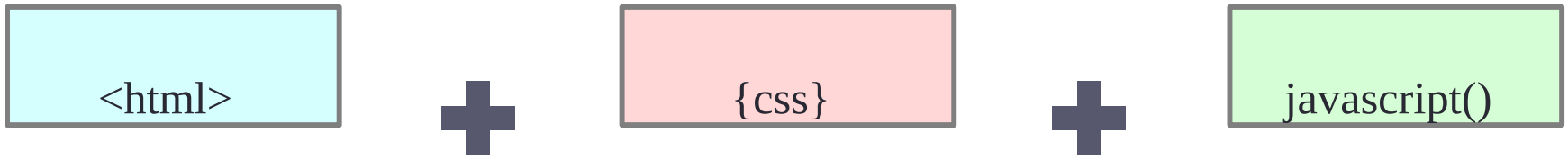
JavaScript

New Programming Languages for Web Applications

Vu Thi Hanh

0948 066 991

How HTML, CSS, and JS fit together



Content layer

The HTML gives the page structure and adds semantics

Presentation layer

The CSS enhances the HTML page with rules that state how the HTML content is presented

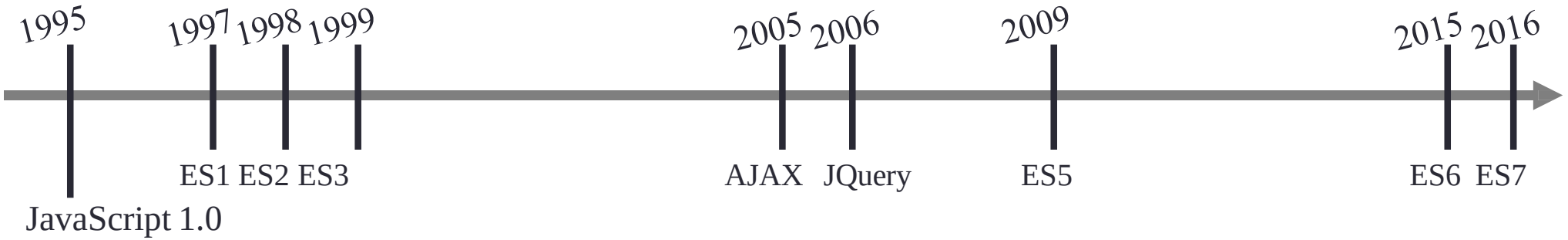
Behavior layer

The JS controls how the page behaves, adding interactivity

JavaScript: Some History

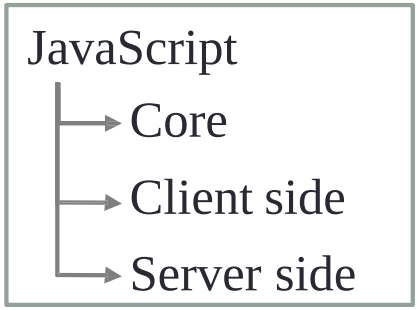
- JavaScript was introduced as part of the Netscape 2.0 browser
- Microsoft soon released its own version called Jscript
- European Computer Manufacturers Association (ECMA) developed a standard language known as ECMAScript
- ECMAScript Edition 6 is widely supported and is what is commonly called "JavaScript"
- JavaScript = ECMAScript + DOMAPI

  Language specification Communication with HTML



JavaScript

- JavaScript (JS) is a high-level, interpreted, dynamic programming language primarily used to make web pages interactive.



- **Key roles of JavaScript:**
  - Client-side web programming
  - Server-side development (Node.js)
  - Mobile apps (React Native)
  - Desktop apps (Electron)
  - Game development
  - Data visualization & AI integration

Key features of JavaScript

- **Interpreted Language**: JS is interpreted, not compiled. Code is executed line by line by the browser or runtime → Makes development faster
- **Client-Side Execution**: JavaScript runs directly in the web browser → Reduces server load and improves performance + Enables instant user interaction (clicks, animations).
- **Dynamic Typing**: Variable types are determined at runtime.
- **Object-Based (Prototype-Based OOP):** JavaScript uses objects and prototypes instead of classical inheritance.
- **Event-Driven Programming:** Responds to events such as clicks, key presses, and page loads.

Key features of JavaScript

- **Asynchronous Programming:** Supports non-blocking operations using (Callbacks, Promises, async / await) → Essential for APIs and real-time apps.
- **Cross-Platform Compatibility:** Works on all major browsers and operating systems.
- **Rich Built-in Functions & APIs**: Built-in support for: DOM manipulation + Math operations + Date handling + Browser APIs (fetch, storage, geolocation)
- **Lightweight and Fast:** JavaScript engines (V8, SpiderMonkey) are highly optimized. Executes quickly in modern environments.

Why learn JavaScript

- It is the programming language of the browser
- Build very interactive user interfaces with framework like React
- Used in building very fast server side and full stack application
- Used in mobile development (React Native, Native Scripts, …)
- Used in desktop application development

JavaScript Characteristics

- JavaScript does not need to be compiled
  - JS is an interpreted language
  - A JS interpreter is software that runs inside a browser that reads and executes JavaScript

- Interpreted vs. compiled languages:
  - Advantage: simplicity
  - Disadvantages: efficiency, maintainability, scalability, reliability

Why and Why Not JavaScript?

- What can be done with JS on the client and cannot be done with other techniques on the server?
  - Monitor user events and take actions
  - Some dynamic effects

- What can be done on both client and server, but are better with JS?
  - Build HTML dynamically when page is loaded
  - Interactive web pages
  - Communicate with the server asynchronously (Ajax)

- What are the drawbacks of JS?
  - Platform dependent
  - Can be turned off by users
  - Performance depends on the user's hardware, not the server
  - Security – JS code is visible in the browser
  - Hard to write reliable and maintainable JS

Embedding JS in HTML

Use the <script> tag to embed JS code in <head> or <body>

```
<script type="text/javascript">
   // code goes here
</script>
```

- Functions and code that may execute multiple times are typically placed in the <head> section
  - These are only interpreted when the relevant function or event- handler are called

- Code that needs to be executed only once, when the document is first loaded is placed in the <body> section

Embedding JS in HTML

- JS code may be written in a separate file.The external file can be included in an HTML file using the src attribute of a <script> tag

```
<script type="text/javascript" src="path/to/file.js"></script>
```

- JS code is visible in the client browser
  - Do not “hardcode” anything that you don’t want the client to see

Script Calls

Some script calls may be embedded in the HTML tags

```
<select name="country" onchange="jmp(url)">
```

or

```
<a href="javascript:newWindow('resources/JSPWebResources.html')">JSP resources</a>
```

```
function newWindow(url)
{
   hWnd = window.open (url,"HelpWindow","width=410,height=180,resizable=yes,scrollbars=yes");
}
```

Script is evaluated once encountered by browser

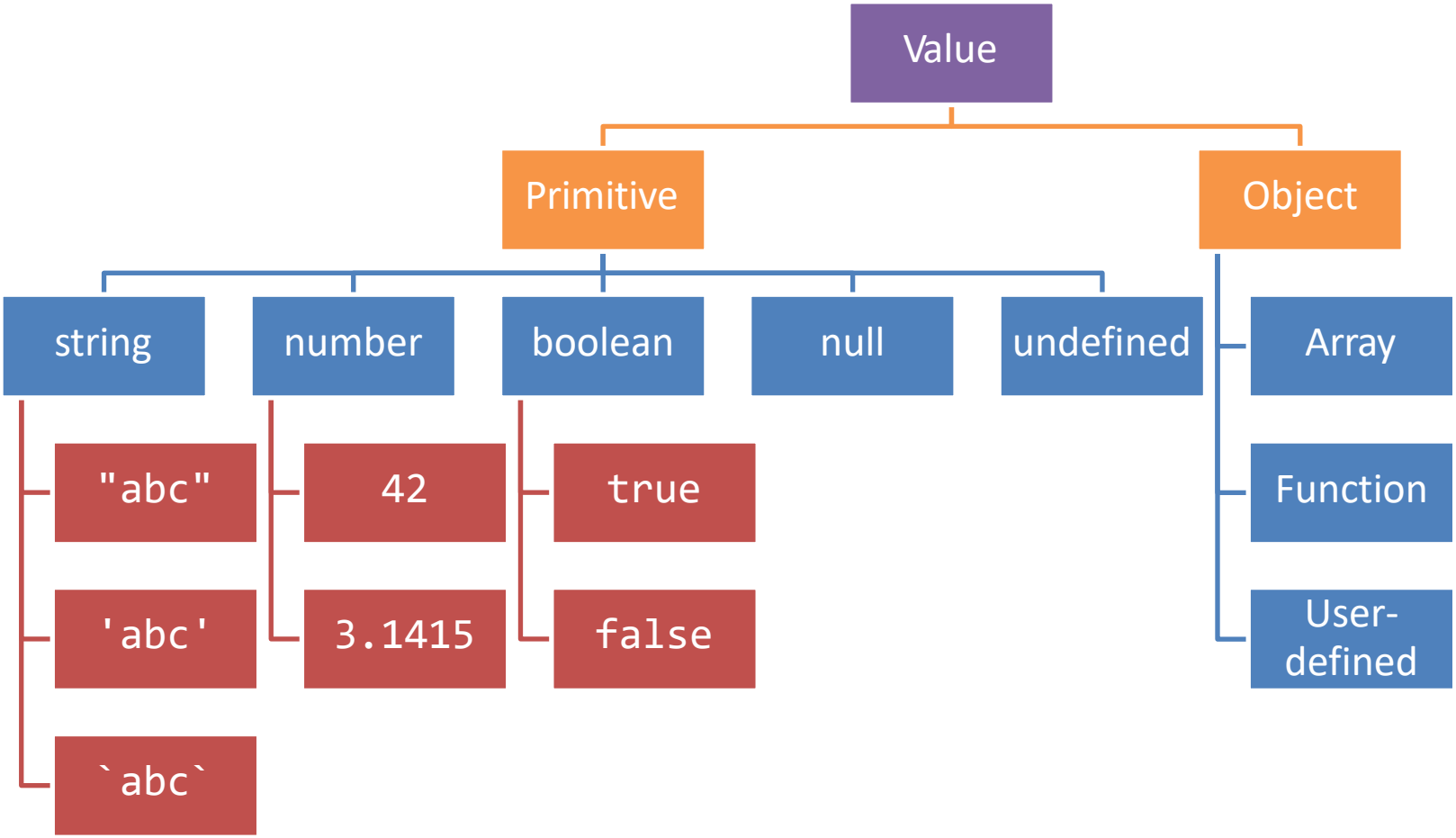
First Example

```
<!DOCTYPE html>
<html>
<head>
<title>First JavaScript Example</title>
</head>
<body>
<h2>This line is straight HTML</h2>
<h3>
<script type = "text/javascript">
    document.write("These lines are produced by<br/>");
    document.write("the JavaScript program<br/>");
    alert("Hey, JavaScript is fun!");
</script>
</h3>
<h2>More straight HTML</h2>
<script type = "text/javascript" src="file.js"></script>
</body>
</html>
```

Variables

- Variables are loosely typed
- Type is determined dynamically based on the value stored
  - The `typeof` operator can be used to check type of a variable
- Declarations are made using the `var` keyword
  - Variables declared but not initialized have the value undefined
  - Variables used and not declared or initialized have value Null
- Names start with letters, $, or _ followed by any sequence of letters, $, _, or digits
- Case sensitive

