

Introduction To Full-Stack Web Development

CS 386

Michael Kremer

Last updated: 10/11/2023 8:06:51 AM





- 14.1 Scripting CSS
- 14.2 Animations
- 14.3 Events

Class 14

14.1 Scripting CSS

- CSS is presentation layer of client-side interface, use Stylesheet to apply formats
- CSS is also of interest to client-side JavaScript programmers because CSS styles can be scripted
- Scripted CSS enables variety of interesting visual effects:
 - ❑ Create animated transitions where document content “slides in” from the right
 - ❑ Create expanding and collapsing outline list where user can control amount of information that is displayed
- Three main ways how to script CSS styles:
 - ❑ Inline styles (CSS style attribute at element level)
 - ❑ CSS class (CSS class attribute at element level)
 - ❑ CSS stylesheet (either inline or external stylesheet)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>CSS Scripting Options</title>
  <!--Scripting CSS style entire style sheet-->
  <link rel="stylesheet" type="text/css" href="Stylesheet.css">
  <script defer type = "text/javascript" src="JavaScript.js"></script>
</head>
<body>
  <h1>CSS Scripting Options</h1>
  <!--Inline style set in html, not recommended-->
  <div id="divFirst" style="border: 1px solid black">First Div</div>
  <br /><!--Scripting CSS style using attribute class-->
  <div class="style" id="divSecond">Second Div</div>
</body>
</html>
```

The diagram illustrates three methods of applying CSS styles in HTML:

- External CSS Stylesheet:** Indicated by an arrow pointing to the `<link rel="stylesheet" type="text/css" href="Stylesheet.css">` line in the `<head>` section.
- Inline CSS Style:** Indicated by an arrow pointing to the `style="border: 1px solid black"` attribute on the `<div id="divFirst">` element.
- HTML Attribute Class:** Indicated by an arrow pointing to the `class="style"` attribute on the `<div class="style" id="divSecond">` element.

14.1 Scripting CSS

Scripting Inline Styles

- Remember, CSS order of precedence (specificity): Inline style has highest priority
- style is property of Element object → can manipulate it in JavaScript
- style property is unusual: Value is not string, but CSSStyleDeclaration object
- **Note:** No CSS semicolon inside property value, but outside (JavaScript semicolon)
- **WARNING:**
 - ❑ Many CSS style properties contain hyphens in their names
 - ❑ In JavaScript, hyphen is interpreted as minus sign, not possible to write expression like:
 - `e.style.font-size = "24pt";` // Syntax error!
 - ❑ Names of properties of CSSStyleDeclaration object are slightly different from names of actual CSS properties
 - ❑ CSS font-size → JavaScript fontSize

Syntax:

```
let e = html element;  
e.style.color = "color_value";
```

14.1 Scripting CSS

Scripting Inline Styles

- All CSS property values in JavaScript are strings
- All positioning values require units
- Can also use shortcut properties in JavaScript
- Other methods of setting CSS attributes:
 - ❑ `setAttribute` (general method for any attributes)
 - ❑ `cssText`
 - ❑ CSS strings can contain multiple CSS properties and values
- Once you have set these values in JavaScript, you can also read those values
- To read CSS attributes:
 - ❑ `getAttribute` (general method for any attributes)
 - ❑ `cssText` with no argument

Syntax:

```
e.setAttribute("style", s);  
e.style.cssText = s;
```

Syntax:

```
s = e.getAttribute("style");  
s = e.style.cssText;
```


14.1 Scripting CSS

➤ Example 14-1:

- ❑ Download Class14_Files.zip from Canvas
- ❑ Create JavaScript file Example14-1.js
- ❑ Use `window.addEventListener("load", fCSS)`
- ❑ Create function `fCSS`
- ❑ In function `fCSS`, style the following elements:
 - First paragraph:
 - font size to 18 pt using style property
 - H1 header:
 - Create variable `strCSS` and assign css string:
 - Blue font color, light gray background
 - Use `setAttribute` method to set style to `strCSS`
 - Display h1 css settings in console

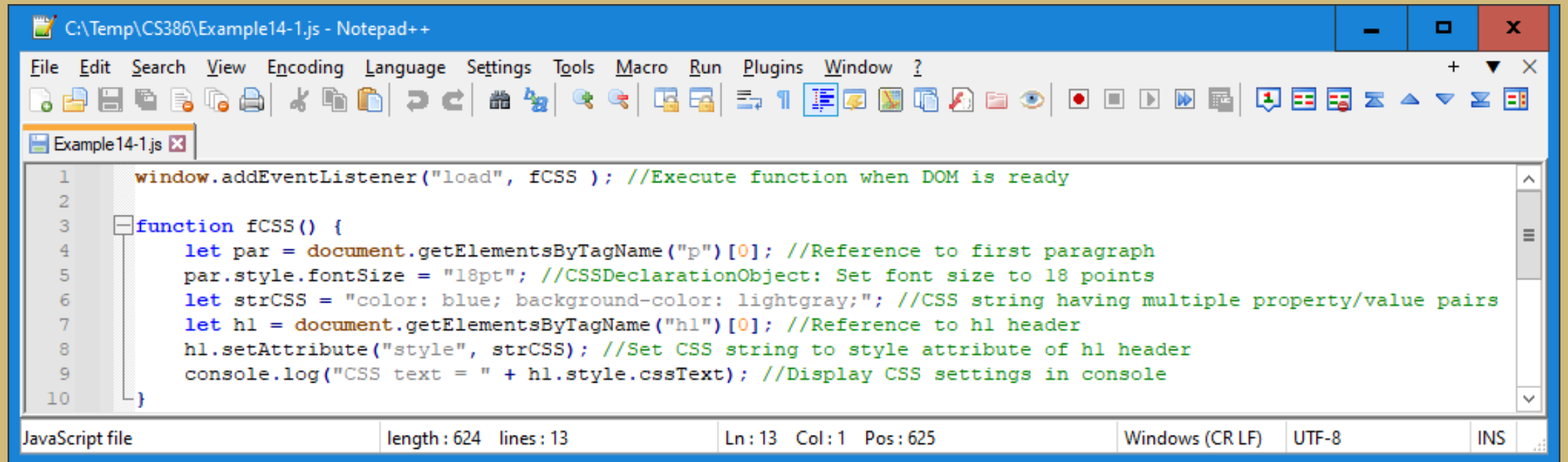


14.1 Scripting CSS

➤ Example 14-1:

❑ IMPORTANT:

- When using CSSDeclarationObject, CSS property names are JavaScript names (no hyphens)
- When using CSS string, property names are CSS property names with hyphens



The screenshot shows a Notepad++ window titled "C:\Temp\CS386\Example14-1.js - Notepad++". The code is as follows:

```
1  window.addEventListener("load", fCSS ); //Execute function when DOM is ready
2
3  function fCSS() {
4      let par = document.getElementsByTagName("p")[0]; //Reference to first paragraph
5      par.style.fontSize = "18pt"; //CSSDeclarationObject: Set font size to 18 points
6      let strCSS = "color: blue; background-color: lightgray;"; //CSS string having multiple property/value pairs
7      let h1 = document.getElementsByTagName("h1")[0]; //Reference to h1 header
8      h1.setAttribute("style", strCSS); //Set CSS string to style attribute of h1 header
9      console.log("CSS text = " + h1.style.cssText); //Display CSS settings in console
10 }
```

The status bar at the bottom indicates: "JavaScript file", "length: 624 lines: 13", "Ln: 13 Col: 1 Pos: 625", "Windows (CR LF)", "UTF-8", and "INS".

14.1 Scripting CSS

Computed Style

- `getComputedStyle()` method returns object containing values of all CSS properties of element
- After applying active stylesheets and resolving any basic computation those values may contain
- Method is part of window object
- Method is read only!!
- Cannot change property values using this method, use previous discussed ones
- Computed style properties are absolute: relative units like percentages and points are converted to absolute values
- Any property that specifies size (such as margin size or font size) will have value measured in pixels

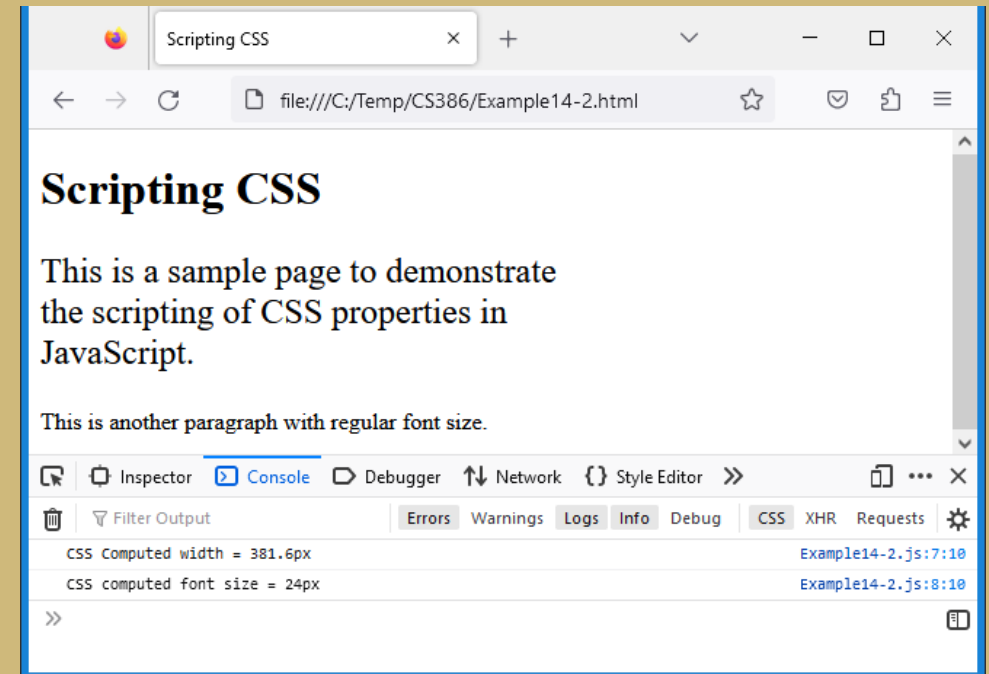
Syntax:

let *prop* = **window.getComputedStyle**(*element*).**CSSproperty**

14.1 Scripting CSS

➤ Example 14-2:

- ❑ Create JavaScript file Example14-2.js
- ❑ Use `window.addEventListener("load", fCompStyle)`
- ❑ Create function `fCompStyle`
- ❑ In function, style the following elements using style property:
 - First paragraph:
 - font size to 18 pt
 - width to 60 percent
 - Display in console computed style of width and font-size



14.1 Scripting CSS

➤ Example 14-2:

- ❑ Notice font size of 18pt is converted to 24px!
- ❑ Points are traditionally used in print media
- ❑ While Pixels used in screen media
- ❑ One point is the equivalent of 1.333(3) pixels
- ❑ One pixel is the equivalent of 0.75 points

The screenshot displays a web browser window titled 'Scripting CSS' and a Notepad++ editor window titled 'C:\Temp\CS386\Example14-2.js - Notepad++'.

The browser window shows a page with the title 'Scripting CSS' and two paragraphs: 'This is a sample page to demonstrate the scripting of CSS properties in JavaScript.' and 'This is another paragraph with regular font size.' The browser's developer tools are open, showing the 'Console' tab with two log messages: 'CSS Computed width = 381.6px' and 'CSS computed font size = 24px'. The 'CSS' tab is also open, showing the computed styles for the selected paragraph.

The Notepad++ editor shows the JavaScript code for 'Example14-2.js':

