FRONTEND WEB DEVELOPMENT PROGRAM

Lecture 3: JavaScript Introduction

BeaconFire

OUTLINE

- About JS
- Data Types (Primitive vs Reference)
- Operators & Flow Control
- DOM Events and Manipulation
- jQuery

JS HISTORY

- 1995: Mocha was invented by Brendan Eich from Netscape in 10 days.
- 1996: Mocha—>LiveScript—>JavaScript. JavaScript to Java is like Carpet to Car; JavaScript has almost nothing to do with Java.
- 1997: ECMAScript1 was released (ECMA is the standard, JS is the language in practice)
- 2009: ES5 was released
- **2015: ES6 was released, biggest update ever.**
- 2022: ES13 was released

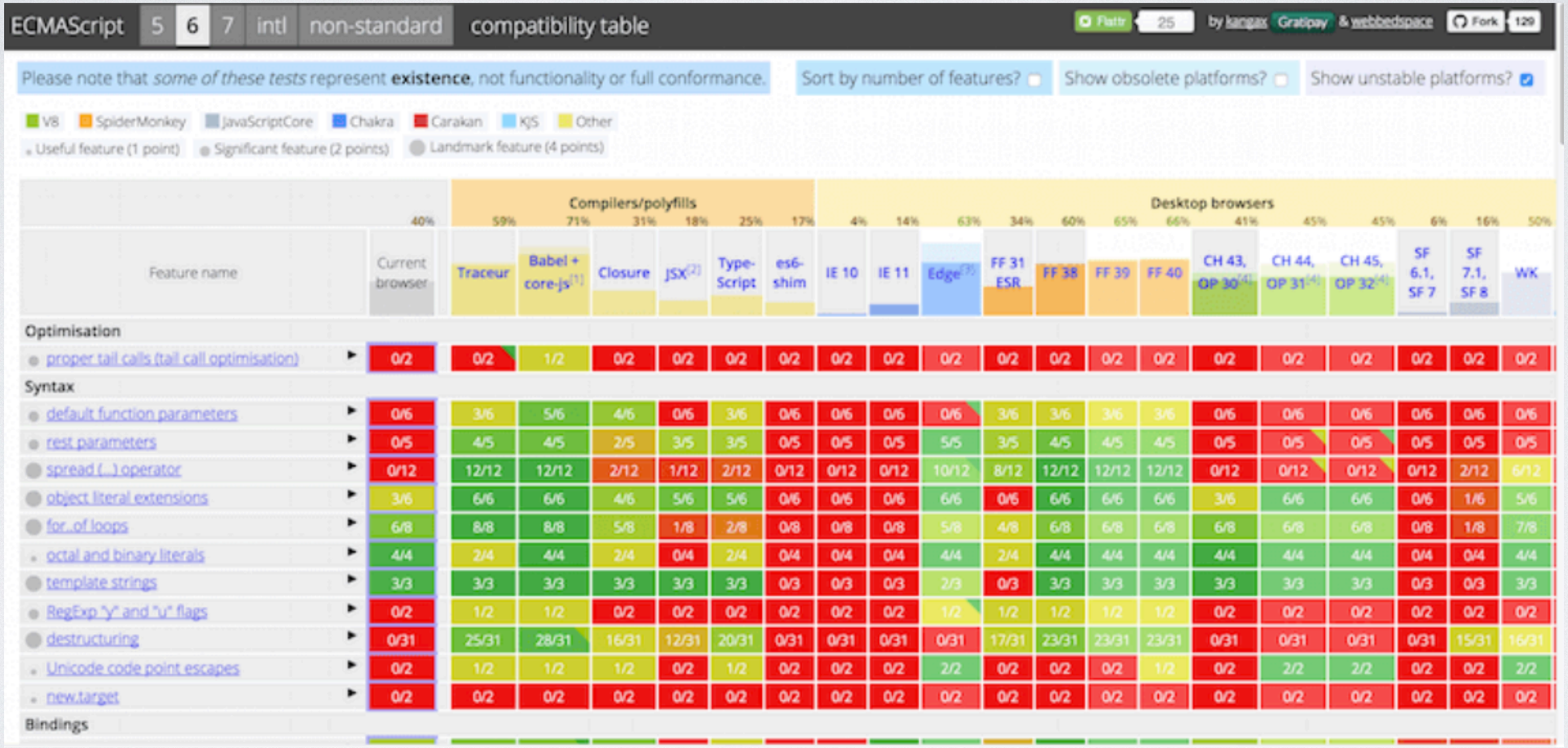
JS COMPATIBILITY

- Backwards-compatible (doesn't remove old features)
  - ES5 JS can run in modern browsers that support ES 6+
  - ES6+ JS can't run in older browsers

- Development: execute JS in a modern browser that supports ES6+

- Production: convert ES6+ back to ES5 for cross-browser compatibility using Babel (a JS transpiler)

ES6 COMPATIBILITY TABLE



BeaconFire

ECMASCRIPT VS JAVASCRIPT

- **ECMAScript**: a standardized specification for a scripting language (language syntax & semantics)

- **JavaScript**: a high-level programming language that conforms to ECMAScript
  - just-in-time compilation
  - dynamic typing
  - first-class functions
  - multi-paradigm programming (how you design solutions)
    - imperative: clearly defined sequence of instructions
    - declarative: a set of expectations without specifying how it should be done
    - functional: apply and compose functions
    - object-oriented: uses prototypes