Variables


Value
Primitive
Object
string
number
boolean
null
undefined
Array
"abc"
'abc'
`abc`
42
3.1415
true
false
Function
User-defined

Variables

Variable is a container that hold things for later use

- String
- Number
- Undefined
- Null
- Objects (including arrays)
- Symbols
- Functions

```
var strVar = 'Hello';
var num = 10;
var undefinedVar;
var nulled = null;
var intArray = [1, 2, 3];
var sym = Symbol('Decscription of the symbol');
function setFocus(ele) {
    document.getElementById(ele).focus();
}
var focusVar = setFocus('firstName');
```

Variables

- Loose typing means that JS figures out the type based on the value

```
var x;      // type: Undefined
x = 2;      // type: Number
x = 'Hi';   // type: String
```

- Variables have block scope.
  - Declarations outside of any function are global
  - Declarations within a function are local to that function

Expressions

- If operator is + and an operand is string, it treats the + as a string concatenation operator and coerce other operand to string

```
var x = 'Hello';
var y = 4;
var result = x + y;   // 'Hello4'
```

- If operator is arithmetic, and string value can be coerced to a number, it will do so
- If string is non-numeric, result is `NaN` (NotaNumber)
- String can be explicitly converted to a number using `parseInt` and `parseFloat`

Using Arithmetic Operators

Let's initialize, y =

Operator	Description	Example	Resulting x
+	Addition	x = y + 5	9
		x = y + "5"	"45"
		x = "Four" + y + "4"	"Four44"
-	Subtraction	x = y - 2	2
++	Increment	x = y++	4
		x = ++y	5
--	Decrement	x = y--	4
		x = --y	3
*	Multiplication	x = y * 4	16
/	Division	x = 10 / y	2.5
%	Modulo	x = y % 3	1

Using Assignment Operators

Let's initialize, x =

Operator	Example	Equivalent arithmetic operators	Resulting x
=	x = 5	x = 5	5
+=	x += 5	x = x + 5	15
-=	x -= 5	x = x - 5	5
*=	x *= 5	x = x * 5	50
/=	x /= 5	x = x / 5	2
%=	x %= 5	x = x % 5	0

Applying Comparison and Conditional operators

Let’s initialize, x = 10

Operator	Description	Example	Result
==	Equal to (value only)	x == 8	false
		x == “10”	true
===	Equal to (value and type)	x === 10	true
		x === “10”	false
!=	Not equal (value only)	x != 5	true
!==	Not equal (value and type)	x !== “10”	true
		x !== 10	false
>	Greater than	x > 5	true
>=	Greater than or equal to	x >= 10	true
<	Less than	x < 5	false
<=	Less than or equal to	x <= 10	true

Chaining Multiple Comparisons with Logical Operators

Let's initialize, x = 10 and y =

Operator	Description	Example	Result
&&	And	(x == 10 && y == 5)	true
		(x == 10 && y > x)	false
||	Or	(x >= 10 || y > x)	true
		(x < 10 && y > x)	false
!	Not	!(x == y)	true
		!(x > y)	false
	Mix	(x >= 10 && y < x || x == y)	true
		((x < y || x >= 10) && y >= 5)	true
		(!(x == y) && y >= 10)	false

Control Structures (if Statement)

```
if (x == 5) {
    do_something();
}

if (x == 5) {
    do_something();
} else {
    do-something_else();
}

if (x < 5) {
    do_something();
} else if (x < 10) {
    do_something_else();
} else {
    do_nothing();
}
```

Control Structures (switch Statement)

```
switch (expression) {
    case value1:
        // code to execute
        break;
    case value2:
        // code to execute
        break;
    default:
        // code to execute if not value1 or value2
}
```

Looping (while Loop)

```
while (condition) {
    // code to execute
    // update to end the loop
}

var i = 1;
while (i < 5) {
    // code to execute
    i++;
}
```

Looping (do-while Loop)

```
do {
    // code to execute
} while (condition);

var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var i = 0;
do {
    var day = days[i++];
    console.log("It's " + day);
} while (day != "Wednesday");
```

Looping (for Loop)

```
for (assignment; condition; update;) {
    // code to execute
}
```

```
var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
var text = "";
var i;
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
```

Looping (for-in Loop)

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days) {
   console.log("It's " + days[idx] + "<br />");
}
```

Array Objects

- An array in JavaScript is a data structure used to store multiple values in a single variable.
- Creating Arrays

```
let numbers = [1, 2, 3, 4, 5];
let fruits = ["apple", "banana", "orange"];
let mixed = [1, "hello", true, null];
```

- Accessing & Modifying Elements

```
let fruits = ["apple", "banana", "orange"];

console.log(fruits[0]);   // apple
fruits[1] = "mango";      // modify
console.log(fruits.length); // 3
```

Array Objects

- Common Array Methods

```
// Common Array Methods
// Add / Remove Elements - push() – add to end
fruits.push("grape");
// pop() – remove from end
fruits.pop();
// unshift() – add to start
fruits.unshift("lemon");
// shift() – remove from start
fruits.shift();
```

Array Objects

- Looping Through Arrays

```
// for loop
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}

// for...of
for (let fruit of fruits) {
  console.log(fruit);
}

// forEach()
fruits.forEach(fruit => console.log(fruit));
```

Array Objects

- Array Methods for Data Processing

```
// map() - transform array
let numbers = [1, 2, 3];
let squared = numbers.map(n => n * n);
// [1, 4, 9]

// filter() - select elements
let numbers = [1, 2, 3, 4, 5];
let even = numbers.filter(n => n % 2 === 0);
// [2, 4]

// reduce() - combine values
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce((total, n) => total + n, 0);
// 10
```

Array Objects

- Search & Check Methods

```
// includes()
fruits.includes("apple"); // true

// indexOf()
fruits.indexOf("banana"); // 1

// find()
let numbers = [10, 20, 30];
let result = numbers.find(n => n > 15);
// 20
```

Array Objects

- Sorting & Reversing

```
// sort()
let nums = [3, 10, 2];
nums.sort((a, b) => a - b);
// [2, 3, 10]

// reverse()
nums.reverse();
```

Functions

- A function in JavaScript is a block of reusable code designed to perform a specific task. Functions help make code modular, reusable, readable, and easier to maintain.

```
function greet(name) {
  return "Hello, " + name;
}

console.log(greet("Alice"));
```

```
function name(paramName, ..., paramName) {
    statements;
}

function myFunction(name) {
    print("Hello, " + name + "!\n");
    print("How are you?\n");
}
```

Functions and Default Values (ES6)

```
function add(num1=10, num2=45) {
  return num1 + num2;
}
var r = add();      // 55
r = add(40);        // 85
r = add(2, 6);      // 8
```

Function Expression

- A function can be stored in a variable.

```
const add = function(a, b) {
  return a + b;
};

console.log(add(3, 4));
```

- Not hoisted
- Often used in callbacks
- Can be anonymous

Arrow Functions (ES6)

- Shorter and cleaner syntax.

```
const multiply = (a, b) => a * b;

console.log(multiply(5, 2));
```

- Short syntax
- No own this
- Ideal for callbacks and simple logic

Functions

- **Anonymous Functions:** Functions without names, often used as callbacks.

```
setTimeout(function() {
  console.log("Hello after 2 seconds");
}, 2000);
```

- **Callback Functions:** A function passed as an argument to another function.

