

HTML5, CSS3 & Javascript

Chapter 3:

Client-Side JavaScript Programming

Chapter Objectives

In this chapter, we will discuss:

- Adding behavior to a web page with JavaScript
- Basic JavaScript syntax
- Manipulating a document via DOM
- Handling events
- The JavaScript security model



Why JavaScript?

Basic Syntax

Working with DOM

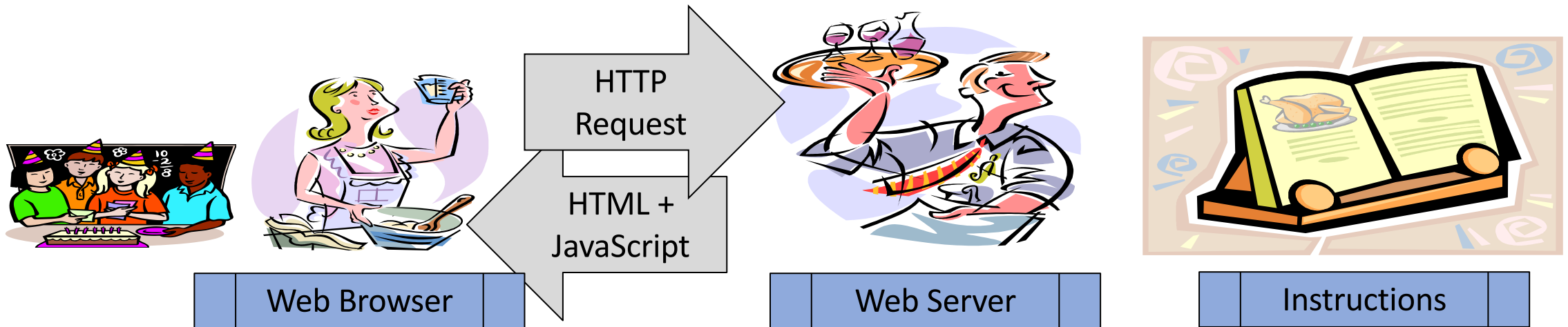
Handling Events

Web Application Security Client Side

Chapter Summary


Behavior

- JavaScript is a way of adding behavior to web pages
 - HTML5 makes JavaScript a standard
 - Prefer JavaScript to Flash, etc.
- A dynamic web application uses:
 - HTML for structure
 - CSS for presentation
 - JavaScript for behavior
 - This is what provides interaction



How CallStack works?

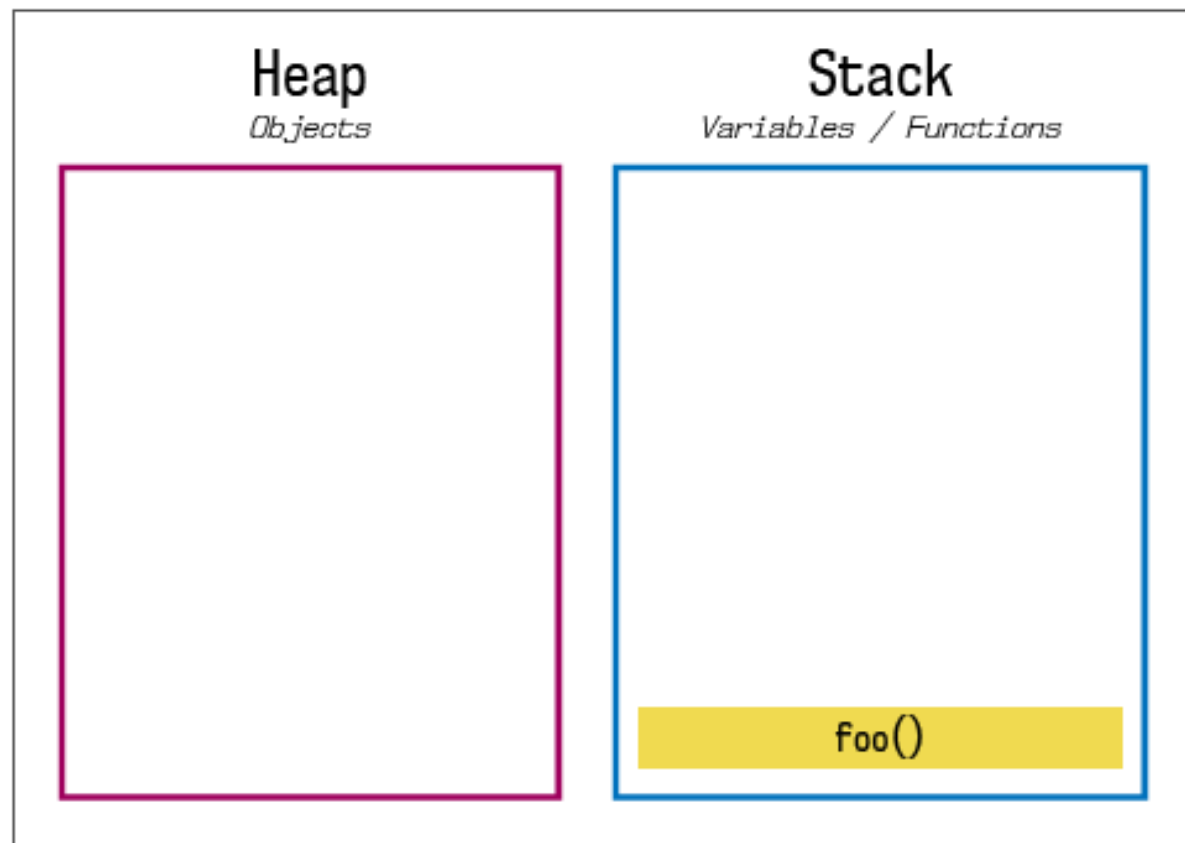
JavaScript Program



```
function baz(){  
  console.log('Hello from baz');  
}  
  
function bar() {  
  baz();  
}  
  
function foo() {  
  bar();  
}  
  
foo();
```