```
1 window.addEventListener("load", fCompStyle ); //Execute function when DOM is ready
2
3 function fCompStyle() {
4     let par = document.getElementsByTagName("p")[0]; //Reference to first paragraph
5     par.style.fontSize = "18pt"; //CSSDeclarationObject: Set font size to 18 points
6     par.style.width = "60%"; //Set width of paragraph to 60%
7     console.log("CSS Computed width = " + window.getComputedStyle(par).width); //Display actual width in pixels
8     console.log("CSS computed font size = " + window.getComputedStyle(par).fontSize); //Notice points converted to pixels
9 }
```

The status bar at the bottom of the Notepad++ window indicates 'JavaScript file', 'length: 570 lines: 12', 'Ln: 12 Col: 1 Pos: 571', 'Windows (CR LF)', 'UTF-8', and 'INS'.

14.1 Scripting CSS

➤ CSS Classes

- ❑ Class attributes allows for grouping of disparate elements together
- ❑ Create styles for classes in stylesheet
- ❑ Then use JavaScript to attach or detach those classes to elements
- ❑ HTML5 added new `classList` property to elements
- ❑ Property contains 4 different methods:
 - **add:** Adds individual class names to element's class attribute
 - **remove:** Removes individual class names from element's class attribute
 - **toggle:** Adds classname if it is not already present and removes it otherwise
 - **contains:** Method tests whether class attribute contains specified classname

Syntax:

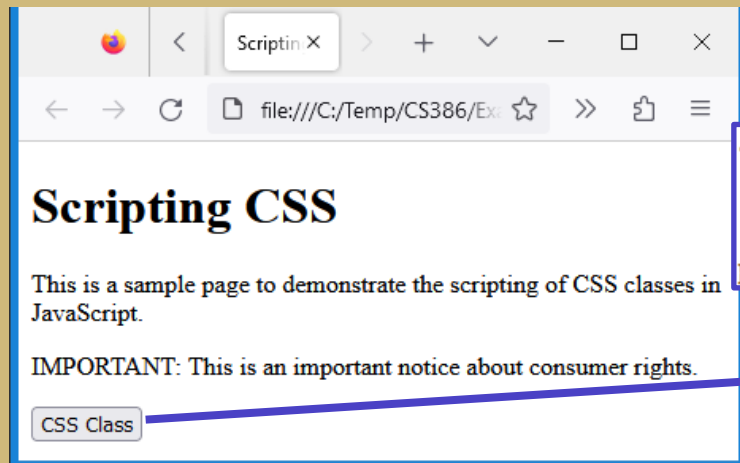
```
element.classList.add(class1, class2, ...classn)  
element.classList.remove(class1, class2, ...classn)  
element.classList.toggle(classname)  
element.classList.contains(classname)
```

14.1 Scripting CSS

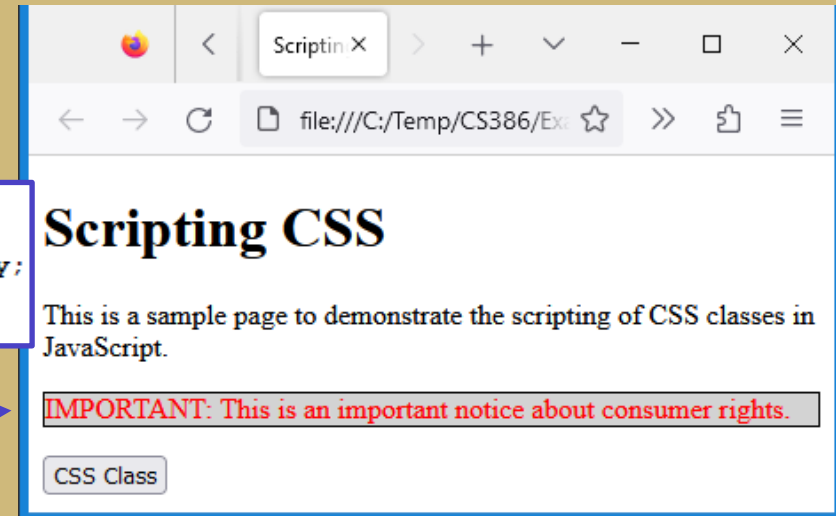
➤ Example 14-3:

- ❑ Create JavaScript file Example14-3.js
- ❑ Create function fCSSClass
- ❑ Add following code at the top:
- ❑ In function fCSSClass, get reference to second paragraph, assign into variable par
- ❑ Use method toggle of classList property and toggle class "emphasize"
- ❑ Class emphasize is defined in stylesheet

```
window.addEventListener("load", function() {  
    //Add click event for button btn  
    document.getElementById("btn").addEventListener("click", fCSSClass );  
});
```

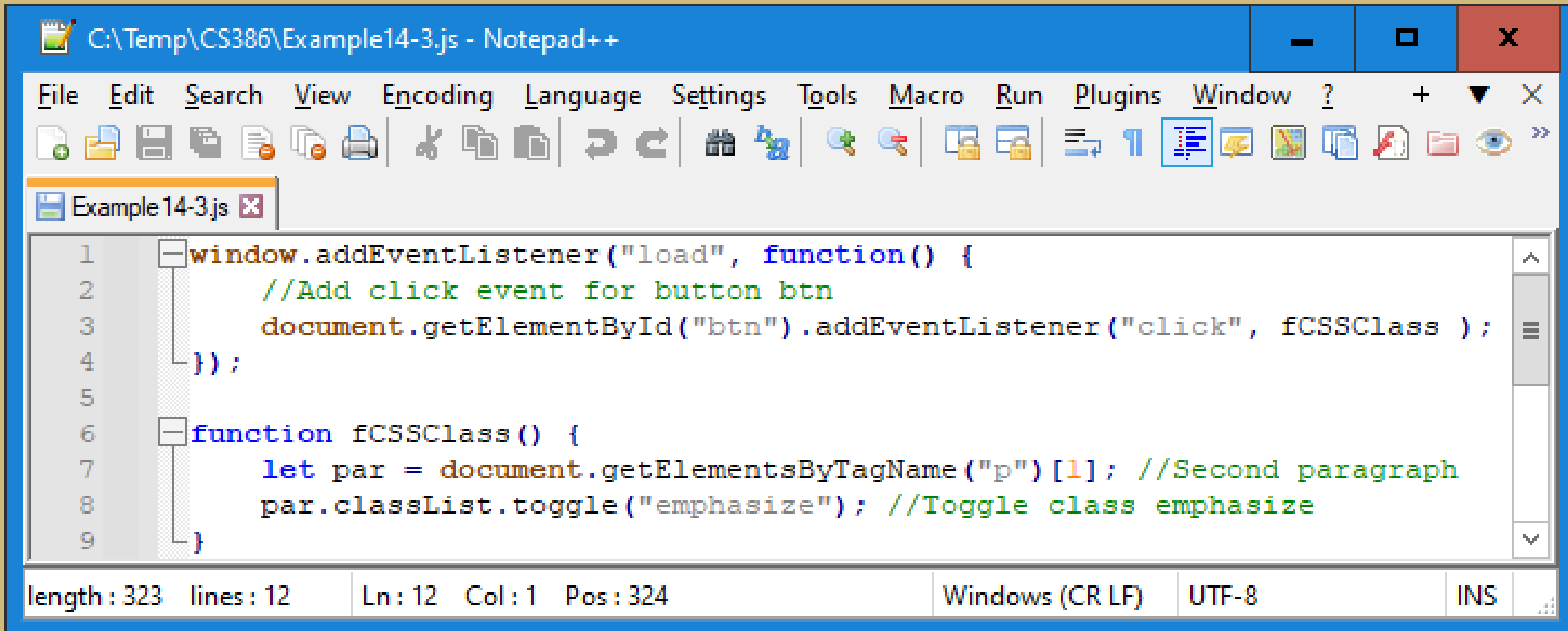


```
.emphasize {  
    color: red;  
    background-color: lightgrey;  
    border: 1px solid black;  
}
```



14.1 Scripting CSS

➤ Example 14-3:



```
C:\Temp\CS386\Example14-3.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? + ▼ X
[Icons]