```
function process(callback) {
  callback();
}

process(function() {
  console.log("Callback executed");
});
```

Objects in JavaScript

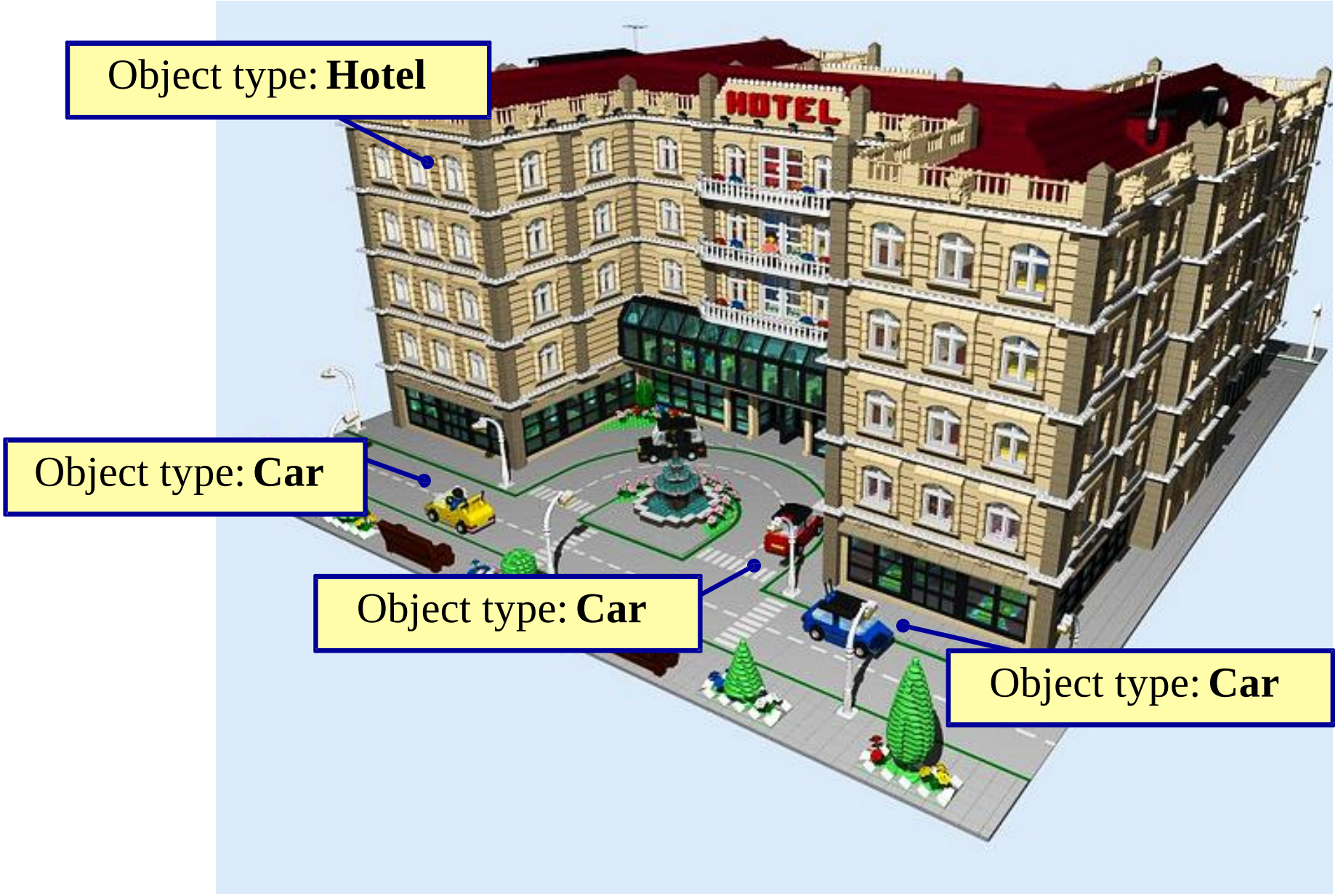
New Programming Languages

Web Applications

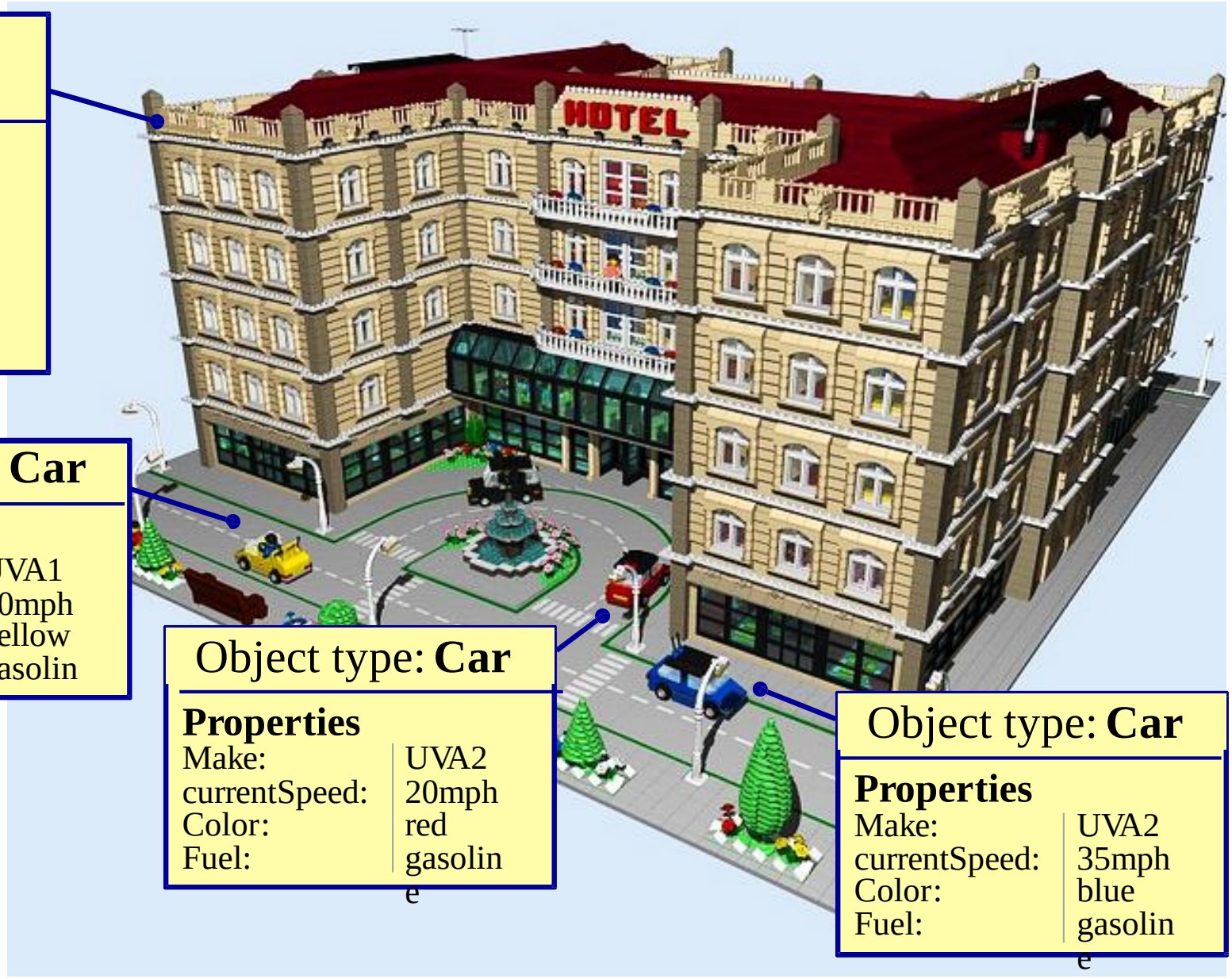
- User interactive software programs, deployed on a web server, accessed via a web browser
  - Browser features may affect the program's execution flow
- Use enabling technologies to
  - Make web site contents dynamic
  - Allow users of the system to implement business logic on the server
- Let users affect state on the server
- Constructed from diverse, distributed, and dynamically generated web components
  - Web components are software modules that implement different parts of the application's functionality

An enabling technology makes web pages interactive and responsive to user input

How do Web Apps fit in with the World Around Them?



Objects and Properties



Object type: **Hotel**

Properties

Name:	Awesome
Rating	5
Rooms:	70
Bookings:	56
Pool:	true
Gym	true

Object type: **Car**

Properties

Make:	UVA1
currentSpeed:	30mph
Color:	yellow
Fuel:	gasoline

Object type: **Car**

Properties

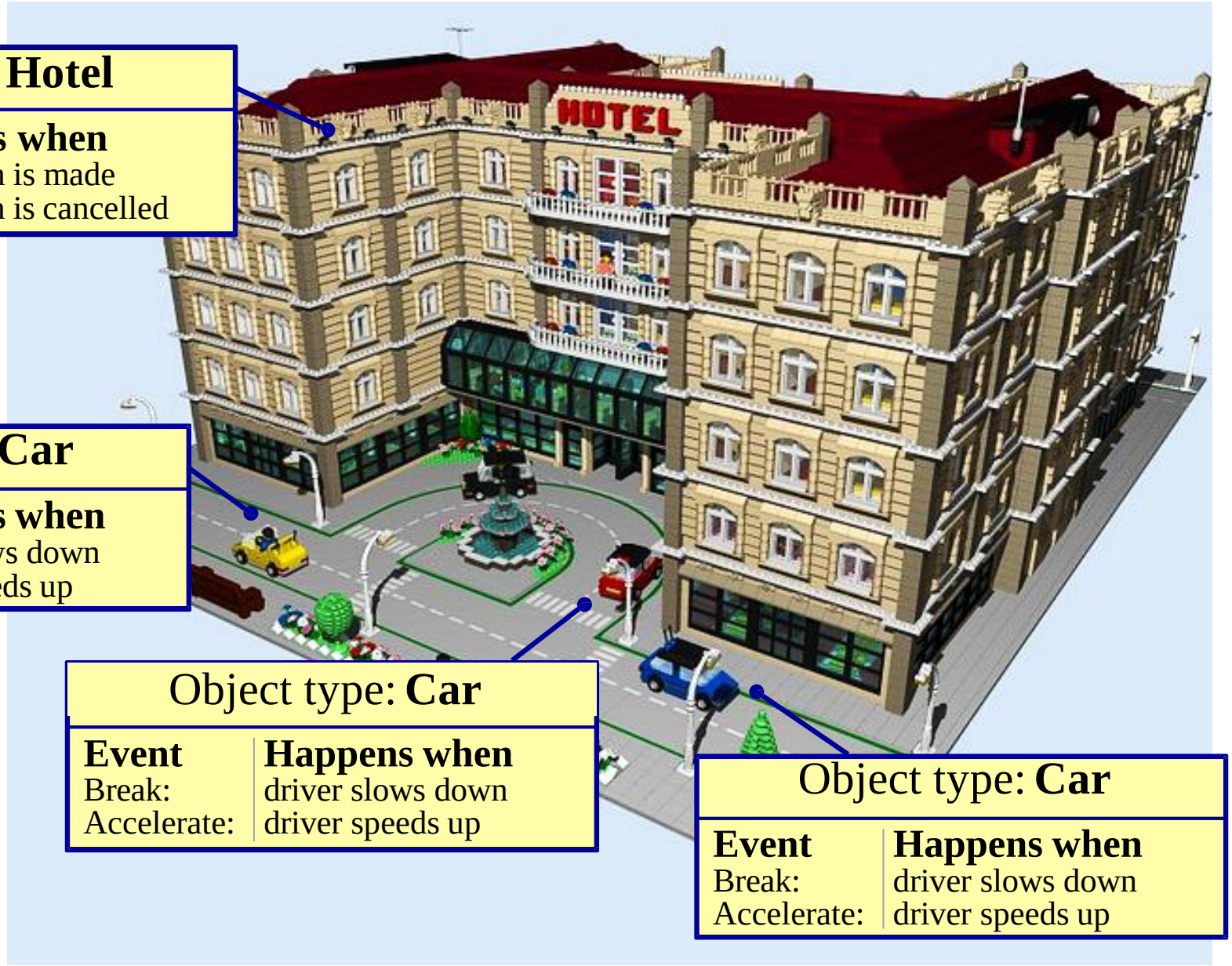
Make:	UVA2
currentSpeed:	20mph
Color:	red
Fuel:	gasoline

Object type: **Car**

Properties

Make:	UVA2
currentSpeed:	35mph
Color:	blue
Fuel:	gasoline

Objects and Events



Object type: **Hotel**

Event	**Happens when**
Reserve	reservation is made
Cancel	reservation is cancelled

Object type: **Car**

Event	**Happens when**
Break:	driver slows down
Accelerate:	driver speeds up

Object type: **Car**

Event	**Happens when**
Break:	driver slows down
Accelerate:	driver speeds up

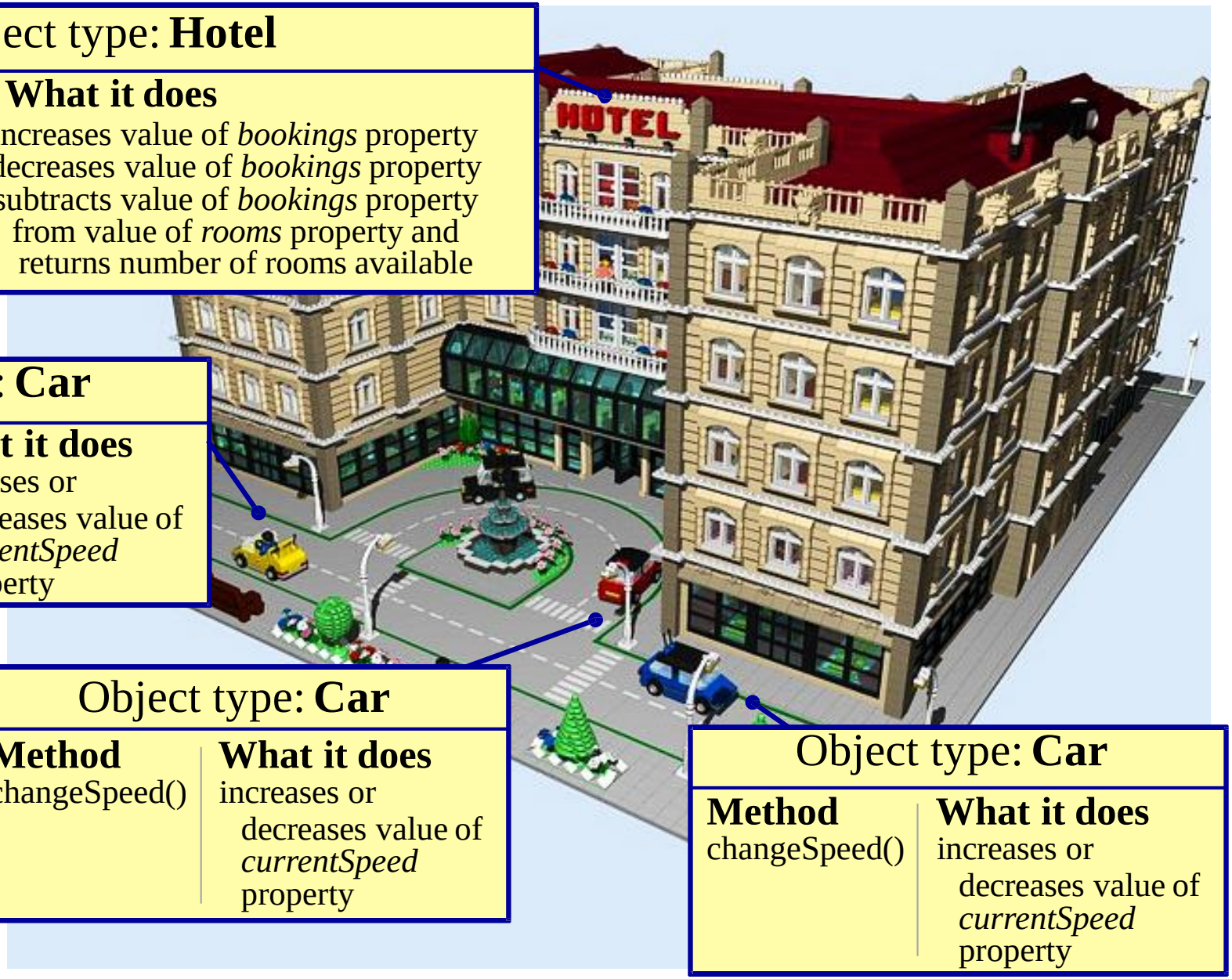
Object type: **Car**

Event	**Happens when**
Break:	driver slows down
Accelerate:	driver speeds up

Objects and Methods

Object type: **Hotel**	
Method	**What it does**
makeReservation()	increases value of *bookings* property
cancelReservation()	decreases value of *bookings* property
