QUESTION 1

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Price Regex Tester</title>
<style>
body {
font-family: Arial, sans-serif;
max-width: 800px;
margin: 0 auto;
padding: 20px;
}
h1, h2 {
color: #333;
}
.description {
background-color: #f5f5f5;
padding: 15px;
border-radius: 5px;
margin-bottom: 20px;
}
.regex-container {
background-color: #f0f8ff;
padding: 15px;
border-radius: 5px;
font-family: monospace;
margin-bottom: 20px;
}
table {
width: 100%;
border-collapse: collapse;
margin-top: 20px;
}
th, td {
padding: 10px;
text-align: left;
border-bottom: 1px solid #ddd;
}
th {
background-color: #f2f2f2;
}
.valid {
```

```
color: green;
font-weight: bold;
}
.invalid {
color: red;
font-weight: bold;
}
.test-input {
margin-top: 30px;
padding: 15px;
background-color: #f5f5f5;
border-radius: 5px;
}
input[type="text"] {
padding: 8px;
width: 200px;
margin-right: 10px;
}
button {
padding: 8px 15px;
background-color: #4CAF50;
color: white;
border: none;
cursor: pointer;
}
#result {
margin-top: 10px;
font-weight: bold;
}
</style>
</head>
<body>
<h1>Price Regular Expression</h1>
<div class="description">
<h2>Requirements:</h2>
<ul>
<li>Must start with a dollar sign ($)</li>
<li>Any amount of numbers can come before the decimal</li>
<li>Two numbers must always follow the decimal</li>
<li>No other characters allowed</li>
</ul>
</div>
<div class="regex-container">
<h2>The Regular Expression:</h2>
<code>/^\$\d*\.\d{2}$/</code>
</div>
<h2>Test Cases:</h2>
```

```
<table>
<tr>
<th>Price</th>
<th>Valid?</th>
<th>Explanation</th>
</tr>
<tr>
<td>$14.99</td>
<td class="valid">Valid</td>
<td>Starts with $, has digits before decimal, has 2 digits after decimal</td>
</tr>
<tr>
<td>$1234567.00</td>
<td class="valid">Valid</td>
<td>Multiple digits before decimal, 2 digits after decimal</td>
</tr>
<tr>
<td>$.90</td>
<td class="valid">Valid</td>
<td>Zero digits before decimal is allowed, 2 digits after decimal</td>
</tr>
<tr>
<td>$14</td>
<td class="invalid">Invalid</td>
<td>Missing decimal point and the 2 required digits after decimal</td>
</tr>
<tr>
<td>$134213.89money</td>
<td class="invalid">Invalid</td>
<td>Contains non-digit characters after the price</td>
</tr>
<tr>
<td>$1.1a</td>
<td class="invalid">Invalid</td>
<td>Only one digit after decimal and contains a non-digit</td>
</tr>
</table>
<div class="test-input">
<h2>Test Your Own Price:</h2>
<input type="text" id="price-input" placeholder="Enter a price...">
<button id="test-button">Test</button>
<div id="result"></div>
</div>

<script>
// The regular expression
const priceRegex = /^\$\d*\.\d{2}$/;
```

```
// Function to validate a price
function validatePrice(price) {
return priceRegex.test(price);
}
// Set up event listener for the test button
document.getElementById('test-button').addEventListener('click', function() {
const price = document.getElementById('price-input').value;
const result = validatePrice(price);
const resultElement = document.getElementById('result');
if (result) {
resultElement.innerHTML = `<span class="valid">Valid price format!</span>`;
} else {
resultElement.innerHTML = `<span class="invalid">Invalid price format!</span>`;
}
});
</script>
</body>
</html>
```

```
// Regular expression to match a price
// Requirements:
// 1. Must start with a dollar sign ($)
// 2. Any amount of numbers can come before the decimal
// 3. Two numbers must follow the decimal
// 4. No other characters allowed

// The regular expression
const priceRegex = /^\$\d*\.\d{2}$/;

// Test function to validate prices
function validatePrice(price) {
return priceRegex.test(price);
}

