Introduction To Full-Stack Web Development

CS 386

Michael Kremer

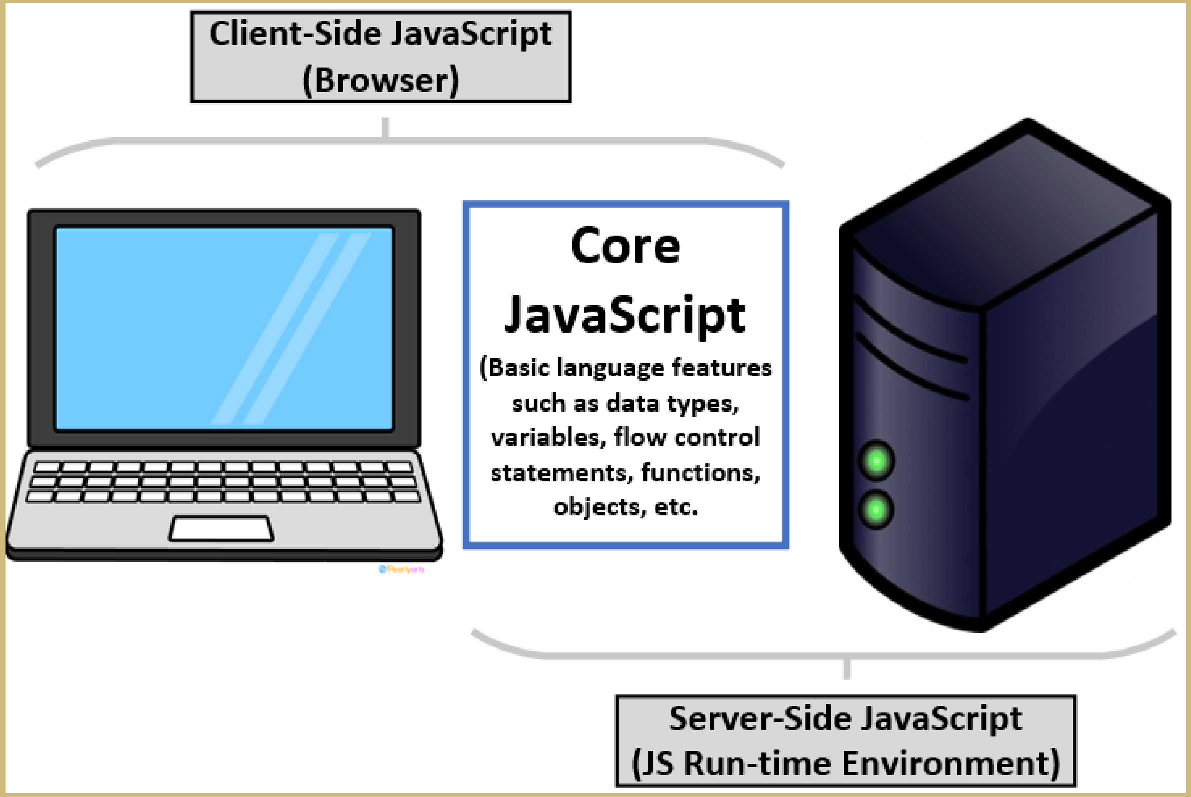
Last updated: 10/8/2023 9:19:35 AM

- 13.1 Introduction to Client-Side JavaScript
- 13.2 Document Object Model (DOM Tree)
- 13.3 Selecting Elements

Class 13

13.1 Introduction to Client-Side JavaScript

- Client Side JavaScript (CSJS) → Extended version of JavaScript
- Enables enhancement and manipulation of web pages and client browsers
- In browser environment , code will have access to “things” provided only by browser:
  - ❑ Document object of current page
  - ❑ Window object
  - ❑ Functions like alert that pop up message, etc.


Client-Side JavaScript (Browser)
Core JavaScript
(Basic language features such as data types, variables, flow control statements, functions, objects, etc.
Server-Side JavaScript (JS Run-time Environment)

13.1 Introduction to Client-Side JavaScript

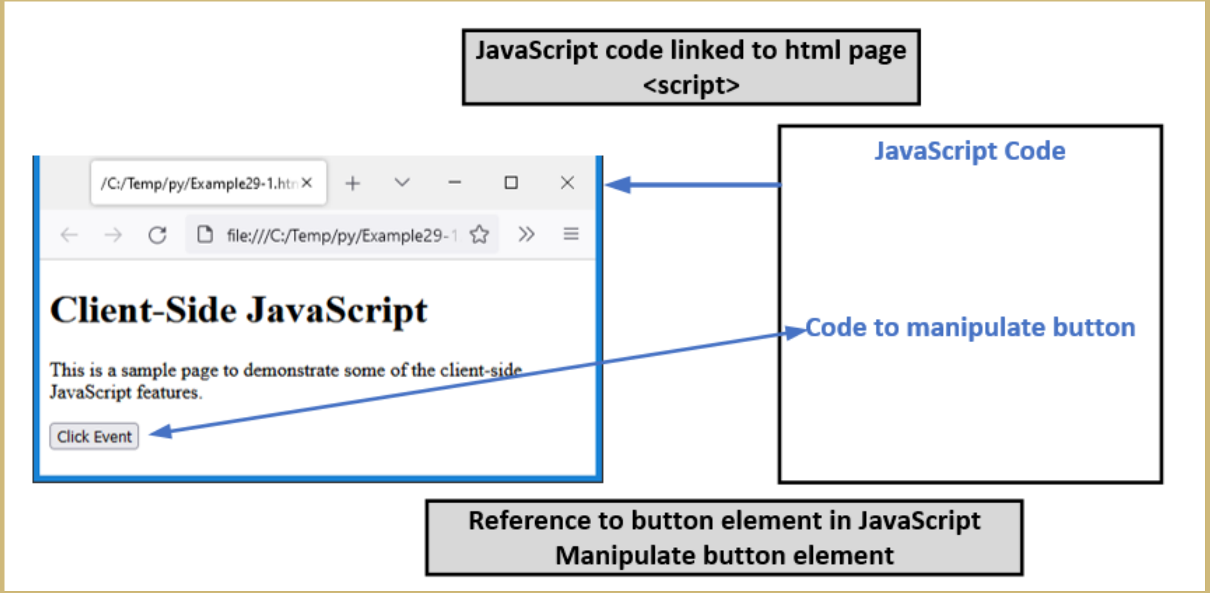
- Main tasks of Client side JavaScript are:
  - ❑ Validating input
  - ❑ Animation
  - ❑ Manipulating UI elements
  - ❑ Applying/Overriding styles (CSS)
  - ❑ Calculations without communicating web back-end server
- In web development it is browser, in user's machine, that runs this code (JavaScript)

13.1 Introduction to Client-Side JavaScript

- Executing JavaScript in HTML
  - Link JavaScript code to html page using script tag (<script>)

Syntax:
<script src = "*js file*" **type=**"text/javascript"**></script>**

  - Create reference to html element and store reference in variable
  - Variable now allows to manipulate element in any way



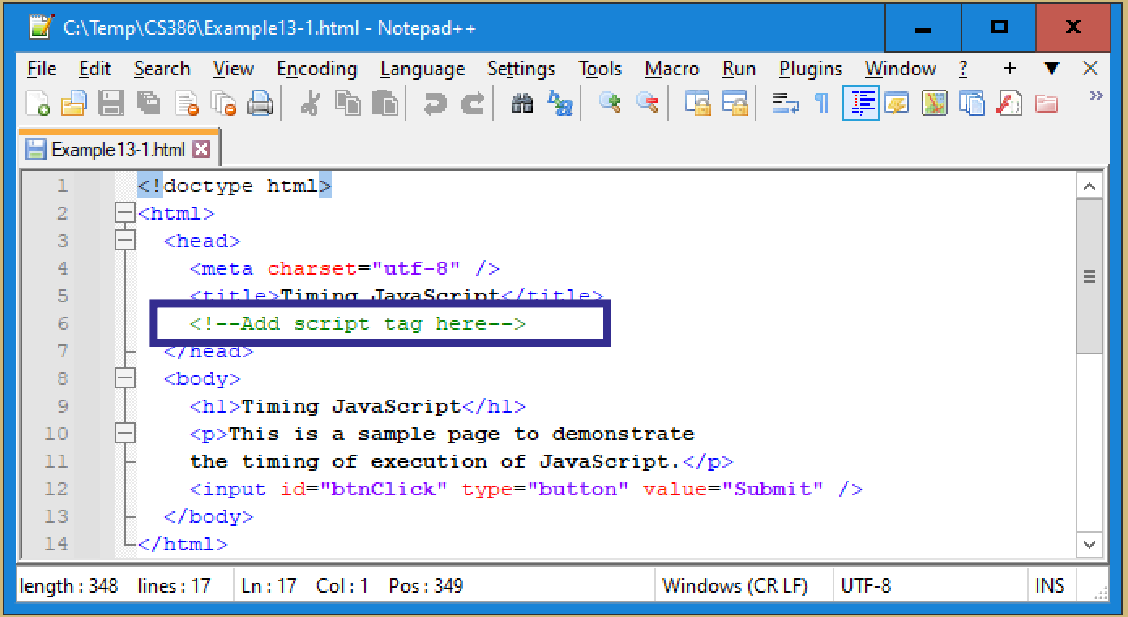
JavaScript code linked to html page
<script>
/C:/Temp/py/Example29-1.htm
file:///C:/Temp/py/Example29-1
Client-Side JavaScript
This is a sample page to demonstrate some of the client-side JavaScript features.
Click Event
JavaScript Code
Code to manipulate button
Reference to button element in JavaScript
Manipulate button element

13.1 Introduction to Client-Side JavaScript

- Timing of executing JavaScript
  - <script> tag is in head section:
    - Blocks html parsing
    - JavaScript code is executed before html page is parsed/executed
    - Notice ending script tag even though there is no content! (historical reasons)
  - Any reference created in JavaScript to html element will fail:
    - No html page yet
    - No elements created yet
  - → Need to wait for JavaScript execution until html page is fully parsed