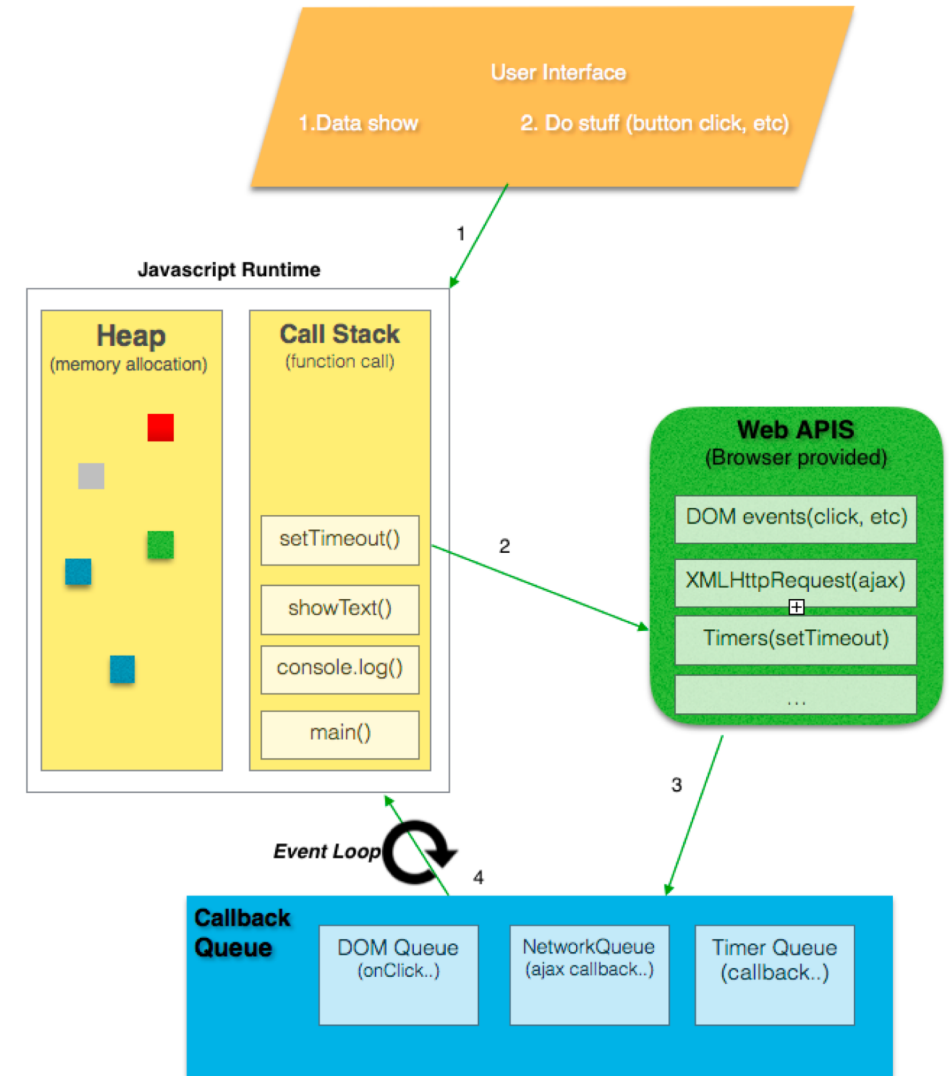


JavaScript Runtime



Javascript Engine

- <http://latentflip.com/loupe/>

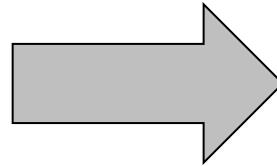
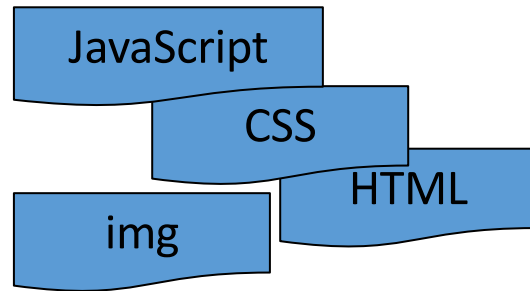


Static vs. Dynamic

- Most web pages do not serve out the exact same content to users
 - For example, different users have different bank balances
 - Cannot use a static HTML page to serve out account information
- Three types of dynamic behavior:
 - Client-side dynamic behavior achieved through the use of JavaScript
 - Browser executes script on client's machine
 - Server-side dynamic behavior where the server runs logic in reaction to a user query
 - Often written in Java, C# or a template language like PHP
 - Typically, user completes some action (e.g., fills out a form)
 - Web page sends a request to the server where any associated data is processed
 - Server generates HTML including appropriate data and returns it
 - Ajax (Asynchronous JavaScript and XML) is a hybrid
 - Server-side sends data in XML (or, more commonly now, in JSON) to client side
 - JavaScript code on client side works with this data