// Test cases
const testCases = [
{ price: '$14.99', expected: true },
{ price: '$1234567.00', expected: true },
{ price: '$.90', expected: true },
{ price: '$14', expected: false },
{ price: '$134213.89money', expected: false },
{ price: '$1.1a', expected: false },
{ price: '14.99', expected: false }, // Missing dollar sign
{ price: '$14.999', expected: false }, // Three digits after decimal
{ price: '$14.9', expected: false }, // One digit after decimal
```

```
{ price: 'price $14.99', expected: false } // Extra characters before
];

// Run tests and print results
console.log("Price Validation Tests:");
testCases.forEach(test => {
const result = validatePrice(test.price);
const status = result === test.expected ? 'PASS' : 'FAIL';
console.log(`${status}: "${test.price}" - Expected: ${test.expected}, Got: ${result}`);
});

// Explanation of the regex:
// ^ - Start of string
// \$ - Dollar sign (escaped because $ is a special character in regex)
// \d* - Zero or more digits (for before the decimal)
// \. - Decimal point (escaped because . is a special character in regex)
// \d{2} - Exactly 2 digits after the decimal
// $ - End of string

// This ensures:
// 1. The string starts with a dollar sign
// 2. It can have any number of digits before the decimal (including none)
// 3. It must have exactly 2 digits after the decimal
// 4. No other characters are allowed anywhere in the string
```

QUESTION 2

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Self-Closing HTML Tag Regex Tester</title>
<style>
body {
font-family: Arial, sans-serif;
max-width: 800px;
margin: 0 auto;
padding: 20px;
}
h1, h2 {
color: #333;
}
.description {
background-color: #f5f5f5;
padding: 15px;
border-radius: 5px;
margin-bottom: 20px;
}
.regex-container {
background-color: #f0f8ff;
padding: 15px;
border-radius: 5px;
font-family: monospace;
margin-bottom: 20px;
word-break: break-all;
}
table {
width: 100%;
border-collapse: collapse;
margin-top: 20px;
}
th, td {
padding: 10px;
text-align: left;
border-bottom: 1px solid #ddd;
}
th {
background-color: #f2f2f2;
}
.valid {
color: green;
```

```
font-weight: bold;
}
.invalid {
color: red;
font-weight: bold;
}
.test-input {
margin-top: 30px;
padding: 15px;
background-color: #f5f5f5;
border-radius: 5px;
}
input[type="text"] {
padding: 8px;
width: 70%;
margin-right: 10px;
}
button {
padding: 8px 15px;
background-color: #4CAF50;
color: white;
border: none;
cursor: pointer;
}
#result {
margin-top: 10px;
font-weight: bold;
}
</style>
</head>
<body>
<h1>Self-Closing HTML Tag Regular Expression</h1>
<div class="description">
<h2>Requirements:</h2>
<ul>
<li>Must start with <code>&lt;</code></li>
<li>Must have a tag name (must start with a letter)</li>
<li>Can have 0, 1, or more properties (attributes)</li>
<li>Must end with <code>/&gt;</code></li>
</ul>
</div>
<div class="regex-container">
<h2>The Regular Expression:</h2>
<code>&lt;([a-zA-Z][a-zA-Z0-9]*)(\s+[a-zA-Z][a-zA-Z0-9]*(?:=(?:"[^"]*"|'[^']*'|[^\s&gt;]*))?)*\s*/&gt;</
code>
</div>
<h2>Test Cases:</h2>
```

```
<table>
<tr>
<th>Tag</th>
<th>Valid?</th>
<th>Explanation</th>
</tr>
<tr>
<td>&lt;img src="foo.jpg" /&gt;</td>
<td class="valid">Valid</td>
<td>Starts with &lt;, has tag name (img), has attribute, ends with /&gt;</td>
</tr>
<tr>
<td>&lt;img /&gt;</td>
<td class="valid">Valid</td>
<td>Starts with &lt;, has tag name (img), no attributes, ends with /&gt;</td>
</tr>
<tr>
<td>&lt;a href="foo.html" id="stuff" /&gt;</td>
<td class="valid">Valid</td>
<td>Starts with &lt;, has tag name (a), has multiple attributes, ends with /&gt;</td>