13.1 Introduction to Client-Side JavaScript

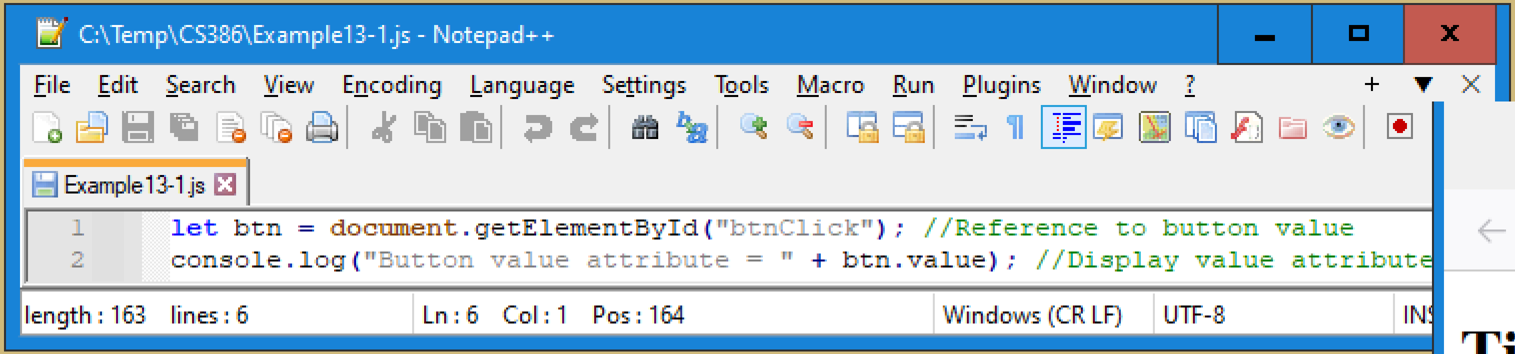
- **Example 13-1:**
  - Demonstration of JavaScript execution timing
  - Download html files in zip file from Canvas under Class 13 page
  - In Example13-1.html add script tag in head section to load file Example13-1.js
  - Create JavaScript file Example13-1.js
  - Create reference to button element:
    - let btn = document.getElementById("btnClick");
  - Display value attribute:
    - console.log("Button value attribute = " + btn.value);


C:\Temp\CS386\Example13-1.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example13-1.html
length : 348 lines : 17 Ln : 17 Col : 1 Pos : 349 Windows (CR LF) UTF-8 INS

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Timing JavaScript</title>
    <!--Add script tag here-->
  </head>
  <body>
    <h1>Timing JavaScript</h1>
    <p>This is a sample page to demonstrate
    the timing of execution of JavaScript.</p>
    <input id="btnClick" type="button" value="Submit" />
  </body>
</html>
```

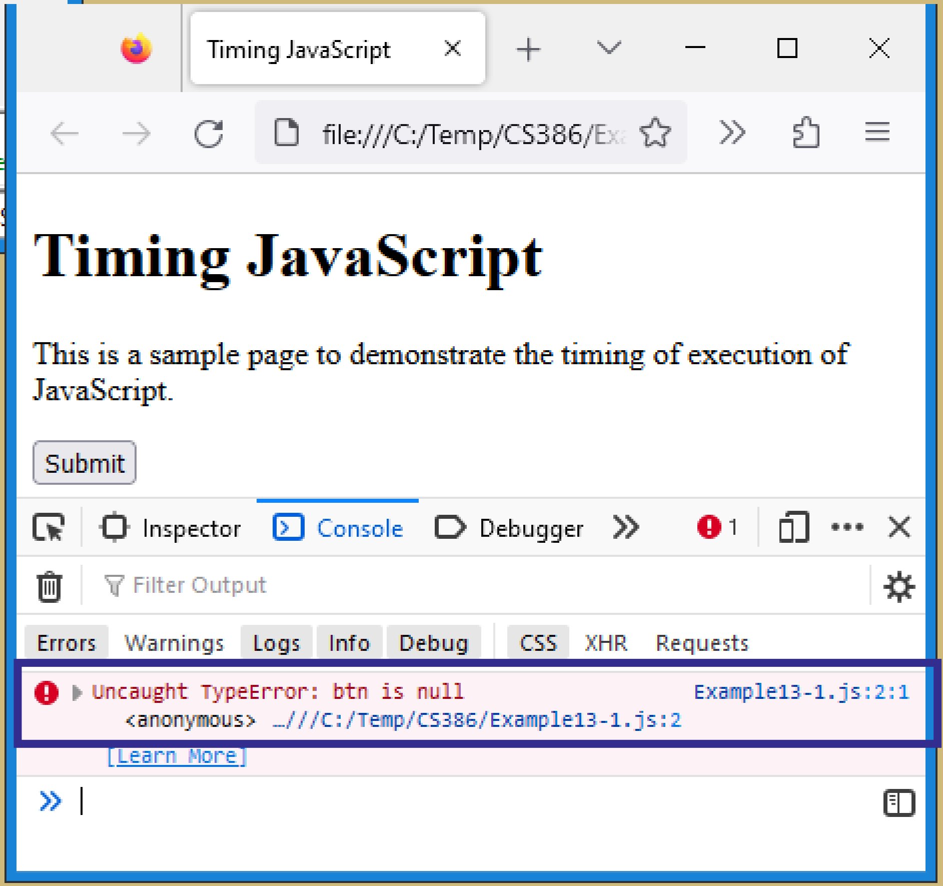
13.1 Introduction to Client-Side JavaScript

- **Example 13-1:**



```
let btn = document.getElementById("btnClick"); //Reference to button value
console.log("Button value attribute = " + btn.value); //Display value attribute
```

```
length : 163    lines : 6    Ln : 6    Col : 1    Pos : 164    Windows (CR LF)    UTF-8
```



Timing JavaScript

This is a sample page to demonstrate the timing of execution of JavaScript.

```
Uncaught TypeError: btn is null                       Example13-1.js:2:1
    <anonymous> …///C:/Temp/CS386/Example13-1.js:2
[Learn More]
```

- Notice error in console:
- btn is null → button does not exist yet!

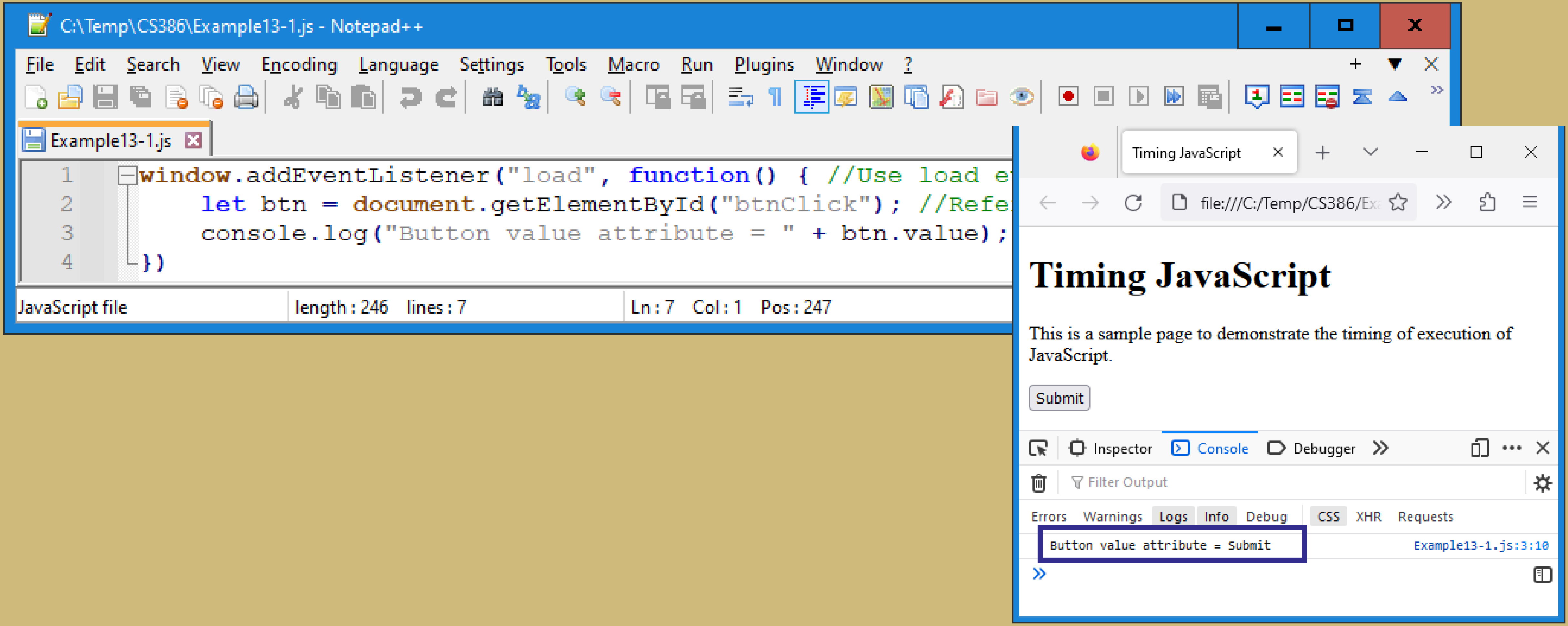
13.1 Introduction to Client-Side JavaScript

- **<u>Example 13-1 (continued):</u>**
- Somehow we need to wait with JavaScript execution until html page is fully parsed
- Use load event on window object in JavaScript:
  - Wrap previous code around load event
  - window.addEventListener("load", function() {

    ```
      previous code
    })
    ```
  - Function here is anonymous function!
  - No function name

13.1 Introduction to Client-Side JavaScript

➢ **Example 13-1 (continued):**



C:\Temp\CS386\Example13-1.js - Notepad++