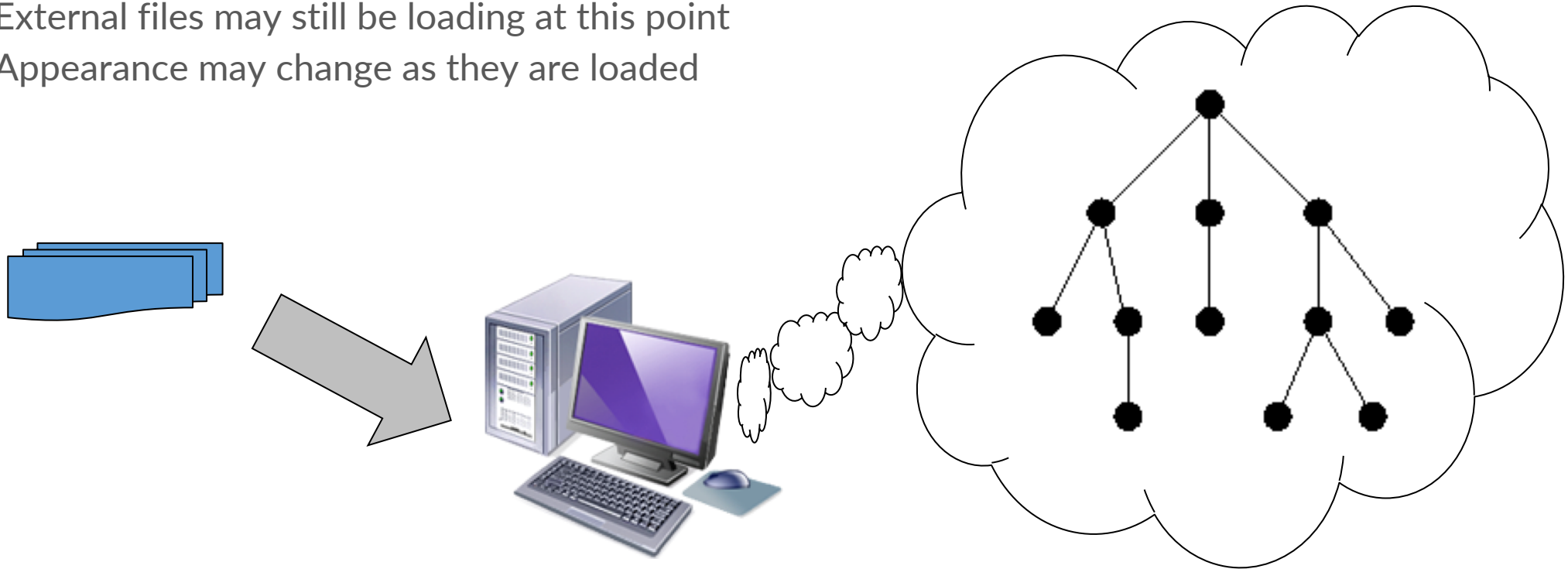
Recap: How HTML Works

- The web browser receives a document
 - Document is written in HTML
 - May have style specified using CSS (in-line or in external files)
 - May have behavior specified using JavaScript (in-line or in external files)
 - Refers to images, which are external files
- The browser starts to parse the document, constructing a **Document Object Model (DOM)**
 - Script files are retrieved immediately, when they are encountered
 - Possible to change this to make them load in parallel
 - Other files are loaded in parallel as parsing of the document continues



Recap: How HTML Works (continued)

- **Script files are executed immediately**
 - May be changed to force them to wait for the DOM to be created
 - **Any script may change the DOM** by adding, removing, or updating elements
 - Important to write scripts so they do not interact with DOM elements that have not been parsed yet
- Finally, the DOM is “rendered” (displayed to the user)
 - External files may still be loading at this point
 - Appearance may change as they are loaded



What JavaScript Can Do

- JavaScript is a full-fledged programming language
 - Use it to carry out computations
 - Is the entered date a weekday?
 - Use it to make conditional decisions
 - On what to display, when to display it, etc.
 - Use it to repeat processing
 - If user changes a value, recalculate total
- Use JavaScript to provide a rich, interactive experience
 - Runs directly in the browser
 - Avoids round-trip to server, or page refresh

Where JavaScript Goes

- Place JavaScript within a script tag
 - Can occur anywhere in the document, for now put it in the <head>

```
<script>  
    // code goes here  
</script>
```

- The JavaScript can also be loaded from external URL
 - Unlike many HTML elements, it requires the closing tag

```
<script src="./js/imageCycle.js"></script>
```

The code in the js file must not have <script> tags

- Having JavaScript in a separate file improves:
 - Reusability: if you have multiple pages but want to use same or similar functions
 - Readability: increases the readability of your HTML code
- In this course, most JavaScript will be in external files

Example JavaScript: Cycling Images

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device
<meta http-equiv="X-UA-Compatible" content=
<link rel="stylesheet" href="./css/style.cs
<title>JavaScript Example</title>

<script src="./js/imageCycle.js"></script>

</head>

<body onload="cycleBanner()">
<h1>Banner Images</h1>
<p>This image below should cycle repeatedly and clicking on the image at any time should take you to an
appropriate page</p>
<a href="javascript:getImageDestination()"></a>
</body>
</html>
```

```
var banner = ['images/600x399-1.jpg', 'images/600x399-2.jpg', ...];
var bannerDestination = ['Algodones_Dunes', 'Antelope_Canyon', ...];
var imageno = 0;
function cycleBanner() {
    if (++imageno == banner.length) {
        imageno = 0;
    }
    document.bannerImage.src = banner[imageno];
    // change to next image every 3 seconds
    setTimeout('cycleBanner()', 3000);
}
function getImageDestination() {
    document.location.href = 'https://en.wikipedia.org/wiki/'
        + bannerDestination[imageno];
}
```

Try It Now: JavaScript



5 min

- View the file `Ch03\ImageCycle\image-cycle.html` in VSC
 - Load it in a browser and verify that it works

Why JavaScript?



Basic Syntax

Working with DOM

Handling Events

Web Application Security Client Side

Chapter Summary

Variables

- Variables in JavaScript are untyped
 - Will be coerced into appropriate type at runtime
 - Are case sensitive
- Naming conventions vary according to the programmer's first language

```
var javaProgrammerHere = 10.32;  
var $usually_write_perl = 'Hello';  
var _systemsProgrammer = true;
```

- Best practice is to use Java naming conventions
 - Map better to XML and JSON
- Uninitialized variables are assigned a special value of `undefined`
 - Can also assign a variable to `null`

Statements and Operators

- JavaScript belongs to the C-family of languages
 - Like C, C++, C#, Java, etc.
 - Statements end in a semicolon
 - Use curly braces to group a set of statements
 - Arithmetic operators: +, -, *, /, %, ++, --, +=, -=
 - Comparison operators: ==, !=, >, >=
 - Logical operators: &&, ||, !
 - Ternary operator

```
var greeting = 'Good ' + (hour < 12) ? 'Morning' : 'Evening';
```

- Comments
 - // comment until end of line
 - /* block comment */
 - <!-- HTML comment also works -->

Loops and Conditions

- Loops and conditions are also like the C-family of languages

We'll talk about
this shortly

```
var numEntries = 100;  
while (numEntries > 10) {  
    // do something  
    --numEntries;  
}
```

```
for (let numEntries = 100; numEntries > 10; --numEntries) {  
    // do something  
}
```

```
if (numEntries > 100) {  
    alert('Too many people in exhibit.');
```

```
} else if (numEntries == 0) {  
    closeExhibit();  
}
```

String in JavaScript

- The String API is very close to that of Java

```
var name = 'O\'Keefe';  
  
var lastChar = name.charAt(name.length - 1);  
  
var aposPosition = name.indexOf('\'); // -1 if not there  
  
var num = parseInt('6');
```

- Strings may use single or double quotes
 - Single quotes generally preferred in applications that handle a lot of embedded HTML
 - Because double quotes are used in HTML
 - Consistency is most important

Arrays

- Arrays grow dynamically
 - Unlike in C or Java
 - More like Java ArrayList
 - Syntax is generally like C array
 - Note the array initializer is [] rather than {} as in Java and C

```
// dynamically grows
var itemCosts = new Array();
itemCosts[0] = 51.2;
itemCosts[1] = 60.6;

// alternative, note: [], not {}
var itemCosts = [51.2, 60.6];

// length of an array
var numItems = itemCosts.length;

// resize array
itemCosts.length = 1;

// sparse array allowed
var itemCosts = [51.2,,,60.6];
```

Exercise 3.1: Jumping JavaScript



10 min

- View the web page Ch03\JavaScript\js-song.html with your browser.
- View the page source.
 - What happens from the time that the page is loaded?
 - Can you explain the resulting web page?

Functions

- Functions are declared with the keyword `function`
- Parameters are always passed by reference
 - Changes that affect the parameter within the function affect the variable that was passed in

```
function updateDetails(zipcode) {  
    zipcode.name = zipcode.city + ', ' + zipcode.state;  
}
```

Variable Scope

- Variables declared with `var` are in the scope of the function that encloses them
 - Global variables are those outside of any function
 - Function parameters are within the scope of the function
- What is the scope of each of the variables below?

