Part I: Complete the following tasks from The JavaScript Language book. Try to answer them before looking at the solutions. You should work on each one for 10 – 20 minutes before looking at the solution. Write your own code for each of these in VSCode before looking at the answers.

Use the Mocha tests in the recursionTest.js file for the Part I exercises. There are no Mocha tests for the Part II exercise. Inspect your output in the console for Part II.

Chapter: Advanced working with functions

Section: Recursion and stack

- Sum all numbers till the given one
- Calculate factorial
- Fibonacci numbers (the dynamic programming solution is optional)

The remaining parts of the assignment can be done tomorrow if we did not cover linked lists in class today.

- Output a single-linked list
- Output a single-linked list in reverse order

Part II:

1. Write a recursive function that will print "name: value" to the console for every node in the following tree data structure. Then write an iterative function that also does this. The following is the output you should have for the recursive version. The iterative version should be similar, but it is not necessary for every line to be in the same order.

```
body:  null
div:  null
label:  Name
input:  this was typed by a user
p:  This is text in the a paragraph
```

2. Modify both versions to return an array containing the "name:value" strings for the entries rather than printing the values to the console.

```
let node3 = {
   name: "p",
   value: "This is text in the a paragraph",
   children: null
};
```

```
let node4 = {
   name: "label",
   value: "Name",
   children: null
};

let node5 = {
   name: "input",
   value: "this was typed by a user",
   children: null
};

let node2 = {
   name: "div",
   value: null,
   children: [node4, node5]
};

let node1 = {
   name: "body",
   children:  [node2, node3],
   value: null,
};
```

Part III

Use the Simpsons code given below to generate a small tree data structure, TreeNode.

Write recursive functions for the following:

1. Log to the console the names of everyone in the tree.

2. Given a target value, return true or false if there is a node in the tree with the target value. E.g.,

   contains(tree, “Lisa”) → true

   contains(tree, “Crusty”) → false

3. Given a target value, return the subtree with the found node or null if no match. Extend the tree to have a more interesting test. Create a mocha test to run at least 2 or 3 tests on your tree.

findSubtree(tree, “Homer”) → subtree with Homer as the root

4. Create a new constructor function ListNode (based on TreeNode below) and use it to generate a linked list of Abe, Homer, Bart, Lisa, Maggie instead of a tree.

5. Given a target value in the list, return the node that contains the target value or null if no match.

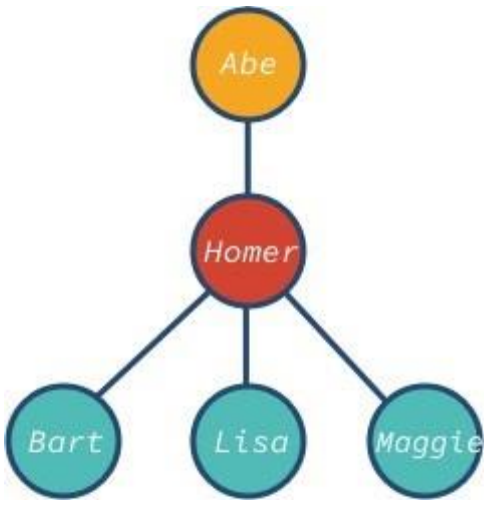
   findListNode(list, “Bart”)

6. Write a recursive function, treeModifier, that will take a tree and a modifier function as parameters. Walk through the tree and apply the function to each node. The function should apply the following operations to each node. treeModifier(tree, modiferFunc)

- allCaps(node) will change the value of a node to be all caps.
- addStars(node) will change the value to have *** in front and behind the node value.
- reverseNode(node) will reverse the string of the node value. Abe -> ebA

   Call your recursive function with each of these modifier functions.

7. Write a recursive function, treeCollector(tree) that will walk the tree and collect the values of each node into an array [“Abe”, “Homer”, “Bart”, “Lisa”, “Maggie”] for this tree. You may want to use an accumulator variable that is external to the function to hold the values.



https://adrianmejia.com/data-structures-for-beginners-trees-binary-search-tree-tutorial/

function TreeNode(value) {

```
    this.value = value;

    this.descendents = [];

  }


// create nodes with values

const abe = new TreeNode('Abe');

const homer = new TreeNode('Homer');

const bart = new TreeNode('Bart');

const lisa = new TreeNode('Lisa');

const maggie = new TreeNode('Maggie');


// associate root with is descendents

abe.descendents.push(homer);

homer.descendents.push(bart, lisa, maggie);
```