```
window.addEventListener("load", function() { //Use load e
    let btn = document.getElementById("btnClick"); //Refe
    console.log("Button value attribute = " + btn.value);
})
```

JavaScript file   length : 246   lines : 7   Ln : 7   Col : 1   Pos : 247

Timing JavaScript

file:///C:/Temp/CS386/Ex

Timing JavaScript

This is a sample page to demonstrate the timing of execution of JavaScript.

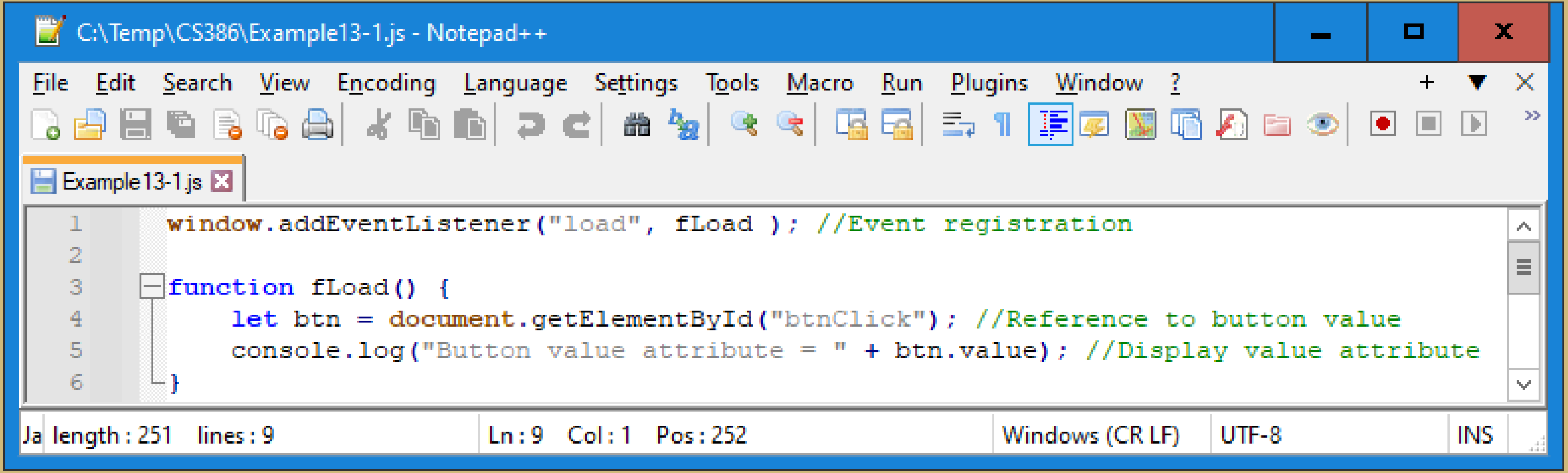
Submit

```
Button value attribute = Submit                    Example13-1.js:3:10
```

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example13-1.js
Inspector
Console
Debugger
Filter Output
Errors Warnings Logs Info Debug CSS XHR Requests

13.1 Introduction to Client-Side JavaScript

- **Example 13-1 (continued):**
- Better yet, create separate function to execute code on load event
- Set reference to function in addEventListener
- **IMPORTANT:** Do not invoke function in event registration → no parentheses

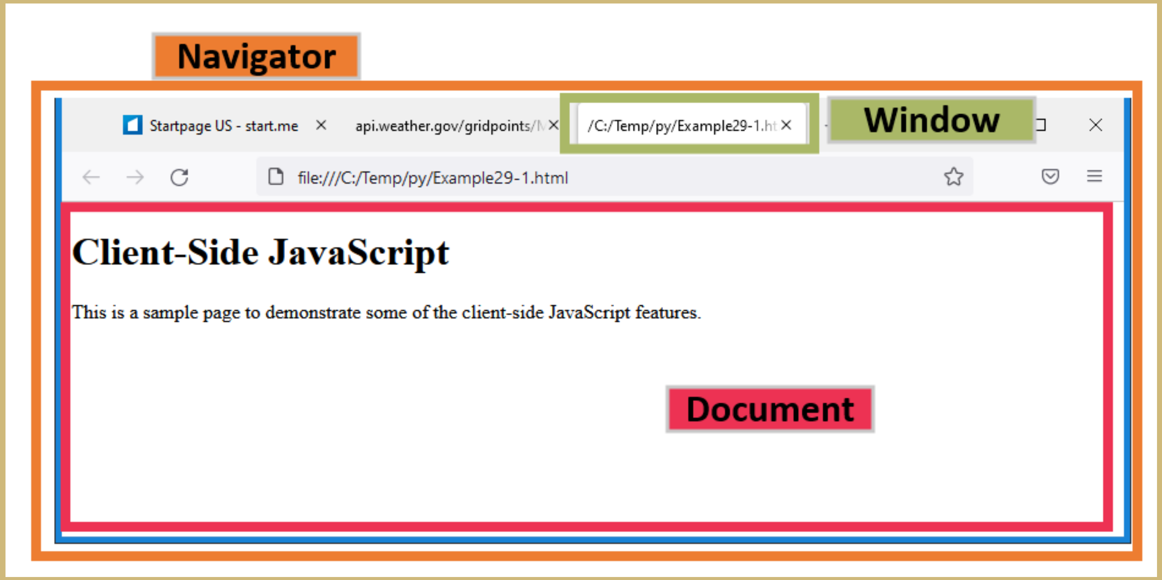

C:\Temp\CS386\Example13-1.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example13-1.js

```
window.addEventListener("load", fLoad ); //Event registration

function fLoad() {
    let btn = document.getElementById("btnClick"); //Reference to button value
    console.log("Button value attribute = " + btn.value); //Display value attribute
}
```

```
Ja length : 251    lines : 9    Ln : 9    Col : 1    Pos : 252    Windows (CR LF)    UTF-8    INS
```

13.2 Document Object Model (DOM Tree)

- When writing web pages and apps:
  - One of most common things to do is manipulate document structure in some way
  - Usually done by using Document Object Model (DOM):
    - Set of APIs for controlling HTML and styling information that makes heavy use of Document object
- Main parts of browser directly involved in viewing web pages:
  - Navigator object: Overall browser
  - Window object: Browser window or tab
  - Document object: HTML page


Navigator
Window
Startpage US - start.me
api.weather.gov/gridpoints/
/C:/Temp/py/Example29-1.ht
file:///C:/Temp/py/Example29-1.html
Document

Client-Side JavaScript

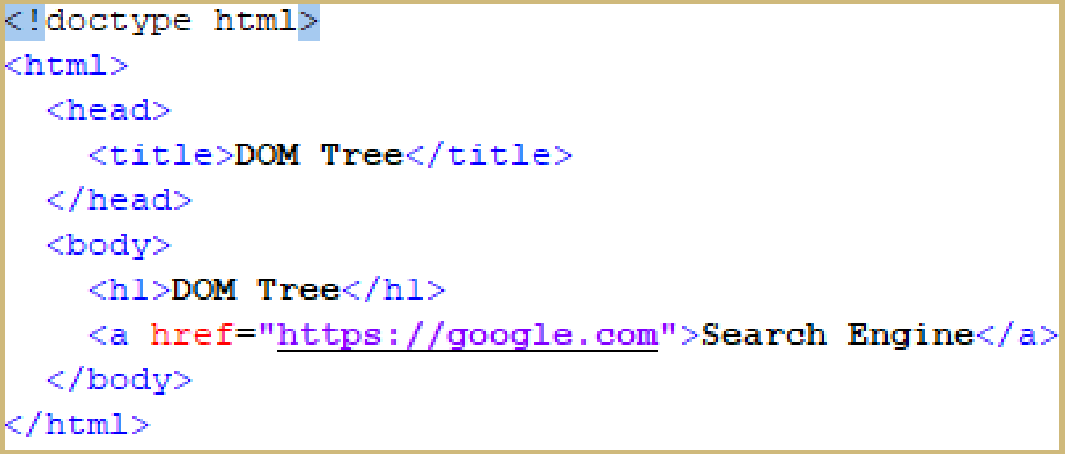
This is a sample page to demonstrate some of the client-side JavaScript features.

13.2 Document Object Model (DOM Tree)

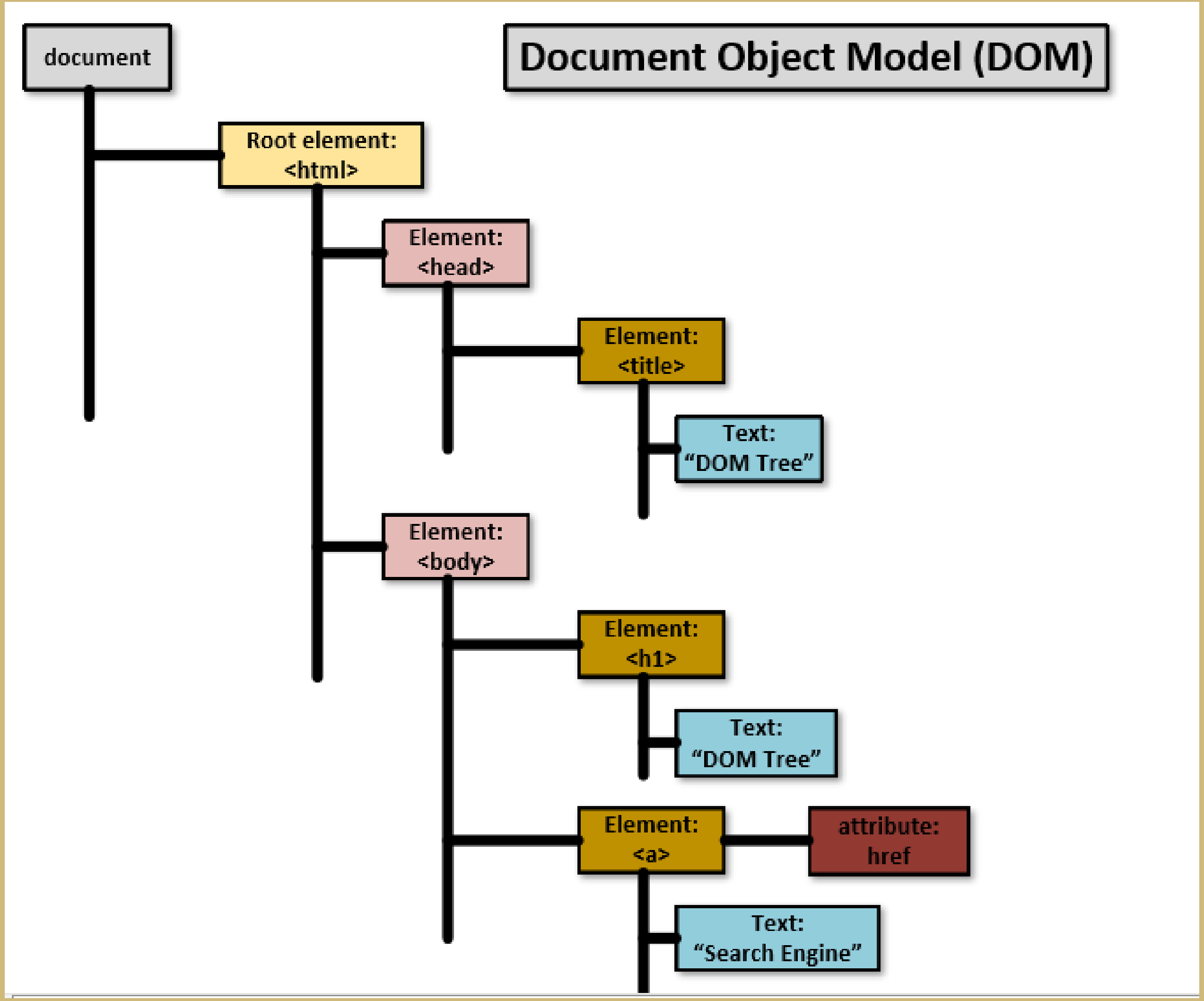
- Document Object Model (DOM) is application programming interface (API) for manipulating HTML documents
- DOM represents HTML document as tree of nodes
- DOM provides functions that allows to add, remove, and modify parts of document effectively
- Use Live DOM Viewer to explore DOM tree
- https://software.hixie.ch/utilities/js/live-dom-viewer/

13.2 Document Object Model (DOM Tree)

- DOM represents HTML document as hierarchy of nodes:



```
<!doctype html>
<html>
  <head>
    <title>DOM Tree</title>
  </head>
  <body>
    <h1>DOM Tree</h1>
    <a href="https://google.com">Search Engine</a>
  </body>
</html>
```



13.2 Document Object Model (DOM Tree)

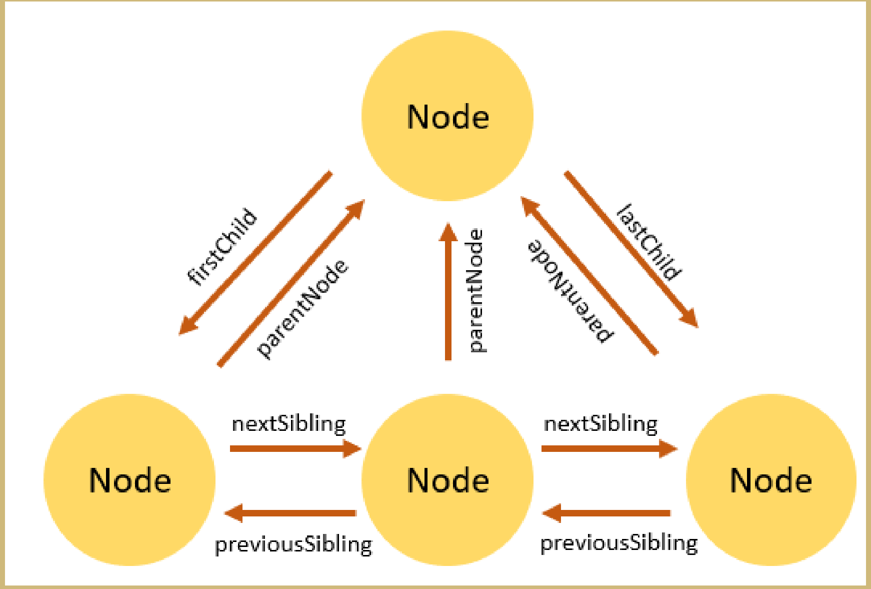
- Tree structures in computer programming borrow terminology from family trees:
  - Node directly above another node is parent of that node
  - Nodes one level directly below another node are children of that node
  - Nodes at same level, and having same parent, are siblings
  - Nodes any number of levels below another node (child, grandchild, etc.) are descendants of that node
  - Nodes any number of levels above another node (parent, grandparent, etc.) are ancestors of that node
- Two APIs to access DOM tree:
  - Tree of Nodes: Accesses all nodes, elements and corresponding text nodes
  - Tree of Elements: Accesses only element nodes

13.2 Document Object Model (DOM Tree)

API to Navigate DOM: Tree of Nodes

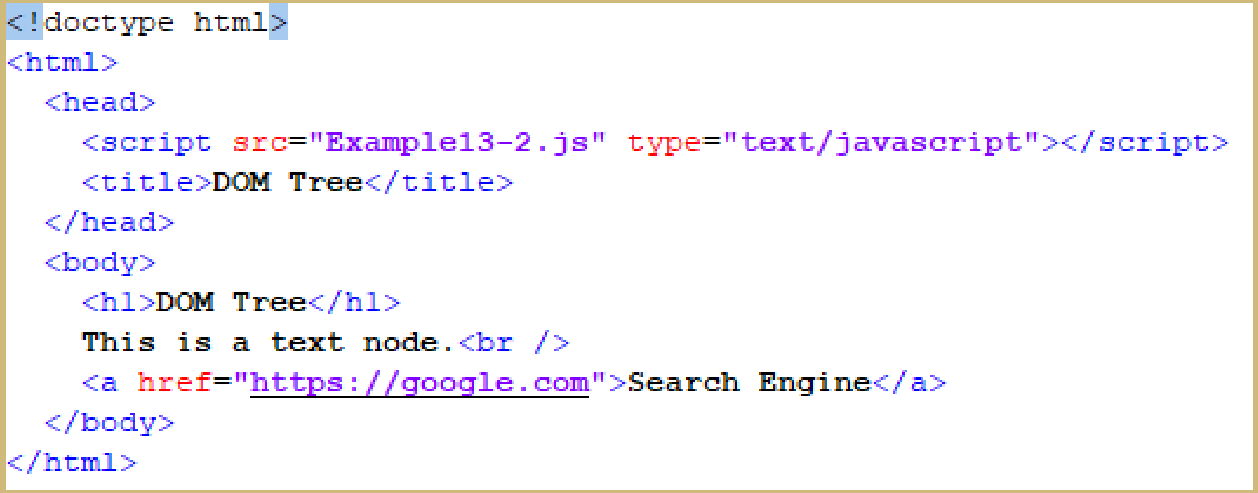
- Accesses all node objects:
  - Document object
  - Its Element objects
  - Text objects that represent runs of text in document
- Node object defines following important properties:

Properties	Description
parentNode	Node that is the parent of this one, or null for nodes like the Document object that have no parent
childNodes	Read-only array-like object (NodeList) that is a live representation of a Node's child nodes
firstChild, lastChild	First and last child nodes of a node, or null if the node has no children
nextSibling, previousSibling	The next and previous sibling node of a node (Two nodes having same parent are siblings, their order reflects the order in which they appear in the document)
nodeType	Document nodes = 9, Element nodes = 1, Text nodes =3, Comments nodes =8
nodeValue	The textual content of a Text or Comment node
nodeName	The tag name of an Element, converted to uppercase



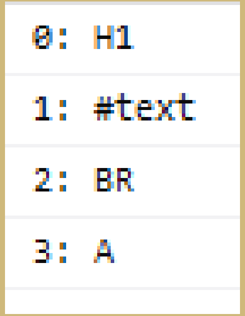
13.2 Document Object Model (DOM Tree)

- **<u>Example 13-2:</u>**
- Use file Example13-2.html
- Create JavaScript file Example13-2.js
- Create window.addEventListener using load event and reference function fLoopDOM (see below)



```
<!doctype html>
<html>
  <head>
    <script src="Example13-2.js" type="text/javascript"></script>
    <title>DOM Tree</title>
  </head>
  <body>
    <h1>DOM Tree</h1>
    This is a text node.<br />
    <a href="https://google.com">Search Engine</a>
  </body>
</html>
```

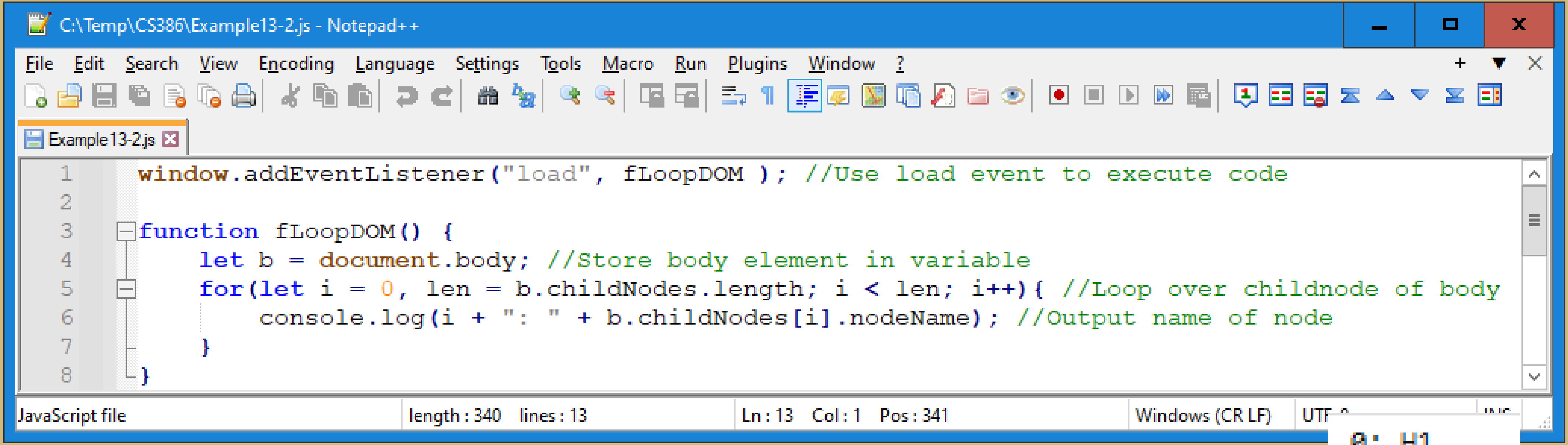
- Create function fLoopDOM:
  - ❑ Create variable b to set reference to body element of html page using document object
  - ❑ Using for loop:
    - o Use iteration variable i
    - o Loop over body (using b) using childNodes property (= array, can use length property)
  - ❑ Display in console iteration variable and nodeName