```
var fullDetails = false;
function showDetails(zipcode) {
    var url = '../jaxrs/zipcode/json/' + zipcode;
    if (fullDetails) {
        var request = new XMLHttpRequest();
        request.open('GET', url, false);
        request.send();
    }
};
```

- Note that variables declared without `var` are always global: avoid this!

```
function showDetails(zipcode) {
    url = '../jaxrs/zipcode/json/' + zipcode; // oops global
    ...
}
```

Other Variable Scoping Rules

- Newer browsers understand the keywords `let` and `const`
 - Support true block-level scope, variable scope is set by the `{}` that contains it

```
var fullDetails = false;
function showDetails(zipcode) {
    const url = '../jaxrs/zipcode/json/' + zipcode;
    if (fullDetails) {
        let request = new XMLHttpRequest();
        request.open('GET', url, false);
        request.send();
    }
};
```

- Generally better than `var`, but it depends on the application scope
 - In a corporate environment, they are usually supported
 - In the wider world, supported by about 90% of all browser installations
 - For this, and other features, check <https://caniuse.com>

Careful with Global Variables!

- Global variables are all within the same global scope
 - Even if they are loaded from different files:

```
var fullDetails = false;
function showDetails(zipcode) {
    var url = '../jaxrs/zipcode/json/' + zipcode;
    if (fullDetails) {
        ...
    }
};
```

show-zipcodes.js

```
var fullDetails = new Object();
function updateDetails(zipcode) {
    fullDetails.zipcode = zipcode;
}
```

update-zipcodes.js

- Best practice is to reduce/eliminate global variables

JavaScript Idioms

- The `||` operator evaluates to `false` if a variable is `undefined`

First “true” value is assigned

```
var numEntries = numStocks || pref.portfolioLength || 10;
```

- Note that `0`, `null`, `NaN` (not a number) and `' '` all also evaluate to `false`
- Can also use the special value `undefined`

```
if ( numEntries == undefined ){  
    numEntries = 10;  
}
```

- Three uses of `=`

```
numEntries = 3;           // assignment  
if (numEntries == 3);      // check equality  
if (entry === this.entry); // strict equality
```

Objects

- Objects in JavaScript are not quite as formal as objects in Java or C#
 - Simply a collection of properties and methods
 - In fact, one alternative syntax is like a Perl associative array

```
// creating an Object and setting its values
var item = new Object();
item.id = 23024;
item.cost = 60.6;

// alternative Perl-like syntax
var item = new Object();
item['id'] = 23024;
item['cost'] = 60.6;

// object-literal alternative syntax
var item = {id: 23024, cost: 60.6};
```

Objects Are Hierarchical

- Objects can be composed of other objects

```
var item = new Object();  
item.id = 23024;  
item.cost = 60.6;  
item.location = new Object();  
item.location.lat = 35.3;  
item.location.lon = -97.1;
```

Objects Are Like Associative Arrays

- Objects are also like associative arrays
 - Can loop through all the properties of an object

```
var item = {  
  id: 23024, cost: 60.6,  
  location: {lat: 35.3, lon: -97.1}  
};  
var prop;  
for (prop in item) {  
  if (prop == 'cost') {  
    // do something with item[prop]  
  }  
}
```

- Properties can be added or deleted

```
item.salePrice = item.cost * 1.05;  
delete item.location;
```

Defining “Classes”

- JavaScript idiom for constructor, private properties, and public methods

```
function Item(id_, cost_, price_) {  
  var id = id_;  
  var cost = cost_;  
  var price = price_;  
  this.getId = function () {  
    return id;  
  }  
  this.getCost = function () {  
    return cost;  
  }  
  this.getPrice = function () {  
    return price;  
  }  
  this.getProfit = function () {  
    return (price - cost);  
  }  
}
```

function starts
behaving like a
constructor/initialize
r when using the
new keyword

```
var item = new Item(23021, 30.2, 31.4);  
var profit = item.getProfit();
```

Dialog Box

- To pop up a dialog box:

```
alert('Please enter a valid state code');
```

- Use sparingly as most users find alerts annoying

- For confirmation:

```
var resp = confirm('Are you sure?');  
if (resp) {  
    // delete all the work  
}
```

- To get a single value from the user

