Javascript Coding Challenge

# Exercise 1

**GIVEN**

Given any web page with dynamic Javascript.

**WRITE**

Write a Javascript library FireflyDomSniffer that does the following:

- Overrides all built-in Javascript functions that allow you to add, remove, or modify DOM nodes on the page. These include appendChild, removeChild, removeAttribute, insertAttribute etc., (a full list should be compiled before starting.)

- When any of those overridden functions are called, the library should print to the console the HTML that’s being changed. For example, if the page calls appendChild to append a node with HTML: <div class=”foo”>bar</div> into the page, the library should print:

Received overridden function call!

Function: appendChild

HTML Delta: <div class=”foo”>bar</div>

**REQUIREMENTS**

- The library should be properly scoped so that all variables and methods are only available through the FireflyDomSniffer object.

- The library should not harm normal page functionality. So all of the functions that are being overwritten should still be callable and should still work normally except for the added functionality.

- The library should be properly commented, documented, and should adhere to DRY (do not repeat yourself) principles.

# Exercise 2

**GIVEN**

Given any web page with dynamic Javascript, and the FireflyDomSniffer library from above.

**WRITE**

Add a function to FireflyDomSniffer library called collectTextNodes that does the following:

- When called, collectTextNodes should collect all text nodes on the page into a JSON array. It should then return the array at the end of the function. The array should be in the following format:

{parentNode1:textNode1, parentNode2:textNode2 …. parentNodeN:textNodeN}

Parent node is the parent node of the text node being added to the array. Text node is the text node that you found.

# Exercise 3

**GIVEN**

Given any web page with dynamic Javascript, and the FireflyDomSniffer library from above.

**WRITE**

Add a function to the FireflyDomSniffer library called sanitizeHTML that does the following:

- Any a time a function that inserts nodes in to the page is called (e.g. appendChild) sanitizeHTML is called

- When given a node with a set of attributes, sanitizeHTML checks a whitelist of allowed attributes. If the attribute is not on the whitelist it is removed from the node. If it is on the whitelist, sanitizeHTML checks the attribute value with a regular expression to ensure that the form of the attribute value is allowed. sanitizeHTML should return the sanitized node from the function.

The whitelist of tag names is the following:

class

id

name

href

All class attributes should be checked with the following regular expression:

/^[a-zA-Z0-9\s,\-\_]+$/

All id attributes should be checked with the following regular expression:

/^[a-zA-Z0-9\:\-\_\.]+$/

All other attributes should be checked with the following regular expression:

/^.\*$/

For example:

Given a dom node:

<div class=”foo” src=”bar”></div>

The function would check to see if class is in the whitelist. It is in the whitelist, so it will then check the attribute value which is ‘foo’. It will check ‘foo’ with the class regular expression: /^[a-zA-Z0-9\s,\-\_]+$/

It passes both tests so it will leave it alone.

The function would then check the attribute ‘src’. It would see that ‘src’ is not on the whitelist and remove it from the node.

Exercise 4

**PUZZLE**

In what cases would the following Javascript regular expression match the given strings? Put a check next to the ones that it would return a ‘true’ match for.

This is the regular expression: /[a-zA-Z0-9,\-\s]+/

The regular expression would be tested on the string like so:

/[a-zA-Z0-9,\-\s]+/.test(‘string’)

Case 1: ‘javascript’

Case 2: ‘ ‘

Case 3: ‘javascript:’

Case 4: ‘javascript:alert(“hi!!”)’

# Exercise 5

**GIVEN**

Let’s say we have a simple HTML page structured like the following:

<html>

<head>

<title>Hello!</title>

</head>

<body>

<div class=”div1”>

<h1>This is a headline!</h1>

<br>

<img src=”header-image.png”>

</div>

<div class=”div2”>

<a href=”http://www.google.com”></a>

</div>

<div class=”div3”>

</div>

</body>

</html>

In order to cobrowse, we need to take this HTML tree and convert it to a JSON representation that looks like the following:

{

"html" : [

{

"head" : [

{ "title" : "hello" }

]

},

{

"body" : [

{ "div1" : [

{ "h1" : "This is a headline!" },

{ "br" : "" },

{ "img" : "header-image.png" }

]},

{ "div2" : [

{ "a" : "http://www.google.com" }

]},

{"div3" : []}

]

}

]

}

**WRITE - Task 1**

Write a recursive function that takes an HTML page and converts it to a JSON object like the one shown above.

**WRITE - Task 2**

The function you wrote in Task 1 works very well for modern browsers like Chrome and Firefox. However, if the tree has a lot of nodes, and we try to run this function on an older browser like IE9, it will freeze the browser. That’s because the browser will wait for the entire recursive function to finish before it does anything else.

Therefore, we must take this recursive synchronous function and turn it into a recursive asynchronous function. Write the recursive function as a recursive asynchronous function to finish the challenge.