```
0: H1
1: #text
2: BR
3: A
```

13.2 Document Object Model (DOM Tree)

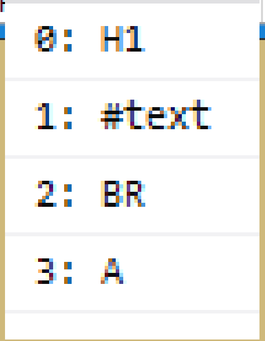
- **Example 13-2:**



C:\Temp\CS386\Example13-2.js - Notepad++

```
window.addEventListener("load", fLoopDOM ); //Use load event to execute code

function fLoopDOM() {
    let b = document.body; //Store body element in variable
    for(let i = 0, len = b.childNodes.length; i < len; i++){ //Loop over childnode of body
        console.log(i + ": " + b.childNodes[i].nodeName); //Output name of node
    }
}
```

JavaScript file   length : 340   lines : 13   Ln : 13   Col : 1   Pos : 341   Windows (CR LF)   UTF-8



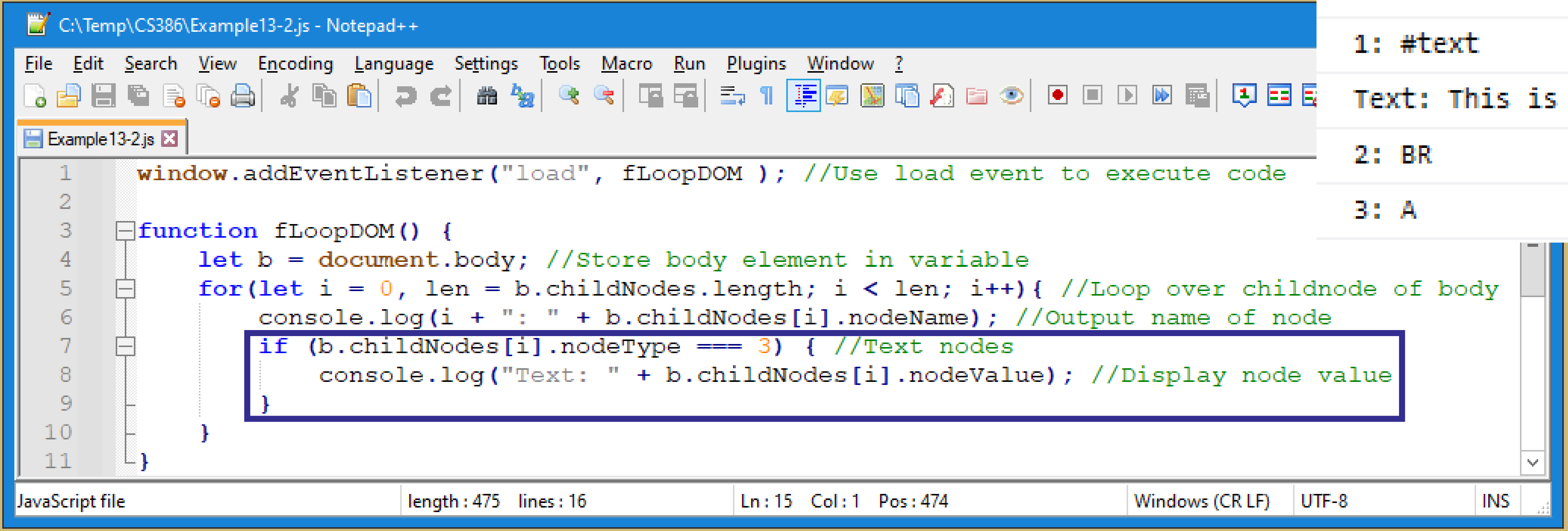
```
0: H1
1: #text
2: BR
3: A
```

- **Note:** More text nodes than expected, should be only one!
- Blank spaces, line breaks are also text nodes
- Put entire html body content into one line

```
<body><h1>DOM Tree</h1>This is a text node.<br /><a href="https://google.com">Search Engine</a></body></html>
```

13.2 Document Object Model (DOM Tree)

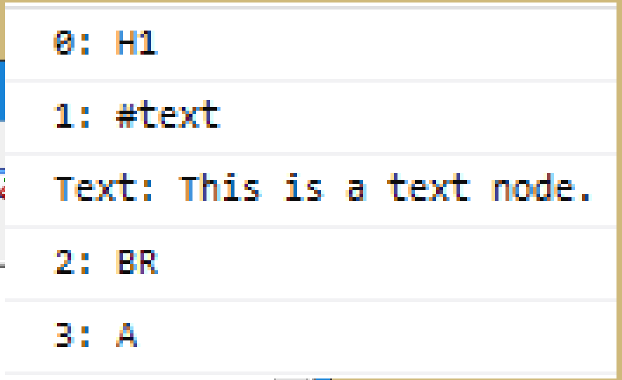
- **Example 13-2(continued):**
- For text nodes, display nodeValue property



C:\Temp\CS386\Example13-2.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example13-2.js

```javascript
window.addEventListener("load", fLoopDOM ); //Use load event to execute code

function fLoopDOM() {
    let b = document.body; //Store body element in variable
    for(let i = 0, len = b.childNodes.length; i < len; i++){ //Loop over childnode of body
        console.log(i + ": " + b.childNodes[i].nodeName); //Output name of node
        if (b.childNodes[i].nodeType === 3) { //Text nodes
            console.log("Text: " + b.childNodes[i].nodeValue); //Display node value
        }
    }
}
```



```
0: H1
1: #text
Text: This is a text node.
2: BR
3: A
```

- **Note:** There are more text nodes, such as text within anchor element
- But this is nested, use recursive function calls to drill into hierarchy/DOM tree

13.2 Document Object Model (DOM Tree)

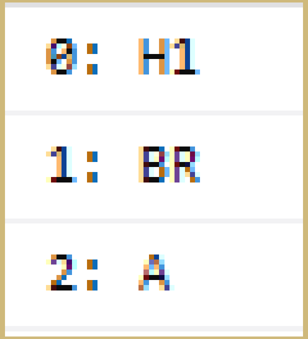
API to Navigate DOM: Tree of Elements

- Simpler API to loop over elements only (excluding text & comment nodes)
- Children property of Element:
  - Read-only children property returns HTMLCollection (live)
  - Contains all child elements of element upon which it was called
- Other properties:

Properties	Description
childElementCount	Returns an elements's number of child elements
firstElementChild	Returns the first child element of an element
lastElementChild	Returns the last child element of an element
nextElementSibling	Returns the next element at the same node tree level
parentElement	Returns the parent element node of an element
previousElementSibling	Returns the previous element at the same node tree level

13.2 Document Object Model (DOM Tree)

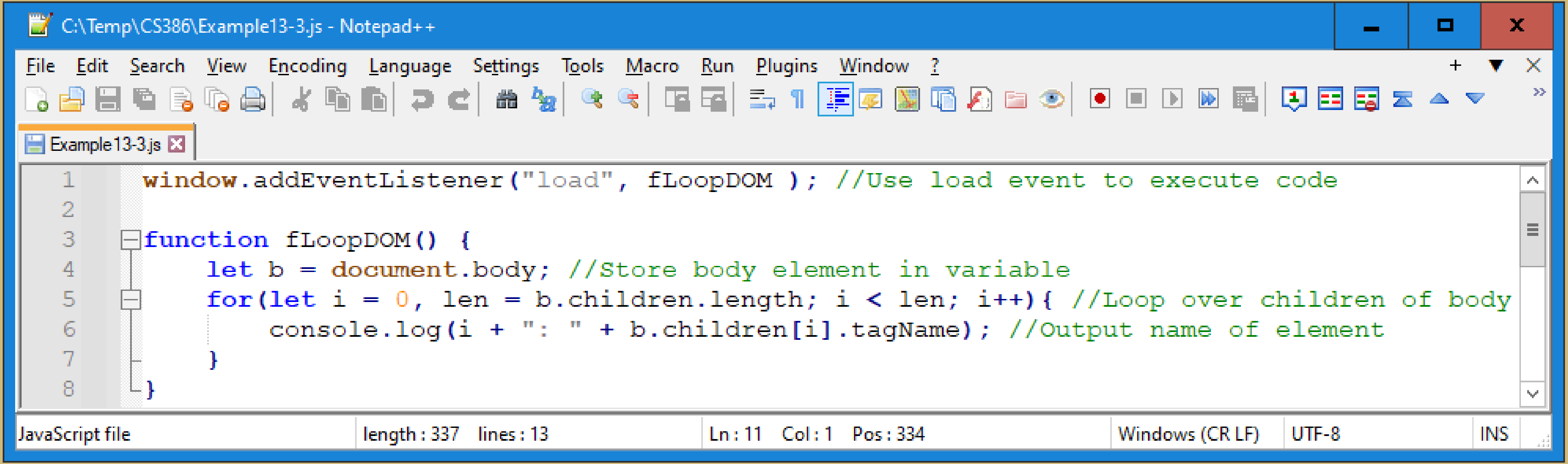
- **Example 13-3:**
- Based on previous example
- Use for loop to loop over children of body
- Display iteration variable and tagName property in console



```
0: H1
1: BR
2: A
```

13.2 Document Object Model (DOM Tree)

- **Example 13-3:**



C:\Temp\CS386\Example13-3.js - Notepad++

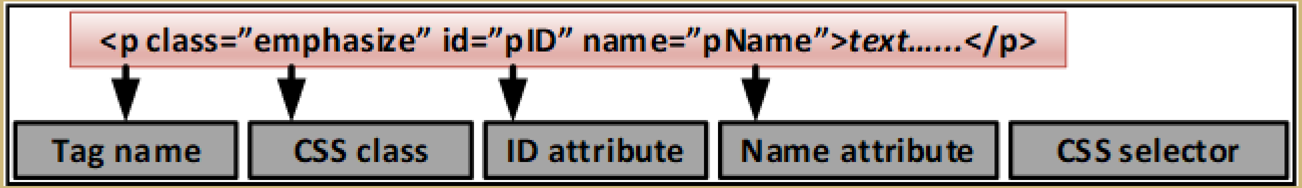
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example13-3.js