```
var resp = prompt('How many zipcodes?', '10');  
if (resp != null && resp != '') {  
    numZips = parseInt(resp);  
}
```

Exercise 3.2: JavaScript Arrays and Objects



15 min

- View the web page Ch03\JavaScript\objects.html with your browser.
- View the page source.
- Answer the following questions:
 - What happens from the time that the page is loaded?
 - Can you explain the syntax of the script?
 - Can you explain the logic? What is the advantage of organizing the varying data into arrays and objects?

Why JavaScript?

Basic Syntax



Working with DOM

Handling Events

Web Application Security Client Side

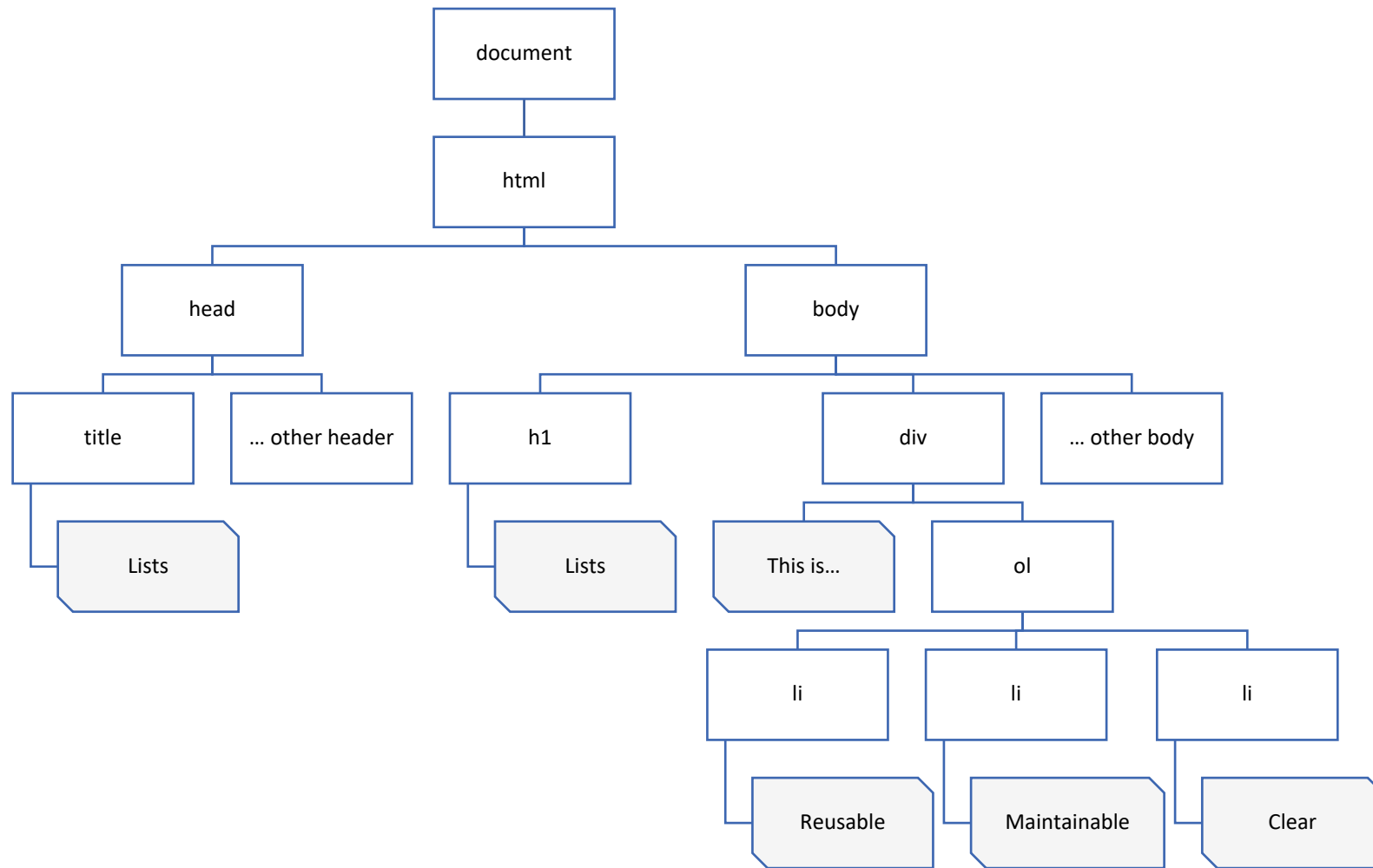
Chapter Summary

The Document Object Model (DOM)

```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Lists</title>
  ... other header content
</head>

<body>
  <h1>Lists</h1>
  <div>
    This is an ordered list:
    <ol>
      <li>Reusable</li>
      <li>Maintainable</li>
      <li>Clear</li>
    </ol>
  </div>
  ... other body content
</body>
</html>
```



Steps to Manipulate the DOM

- Can use JavaScript to manipulate the DOM using the DOM API
 - Write a JavaScript function
 - In a script tag or in an external file sourced from a script tag
 - Function will access one or more DOM elements using the API
 - Specify that function is to be invoked when a particular event occurs
 - onload, onmousemove, etc.
 - We will cover these events later
 - Write HTML page with the required elements

getElementById
getElementsByClassName
getElementsByTagName
querySelectorAll

```
function insertSong() {  
    numZips = getLoopLength();  
    document.getElementById('song').innerHTML  
        = createSong(numZips);  
}
```

```
<body onload="insertSong()">  
    ...  
</body>
```

```
<h1>10 Little Zipcodes Event Driven</h1>  
<div id="song">  
    Song replaces this text when JavaScript finishes executing.  
</div>
```

Manipulating the DOM

```
<body onload="insertSong()">
```

Run this JavaScript code when the element has been loaded. Important not to try to manipulate the DOM before the document has been properly parsed.

```
<h1>10 Little Zipcodes Event Driven</h1>
```

```
<div id="song">
```

Song replaces this text when JavaScript finishes executing.

```
</div>
```

```
</body>
```

The id of element

```
function createSong(zipCount) {  
  var song = '';  
  while (zipCount > 0) {  
    song += zipCount + ' little zipcodes jumping on the bed<br>';  
    song += ' one fell down and broke his head.<br>';  
    --zipCount;  
  }  
  song += 'No more zipcodes jumping on the bed<br>';  
  return song;  
}
```

The id of element

Alternative way of declaring the event handler (we will cover this later)

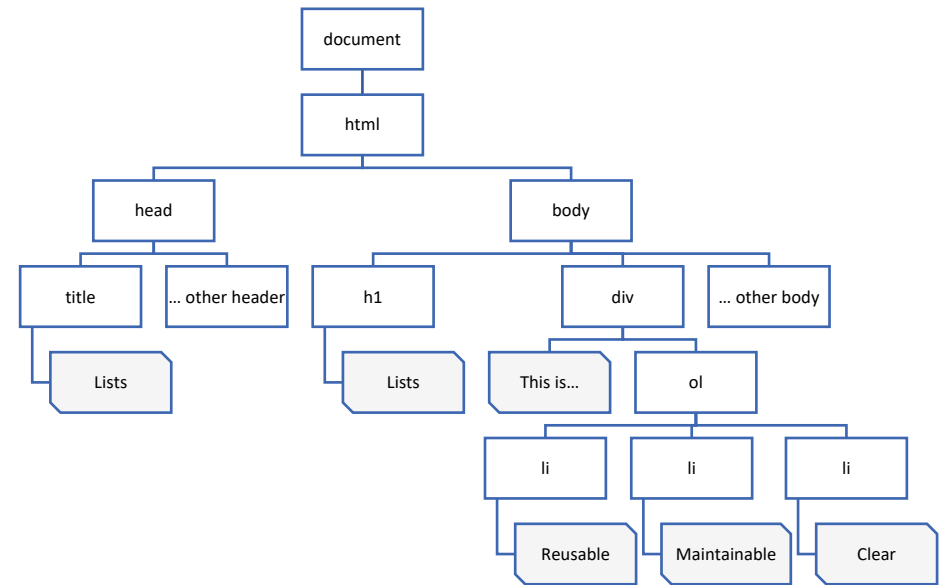
```
function insertSong() {  
  document.getElementById('song').innerHTML = createSong(10);  
}
```