checkAvailability()	subtracts value of *bookings* property from value of *rooms* property and returns number of rooms available

Object type: **Car**	
Method	**What it does**
changeSpeed()	increases or decreases value of *currentSpeed* property



Object type: **Car**	
Method	**What it does**
changeSpeed()	increases or decreases value of *currentSpeed* property

Object type: **Car**	
Method	**What it does**
changeSpeed()	increases or decreases value of *currentSpeed* property

All Together

Object type: **Hotel**

Event	Happens when	Method called
Reserve	reservation is made	makeReservation()
Cancel	reservation is cancelled	cancelReservation()

Method	What it does
makeReservation()	increases value of *bookings* property
cancelReservation()	decreases value of *bookings* property
checkAvailability()	subtracts value of *bookings* property from value of *rooms* property and returns number of rooms available

Properties

Name:	Awesome
Rating:	5
Rooms:	70
Bookings:	56
Pool:	true
Gym:	true

Object type: **Car**

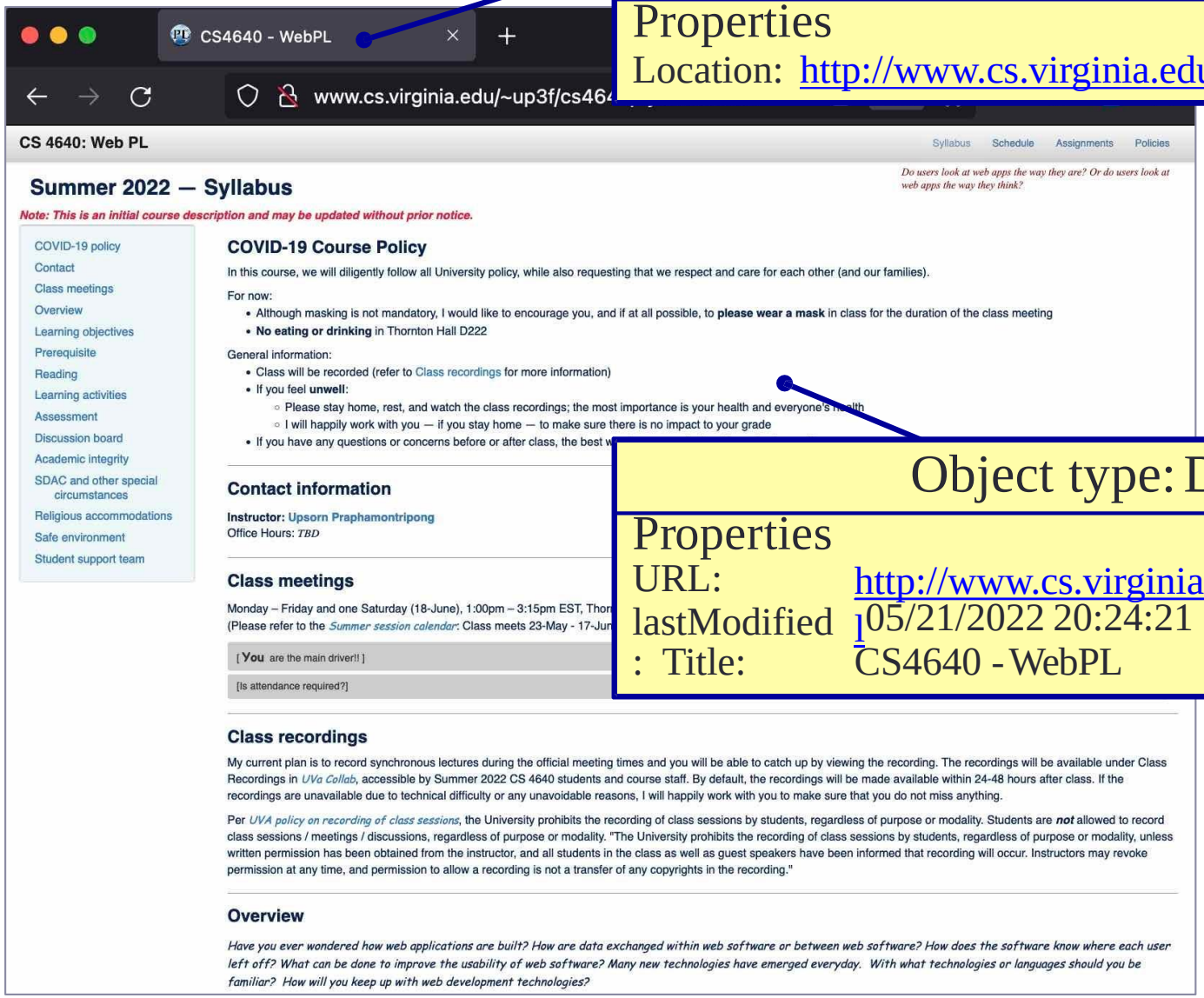
Event	Happens when	Method called
Break	driver slows down	changeSpeed()
Accelerate	driver speeds up	changeSpeed()

Method	What it does
changeSpeed()	increases or decreases value of *currentSpeed* property

Properties

Make:	UVA1
currentSpeed:	30
Color:	yellow
Fuel:	gasoline

Web Browsers and Objects



Object type: Window

Properties

Location: http://www.cs.virginia.edu/~up3f/cs4640/syllabus.html

Object type: Document

Properties

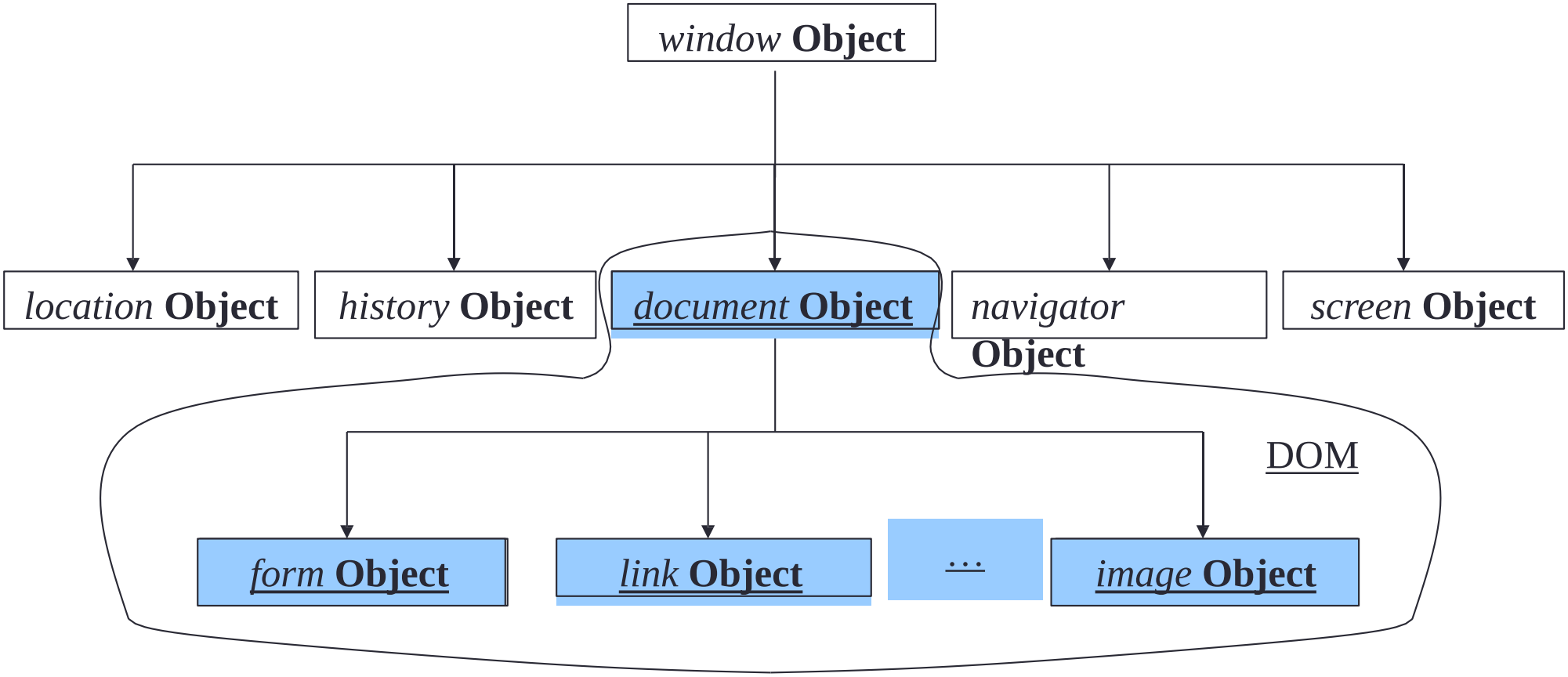
URL: http://www.cs.virginia.edu/~up3f/cs4640/syllabus.html

lastModified: 05/21/2022 20:24:21

Title: CS4640 - WebPL

BOM: Browser Object Model

- BOM – collection of objects that the browser makes available to us for use with JavaScript



BOM: Browser Object Model

- The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
- The window object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Global variables are properties of the window object.
- Global functions are methods of the window object.
- Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById("header");
```