```
window.addEventListener("load", fLoopDOM ); //Use load event to execute code

function fLoopDOM() {
    let b = document.body; //Store body element in variable
    for(let i = 0, len = b.children.length; i < len; i++){ //Loop over children of body
        console.log(i + ": " + b.children[i].tagName); //Output name of element
    }
}
```

JavaScript file | length : 337 lines : 13 | Ln : 11 Col : 1 Pos : 334 | Windows (CR LF) | UTF-8 | INS

13.3 Selecting Elements

- Most client-side JavaScript programs work by somehow manipulating one or more document elements
- In order to manipulate elements of document:
  - Must obtain or select Element objects that refer to those document elements
  - DOM defines number of ways to select elements (can query document for one or more elements):
    - with specified id attribute (document only)
    - with specified name attribute (document only)
    - with specified tag name (document or element)
    - with specified CSS class or classes (document or element)
    - matching specified CSS selector (document or element)



13.3 Selecting Elements

- Selecting Elements by ID
  - Any HTML element can have id attribute
  - Value of this attribute must be unique within document (no two elements in same document can have same ID)
  - Select element based on this unique ID with getElementById() method of Document object

    Syntax:
    let *el* **= document.getElementById("***id value***")**

  - Reference to element is object with many properties and methods
  - **Note:** This method only exists on document object

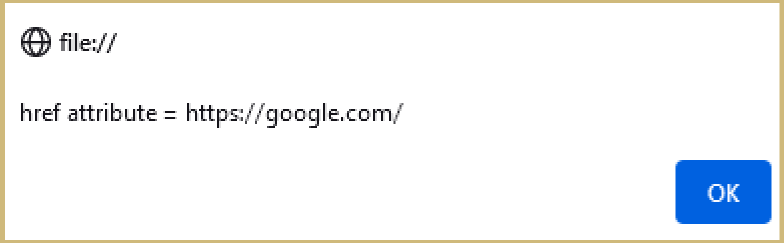
13.3 Selecting Elements

- Selecting Elements by Name
  - HTML name attribute was originally intended to assign names to form elements
  - Value of this attribute is used when form data is submitted to server
  - Like id attribute, name assigns name value to element
  - Unlike id, value of name attribute does not have to be unique:
    - Multiple elements may have same name
    - Common in case of radio buttons and checkboxes in forms
  - **Note:** Unlike id, name attribute is only valid on handful of HTML elements, including forms, form elements, <iframe>, and <img> elements
  - To select HTML elements based on value of their name attributes, use getElementsByName() method of Document object (only)
  - **IMPORTANT:** Note the plural s!
    - Returns NodeList (static) object that behaves like read-only array of Element objects

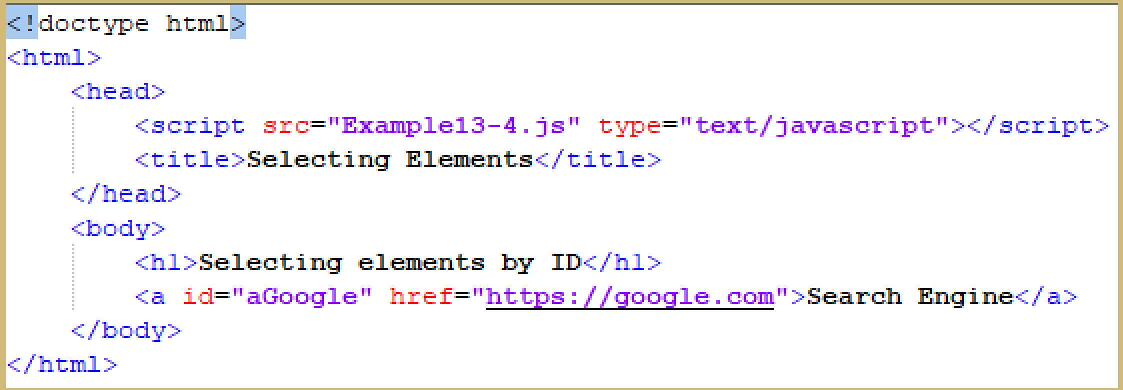
Syntax:
let *els* **= document.getElementsByName("***name value***")**

13.3 Selecting Elements

- **Example 13-4:**
  - Use file Example13-4.html
  - Create JavaScript file Example13-4.js
  - Set load event to fSelElements
  - Create function fSelElements:
    - Select anchor element by using id value
    - Assign into variable anchor
    - Display href attribute in alert



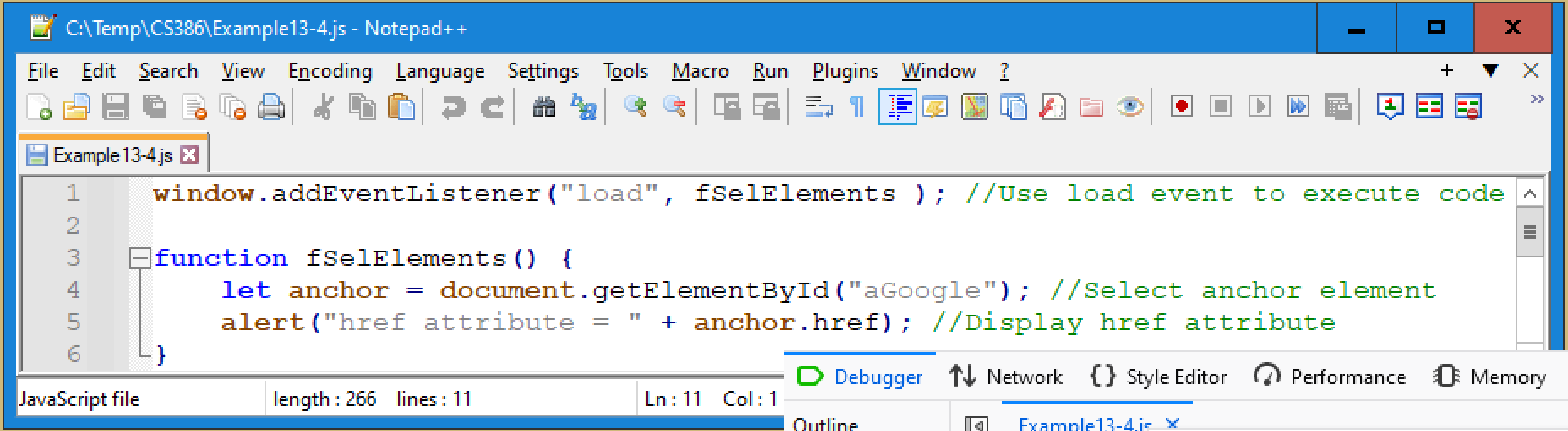
```
file://

href attribute = https://google.com/

OK
```



```html
<!doctype html>
<html>
    <head>
        <script src="Example13-4.js" type="text/javascript"></script>
        <title>Selecting Elements</title>
    </head>
    <body>
        <h1>Selecting elements by ID</h1>
        <a id="aGoogle" href="https://google.com">Search Engine</a>
    </body>
</html>
```

13.3 Selecting Elements

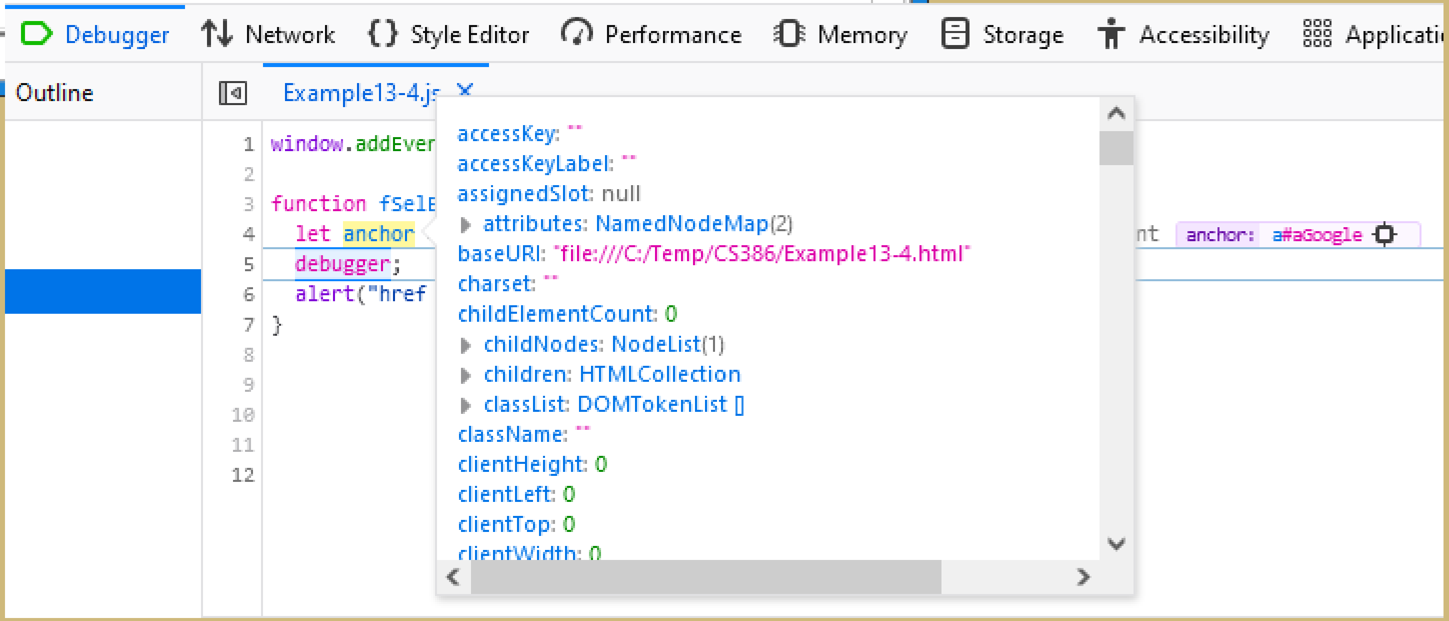
- **Example 13-4:**



C:\Temp\CS386\Example13-4.js - Notepad++