</tr>
<tr>
<td>&lt;/&gt;</td>
<td class="invalid">Invalid</td>
<td>Missing tag name</td>
</tr>
<tr>
<td>&lt;img src= /&gt;</td>
<td class="invalid">Invalid</td>
<td>Attribute missing value</td>
</tr>
<tr>
<td>&lt;img src="foo.jpg"&gt;</td>
<td class="invalid">Invalid</td>
<td>Not a self-closing tag (missing the /)</td>
</tr>
</table>
<div class="test-input">
<h2>Test Your Own HTML Tag:</h2>
<input type="text" id="tag-input" placeholder="Enter an HTML tag...">
<button id="test-button">Test</button>
<div id="result"></div>
</div>

<script>
// The regular expression for self-closing HTML tags
```

```
const selfClosingTagRegex = /<([a-zA-Z][a-zA-Z0-9]*)(\s+[a-zA-Z][a-zA-Z0-9]*(?:=(?:"[^"]*"|'[^']*'|
[^\s>]*))?)*\s*\/>/;
// Function to validate a self-closing HTML tag
function validateSelfClosingTag(tag) {
return selfClosingTagRegex.test(tag);
}
// Set up event listener for the test button
document.getElementById('test-button').addEventListener('click', function() {
const tag = document.getElementById('tag-input').value;
const result = validateSelfClosingTag(tag);
const resultElement = document.getElementById('result');
if (result) {
resultElement.innerHTML = `<span class="valid">Valid self-closing HTML tag!</span>`;
} else {
resultElement.innerHTML = `<span class="invalid">Invalid self-closing HTML tag!</span>`;
}
});
</script>
</body>
</html>
```

```
// Regular expression to match a self-closing HTML tag
// Requirements:
// 1. Must start with "<"
// 2. Must have a tag name
// 3. Can have 0, 1, or more properties
// 4. Must end with "/>"
// 5. Don't need to worry about whitespace inside the tag

// The regular expression
const selfClosingTagRegex = /<([a-zA-Z][a-zA-Z0-9]*)(\s+[a-zA-Z][a-zA-Z0-9]*(?:=(?:"[^"]*"|'[^']*'|
[^\s>]*))?)*\s*\/>/;

// Test function to validate self-closing HTML tags
function validateSelfClosingTag(tag) {
return selfClosingTagRegex.test(tag);
}

// Test cases
const testCases = [
{ tag: '<img src="foo.jpg" />', expected: true },
{ tag: '<img />', expected: true },
{ tag: '<a href="foo.html" id="stuff" />', expected: true },
{ tag: '</>', expected: false },
{ tag: '<img src= />', expected: false },
```

```
{ tag: '<1img />', expected: false }, // Invalid tag name starting with number
{ tag: '<img src="foo.jpg">', expected: false }, // Not self-closing
{ tag: '<img src="foo.jpg', expected: false }, // Missing closing />
{ tag: 'img src="foo.jpg" />', expected: false }, // Missing opening <
{ tag: '<img src="foo.jpg" / >', expected: false } // Space between / and >
];

// Run tests and print results
console.log("Self-Closing HTML Tag Validation Tests:");
testCases.forEach(test => {
const result = validateSelfClosingTag(test.tag);
const status = result === test.expected ? 'PASS' : 'FAIL';
console.log(`${status}: "${test.tag}" - Expected: ${test.expected}, Got: ${result}`);
});

// Explanation of the regex parts:
// < - Matches opening angle bracket
// ([a-zA-Z][a-zA-Z0-9]*) - Captures tag name (must start with letter, can include letters and numbers)
// ( )* - Group for attributes (0 or more)
// \s+ - At least one whitespace character
// [a-zA-Z][a-zA-Z0-9]* - Attribute name (must start with letter)
// (?: )? - Optional attribute value
// = - Equals sign
// (?: ) - Value options (one of the following)
// "[^"]*" - Double-quoted value
// |'[^']*' - Single-quoted value
// |[^\s>]* - Unquoted value (no whitespace or >)
// \s* - Optional whitespace before closing
// \/> - Self-closing tag end (/ followed by >)
```