Example14-3.js x
1 window.addEventListener("load", function() {
2     //Add click event for button btn
3     document.getElementById("btn").addEventListener("click", fCSSClass );
4 });
5
6 function fCSSClass() {
7     let par = document.getElementsByTagName("p")[1]; //Second paragraph
8     par.classList.toggle("emphasize"); //Toggle class emphasize
9 }

length: 323 lines: 12 Ln: 12 Col: 1 Pos: 324 Windows (CR LF) UTF-8 INS
```

14.2 Animations

➤ Two JavaScript methods for creating animations:

- ❑ setTimeout
- ❑ setInterval

Syntax:

let *timeoutID* = **setTimeout**(*callback function*[, *delay*])

let *intervalID* = **setInterval**(*callback function*[, *delay*])

➤ Pass *timeoutID*, *intervalID* into `clearTimeout`, `clearInterval` methods to cancel function execution:

Syntax:

clearTimeout(*timeoutID*)

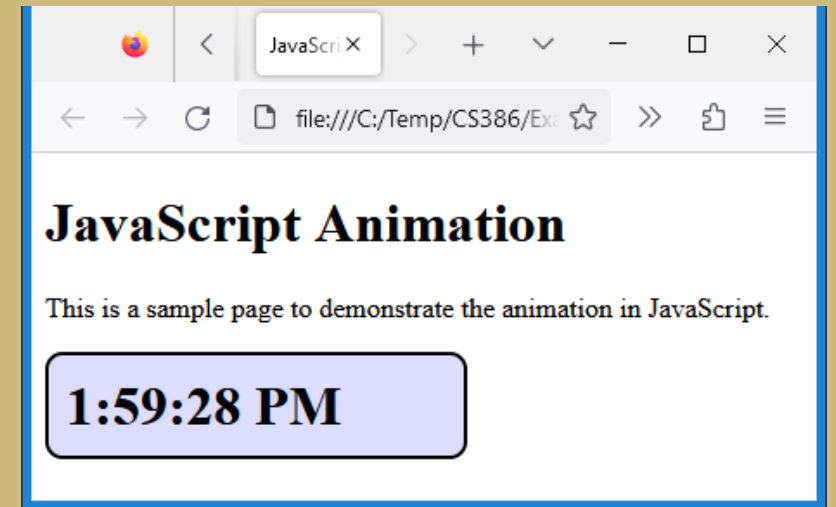
clearInterval(*intervalID*)

➤ Both methods:

- ❑ On window object
- ❑ Are asynchronous using callback
- ❑ Execute callback function after specified delay in ms
- ❑ Are non-blocking:
 - Statements after `setTimeout` and `setInterval` are executed immediately
 - Callback functions return after statements

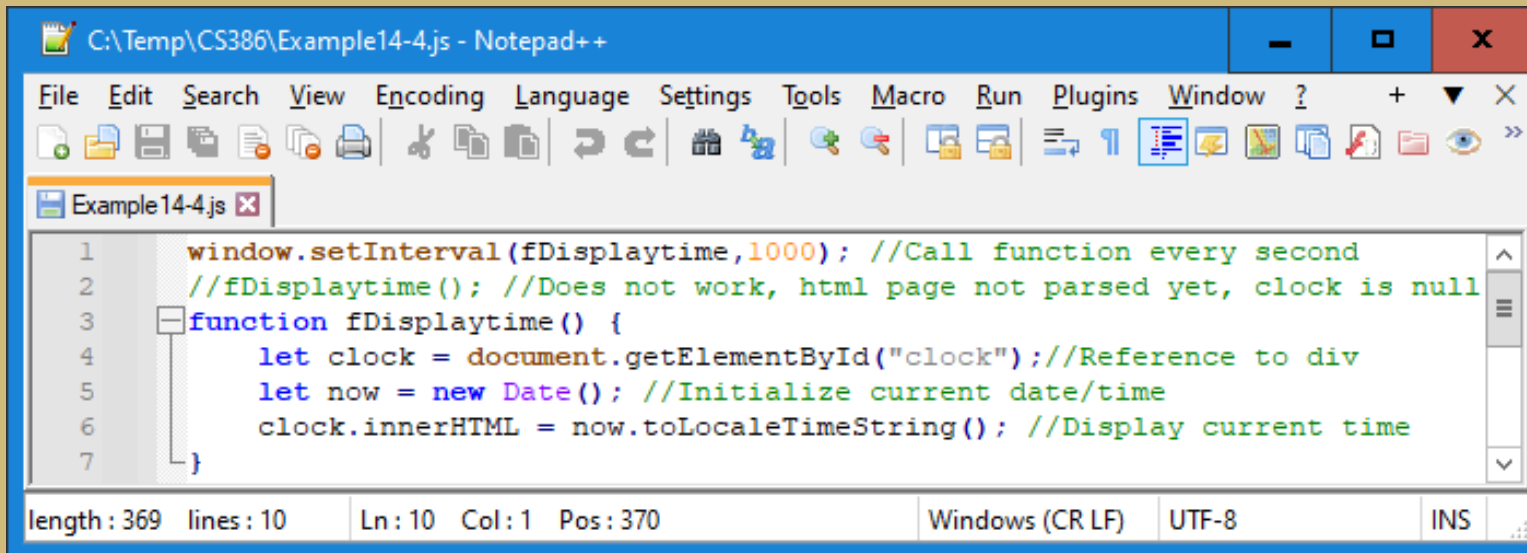
14.2 Animations

- **Example 14-4:**
- Create JavaScript Example14-4.js
- Use setInterval method to call function fDisplayTime after 1 second
- Create function fDisplayTime
- Store reference to div element in variable clock using ID value
- Create variable now and store current date/time
- Set innerHTML of clock element to current time using method toLocaleTimeString() of variable now



14.2 Animations

- **Example 14-4:**
- Notice no load event this time (why??)
- First call to function is made in 1 second
- Function in setInterval is callback, does not block code
- By that time html page is fully parsed!

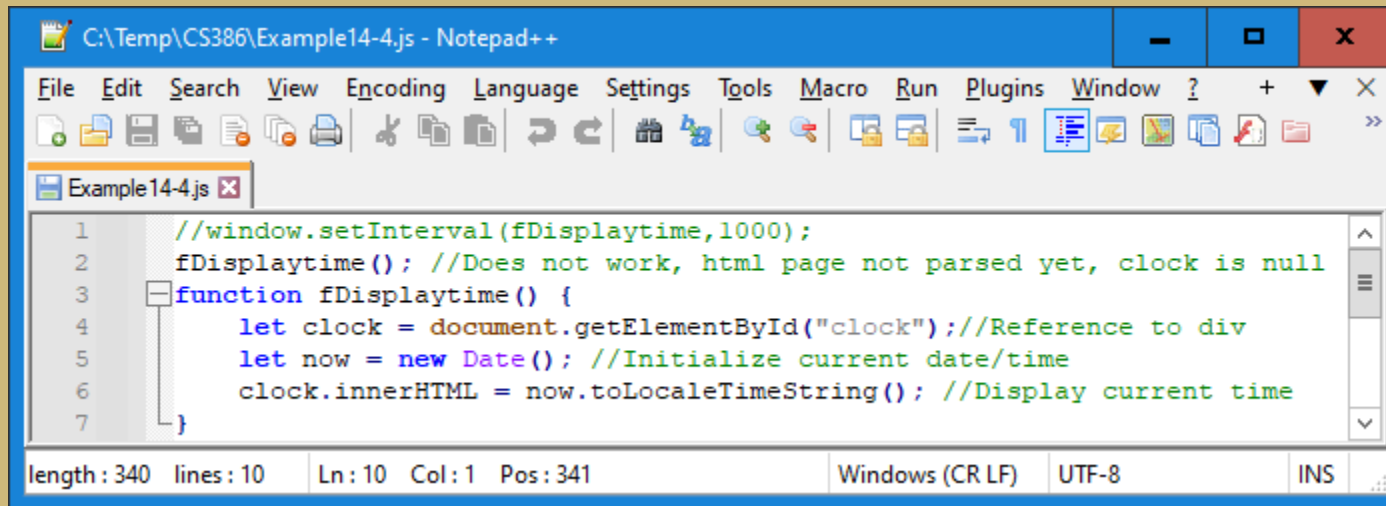


```
1  window.setInterval(fDisplaytime,1000); //Call function every second
2  //fDisplaytime(); //Does not work, html page not parsed yet, clock is null
3  function fDisplaytime() {
4      let clock = document.getElementById("clock");//Reference to div
5      let now = new Date(); //Initialize current date/time
6      clock.innerHTML = now.toLocaleTimeString(); //Display current time
7  }
```

length: 369 lines: 10 Ln: 10 Col: 1 Pos: 370 Windows (CR LF) UTF-8 INS

14.2 Animations

- **Example 14-4 (continued):**
- Call function fDisplayTime immediately
- Clock element is null since html page is not parsed yet

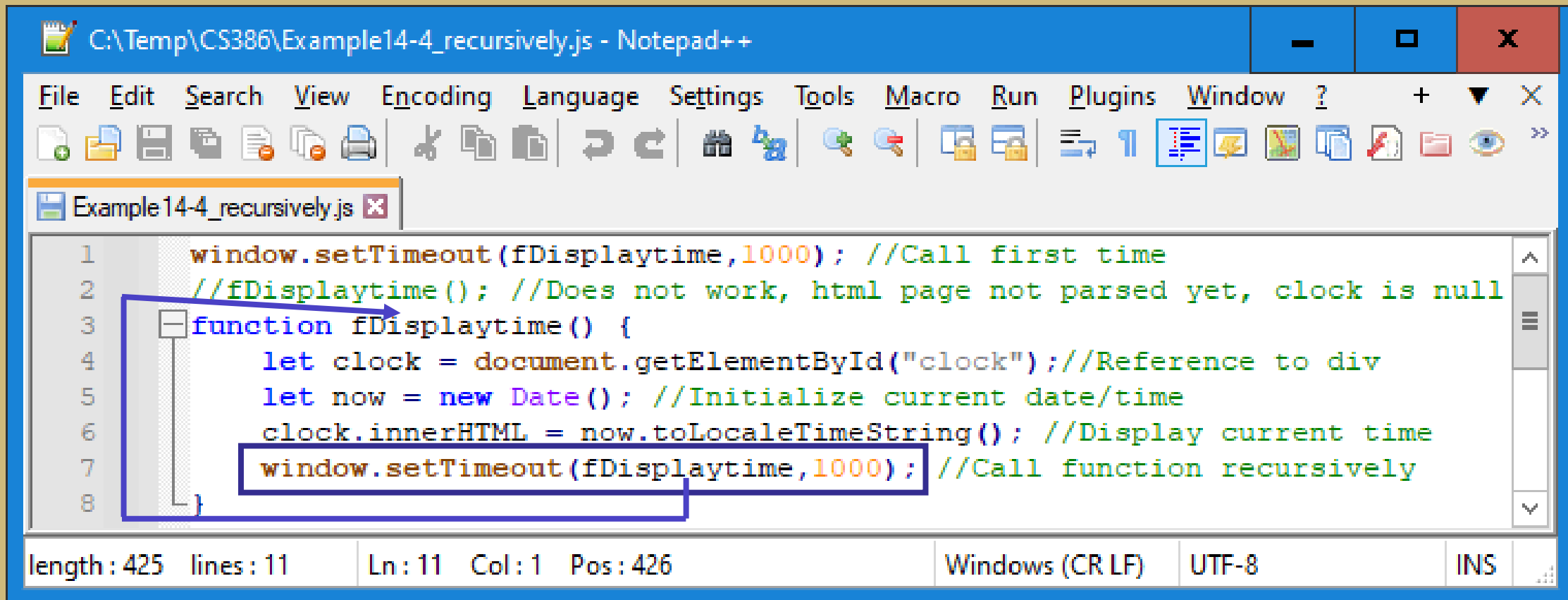