```
window.addEventListener("load", fSelElements ); //Use load event to execute code

function fSelElements() {
    let anchor = document.getElementById("aGoogle"); //Select anchor element
    alert("href attribute = " + anchor.href); //Display href attribute
}
```

JavaScript file   length : 266   lines : 11   Ln : 11   Col : 1



Example13-4.js

```
window.addEver
function fSelE
  let anchor
  debugger;
  alert("href
}
```

```
accessKey: ""
accessKeyLabel: ""
assignedSlot: null
▸ attributes: NamedNodeMap(2)
baseURI: "file:///C:/Temp/CS386/Example13-4.html"
charset: ""
childElementCount: 0
▸ childNodes: NodeList(1)
▸ children: HTMLCollection
▸ classList: DOMTokenList []
className: ""
clientHeight: 0
clientLeft: 0
clientTop: 0
clientWidth: 0
```

anchor: a#aGoogle

- Put debugger statement in function before alert
- In console, hover over anchor
- Notice the properties and methods

13.3 Selecting Elements

- Selecting Elements by Tag Name(Type)
  - Select all HTML elements of specified type (or tag name) using getElementsByTagName() method of Document or Element object
  - Returns NodeList object

Syntax:
let *els* **= document.getElementsByTagName(“**tag**”)**
Or
let *els* **=** *element***.getElementsByTagName(“**tag**”)**

  - Elements of returned NodeList are in document order
  - Example:
    - Can select the first <p> element in document like this:
    - let firstparagraph = document.getElementsByTagName("p")[0];
  - **Note:** HTML tags are case-insensitive
  - Element class also defines getElementsByTagName() method
  - Works in same way as Document version:
    - But only selects elements that are descendants of element on which it is invoked

13.3 Selecting Elements

- Selecting Elements by CSS Class
  - Class attribute of HTML element is a space-separated list of zero or more identifiers
  - Describes way to define sets of related document elements:
    - Any elements that have same identifier in their class attribute are part of same set
  - Class attribute is usually used in conjunction with CSS stylesheet to apply same presentation styles to all members of set
  - HTML5 defines method, getElementsByClassName():
    - Allows to select sets of document elements based on identifiers in their class attribute
  - Can be used on document or element

Syntax:
let *els* **= document.getElementsByClassName(“***class name***”)**
Or
let *els* **=** *element***.getElementsByClassName(“***class name***”)**

13.3 Selecting Elements

➢ **Example 13-5:**

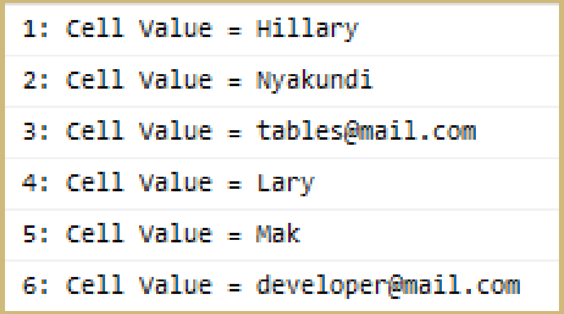
❑ Use file Example13-5.html

❑ Create JavaScript file Example13-5.js

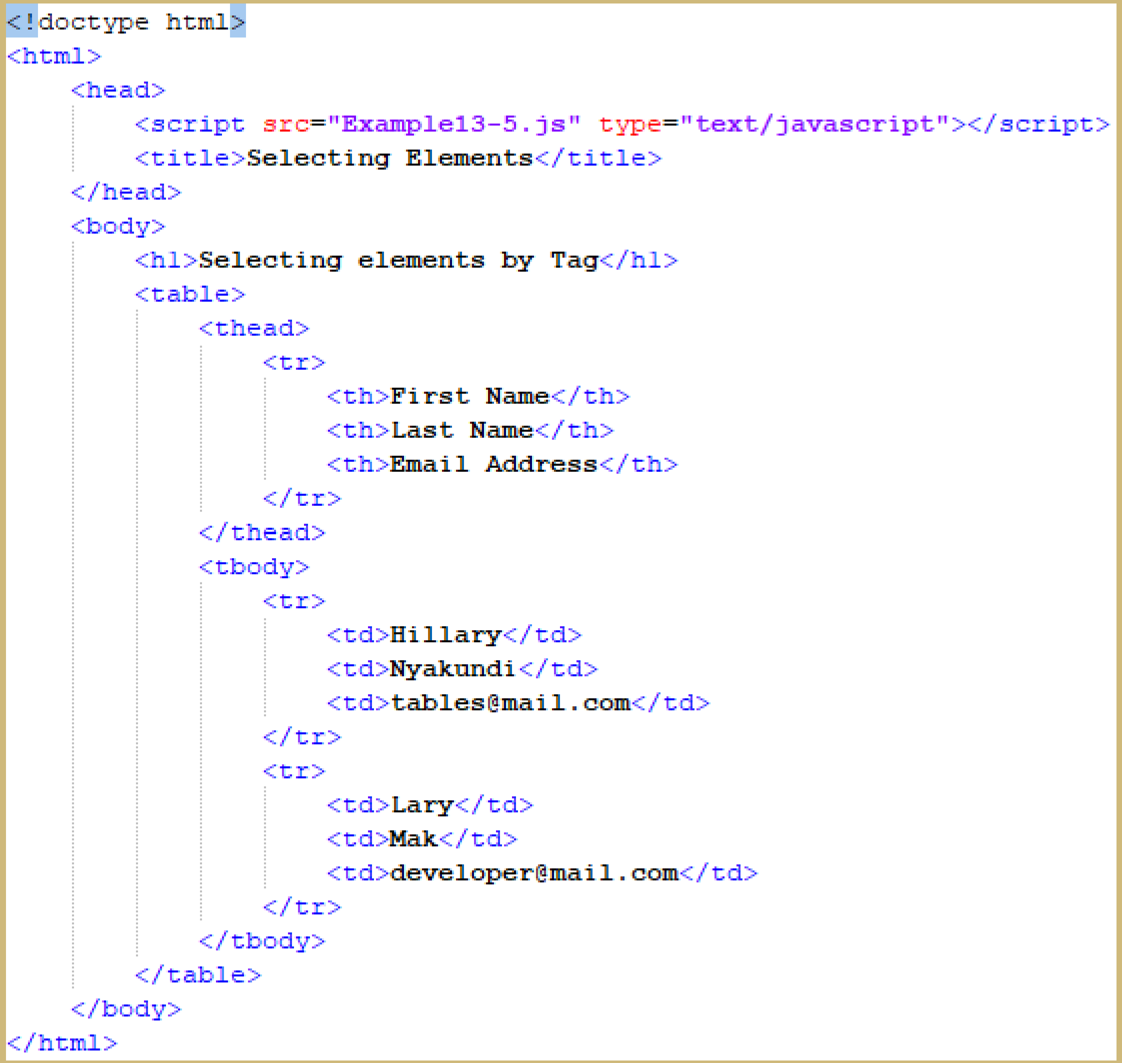
❑ Set load event to function fSelCells

❑ Create function fSelCells:

  - Create variable table assigning reference to table element using getElementsByTagName
  - Create variable tds
  - Assign nodelist of td elements
  - Output i incremented by 1 and cell value (innerHTML)



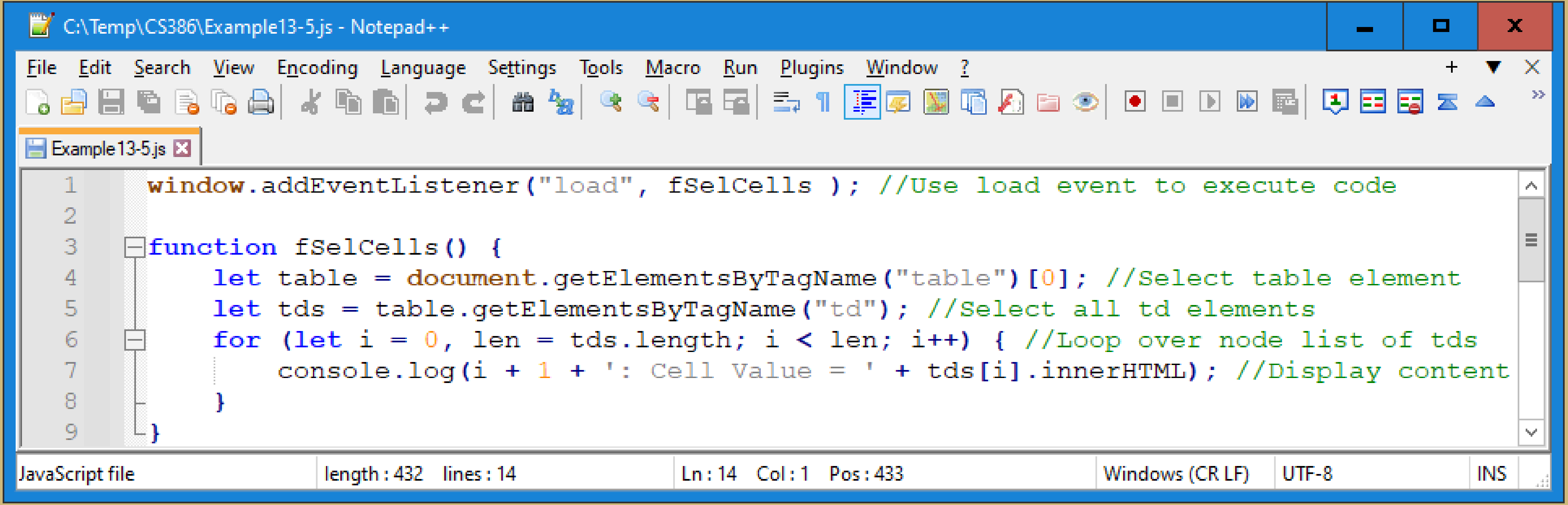
```
1: Cell Value = Hillary
2: Cell Value = Nyakundi
3: Cell Value = tables@mail.com
4: Cell Value = Lary
5: Cell Value = Mak
6: Cell Value = developer@mail.com
```