```
window.onload = insertSong;
```

Not document.write() but ...

DOM API

- The DOM is represented as a tree of nodes and elements
 - Every HTML element is a node
 - Nodes are also created for element contents (e.g., textual content)
- The normal entry point is the global `document` object to get one or more elements
- Methods that return a “live” list of elements
 - `getElementsByClassName()`
 - `getElementsByTagName()`
- Method that returns a single element
 - `getElementById()`
- Methods that return a static list of nodes from CSS selectors
 - `querySelector()`
 - `querySelectorAll()`



Examples of Manipulating Elements

- Replace the tree below an element
 - All the HTML contained in it
- Change the HTML class associated with an element
 - It is really a list of classes, better to use `classList`
- Change the contents of a text node
- Change an attribute

```
document.getElementById('song').innerHTML = song;
```

```
document.getElementById('songtitle').className = 'highlight';
```

```
var classes = document.getElementById('songtitle').classList;  
classes.add('highlight');  
classes.remove('highlight');
```

```
var title = document.getElementsByTagName('title')[0];  
title.childNodes[0].nodeValue = 'No more zipcodes!';
```

```
document.getElementById('ad').setAttribute('src', urlToShow);
```

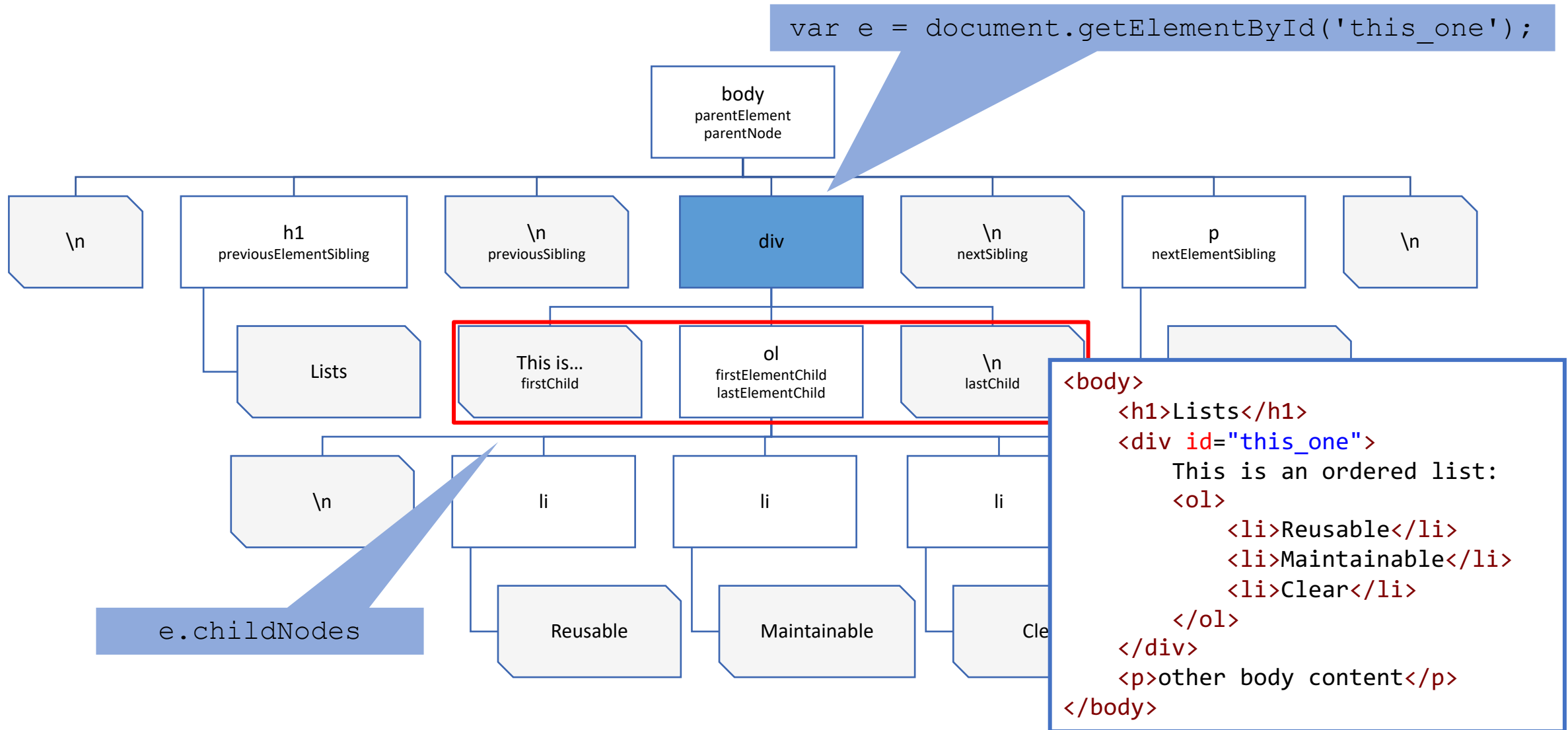
Navigating from an Element

- Can navigate from an Element:
 - `childNodes`
 - `firstChild`, `firstElementChild`
 - `lastChild`, `lastElementChild`
 - `nextSibling`, `nextElementSibling`
 - `parentNode`, `parentElement`
 - `previousSibling`, `previousElementSibling`
- To remove an element from the document:

```
var sep = document.getElementById('separator');  
sep.parentNode.removeChild(sep);
```

- Similarly, `appendChild()`, `replaceChild()`, `insertBefore()`

Relative Position in the DOM



Exercise 3.3: Manipulating the DOM



10 min

- View the web page Ch03\ChangeDOM\change-dom.html with your browser.
- View the HTML page and its associated JavaScript file in VSC.
- Modify the web page to include a new div section.
- In the JavaScript file, define a new function that displays a positive, uplifting message of the day in the new `div` element that you just defined in the web page.
- Modify the web page so that your new function will be called when the page is loaded.
 - Your message should then be displayed when the page is viewed.
- How can you have both the original function and your new function be called when the page is loaded?

JavaScript Built-in Classes

- Math

```
var lower = Math.floor(32.2); // 32
var rnd = Math.random(); // 0 to 1
```

- Date

```
var now = new Date(); // a Date object
var alsoNow = Date.now(); // a number (milliseconds since epoch)
var fixedTime = new Date(2019, 1, 31, 12, 1, 31, 300);
var timeString = fixedTime.toLocaleString('de-DE');
```

- Number

