Logan Watanabe

67-328 Sept.3, 2013

Assignment 1 – JavaScript Class and String method

1. JavaScript class

My JavaScript class is a simple “Team” class, meant for sports teams. The class has attributes “location,” “name,” “league,” “division,” “wins,” and “losses.” This is just a basic class that could be used for NBA, MLB, NFL, etc.

The constructor for the class takes in parameters for the attributes and assigns them. There are no defaults because I can’t predict what sport it will be.

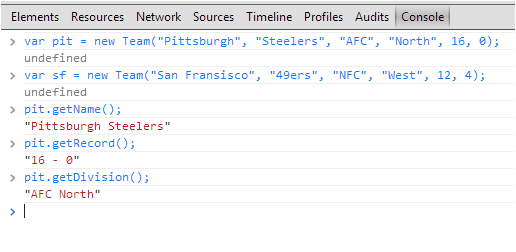
The methods are basic getter methods that concatenate the attributes to return a nicely formatted string. So the “getName” method returns a team name (i.e. Pittsburgh Steelers), the “getRecord” method returns a record with a hyphen in the middle, and the “getDivision” method returns the team’s league division (i.e. AFC North). All these methods are assigned to the team prototype, so they can be called from any instance of team.

Figure 1 is a screenshot of the constructor and methods input to the console on chrome.

Figure 2 is a screenshot of me instantiating 2 instances of the team class, and also testing each of the methods. It all works.

Figure 1



 Figure 2

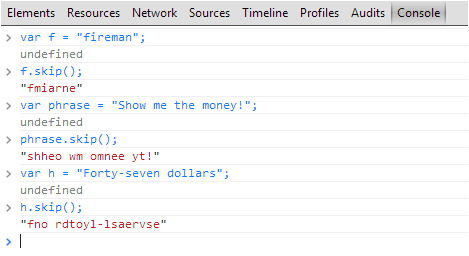
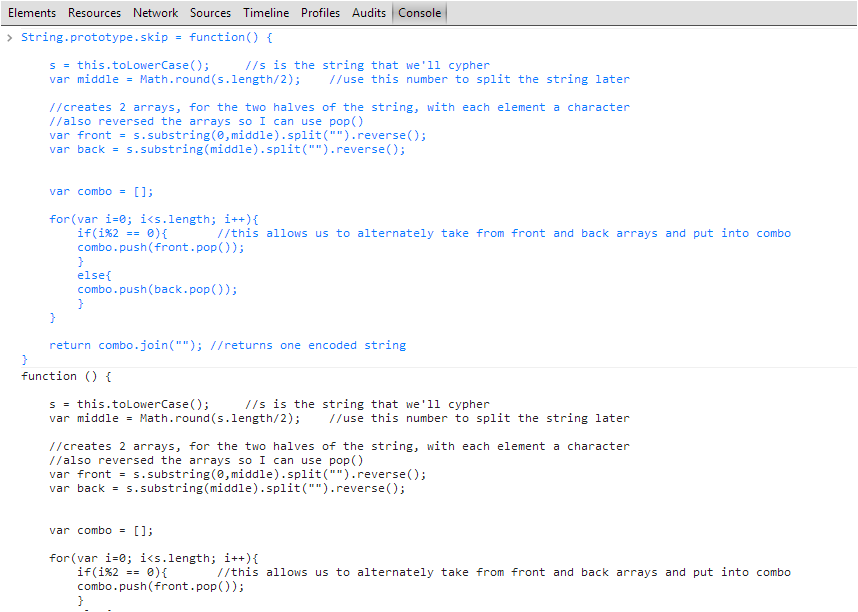
1. String Manipulator

The method that I added to the String class in JavaScript is sort of a cypher, called “skip a letter code.” I found out how to do the code online at [http://www.wikihow.com/Write-in-Skip-a-Letter-Code](http://www.wikihow.com/Write-in-Skip-a-Letter-Code.) . Basically, for a given phrase you split it in half and then rewrite it out using the letters from the first half and second half in an alternating order. The difference between my version and the website’s version is that in my code I include the spaces as characters, so that the final encoded phrase has spaces and is easier on the eyes, in my opinion.

In the method applied to the String prototype, I accomplished this code by taking several steps. First, I receive the input string (lowercase) and take the length and divide it by two (rounding up for odd lengthened strings) in order to get the middle where I will split the string into the two halves. I split the string into these two halves using substring and make them arrays. I also reverse these arrays of characters so that I can use the pop method later. Then I use a for loop and a modulus function (so that I start on the first half) to alternately add the characters into a new array. Then, I join the array into a string and return the coded string.

I did not write a decoder for this method, but if I did it would involve taking the even and odd numbered characters of a string and putting those two halves together again.

Figure 1 shows the String prototype method being put into the console, and figure 2 shows me testing it on a couple of strings. It’s all good.

 Figure 1 Figure 2