QUESTION 3

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>To-Do List</title>
<script src="todo-list.js" type="text/javascript"></script>
<style>
body {
font-family: Arial, sans-serif;
max-width: 600px;
margin: 0 auto;
padding: 20px;
}
h1 {
color: #333;
}
ul {
list-style-type: disc;
padding-left: 40px;
}
li {
padding: 10px;
margin: 5px 0;
border: 1px solid #ddd;
position: relative;
}
input[type="text"] {
padding: 8px;
width: 70%;
margin-right: 10px;
}
button {
padding: 8px 15px;
background-color: #4CAF50;
color: white;
border: none;
cursor: pointer;
margin-right: 5px;
}
button:hover {
background-color: #45a049;
}
#remove {
background-color: #f44336;
```

```
}
#remove:hover {
background-color: #d32f2f;
}
</style>
</head>
<body>
<h1>My super nifty to-do list</h1>
<ul id="list"></ul>
<div>
<input type="text" id="item" />
<button id="add">add</button>
<button id="remove">remove</button>
</div>
</body>
</html>
```

```
// JavaScript code for to-do list functionality

// Wait for the document to be fully loaded
document.addEventListener('DOMContentLoaded', function() {
// Get references to all required elements
const listElement = document.getElementById('list');
const itemInput = document.getElementById('item');
const addButton = document.getElementById('add');
const removeButton = document.getElementById('remove');
// Add button event listener
addButton.addEventListener('click', function() {
const itemText = itemInput.value.trim();
// Only add if the input is not empty
if (itemText !== '') {
// Create a new list item
const newItem = document.createElement('li');
newItem.textContent = itemText;
// Set background color based on alternating pattern
// Count current items to determine if this should be white or yellow
const currentItems = listElement.getElementsByTagName('li');
if (currentItems.length % 2 === 0) {
// Even number of current items, so new one is white (first item)
newItem.style.backgroundColor = 'white';
} else {
// Odd number of current items, so new one is yellow (second item)
newItem.style.backgroundColor = 'yellow';
}
// Add to the list
```

```
listElement.appendChild(newItem);
// Clear the input field
itemInput.value = '';
}
});
// Remove button event listener
removeButton.addEventListener('click', function() {
const itemText = itemInput.value.trim().toLowerCase();
// Only proceed if the input is not empty
if (itemText !== '') {
// Get all list items
const items = listElement.getElementsByTagName('li');
// Find the first occurrence of the item (case insensitive)
let foundIndex = -1;
for (let i = 0; i < items.length; i++) {
if (items[i].textContent.toLowerCase() === itemText) {
foundIndex = i;
break;
}
}
// If found, remove it
if (foundIndex !== -1) {
listElement.removeChild(items[foundIndex]);
// Update background colors to maintain alternating pattern
updateBackgroundColors();
}
// Clear the input field
itemInput.value = '';
}
});
// Function to update background colors after removing an item
function updateBackgroundColors() {
const items = listElement.getElementsByTagName('li');
// Reset all background colors according to alternating pattern
for (let i = 0; i < items.length; i++) {
if (i % 2 === 0) {
// Even index (0, 2, 4...) - white background
items[i].style.backgroundColor = 'white';
} else {
// Odd index (1, 3, 5...) - yellow background
items[i].style.backgroundColor = 'yellow';
}
}
}
});
```

QUESTION 4

```
<html lang="en">
<head>
<script src="sesame-street.js" type="text/javascript"></script>
<script src="fetch-pie.js" type="text/javascript"></script>
</head>
<body>
<h1>Whitaker's Desserts</h1>
<h2 id="cookie-header">Whitaker's Cookie Jar:</h2>
<ul id="cookie-jar">
<li class="cookie">Chocolate Chip</li>
<li class="cookie">Oatmeal raisin</li>
<li class="cookie">Macaroon</li>
</ul>
<p id="cookie-count"></p>
<h2>Whitaker's Pie Cupboard:</h2>
<ul id="pie-cupboard">
</ul>
<button id="moar-pie">Moar Pie!</button>
</body>
</html>
```