```
var maxInt = Number.MAX_SAFE_INTEGER;
var num = Number.parseFloat("1234.56");
var output = num.toFixed(1); // "1234.6"
var exp = num.toExponential(3); // "1.235e+3"
var fmt = num.toLocaleString('fr-FR'); // "1 234,56"
```

Optional after month,
default to lowest value

Prefer primitive numbers, but Number is
helpful for type conversions

Exercise 3.4: Working with Built-In Classes



15 min

- View the web page `Ch03\Random\random.html` with your browser.
- Notice that the image cycles as it did before, but this time the images are spresented in a random sequence.
- View the HTML page and its associated JavaScript file in VSC.
- Create a new page that will randomly select an inspirational message and display it in your page.
- If you have time, display the date along with the message. Display the date in a foreign format or convert it to UTC.

Why JavaScript?

Basic Syntax

Working with DOM



Handling Events

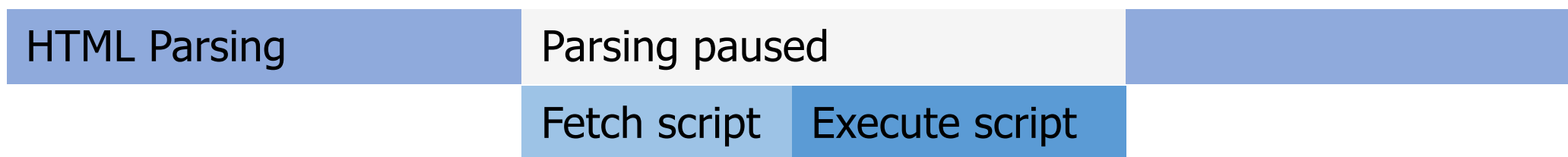
Web Application Security Client Side

Chapter Summary

1. event
2. event handler
3. trigger/fire event
4. register and event
5. unregister an event
6. callback function

The `onload` Event Handler

- Have already seen this
 - Define a JavaScript function to execute when the page has finished loading
 - `onload` declares an event handler for the `load` event
 - *Note:* `onload` attribute in `body` tag and `window.onload` are the same
- Can also use `onload` with other items that are “loaded”, such as:
 - `img`
 - `iframe`
- Why?
 - Remember JavaScript executes as soon as it is encountered



- If the script refers to a DOM Element that has not been parsed, it will fail

When?

- When is the `load` event fired?
 - After document has been parsed *and* other resources (images etc.) have been loaded
- If the purpose is to wait until the DOM has been parsed, there are better options

- `DOMContentLoaded`
(no “on” attribute)

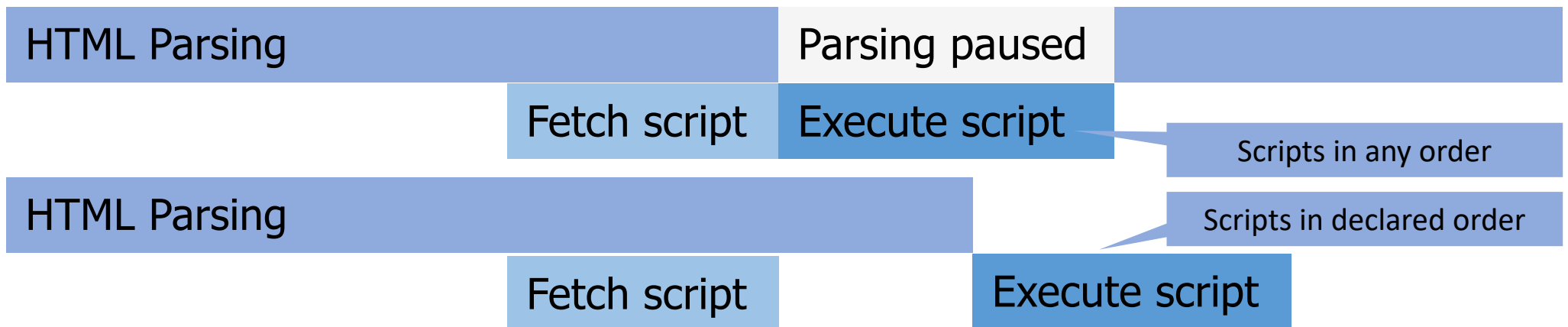
```
window.addEventListener('DOMContentLoaded', (event) => {  
    console.log('DOM fully loaded and parsed');  
});
```

- `defer` and `async`

```
<script defer src="randomDefer.js"></script>
```

`async`

`defer`



JavaScript Event Handlers

- Here are some of the event handlers that can be registered:

| Event handler name | Event name | Example of when it would be triggered |
|--------------------|-------------|--|
| onabort | abort | User aborts the loading of an image |
| onblur | blur | Item loses focus |
| onchange | change | User checks on/off a checkbox |
| onclick | click | User clicks item |
| oncontextmenu | contextmenu | User right-clicks, before menu is shown |
| ondblclick | dblclick | User double-clicks item |
| onerror | error | Link/URL not available |
| onfocus | focus | Item gains focus |
| onload | load | Page is loaded |
| onmouseenter | mouseenter | When mouse enters the area of an element |

Attaching Event Handlers

- Prefer assigning event handlers in code: keep separate from HTML
- Attach event handlers after the DOM has been parsed
 - Either use `defer` on a script
 - Or assign the handler in the `onload` event handler

```
window.onload = function () {  
    document.getElementById('active').onmouseover = insertSong;  
}
```

Function reference, no parentheses

- This code uses an anonymous function
 - Hard to test, restrict use to trivial code that does not need to be tested, as here
- As we saw earlier, can also attach a handler using `addEventListener`
 - Handlers can be attached to a wider range of objects
 - Some events cannot have handlers attached any other way



```
document.getElementById('active').addEventListener('mouseover', insertSong);
```

Passing Parameters to an Event Handler

- As already noted, the event handlers are function references
 - There are no parentheses and no parameters

Function reference, no parentheses

```
document.getElementById('clickMe1').onclick = insertNewSong;
```

- The handler may be defined with an optional parameter of type `Event`

```
function insertNewSong(e) {  
    console.log(e);  
}
```

- To pass other parameters, need to create an anonymous function