```
C:\Temp\CS386\Example14-4.js - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Example14-4.js
1 //window.setInterval(fDisplaytime,1000);
2 fDisplaytime(); //Does not work, html page not parsed yet, clock is null
3 function fDisplaytime() {
4     let clock = document.getElementById("clock");//Reference to div
5     let now = new Date(); //Initialize current date/time
6     clock.innerHTML = now.toLocaleTimeString(); //Display current time
7 }
length: 340 lines: 10 Ln: 10 Col: 1 Pos: 341 Windows (CR LF) UTF-8 INS
```

! ▶ Uncaught TypeError: clock is null
fDisplaytime file:///C:/Temp/CS386/Example14-4.js:6
<anonymous> file:///C:/Temp/CS386/Example14-4.js:2
[\[Learn More\]](#)

14.2 Animations

- **Example 14-4 (continued):**
- Can also call function `fDisplayTime` recursively using `setTimeout`



The screenshot shows a Notepad++ window titled "C:\Temp\CS386\Example14-4_recursively.js - Notepad++". The code is as follows:

```
1  window.setTimeout(fDisplaytime,1000); //Call first time
2  //fDisplaytime(); //Does not work, html page not parsed yet, clock is null
3  function fDisplaytime() {
4      let clock = document.getElementById("clock");//Reference to div
5      let now = new Date(); //Initialize current date/time
6      clock.innerHTML = now.toLocaleTimeString(); //Display current time
7      window.setTimeout(fDisplaytime,1000); //Call function recursively
8  }
```

A blue box highlights the recursive call `window.setTimeout(fDisplaytime,1000);` on line 7. A blue arrow points from this call to the function definition `function fDisplaytime() {` on line 3. The status bar at the bottom indicates "length: 425 lines: 11 Ln: 11 Col: 1 Pos: 426 Windows (CR LF) UTF-8 INS".

14.2 Animations

- Problem with using setTimeout/setInterval for executing code that changes something on screen is twofold:
 - ❑ Specified delay (ie: 50 milliseconds) inside these functions are often times not honored:
 - Due to changes in user system resources at time
 - Leading to inconsistent delay intervals between animation frames
 - ❑ Even worse, using setTimeout() or setInterval() to continuously make changes to the user's screen often induces "layout thrashing"
- Browser version of cardiac arrest:
 - ❑ Forced to perform unnecessary reflows of page before the user's screen is physically able to display changes
 - ❑ This is bad -very bad- due to taxing nature of page reflows
 - ❑ Especially on mobile devices where problem is most apparent, with janky page loads and battery drains

14.2 Animations

- For those reasons `requestAnimationFrame()` was introduced
- Allows to execute code on next available screen repaint:
 - ❑ Taking guess work out of getting in sync with user's browser and hardware readiness to make changes to screen
- When `requestAnimationFrame()` is called repeatedly to create animation:
 - ❑ Assured that animation code is called when user's computer is actually ready to make changes to screen each time
 - ❑ Resulting in a smoother, more efficient animation
- Notice no more delay time is needed
- It runs automatically at best points in time
- To execute it repeatedly, use recursive function call
- To cancel animation, use `cancelAnimationFrame`

Syntax:

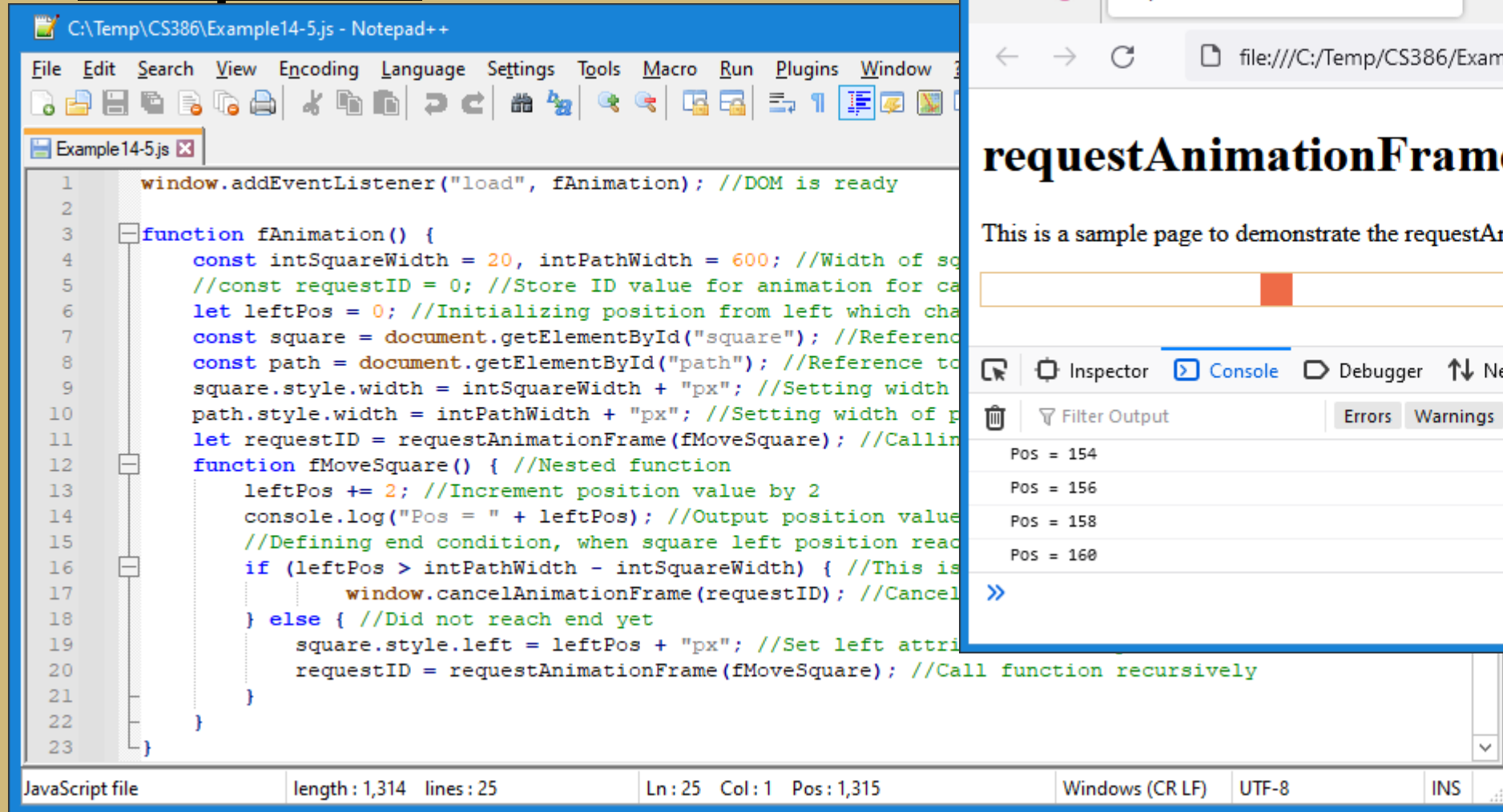
let *requestID* = **requestAnimationFrame**(*function*)

Syntax:

cancelAnimationFrame(*requestID*)

14.2 Animations

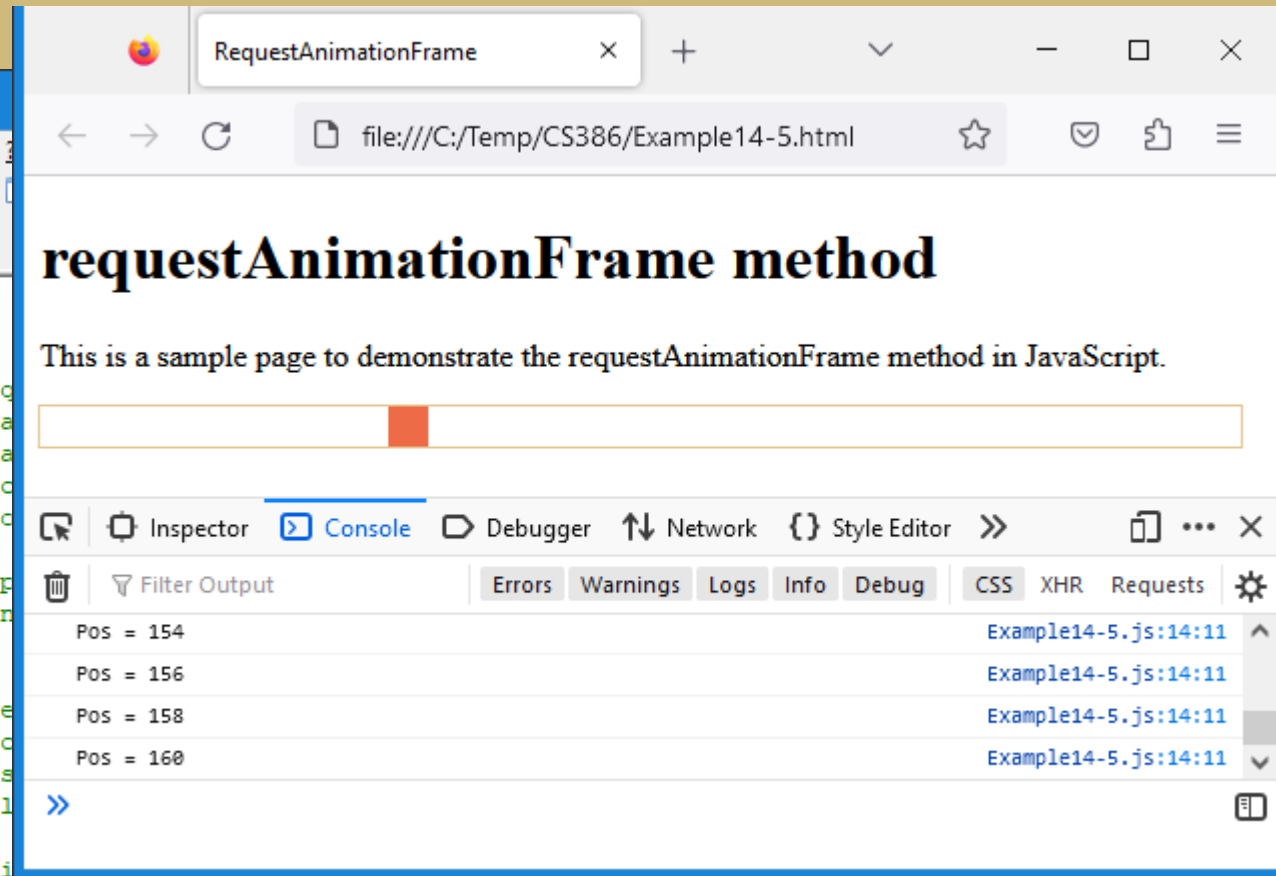
➤ Example14-5:



The screenshot shows a Notepad++ window with the file `C:\Temp\CS386\Example14-5.js`. The code implements a square moving along a path using the `requestAnimationFrame` method.