```
<!doctype html>
<html>
    <head>
        <script src="Example13-5.js" type="text/javascript"></script>
        <title>Selecting Elements</title>
    </head>
    <body>
        <h1>Selecting elements by Tag</h1>
        <table>
            <thead>
                <tr>
                    <th>First Name</th>
                    <th>Last Name</th>
                    <th>Email Address</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>Hillary</td>
                    <td>Nyakundi</td>
                    <td>tables@mail.com</td>
                </tr>
                <tr>
                    <td>Lary</td>
                    <td>Mak</td>
                    <td>developer@mail.com</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```

13.3 Selecting Elements

➢ **Example 13-5:**


C:\Temp\CS386\Example13-5.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example13-5.js

```
window.addEventListener("load", fSelCells ); //Use load event to execute code

function fSelCells() {
    let table = document.getElementsByTagName("table")[0]; //Select table element
    let tds = table.getElementsByTagName("td"); //Select all td elements
    for (let i = 0, len = tds.length; i < len; i++) { //Loop over node list of tds
        console.log(i + 1 + ': Cell Value = ' + tds[i].innerHTML); //Display content
    }
}
```

JavaScript file length : 432 lines : 14 Ln : 14 Col : 1 Pos : 433 Windows (CR LF) UTF-8 INS

13.3 Selecting Elements

- Selecting Elements by CSS Selectors
  - CSS stylesheets have very powerful syntax, known as selectors, for describing elements or sets of elements within document
  - CSS selectors allow elements to be selected in all of ways described below:
    - ID
    - Name
    - Tag name
    - Class name
  - New HTML5 JavaScript API methods for obtaining elements that match given CSS selector
  - One of main reason why jQuery library was developed:
    - To implement CSS selector syntax in JavaScript to select elements in same way as in CSS

13.3 Selecting Elements

- Selecting Elements by CSS Selectors
  - Key to this API is method querySelectorAll() (document or element)

    <u>Syntax:</u>
    let *els* **= document.querySelectorAll(“***CSS selector syntax***”)**
    Or
    let *els* **=** *element***.querySelectorAll(“***CSS selector syntax***”)**

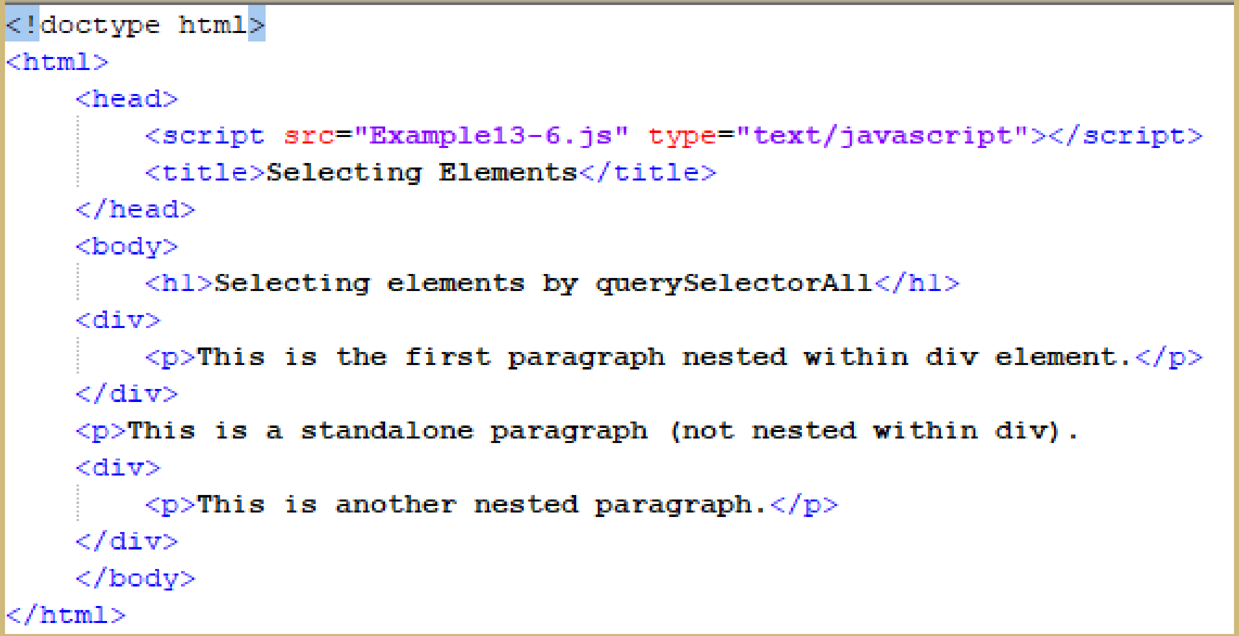
  - querySelectorAll() exception cases:
    - If no elements match, querySelectorAll() returns empty NodeList
    - If selector string is invalid, querySelectorAll() throws exception
  - Also defines querySelector():
    - Returns only first (in document order) matching element or null if there is no matching element

      <u>Syntax:</u>
      let *el* **= document.querySelector(“***CSS selector syntax***”)**
      Or
      let *el* **=** *element***.querySelector(“***CSS selector syntax***”)**

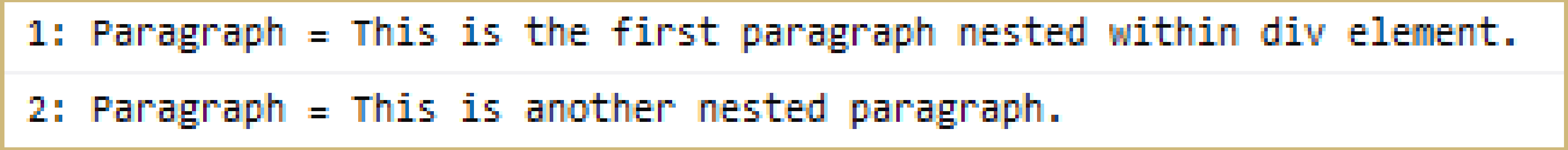
13.3 Selecting Elements

➢ **Example 13-6:**

- ❑ Use file Example13-6.html
- ❑ Create JavaScript file Example13-6.js
- ❑ Set load event to function fSelAll
- ❑ Create function fSelAll:
  - o Create variable divp
  - o Assign all paragraph elements that are nested within div elements using querySelectorAll
  - o Loop over divp and display i incremented by one and paragraph value (innerHTML)



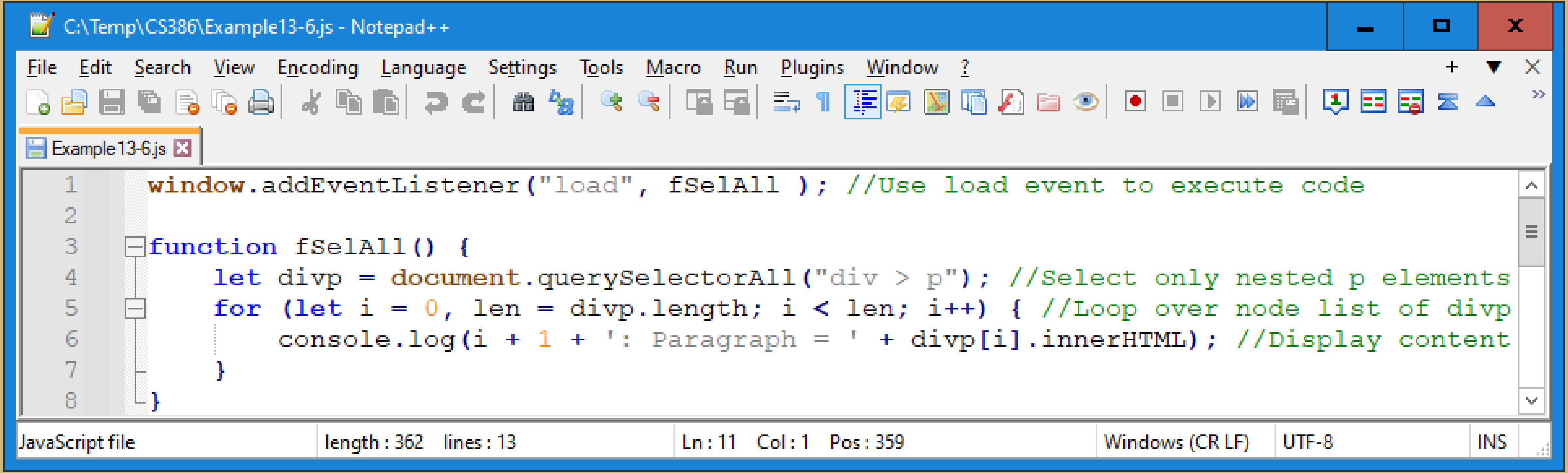
```html
<!doctype html>
<html>
    <head>
        <script src="Example13-6.js" type="text/javascript"></script>
        <title>Selecting Elements</title>
    </head>
    <body>
        <h1>Selecting elements by querySelectorAll</h1>
    <div>
        <p>This is the first paragraph nested within div element.</p>
    </div>
    <p>This is a standalone paragraph (not nested within div).
    <div>
        <p>This is another nested paragraph.</p>
    </div>
    </body>
</html>
```



```
1: Paragraph = This is the first paragraph nested within div element.
2: Paragraph = This is another nested paragraph.
```

13.3 Selecting Elements

- **Example 13-6:**


C:\Temp\CS386\Example13-6.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example13-6.js

```
window.addEventListener("load", fSelAll ); //Use load event to execute code

function fSelAll() {
    let divp = document.querySelectorAll("div > p"); //Select only nested p elements
    for (let i = 0, len = divp.length; i < len; i++) { //Loop over node list of divp
        console.log(i + 1 + ': Paragraph = ' + divp[i].innerHTML); //Display content
    }
}
```

JavaScript file
length : 362 lines : 13
Ln : 11 Col : 1 Pos : 359
Windows (CR LF)
UTF-8
INS