```
document.getElementById('clickMe2').onclick = function() {  
    insertNewerSong(2);  
}
```

As well as constants, has access to any variables in scope when it is defined

This is the event handler, not the function it calls

event propagation inside-out bubbling
capturing

Exercise 3.5: Responding to Events



15 min

- View the web page `Ch03\EventHandler\one-by-one.html` with your browser.
- Open the HTML and its associated JavaScript file in VSC.
- Modify the `addEventListener` function to register an event handler for the `mouseout` event.
- Verify the web page performs as expected.
- Try the `click` event instead.

Event Bubbling `this` and `event.target`

- After an event triggers on an element, it then triggers on its parents in nesting order
- The deepest element (the one triggered) is the target
 - In W3C compliant browsers: `event.target`
 - In IE8: `event.srcElement`
- The element the event has bubbled up to is `this`
 - The element currently handling the event

```
<div id="d1">
  Grandparent div 1
  <div id="d2">
    Parent div 2
    <div id="d3">
      Child div 3
    </div>
  </div>
</div>
```

```
let divs = document.getElementsByTagName('div');
for (let i = 0; i < divs.length; i++) {
  divs[i].onclick = function (e) {
    e = e || event;
    let target = e.target || e.srcElement;
    console.log('target=' + target.id + ', this=' +
this.id);
  }
}
```

Stopping Events

- Can stop event bubbling using the method `event.stopPropagation()`
 - In IE8, this was `event.cancelBubble()` on the global event object
- This may seem like a good idea sometimes, but it almost never is
 - Avoid doing it! Find another way
 - Stopping event propagation makes it hard to have event handlers for standard behavior
 - Hard to predict every single case when events should, or should not, propagate
- In most cases, there is a way to achieve a similar effect by stopping the default behavior
 - `event.preventDefault()`
 - E.g., stop a form being submitted, prevent a checkbox from changing state
 - Events that can be stopped have `event.cancelable == true`
 - Can check whether it has been done: `event.defaultPrevented`
 - Event handlers can decide for themselves whether they should handle such an event

Event Bubbling



5 min

- Your instructor will demonstrate event bubbling

Form Validation

- JavaScript used to be the only way to validate forms
 - Since HTML5, most validation is better done declaratively in HTML
- Continue to use JavaScript for more advanced validation
 - Cross-field validation
 - Validation that cannot be implemented as a regular expression or limit
- When working with an individual form control, can override validation
 - `setCustomValidity(message)`
 - If `message` is not empty, this marks the control as invalid and displays `message`
 - Must manually reset validity when the control value changes
 - Enables the pseudo-classes `:invalid` and `:valid` to give visual feedback

Form Events

- Common events in form validation

| Event | Triggered When... |
|----------------|--|
| change | ... a form control changes value. For a text field, this is when tabbing out or clicking away. For a <code>SELECT</code> , when choosing a value. For a checkbox or radiobutton, when clicking on the control. |
| input | ... the item value changes (for every change). |
| keydown, keyup | ... a key is pressed in the control. These events occur before <code>change</code> or <code>input</code> . |
| invalid, valid | ... validation is run, depending on the result. |
| reset | ... the reset button is pressed. |
| submit | ... the submit button is pressed. |

- Or use the submit button `click` event and call `setCustomValidity()` on a control

Exercise 3.6: Form Validation with JavaScript



20 min

- View the web page `Ch03\Forms\form-exercise.html` with your browser.
 - Try submitting the form with a “to” date that is later than the “from date”.
 - What happens?
- Open the HTML in VSC.
- Add validation to ensure the dates are the right way around.
- There are two main approaches (shown below). Choose one and follow the instructions for that section.
 - Put validation in the click event of the button.
 - Put validation in the submit event (and optionally in the change event).
- While you are working, you may wish to assign an event handler to the submit event that just calls `event.preventDefault()` or remove the action from the form. Make sure to remove this code when you have it working

Why JavaScript?

Basic Syntax

Working with DOM

Handling Events



Web Application Security Client Side

Chapter Summary

How JavaScript Security Works

- Executable content within web pages are a security issue
 - Much more than static content
 - Arbitrary code can be executed on user's computer
- Have to be aware of security constraints on JavaScript
- Who executes the code?
 - The browser (trusted by the user)
- Where is the code?
 - Served out from a website (not trusted by the user)
- JavaScript security relies on the fact that the code is interpreted and run by a trusted program
 - Not possible for JavaScript code to access resources that the JavaScript engine does not allow access to

JavaScript Sandbox

- JavaScript does not provide:
 - Any way to write or delete files or directories on client computer
 - No way to plant a virus on user's system
 - No way to delete a user's data
 - Any networking primitives
 - Has a way to load URLs and send HTML form data to URLs
 - But cannot open a socket connection
 - Therefore, safe when accessed behind firewall
- The sandbox model fails if JavaScript can control a browser plugin which can write or delete files or open socket connections
 - This is a frequent issue with Microsoft ActiveX controls

JavaScript Restrictions

- Browsing history
 - The JavaScript program can tell the browser to navigate `back()`, `forward()`, or `go()`, but the actual URL being navigated to is not known to the program
 - Otherwise, privacy issues arise
- Restricts the reading of files
 - The filename to be uploaded can be set by the user via a dialog box
 - The filename cannot be set by the script
 - Otherwise, the script can read arbitrary files on user's machine
- Privacy of email addresses
 - The script cannot submit a form to a `mailto:` URL
- Protection of browser sessions
 - A script cannot close a window it did not open
 - Windows opened by script have be larger than 100px on a side, cannot be larger than screen size and cannot be moved off screen

Same-Origin Restriction

- Prevent event spoofing
 - A script cannot set properties of an Event object
 - Cannot register listeners for documents loaded from a different server than the script
 - A specific case of what is known as the same-origin restriction
- Cannot access predefined properties of client-side objects from a different origin
 - If your web page is served out from `widgets.acme.com`, the only documents you have unrestricted access to are documents from `widgets.acme.com`
- If you are serving out a document from `widgets.acme.com` and you want the document to be available to scripts in documents served out from `sales.acme.com`, then you need to explicitly set the domain property on both the documents to `acme.com`

```
document.domain = 'acme.com';
```

Chapter Concepts

Why JavaScript?

Basic Syntax

Working with DOM

Handling Events

Web Application Security Client Side



Chapter Summary

Chapter Summary

In this chapter, we have discussed:

- JavaScript is a way of adding behavior to web pages
 - Can be used to carry out client-side computation and to update the web page in-place
 - JavaScript is a dynamically typed language
 - Syntax is like the C-family of languages
- The browser parses a HTML document and creates a DOM
 - Can manipulate the DOM using JavaScript
 - Can assign a JavaScript function to be called when an event happens
- JavaScript security relies on a sandbox where dangerous operations are restricted
 - Events, document properties, etc. are subject to a same-origin policy