```
1 window.addEventListener("load", fAnimation); //DOM is ready
2
3 function fAnimation() {
4     const intSquareWidth = 20, intPathWidth = 600; //Width of square and path
5     //const requestId = 0; //Store ID value for animation for cancellation
6     let leftPos = 0; //Initializing position from left which changes over time
7     const square = document.getElementById("square"); //Reference to square element
8     const path = document.getElementById("path"); //Reference to path element
9     square.style.width = intSquareWidth + "px"; //Setting width of square
10    path.style.width = intPathWidth + "px"; //Setting width of path
11    let requestId = requestAnimationFrame(fMoveSquare); //Calling requestAnimationFrame
12    function fMoveSquare() { //Nested function
13        leftPos += 2; //Increment position value by 2
14        console.log("Pos = " + leftPos); //Output position value to console
15        //Defining end condition, when square left position reaches end of path
16        if (leftPos > intPathWidth - intSquareWidth) { //This is the end condition
17            window.cancelAnimationFrame(requestId); //Cancel the animation
18        } else { //Did not reach end yet
19            square.style.left = leftPos + "px"; //Set left attribute of square
20            requestId = requestAnimationFrame(fMoveSquare); //Call function recursively
21        }
22    }
23 }
```

The status bar at the bottom indicates: JavaScript file, length: 1,314 lines: 25, Ln: 25 Col: 1 Pos: 1,315, Windows (CR LF), UTF-8, INS.



The screenshot shows a web browser window with the title `requestAnimationFrame method`. The address bar shows the file path `file:///C:/Temp/CS386/Example14-5.html`. The page content includes the heading `requestAnimationFrame method` and a paragraph: "This is a sample page to demonstrate the requestAnimationFrame method in JavaScript." Below the text is a visual representation of a square moving along a path, with a red square on a horizontal line.

The browser's developer console is open, showing the following log entries:

| Message | Source |
|-----------|----------------------|
| Pos = 154 | Example14-5.js:14:11 |
| Pos = 156 | Example14-5.js:14:11 |
| Pos = 158 | Example14-5.js:14:11 |
| Pos = 160 | Example14-5.js:14:11 |

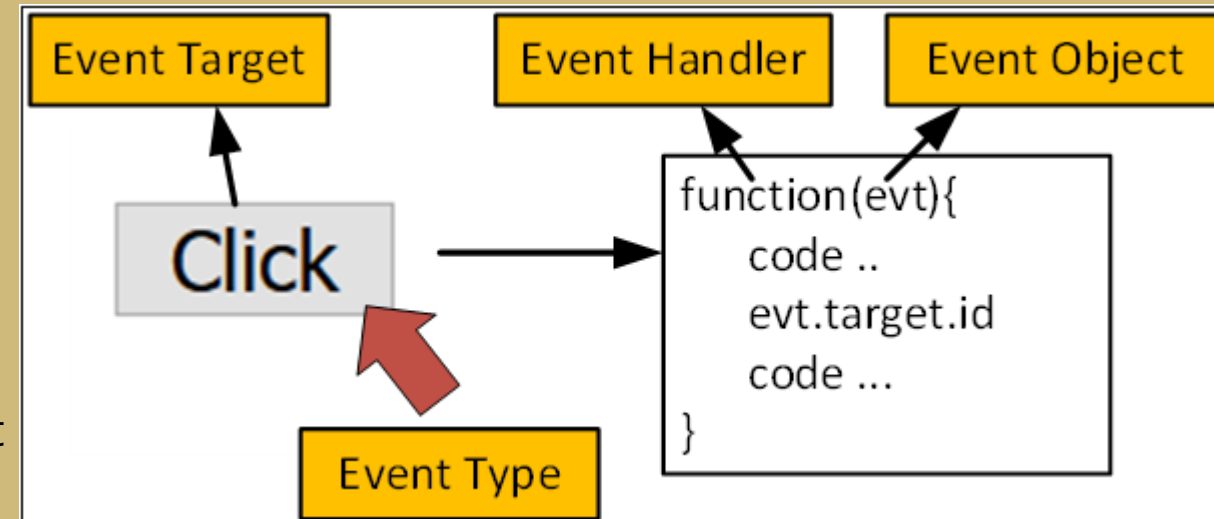
14.3 Events

- Event is signal that something has happened
- All DOM nodes generate such signals (but events are not limited to DOM)
 - ❑ Mouse events:
 - click - when mouse clicks on element (touchscreen devices generate it on a tap)
 - contextmenu - when mouse right-clicks on element
 - mouseover / mouseout - when mouse cursor comes over / leaves element
 - mousedown / mouseup - when mouse button is pressed / released over element
 - mousemove - when mouse is moved
 - ❑ Keyboard events:
 - keydown and keyup - when keyboard key is pressed and released
 - ❑ Form element events:
 - submit - when visitor submits <form>
 - focus - when visitor focuses on element, e.g. on <input>
 - ❑ Document events:
 - DOMContentLoaded - when HTML is loaded and processed, DOM is fully built
 - ❑ CSS events:
 - transitionend - when CSS-animation finishes

14.3 Events

Terminology

- Event Type
 - ❑ String that specifies what kind of event occurred (mousemove, click)
 - ❑ Because type of event is just string → sometimes called event name
- Event Target
 - ❑ Object on which event occurred or with which event is associated
 - ❑ For events must specify both type and target
- Event Handler/Listener
 - ❑ Function that handles or responds to event
- Event Object
 - ❑ Object associated with particular event and contains details about that event
 - ❑ Event objects are implicitly passed as argument to event handler function



14.3 Events

Event handlers

- To react on events assign handler:
 - ❑ Function that runs in case of event
- Handlers are way to run JavaScript code in case of user actions
- Two ways to assign handlers:
 - ❑ Event property (starting with on, i.e. onclick)
 - ❑ Method `addEventListener` (without on, i.e. click))

Syntax:

target.on<event type> = event handler

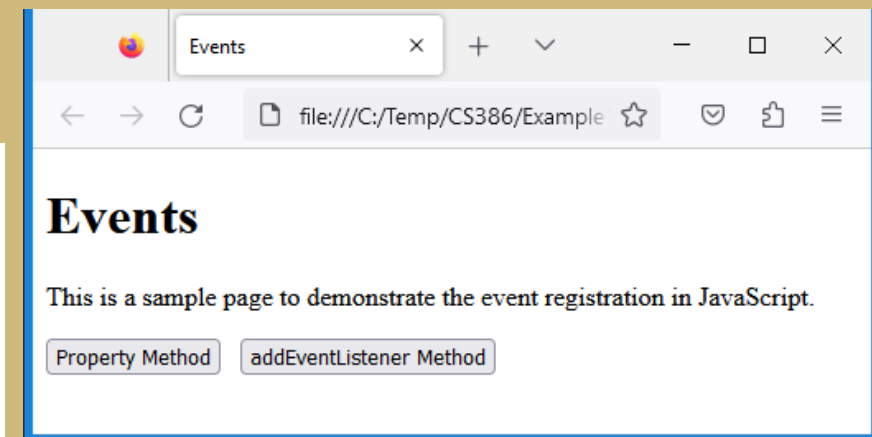
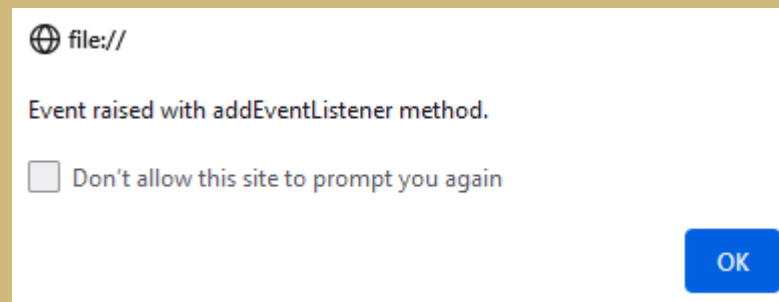
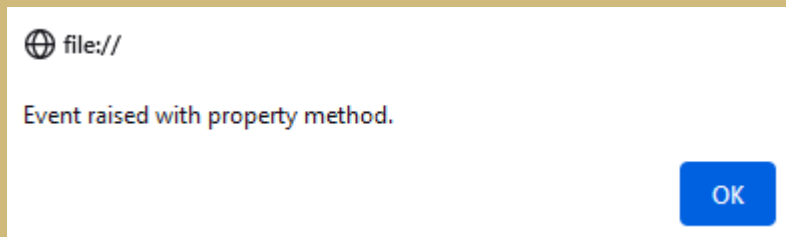
Syntax:

*target.**addEventListener**(event type, event handler)*

14.3 Events

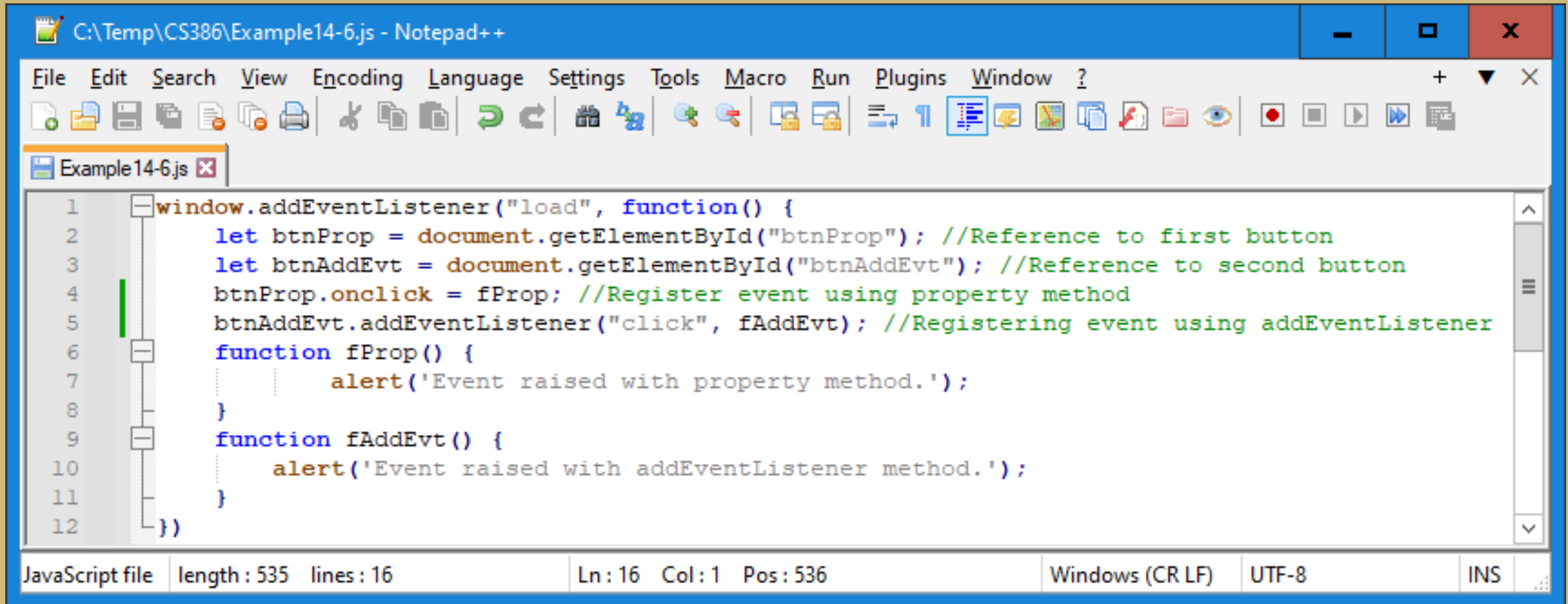
➤ Example 14-6:

- ❑ Create JavaScript file Example14-6.js
- ❑ Wrap code in `window.addEventListener("load", function() { code..... })` (inline function)
- ❑ Create variable `btnProp` and store reference to first button
 - Using `btnProp` register event using property method assigning `fProp`
- ❑ Create variable `btnAddEvt` and store reference to second button
 - Using `btnAddEvt` register event using `addEventListener` responding with `fAddEvt`
- ❑ Create functions `fProp` and `fAddEvt` and issue alerts as shown below:



14.3 Events

➤ Example 14-6:



```
C:\Temp\CS386\Example14-6.js - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

Example14-6.js x

1 window.addEventListener("load", function() {
2     let btnProp = document.getElementById("btnProp"); //Reference to first button
3     let btnAddEvt = document.getElementById("btnAddEvt"); //Reference to second button
4     btnProp.onclick = fProp; //Register event using property method
5     btnAddEvt.addEventListener("click", fAddEvt); //Registering event using addEventListener
6     function fProp() {
7         alert('Event raised with property method.');
```

JavaScript file length : 535 lines : 16 Ln : 16 Col : 1 Pos : 536 Windows (CR LF) UTF-8 INS

14.3 Events

- Can call `addEventListener()` multiple times to register more than one handler function for same event type on same target
- When using event handler function with parameters, use anonymous function wrapper to register the event passing arguments:

Syntax:

```
target.addEventListner(event type, function() {func_name(arg1, arg2...)}))
```

- `removeEventListener()` method that expects same arguments but removes event handler function from target rather than adding it
- Often useful to temporarily register event handler and then remove it soon afterward

14.3 Events

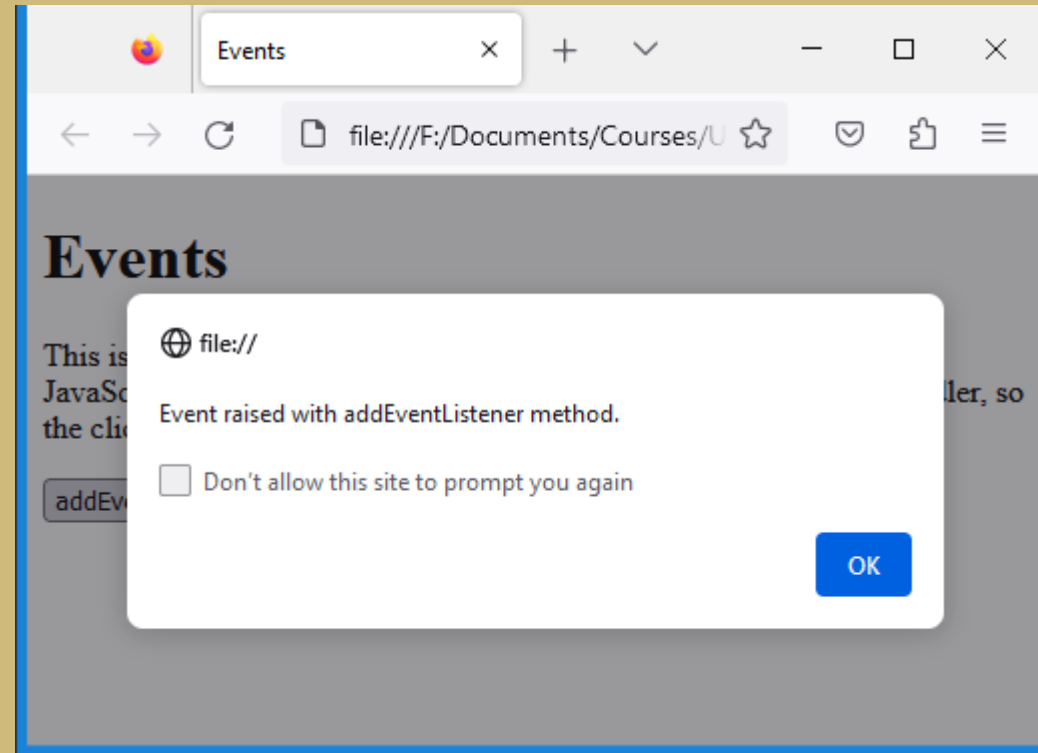
- **Example 14-7:**
- Demonstrating of registering events with parameters and removing event handler

file://

Enter your name

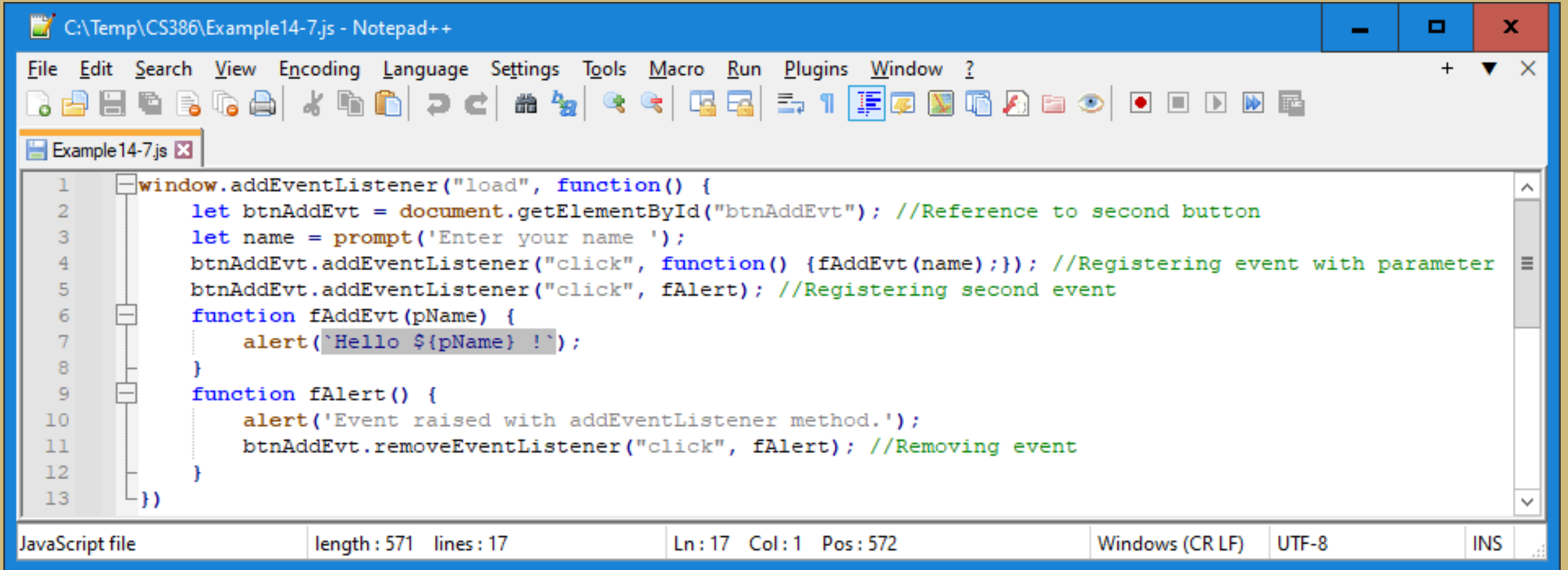
John Doe

OK Cancel



14.3 Events

- Example 14-7:
- Demonstrating of registering events with parameters and removing event handler



