You have a typo on line 30 of part2.html

Download the three starter files (part1.html, part2.html, and bst.html, and put them in your webpages directory under a new directory called *lab4*

Make sure to check your directory and file permissions are set to 755. Go to a browser and go to your page and load each of the three files.

Part 1 - Variables (20 pts)

Go to http://students.engr.scu.edu/~your_username/coen161/lab4/part1.html and examine the outputs in the console.

Open the HTML file in a text editor.

1. Go through the examples and answer the questions in the comments.
    - Answer the question directly in the comment
    - Write any code directly into the script
    - Upload your completed file

Part 2 - Functions (30 pts)

Go to http://students.engr.scu.edu/~your_username/coen161/lab4/part2.html

Open the HTML file in a text editor and implement the three functions.

1. Write a function that takes an array and returns the minimum value in that array (Do not use Math.min, I want you to implement this without using the Math library)
2. Write a function that checks whether or not the given string is a palindrome
3. Write a function that returns the string that occurs the most in an array of strings
   (For example, ['green','red','red','blue'] would return red. Assume that there is only one string that occurs the most)

Note: While test data is provided, they don't test every case. You should come up with additional test cases to ensure the correctness of your functions

Part 3 - Binary Search Tree (50 pts)

Go to http://students.engr.scu.edu/~your_username/coen161/lab4/part3.html

In this file, your task is to implement a binary search tree using JavaScript constructors and prototypes as discussed in class.

As a reminder, a binary search tree stores values less than the root in the left subtree, and values greater than the root in the right subtree.

In C, you may have defined a structure like the one below for each tree node.

```
typedef struct node {
    struct node *left;
    struct node *right;
    int val;
} NODE;
```

You may have also had a second structure to abstract your tree's implementation.

```
typedef struct bst {
    struct node *root;
} BST;
```

With this in mind, and using what you learned about JavaScript constructors, build a class (or blueprint) for a binary search tree. You might also find it useful to build a separate class (or blueprint) for nodes.

Your BST class should be able to do two things, insert values into the tree, and search for a value in the tree.

These two functionalities are essentially the same for all BSTs so it's best to add them to the BST prototype.

You will also need to write a function to traverse the tree **in order.** This function will take one argument, a **function** that is called with the value of each node in your traversal. It will look something like this:

```
BinarySearchTree.prototype.inorder = function(fn) {
    ...
    fn(node.val);
    ...
};
```

To test your implementation, I provided a library function, *testBst*, which takes a BST object. This function will insert test data into your BST and display the values in order using your insert and inorder functions. To test your search, just enter a value into the text

box and hit submit to search your tree using your search function. You will see an alert message telling you whether or not the data was found.