BOM: Browser Object Model

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Window</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
"Browser inner window width: " + window.innerWidth + "px<br>" +
"Browser inner window height: " + window.innerHeight + "px";
</script>

</body>
</html>
```

JavaScript Window

Browser inner window width: 664px
Browser inner window height: 571px

Using BOM Objects (Some properties)

Property	Description
window.screenX	X-coordinate of pointer, relative to top left corner of screen (in pixels)
window.screenY	Y-coordinate of pointer, relative to top left corner of screen (in pixels)
window.location	Current URL of window object
window.document	Reference to document object
window.history	Reference to history object for browser window or tab, which contains details of the pages that have been viewed in that window or tab
window.history.length	Number of items in history object
window.screen	Reference to screen object
window.screen.width	Accesses width property of screen object
window.screen.height	Accesses height property of screen object

Using BOM Objects (Some methods)

Method	Description
window.alert( )	Create modal dialog box with message (user must click OK button to close it)
window.open(url)	Open new browser window with the specified URL
window.print( )	Tell browser that user wants to print contents of current page (act like user has clicked a print option)
window.history.back( )	Move backward through history
window.history.forward( )	Move forward through history
window.history.go(step)	Move to specific page from session history (step specifies the number of pages, forward or backward)
history.pushState(state, title, url)	Create a new entry (or add a URL) at the top of the browser history
history.replaceState(state, title, url)	Modify the current entry (current URL at the top) of the browser history

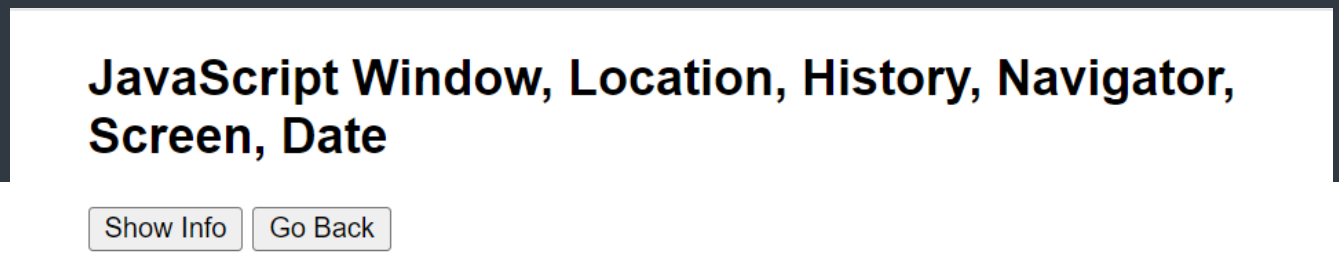
Using BOM Objects

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JS Window, Location, History, Navigator, Screen, Date</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        #container { max-width: 600px; margin: auto; }
        pre { background: #f4f4f4; padding: 10px; }
    </style>
</head>
<body>
    <div id="container">
        <h2>JavaScript Window, Location, History, Navigator, Screen, Date</h2>
        <button onclick="showInfo()">Show Info</button>
        <button onclick="goBack()">Go Back</button>
        <pre id="info"></pre>
    </div>
```

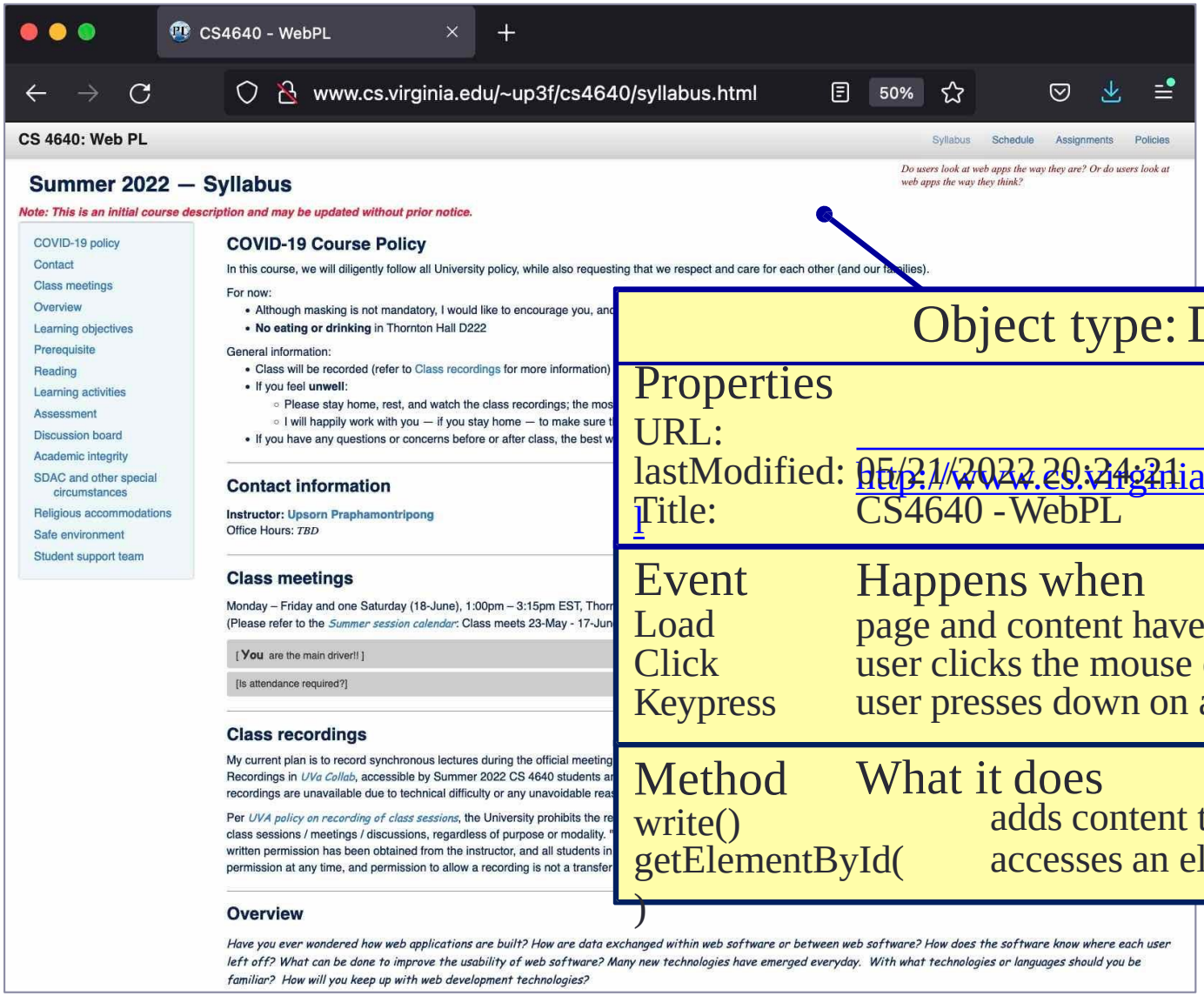
Using BOM Objects

```
    <script>
        function showInfo() {
            let info = `
            Window Inner Size: ${window.innerWidth} x ${window.innerHeight}
            Screen Size: ${screen.width} x ${screen.height}
            URL: ${location.href}
            Protocol: ${location.protocol}
            Hostname: ${location.hostname}
            Pathname: ${location.pathname}
            User Agent: ${navigator.userAgent}
            Platform: ${navigator.platform}
            Language: ${navigator.language}
            Online Status: ${navigator.onLine}
            Current Date & Time: ${new Date().toLocaleString()}
            `;
            document.getElementById("info").textContent = info;
        }

        function goBack() {
            if (history.length > 1) {
                history.back();
            } else {
                alert("No previous page in history.");
            }
        }
    </script>
</body>
```



DOM: Document Object Model



Object type: Document

Properties

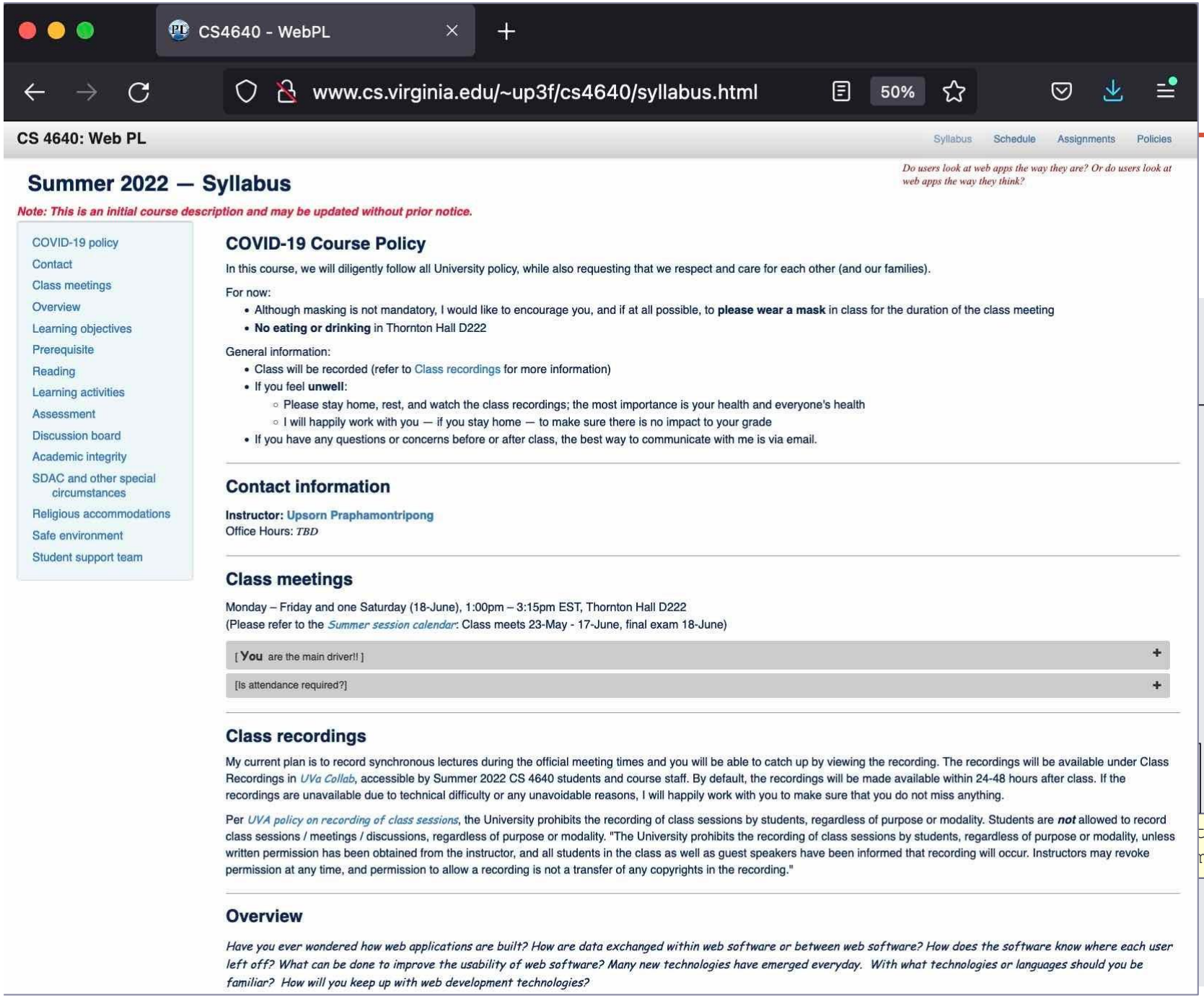
URL:	http://www.cs.virginia.edu/~up3f/cs4640/syllabus.html
lastModified:	05/21/2022 20:24:31
Title:	CS4640 - WebPL

Event	Happens when
Load	page and content have finished loading
Click	user clicks the mouse over the page
Keypress	user presses down on a key

