematician name Lothar Collatz defined the following sequence of positive integers:

$$a_{n+1} = \begin{cases} \text{undefined} & \text{if } a_n = 1 \\ a_n/2 & \text{if } a_n \text{ is even} \\ 3a_n + 1 & \text{if } a_n \text{ is odd} \end{cases}$$

where  $a_0 > 1$  is specified (i.e. set to anything you like), and the sequence terminates as soon as the value 1 is attained (which, as far as anybody knows, will always happen eventually). For example, starting with  $a_0 = 3$ , we get the following sequence: 3, 10, 5, 16, 8, 4, 2, 1. Starting with  $a_0 = 5$ , we get the following sequence: 5, 16, 8, 4, 2, 1. This type of sequence has come to be called a *Collatz sequence*. Note that the length of the Collatz sequence varies wildly and is impossible to predict from the starting value.

For this part of the assignment you are to define a Python function called `CollatzSequence` that takes a single argument, call it  $N$ , and returns a list containing the entire Collatz sequence that begins with  $a_0 = N$ . For example, `CollatzSequence( 5 )` should return the list [ 5, 16, 8, 4, 2, 1 ]. (Do not worry about catching erroneous input, such as non-integer values of  $N$ . You may simply assume that  $N$  will always be an integer greater than 1, and that the sequence will always terminate.)

3. Define another function called `CollatzSearch`, which takes two arguments, let’s call them *num\_examples* and *min\_length* (in that order), which are both assumed to be positive integers. The function `CollatzSearch` is to search for Collatz sequences whose lengths are equal to or greater than *min\_length*, and it is to find exactly *num\_examples* such distinct sequences. The function is to begin searching at  $N = 2$ , incrementing by one and testing each subsequent value of  $N$  using the function `CollatzSequence` from the previous problem. Each time it finds a Collatz sequence that is at least as long as *min\_length* it is to add the pair  $(N, S)$  to a *dictionary*, where  $N$  is used as the “key” and  $S$  is the data associated with the key; here  $S$  is to be the actual Collatz sequence generated by  $a_0 = N$ . When your dictionary contains exactly *num\_examples* entries (which you can test using the `len` function) the function `CollatzSearch` should simply return the dictionary as its function value. For example, `CollatzSearch(`

3, 8 ) should return {3: [3, 10, 5, 16, 8, 4, 2, 1], 6: [6, 3, 10, 5, 16, 8, 4, 2, 1], 7: [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]}, because 3, 6, and 7 are the first three integers that produce a Collatz sequence of length 8 or longer.

4. Write a function called `IsAlphabetic`, which takes a single argument, assumed to be a string of length 1 (i.e. a single character). The function is to return `True` if the single character is *alphabetic* (i.e. “a” to “z”, or “A” to “Z”), or an apostrophe. (You will see why you need to include the apostrophe shortly.) It should return `False` otherwise. Note that you can use the “in” keyword to test whether one string is a sub-string of another. For example “ab” in “cdefg” returns `False`, while “ab” in “cdefabgh” returns `True`.
5. Write a function called `GetTokens`, which is to accept a single argument, assumed to be a string. `GetTokens` is to return a *list* of the “tokens” that are found in the string, in exactly the order they are found. Here we shall define a token to be a sequence of alphabetic characters (including the apostrophe symbol) that are delimited by white space (blanks, tabs, etc.), the beginning or end of the string, or any other non-alphabetic character, such as commas and other punctuation. This function will be used to separate out the words and the punctuation found in English sentences. Here are some examples of what `GetTokens` should return:

- `GetTokens( "Methinks it is like a weasel." )` should return  
[ 'Methinks', 'it', 'is', 'like', 'a', 'weasel', '.' ]
- `GetTokens( "It's a line from Shakespeare's Hamlet!" )` should return  
[ "It's", 'a', 'line', 'from', "Shakespeare's", 'Hamlet', '!' ]

Note that the tokens “it’s” and “Shakespeare’s” each contain an apostrophe, which is why we included apostrophes among the alphabetic characters. It’s also why Python decided to print those tokens out using double quotes rather than single quotes (apostrophes). We will build upon this function in subsequent assignments to do some interesting processing of English text.

More test cases for you to check your functions with will be posted for you on the *downloads* page. Be sure to test your functions thoroughly. Also, remember to document your functions internally.

You will need to understand strings, lists, dictionaries, the `append` method, the `len` function, `if-elif-else` tests, `while` loops, and the `in` keyword (among other things) to do this assignment.