```
1 window.addEventListener("load", function() {
2     let btnAddEvt = document.getElementById("btnAddEvt"); //Reference to second button
3     let name = prompt('Enter your name ');
4     btnAddEvt.addEventListener("click", function() {fAddEvt(name)}); //Registering event with parameter
5     btnAddEvt.addEventListener("click", fAlert); //Registering second event
6     function fAddEvt(pName) {
7         alert(`Hello ${pName} !`);
8     }
9     function fAlert() {
10        alert('Event raised with addEventListener method. ');
11        btnAddEvt.removeEventListener("click", fAlert); //Removing event
12    }
13 })
```

JavaScript file length: 571 lines: 17 Ln: 17 Col: 1 Pos: 572 Windows (CR LF) UTF-8 INS

14.3 Events

Event Object

- Event handler functions are automatically/implicitly passed argument:
 - ❑ Event object
- This object holds additional information about event:
 - ❑ Which mouse button was pressed
 - ❑ What id is target element
 - ❑ What are the x and y coordinates mouse was clicked
- Use parameter variable to receive event object
- Common names are e, evt, event

Syntax:

target.addEventListener(event type, func_name)

```
func_name(evt) {  
    evt.methods/properties  
}
```

14.3 Events

Event Object

- In case of passing additional arguments, use anonymous function wrapper function (as previously discussed)
- Pass event object parameter into anonymous wrapper, then pass it down to event handler function

Syntax:

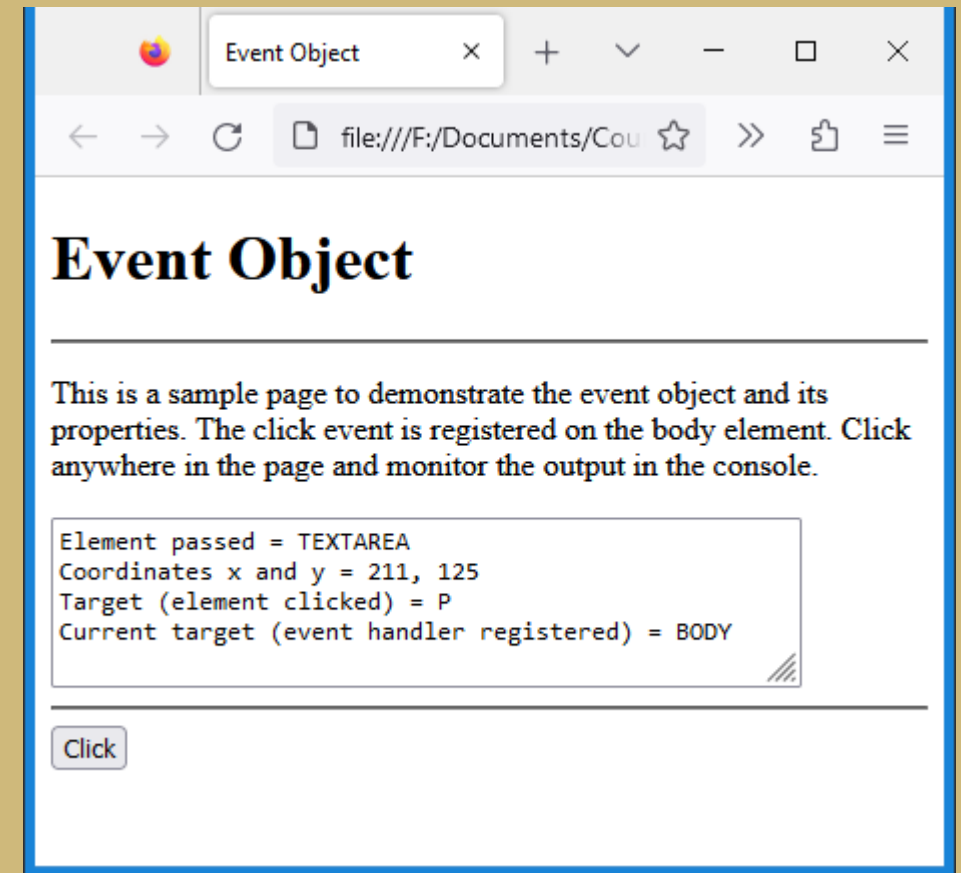
```
target.addEventListner(event type, function(evt) {func_name(evt, arg1, arg2) } )
```

```
func_name(evt, par1, par2) {  
    evt.methods/properties  
}
```

14.3 Events

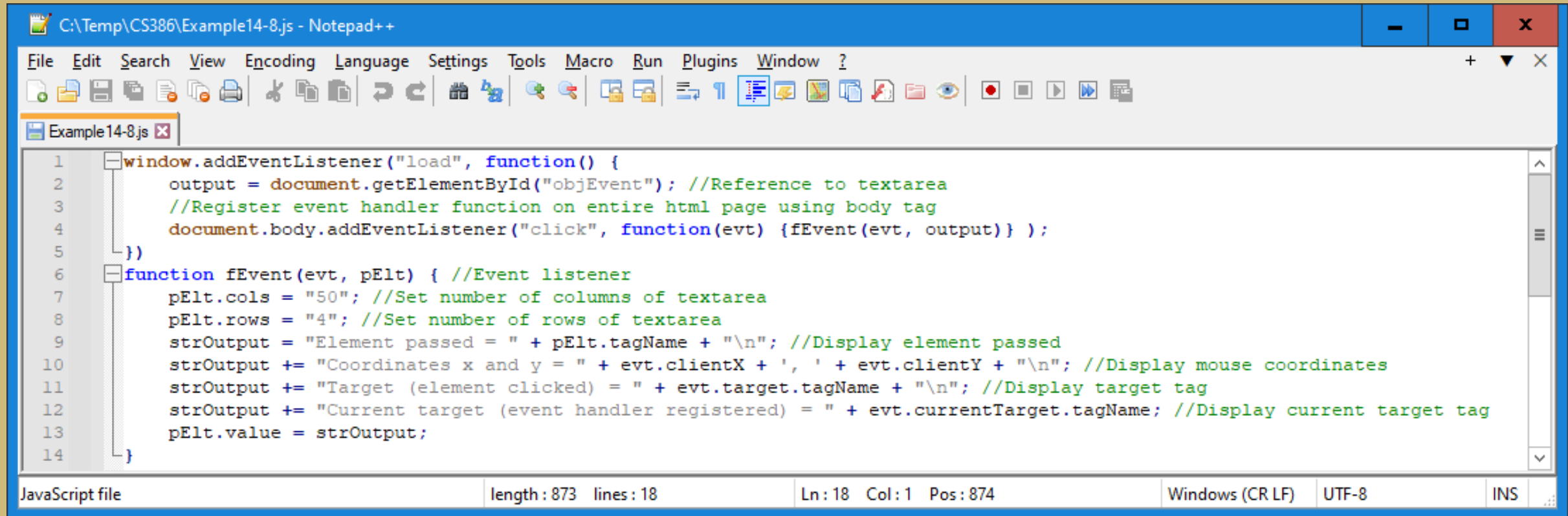
➤ Example 14-8:

- ❑ Create file Example14-8.js
- ❑ Create load event handler:
 - Store reference to textarea in variable output
 - Register click event on body passing event object and output variable into function fEvent
- ❑ Create function fEvent with parameters evt and pElt
- ❑ Set cols and rows properties for pElt to 50 and 4
- ❑ Create variable strOutput and assign string as shown:
 - Use passed element tag name
 - Use event object clientX and clientY properties
 - Use event object target tag name
 - Use event object current target tag name
- ❑ Assign strOutput to value of pElt



14.3 Events

➤ Example 14-8:



```
1 window.addEventListener("load", function() {
2     output = document.getElementById("objEvent"); //Reference to textarea
3     //Register event handler function on entire html page using body tag
4     document.body.addEventListener("click", function(evt) {fEvent(evt, output)} );
5 })
6 function fEvent(evt, pElt) { //Event listener
7     pElt.cols = "50"; //Set number of columns of textarea
8     pElt.rows = "4"; //Set number of rows of textarea
9     strOutput = "Element passed = " + pElt.tagName + "\n"; //Display element passed
10    strOutput += "Coordinates x and y = " + evt.clientX + ', ' + evt.clientY + "\n"; //Display mouse coordinates
11    strOutput += "Target (element clicked) = " + evt.target.tagName + "\n"; //Display target tag
12    strOutput += "Current target (event handler registered) = " + evt.currentTarget.tagName; //Display current target tag
13    pElt.value = strOutput;
14 }
```

JavaScript file length : 873 lines : 18 Ln : 18 Col : 1 Pos : 874 Windows (CR LF) UTF-8 INS

14.3 Events

Event Object

- Many events automatically lead to certain actions performed by browser:
 - ❑ Click on anchor link → initiates navigation to its URL
 - ❑ Form submit button → initiates submission to server
 - ❑ Pressing mouse button over text and moving it → selects the text
- Sometimes need to cancel default actions:
 - ❑ Form validation fails → do not submit form
 - ❑ Prevent certain characters from being entered in inputs
- When default action is canceled, subsequent events are also canceled
- Example:
 - ❑ Canceled mousedown event in text input prevents focus event

Syntax:

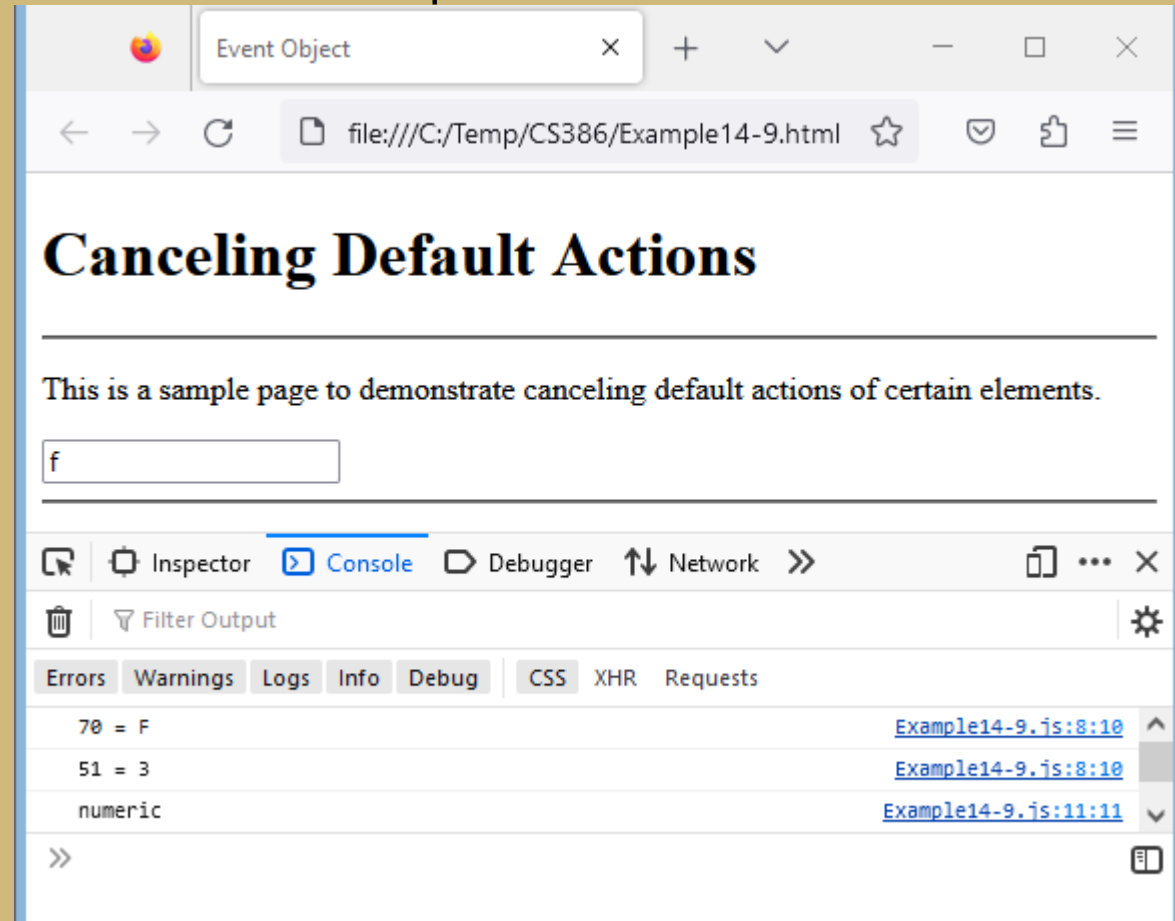
//When using addEventListener
event_object.preventDefault()

Syntax

//When using event property
//Return false in event handler
return false

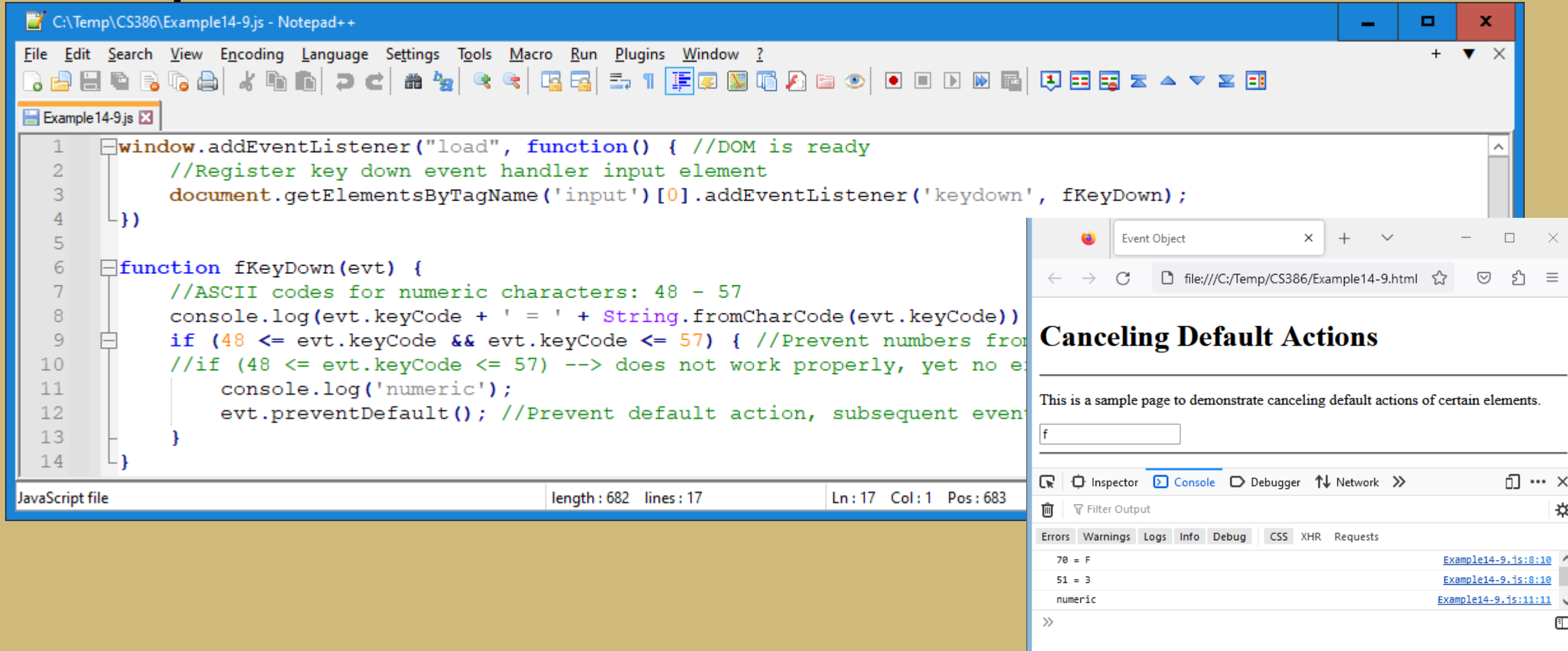
14.3 Events

- **Example 14-9:**
- Prevent numeric characters from being entered in text input:
 - ❑ Numeric characters are codes 48 - 57
- Keydown event has property keyCode of event object for entered character
- In load event, register event on first input element calling function fKeyDown
- In function, test for codes 48 - 57, print numeric and cancel default action
- Once keydown event is canceled, subsequent events keypress and keyup are also canceled



14.3 Events

➤ Example 14-9:



The screenshot displays a Notepad++ window titled "C:\Temp\CS386\Example14-9.js - Notepad++" with a menu bar (File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window) and a toolbar. The code in the editor is as follows:

```
1 window.addEventListener("load", function() { //DOM is ready
2     //Register key down event handler input element
3     document.getElementsByTagName('input')[0].addEventListener('keydown', fKeyDown);
4 })
5
6 function fKeyDown(evt) {
7     //ASCII codes for numeric characters: 48 - 57
8     console.log(evt.keyCode + ' = ' + String.fromCharCode(evt.keyCode))
9     if (48 <= evt.keyCode && evt.keyCode <= 57) { //Prevent numbers from
10        //if (48 <= evt.keyCode <= 57) --> does not work properly, yet no e
11        console.log('numeric');
12        evt.preventDefault(); //Prevent default action, subsequent event
13    }
14 }
```

Below the code editor, a status bar shows "JavaScript file", "length: 682 lines: 17", and "Ln: 17 Col: 1 Pos: 683".

Overlaid on the bottom right is a web browser window titled "Event Object" showing the file "file:///C:/Temp/CS386/Example14-9.html". The page content includes the heading "Canceling Default Actions" and the text "This is a sample page to demonstrate canceling default actions of certain elements." Below this text is an input field containing the letter "f".

The browser's developer tools are open, showing the "Console" tab. The console output is as follows:

| Message | Source |
|---------|----------------------|
| 70 = F | Example14-9.js:8:10 |
| 51 = 3 | Example14-9.js:8:10 |
| numeric | Example14-9.js:11:11 |