Method	What it does
write()	adds content to the document
getElementById( )	accesses an element of a given *id* attribute

revisit

Page



The browser receives an HTML page

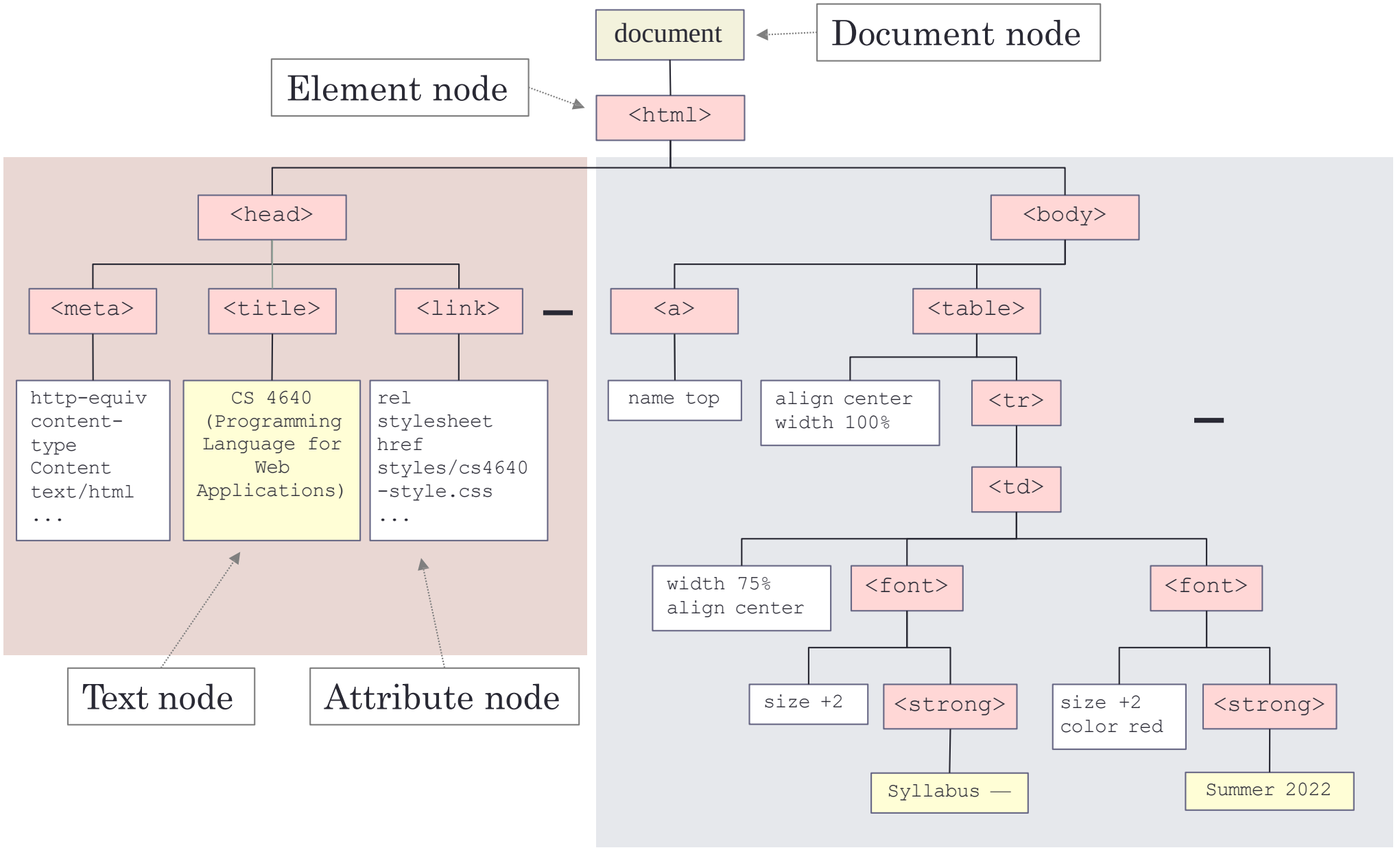
It creates a model of the page and stores it in memory

It shows the page on screen using a rendering engine

Using DOM Objects

- When a web page is loaded, the browser creates a Document Object Model of the page.
- With the object model, JavaScript gets all the power it needs to create dynamic HTML:
  - JavaScript can change all the HTML elements in the page
  - JavaScript can change all the HTML attributes in the page
  - JavaScript can change all the CSS styles in the page
  - JavaScript can remove existing HTML elements and attributes
  - JavaScript can add new HTML elements and attributes
  - JavaScript can react to all existing HTML events in the page
  - JavaScript can create new HTML events in the page

DOM: Four Types of Nodes



Using DOM Objects (Some properties methods)

Property	Description
document.title	Title of current document
document.lastModified	Date on which document was last modified
document.URL	String containing URL of current document
document.domain	Domain of current document

Method	Description
document.write( )	Write text to document
document.getElementById(id)	Return element whose id attribute matches the specified id
document.querySelectorAll(selector)	Return list of elements that match the specified CSS selector
document.createElement(element)	Create new element
document.createTextNode(text)	Create new text node

Using DOM Objects (Some properties methods)

Accessing DOM Elements	Description
document.getElementById(id)	Select an element by its id
document.getElementsByClassName (className)	Selects all elements with the given class.
document.querySelector(selector)	Selects the first matching element
document.querySelectorAll(selector)	Selects all matching elements

Modifying Content	Description
element.innerHTML	Gets or sets the HTML content inside an element
element.textContent	Gets or sets only the text inside an element (ignores HTML)
element.innerText	Similar to textContent, but respects styling

Using DOM Objects (Some properties methods)

Modifying Attributes	Description
element.getAttribute(attribute)	Gets the value of an attribute
element.setAttribute(attribute, value)	Sets the value of an attribute
element.removeAttribute(attribute)	Removes an attribute

Modifying Styles	Description
element.style.property = "value"	Changes an inline CSS property
element.classList.add(className)	Adds a class
element.classList.remove(className)	Removes a class
element.classList.toggle(className)	Toggles a class

Using DOM Objects (Some properties methods)

Event Handling	Description
element.addEventListener(event, function)	Adds an event listener to an element
element.removeEventListener(event, function)	Removes an event listener

Creating and Removing Elements	Description
document.createElement(tagName)	Creates a new HTML element
parent.appendChild(child)	Appends a child element to a parent
parent.insertBefore(newElement, referenceElement)	Inserts an element before another
parent.removeChild(child)	Removes a child element from the parent

Using DOM Objects (Some properties methods)

Traversing the DOM	Description
element.parentElement	Gets the parent element
element.children	Gets the child elements
element.nextElementSibling	Gets the next sibling element
element.previousElementSibling	Gets the previous sibling element

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JS HTML DOM Example</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        #container { max-width: 400px; margin: auto; }
        ul { list-style: none; padding: 0; }
        li { background: #f4f4f4; margin: 5px 0; padding: 10px; display: flex;
            justify-content: space-between; }
        button { cursor: pointer; }
    </style>
</head>
<body>
    <div id="container">
        <h2>To-Do List</h2>
        <input type="text" id="taskInput" placeholder="Enter a task">
        <button onclick="addTask()">Add Task</button>
        <ul id="taskList"></ul>
    </div>
```

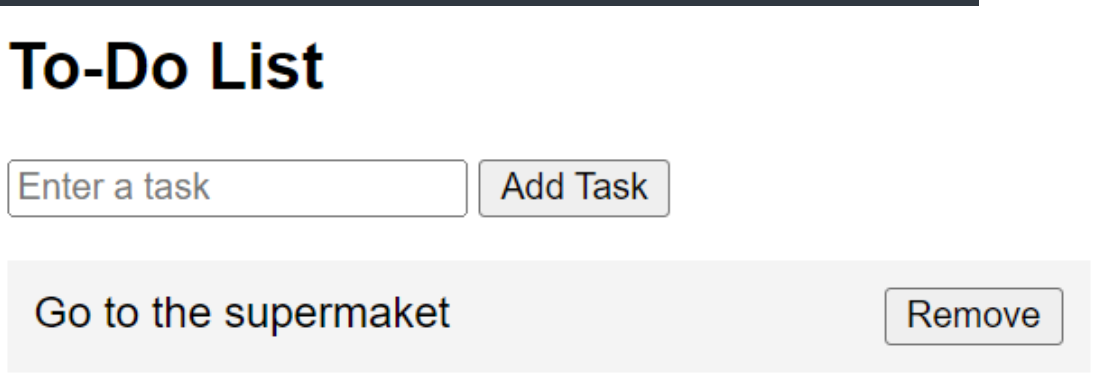
Example

```
    <script>
        function addTask() {
            let input = document.getElementById("taskInput");
            let taskText = input.value.trim();
            if (taskText === "") return;

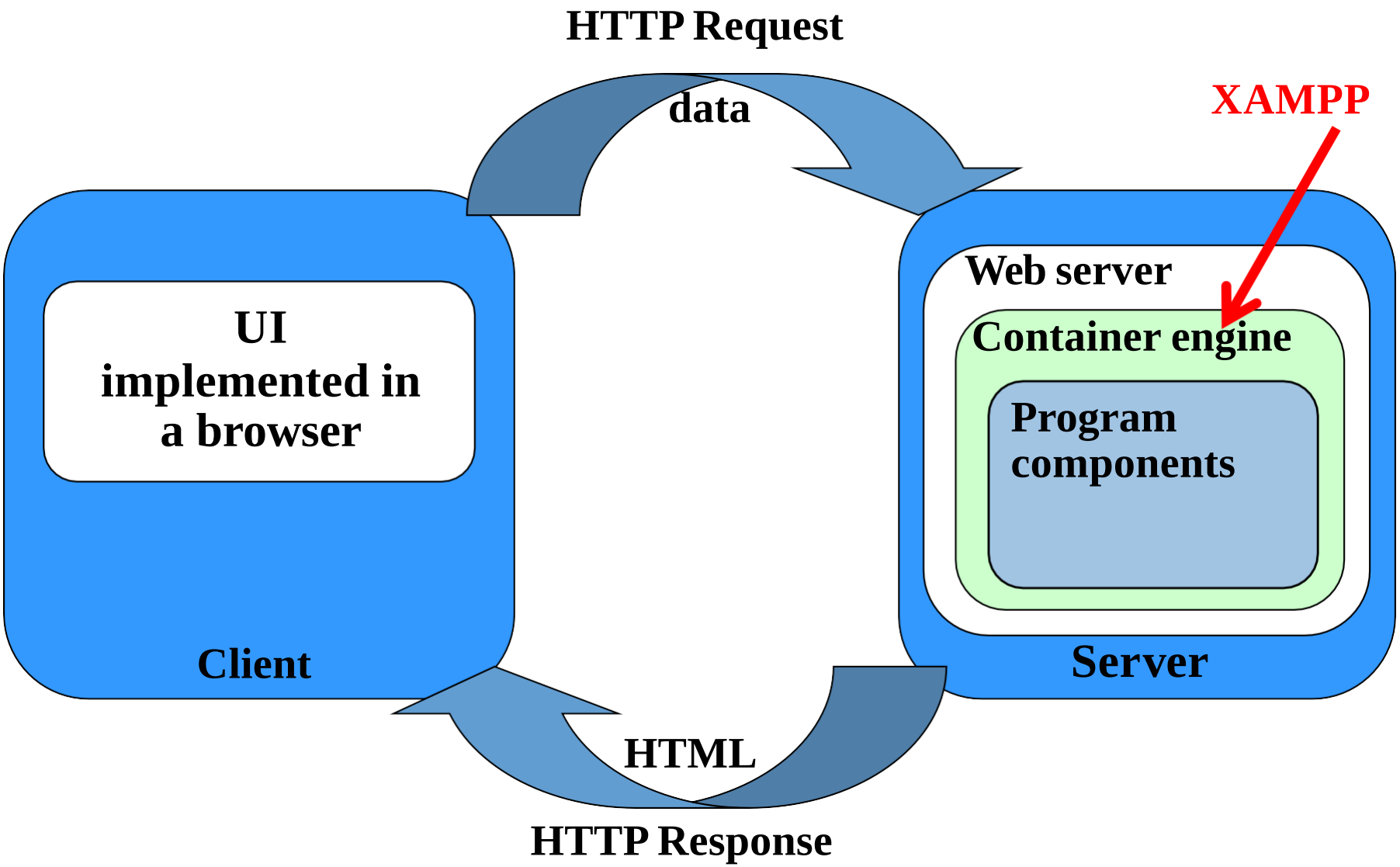
            let li = document.createElement("li");
            li.textContent = taskText;