```
// sesame-street.js - Implementation for cookie-related functionality

// Wait for the document to be fully loaded
document.addEventListener('DOMContentLoaded', function() {
// Big Bird Yellow: Apply styling to the cookie header when page loads
const cookieHeader = document.getElementById('cookie-header');
if (cookieHeader) {
// Apply yellow styling (Big Bird's color)
cookieHeader.style.backgroundColor = '#FFDF00'; // Big Bird yellow
cookieHeader.style.color = '#000000'; // Black text for contrast
cookieHeader.style.padding = '10px';
cookieHeader.style.borderRadius = '5px';
cookieHeader.style.boxShadow = '0 2px 4px rgba(0, 0, 0, 0.2)';
}
// Count Chocula: Update the cookie count display
const cookieJar = document.getElementById('cookie-jar');
const cookieCount = document.getElementById('cookie-count');
function updateCookieCount() {
if (cookieJar && cookieCount) {
const cookies = cookieJar.getElementsByClassName('cookie');
const numCookies = cookies.length;
```

```
// Set the formatted text as requested
cookieCount.textContent = `${numCookies}! There are ${numCookies} cookie(s) in the cookie jar!`;
// Set the text color to the specified hex value (#f7f16d - a light yellow color)
cookieCount.style.color = '#f7f16d';
// Add Count Chocula-inspired styling
cookieCount.style.fontWeight = 'bold';
cookieCount.style.fontFamily = 'cursive, fantasy';
cookieCount.style.textShadow = '1px 1px 2px #000';
cookieCount.style.padding = '8px';
cookieCount.style.backgroundColor = '#5c2d91'; // Purple background for Count Chocula
cookieCount.style.borderRadius = '5px';
cookieCount.style.display = 'inline-block';
// If all cookies are gone, add a message
if (numCookies === 0) {
cookieCount.textContent += " Cookie Monster ate them all! Om nom nom!";
}
}
}
// Initial count update
updateCookieCount();
// Cookie Monster: Remove a cookie every 30 seconds - using 3 seconds for testing
const cookieInterval = setInterval(function() {
// Check if there are still cookies in the jar
if (cookieJar && cookieJar.getElementsByClassName('cookie').length > 0) {
// Remove the last cookie from the jar
const lastCookie = cookieJar.getElementsByClassName('cookie')
[cookieJar.getElementsByClassName('cookie').length - 1];
if (lastCookie && lastCookie.parentNode) {
cookieJar.removeChild(lastCookie);
// Update the cookie count display
updateCookieCount();
console.log("Cookie removed! Remaining cookies:",
cookieJar.getElementsByClassName('cookie').length);
}
} else {
// If no cookies left, we can clear the interval (optional)
// clearInterval(cookieInterval);
console.log("No more cookies to remove!");
}
}, 30000);
});
```

```
// fetch-pie.js - Implementation for pie-related functionality

document.addEventListener('DOMContentLoaded', function() {
// Get references to the button and pie cupboard
const moarPieButton = document.getElementById('moar-pie');
const pieCupboard = document.getElementById('pie-cupboard');
// Array of pie types
const pieTypes = [
'Apple Pie',
'Cherry Pie',
'Blueberry Pie',
'Pumpkin Pie',
'Pecan Pie',
'Lemon Meringue Pie',
'Key Lime Pie',
'Chocolate Cream Pie',
'Coconut Cream Pie',
'Banana Cream Pie'
];
// Add event listener to the Moar Pie! button
if (moarPieButton) {
moarPieButton.addEventListener('click', function() {
// Get a random pie from the array
const randomIndex = Math.floor(Math.random() * pieTypes.length);
const randomPie = pieTypes[randomIndex];
// Create a new list item for the pie
const pieItem = document.createElement('li');
pieItem.textContent = randomPie;
pieItem.className = 'pie';
// Add the pie to the cupboard
if (pieCupboard) {
pieCupboard.appendChild(pieItem);
}
});
}
});
```