            let removeBtn = document.createElement("button");
            removeBtn.textContent = "Remove";
            removeBtn.onclick = function() {
                li.remove();
            };

            li.appendChild(removeBtn);
            document.getElementById("taskList").appendChild(li);
            input.value = "";
        }
    </script>
</body>
</html>
```



Server Side Processing



JavaScript Strings

- A JavaScript string stores a series of characters
- A string can be any text inside double or single quotes
- JavaScript String Methods

Name	Description
at()	Returns an indexed character from a string
charAt()	Returns the character at a specified index (position)
charCodeAt()	Returns the Unicode of the character at a specified index
codePointAt()	Returns the Unicode value at an index (position) in a string
concat()	Returns two or more joined strings
constructor	Returns the string's constructor function

JavaScript Strings

Name	Description
endsWith()	Returns if a string ends with a specified value
fromCharCode()	Returns Unicode values as characters
includes()	Returns if a string contains a specified value
indexOf()	Returns the index (position) of the first occurrence of a value in a string
lastIndexOf()	Returns the index (position) of the last occurrence of a value in a string
length	Returns the length of a string
localeCompare()	Compares two strings in the current locale
match()	Searches a string for a value, or a regular expression, and returns the matches
padEnd()	Pads a string at the end
padStart()	Pads a string from the start
prototype	Allows you to add properties and methods to an object
repeat()	Returns a new string with a number of copies of a string
replace()	Searches a string for a pattern, and returns a string where the first match is replaced
replaceAll()	Searches a string for a pattern and returns a new string where all matches are replaced
search()	Searches a string for a value, or regular expression, and returns the index (position) of the match

JavaScript Strings

slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
startsWith()	Checks whether a string begins with specified characters
substr()	Extracts a number of characters from a string, from a start index (position)
substring()	Extracts characters from a string, between two specified indices (positions)
toLocaleLowerCase()	Returns a string converted to lowercase letters, using the host's locale
toLocaleUpperCase()	Returns a string converted to uppercase letters, using the host's locale
toLowerCase()	Returns a string converted to lowercase letters
toString()	Returns a string or a string object as a string
toUpperCase()	Returns a string converted to uppercase letters
trim()	Returns a string with removed whitespaces
trimEnd()	Returns a string with removed whitespaces from the end
trimStart()	Returns a string with removed whitespaces from the start
valueOf()	Returns the primitive value of a string or a string object

JS String Templates

- Back-Tics Syntax
- Template Strings use back-ticks (``) rather than the quotes ("") to define a string

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Template Strings</h1>
<p>Templates allow multiline strings:</p>

<p id="demo"></p>

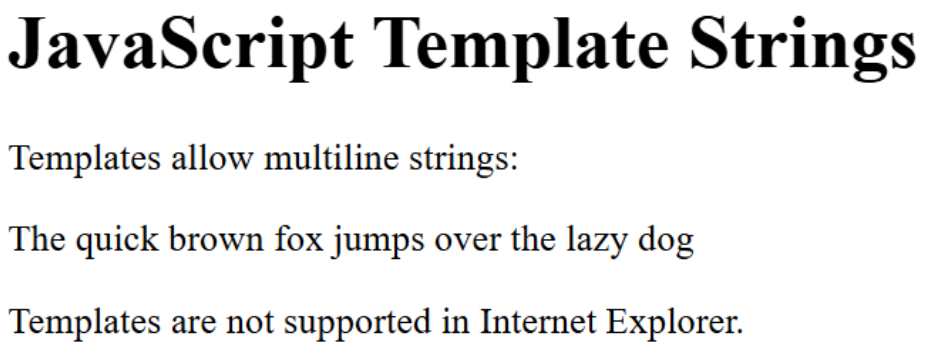
<p>Templates are not supported in Internet Explorer.</p>

<script>
let text =

`The quick
brown fox
jumps over
the lazy dog`;

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```



JS String Templates

- **Interpolation**: Template String provide an easy way to interpolate variables and expressions into strings.
- The method is called string interpolation.
- The syntax is: ${...}

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Template Strings</h1>
<p>Templates allow variables in strings:</p>
<p id="demo"></p>


<script>
let firstName = "John";
let lastName = "Doe";

let text = `Welcome ${firstName}, ${lastName}!`;

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JS String Templates

- Expression Substitution
- Template Strings allow expressions in string

```
let price = 10;
let VAT = 0.25;

let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;
```

- HTML Templates

```
let header = "Template Strings";
let tags = ["template strings", "javascript", "es6"];

let html = `<h2>${header}</h2><ul>`;
for (const x of tags) {
  html += `<li>${x}</li>`;
}

html += `</ul>`;
```

Template Strings

- template strings
- javascript
- es6

JS Numbers

- Adding Numbers and Strings
- ```
  let x = 10;
  let y = 20;
  let z = x + y; // Output z = 30
  ```
- ```
  let x = "10";
  let y = "20";
  let z = x + y; // Output z = 1020
  ```
- If you add a number and a string, the result will be a string concatenation
- ```
  let x = 10;
  let y = "20";
  let z = x + y; // Output = 1020
  ```
- A common mistake is
- ```
  let x = 10;
  let y = 20;
  let z = "The result is: " + x + y; // Output: The result is:
  1020
  ```

JS Numbers

- JavaScript will try to convert strings to numbers in all numeric operations
- ```
  let x = "100";
  let y = "10";
  let z = x / y; // Output z = 10
  ```

- ```
  let x = "100";
  let y = "10";
  let z = x * y; // Output z = 1000
  ```

- ```
  let x = "100";
  let y = "10";
  let z = x - y; // Output z = 90
  ```

- This will not work
- ```
  let x = "100";
  let y = "10";
  let z = x + y; // Output z = 10010
  ```

JS Numbers

- NaN - Not a Number; NaN is a JavaScript reserved word indicating that a number is not a legal number.
- ```
  let x = NaN;
  let y = "5";
  let z = x + y;
  ```
  ```

Advanced JavaScript

New Programming Languages

Advanced JavaScript

- Closures
- Prototypes & this Keyword
- Asynchronous JavaScript (Callbacks, Promises, Async/Await)
- Error Handling & Debugging

Closures

- A closure is a function that remembers the variables from its outer scope even after the outer function has finished execution.

```
<script>
    function outerFunction(outerVariable) {
        return function innerFunction(innerVariable) {
            const result = `Outer: ${outerVariable}, Inner: ${innerVariable
                }`;
            console.log(result);
            document.getElementById("output").textContent = result;
        };
    }

    function runClosure() {
        const newFunc = outerFunction("Hello");
        newFunc("World"); // Output: Outer: Hello, Inner: World
    }
</script>
```

Closures

- A closure is a function that remembers the variables from its outer scope even after the outer function has finished execution.

```
function Counter() {
    let count = 0; // Private variable
    function updateDisplay() {
        document.getElementById("countDisplay").textContent = count;
    }

    return {
        increment: function () {
            count++;
            updateDisplay();
            console.log(`Count: ${count}`);
        },
        decrement: function () {
            count--;
            updateDisplay();
            console.log(`Count: ${count}`);
        }
    };
}

const counter = Counter();
counter.increment();
counter.decrement();
console.log(counter.count); // Undefined (count is private)
```

Prototypes & this Keyword

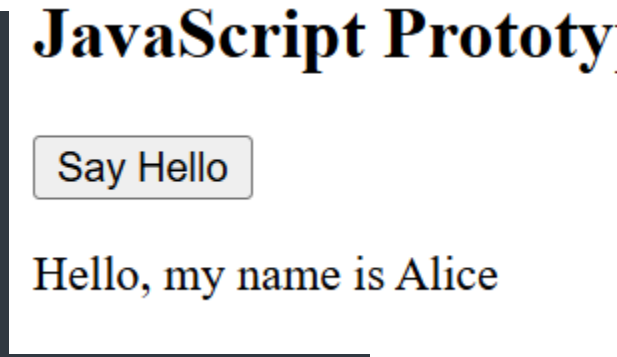
- JavaScript uses prototypes for inheritance. Every object in JS has a __proto__ property that points to its prototype.
- The method sayHello() is defined on the prototype, meaning all instances of Person share this method, saving memory.

```
// Propotypes and this keyword
function Person(name) {
            this.name = name;
}

Person.prototype.sayHello = function () {
    const message = `Hello, my name is ${this.name}`;
    console.log(message);
    document.getElementById("greeting").textContent = message;
};

const person1 = new Person("Alice");

// Attach event listener to button
document.getElementById("greetBtn").addEventListener("click", function() {
    person1.sayHello();
});
```



Asynchronous JavaScript

- JavaScript Asynchronous Handling: Callbacks, Promises, and Async/Await
- JavaScript is single-threaded, meaning it executes one operation at a time. Asynchronous programming allows JS to handle tasks like fetching data, file reading, or timers without blocking execution.
- **Callback**: A function passed as an argument to another function, executed once the task is complete
- **Promises**: A better way to handle async tasks, avoiding callback hell. A Promise is an object that represents a task's future result (either success or failure).

Asynchronous JavaScript

- JavaScript Asynchronous Handling: Callbacks, Promises, and Async/Await
- Async/Await (Best & Simplest): A cleaner way to handle Promises using async (declares function as asynchronous) and await (waits for the Promise to resolve).

Asynchronous JavaScript

- JavaScript is single-threaded, meaning it executes one task at a time. To handle asynchronous tasks (API calls, timers, etc.), we use **Callback()** as old method

```
function fetchData(callback) {
    setTimeout(() => {
        callback("Data received!");
    }, 2000);
}

fetchData((data) => {
    console.log(data); // Output after 2 seconds: Data received!
});
```

- Problem: nested callbacks make code messy

Asynchronous JavaScript

- **Promises (Better Alternative)**
- A Promise is an object representing an asynchronous operation.
  - resolve() for success
  - reject() for failure

```
//Promise
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const success = true;
            if (success) resolve("Data received!");
            else reject("Error fetching data.");
        }, 2000);
    });
}

fetchData()
    .then(data => console.log(data)) // Data received!
    .catch(error => console.log(error));
```

Asynchronous JavaScript

- **Promises (Better Alternative)**
- A Promise is an object representing an asynchronous operation.
  - resolve() for success
  - reject() for failure

```
//Promise
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const success = true;
            if (success) resolve("Data received!");
            else reject("Error fetching data.");
        }, 2000);
    });
}

fetchData()
    .then(data => console.log(data)) // Data received!
    .catch(error => console.log(error));
```

Asynchronous JavaScript

- Async/Await (Best Approach): Makes asynchronous code look synchronous.

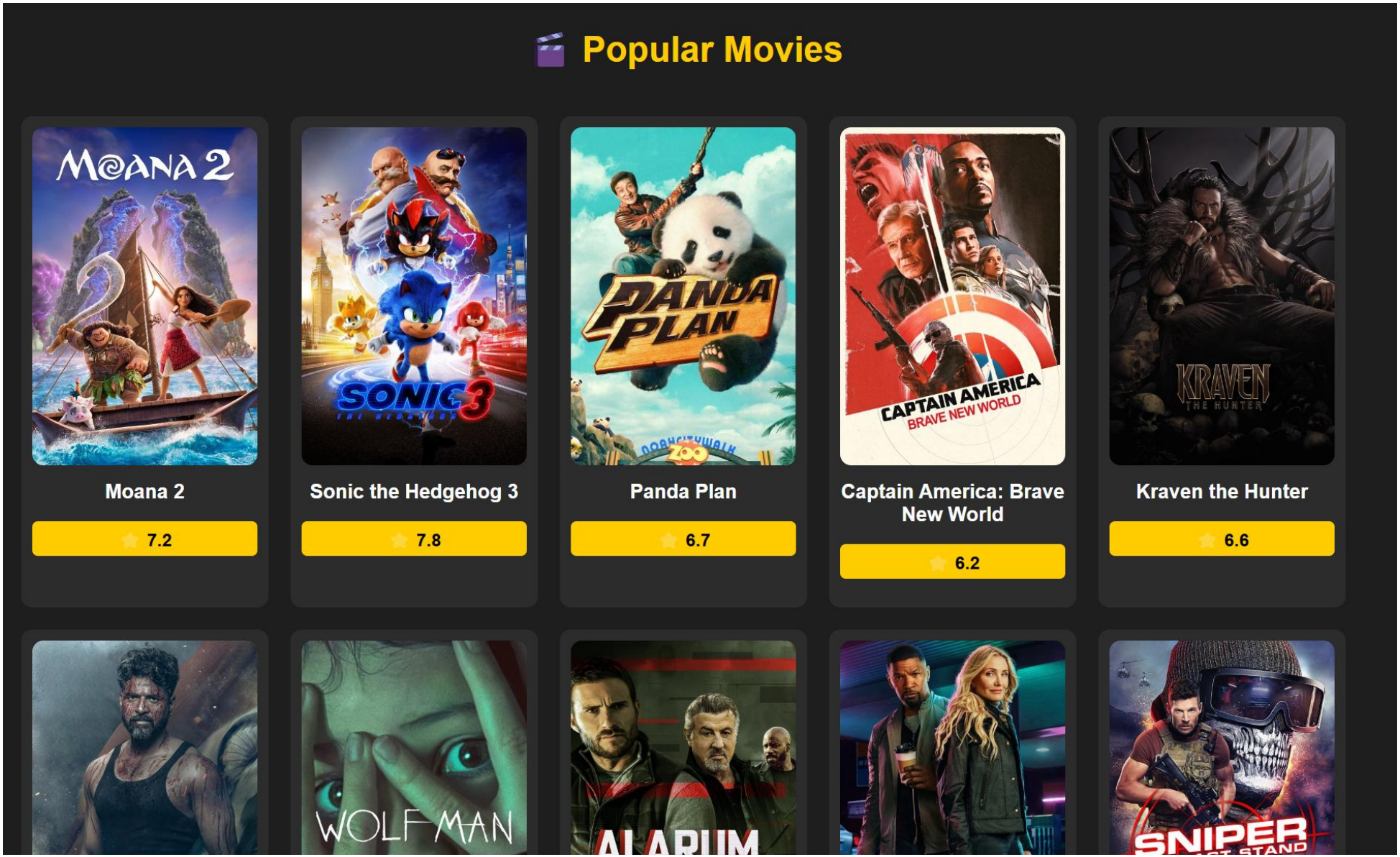
```
// Async/Await
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const success = true; // Simulate success/failure
            if (success) {
                resolve("Data received!"); // Simulated response
            } else {
                reject("Error fetching data.");
            }
        }, 2000);
    });
}
// Async function to get data
async function getData() {
    try {
        const response = await fetchData();
        console.log(response);
        document.getElementById("result").textContent = response;
    } catch (error) {
        console.log(error);
        document.getElementById("result").textContent = error;
    }
}
// Attach event listener to the button
document.getElementById("fetchBtn").addEventListener("click", getData);
```

Asynchronous JavaScript

- Callback, Promise Async/Await (Best Approach)

Method	Pros	Cons	Best use
Callback	Simple, widely use	Callback hell, harder error handling	Simple async tasks (not recommended for complex logic)
Promise	Avoids callback hell, readable	Still uses .then() & .catch()	API calls, fetching data
Async/Await	Cleanest syntax, easy debugging	Needs try/catch for errors	Modern async handling (Best for most cases)

Movie app


Popular Movies
Moana 2
7.2
Sonic the Hedgehog 3
7.8
Panda Plan
6.7
Captain America: Brave New World
6.2
Kraven the Hunter
6.6
WOLFMAN

Movie app

```
<script>
    const API_URL = 'https://api.themoviedb.org/3/discover/movie?sort_by=popularity.desc&api_key=3fd2be6f0c70a2a598f084ddfb75487c&page=1';
    const IMG_PATH = 'https://image.tmdb.org/t/p/w500';

    async function fetchMovies() {
        try {
            const response = await fetch(API_URL);
            const data = await response.json();
            displayMovies(data.results);
        } catch (error) {
            console.error("Error fetching movies:", error);
            document.getElementById("movieContainer").innerHTML = "<p>Failed to load movies.</p>";
        }
    }
    function displayMovies(movies) {
        const movieContainer = document.getElementById("movieContainer");
        movieContainer.innerHTML = movies.map(movie => `
            <div class="movie">
                <img src="${IMG_PATH + movie.poster_path}" alt="${movie.title}">
                <h2>${movie.title}</h2>
                <p class="rating">⭐ ${movie.vote_average.toFixed(1)}</p>
            </div>
        `).join('');
    }
    fetchMovies();
</script>
```

Exercise: Build a Book App

- Create a web app that fetches books from the Google Books API and displays them on the page.
- **Requirements:**
  - Fetch data from the Google Books API using async/await.
  - Display book title, cover image, and author.
  - Use a search bar to find books by title.
  - Display a list of books dynamically.
  - Show a loading state when fetching data.
  - API URL for searching books:

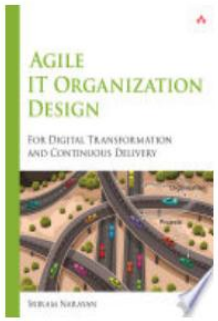
    https://www.googleapis.com/books/v1/volumes?q=SEARCH_

Exercise: Build a Book App

Mặc Kệ Nó, Làm Tới Đi! - Screw Tt, Let's Do It

Author: Richard Branson

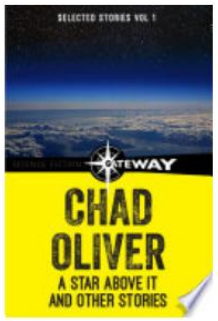
⭐ N/A



Agile IT Organization Design

Author: Sriram Narayan

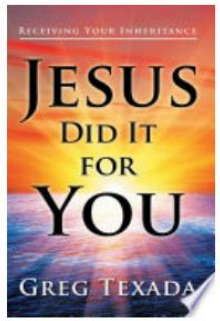
⭐ N/A



A Star Above It and Other Stories

Author: Chad Oliver

⭐ N/A

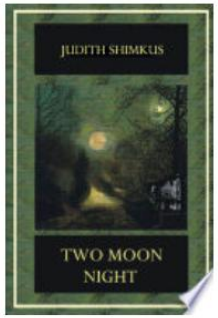


Jesus Did It for You

Author: Greg Texada

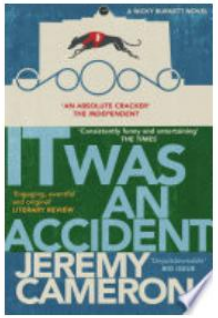
⭐ N/A

Your Mind And How To Use It



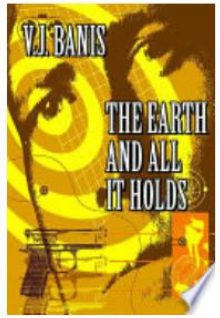
How Aliens Do It

Author: Judith Shimkus



It Was An Accident

Author: Jeremy Cameron



The Earth and All It Holds

JS AJAX

- AJAX (Asynchronous JavaScript and XML) is a technique for updating web pages without reloading them. It enables asynchronous communication between the client and the server.
- **How AJAX Works in Web Apps**
  - A user interacts with the webpage.
  - JavaScript sends an HTTP request to the server asynchronously.
  - The server processes the request and sends back a response.
  - JavaScript processes the response and updates the webpage dynamically.

JS AJAX

- **Methods to Implement AJAX**
  - Using XMLHttpRequest (Old, but still in use)
  - Using Fetch API (Modern and recommended)
  - Using Axios (A third-party library for AJAX calls)

AJAX with fetch() and async/await

- Use case: Making multiple AJAX requests sequentially.

```
async function loadData() {
    try {
        let response = await fetch("https://jsonplaceholder.typicode.com/posts/1");
        let data = await response.json();
        document.getElementById("output").innerHTML = `<h3>${data.title}</h3><p>${data.body}</p>`;
    } catch (error) {
        console.error("Error:", error);
    }
}
```

AJAX with fetch() and async/await

- Use case: Making multiple AJAX requests sequentially.

```
async function loadPost(postId) {
    const output = document.getElementById("output");
    output.innerHTML = "<p>Loading...</p>"; // Show loading message

    try {
        let response = await fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`);

        if (!response.ok) {
            throw new Error("Failed to fetch post");
        }

        let data = await response.json();

        output.innerHTML = `
            <h3>${data.title}</h3>
            <p>${data.body}</p>
        `;
    } catch (error) {
        output.innerHTML = "<p style='color: red;'>Error loac
        console.error("Error:", error);
    }
}
```

AJAX with fetch() for Sending Data (POST request)

- Sends a new post to the server via a POST request.
- Use case: Form submissions, creating new database records.

```
async function sendData() {
    let newData = {
        title: "New Post",
        body: "This is the content of the new post.",
        userId: 1
    };

    try {
        let response = await fetch("https://jsonplaceholder.typicode.com/posts", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify(newData)
        });

        let result = await response.json();
        console.log("Data Sent Successfully:", result);
    } catch (error) {
        console.error("Error:", error);
    }
}
```

AJAX with fetch() for Sending Data (POST request)

- Sends a new post to the server via a POST request.

```
document.getElementById("postForm").addEventListener("submit", async function(event) {
event.preventDefault();
const title = document.getElementById("title").value;
const body = document.getElementById("body").value;
const output = document.getElementById("output");
output.innerHTML = "<p>Sending data...</p>";
try {
    let response = await fetch("https://jsonplaceholder.typicode.com/posts", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ title, body, userId: 1 })
    });

    let data = await response.json();
    console.log("Server Response:", data); // Log resp

    output.innerHTML = `
        <h3>Data Sent Successfully!</h3>
        <p><strong>Title:</strong> ${data.title}</p>
        <p><strong>Body:</strong> ${data.body}</p>
        <p><strong>Post ID:</strong> ${data.id}</p>
    `;
} catch (error) {
    output.innerHTML = "<p style='color: red;'>Error sending data.</p>";
    console.error("Error:", error);
}
});
```