JUST-IN-TIME (JIT) COMPILATION

- **Ahead-of-time (AOT) compilation**: before runtime, convert the program entirely into machine code, write it to a binary file, execute this file.
- **Interpretation**: during runtime, interpreter parses the source code and directly executes instructions line by line.
- **Just-In-Time (JIT) compilation**: during runtime, entire code is converted into machine code, then executed immediately, which allows for runtime optimizations

JS ENGINE

- **JavaScript engine**: a software program that optimizes and executes JavaScript code.
  - **V8**: Google Chrome, Node.js
  - **SpiderMonkey**: Mozilla Firefox
  - **JavaScriptCore**: Safari
  - **Chakra**: Internet Explorer, Microsoft Edge

HOW TO RUN JAVASCRIPT

- In browsers:
  - Import a <script> in HTML file at the end of </body> to download & execute the script after loading the DOM.
  - For chrome, open Chrome dev tools and click the 'console' tab to view outputs.

- Outside of the browser:
  - Node.js

REPL

- **Read-Eval-Print-Loop (REPL)**: a programming language environment that accepts user input (statements, expressions…) and outputs the result to the console after execution.
  - Can test basic syntax and operations

PRIMITIVE DATA TYPES

- **Dynamic typing**: types are not specified when declaring variables; they are assigned based on the value at runtime.

- 7 Primitive Data Types
  - string, number, boolean, undefined, null, symbol (ES6), bigint (ES11, 2020)

- **Immutability**: the structure or data cannot be changed.

- **typeof**: a unary operator that takes 1 operand and evaluates to a string that represents its data type.

COERCION & CONVERSION

- **Coercion**: when a value's data type is implicitly changed to another data type
  - Happens on operations between values of different data types.
  - ex: '5' + 9, true + 9

- **Conversion (typecasting)**: when a value's data type is explicitly changed to another data type
  - String()
  - Boolean(), !
  - Number(), +

NUMBER, STRING, BOOLEAN

- Numbers, strings, and booleans are either primitive data types or objects.
- You'll probably never need to create them as objects using "new ___".

```
var num = 1;
var num2 = new Number();

var str = "Hello";
var str2 = new String();

var bool = true;
var bool2 = new Boolean();

console.log(typeof num, typeof num2);
console.log(typeof str, typeof str2);
console.log(typeof bool, typeof bool2);
```

```
number object
string object
boolean object
```

STRINGS

- String: a sequence of characters
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

- Access characters:
  - string**[**index**]**
  - string**.charAt(index)**

- String properties/methods:
  - string**.length**: returns the length of the string as a number
  - string**.slice(start, end?)**: returns a substring based on the specified indices
  - string**.split(separator?)**: returns an array of substrings from splitting the string at all occurrences of separator
  - string**.replaceAll(str1, str2)**: replace all occurrences of str1 with str2
  - string**.trim()**: removes whitespace from both ends of a string
  - string**.includes()**: return true/false based on whether the string has the specified characters

TEMPLATE LITERALS

- **Template literals/template strings (`${}`)**: (ES6) string literals that are delimited with back-ticks and allow expressions for dynamic values

- "+" can be used to concatenate strings, but we can do the same with template literals.

```
var name = 'Hello World';
var combinedStr = 'My name is ' + name;
console.log(combinedStr);
console.log(`My name is ${name}`);
```

EXPRESSION VS STATEMENT

- **Expression**: code that can be evaluated and returns a value
  - 1+1, 'hello'+'world', 3<4, 'hi', 100

- **Statement**: an instruction that performs a specific action
  - var num = 0, console.log('hi')

- Template Literals can only accept expressions.

OPERATORS

- Mathematical: +, +=, ++ -, -=, -- *, *=, ** /, /= %, %=
- Comparison: >, <, >=, <=, ==, ===
- Boolean:
  ! : negates the boolean value
  && : true when left and right are true, otherwise false
  || : false when left and right are false, otherwise true
- **Falsy values**: 0, empty string (’’, ‘‘’’, ``), undefined, null, NaN
- **Short-circuiting**: when evaluating a boolean expression with (&& ||) from left to right, the right operand is evaluated only if the left operand is not enough to determine the final value.
  - When an argument is considered ‘enough’, it becomes the final value.

== VS ===

- **==**: a comparison operator that converts the operands to the same type before the comparison.
  - empty string => 0
  - string with no numeric value => NaN

- **===**: a strict-equality comparison operator that returns false if the operands are not the same type.

FLOW CONTROL

- if else, ternary
- switch
- while
- for loop

REFERENCE DATA TYPES

- There is 1 reference data type: Object
- **Object**: mutable keyed collection of properties; they store a collection of data instead of a single value.
  - ex: Functions, Array, Map, Set, …

```
var obj1 = {
  name: 'Object 1',
  id: 'NJ',
  greeting: function () {
    console.log('hi there');
  }
};
```

- **Data property**: associates keys to values
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

OBJECT

- Objects can be created by using the Object() constructor or the object literal syntax ({}).

- Iterating through an Object
  - for … in …
  - Object.keys(), Object.values(), Object.entries()

- **JavaScript Object Notation (JSON)**: a lightweight format for storing & sending data that is used by many programming languages
  - JSON.stringify(): convert JSON object to string
  - JSON.parse(): convert JSON string to object

```
var obj1 = {
  name: 'Object 1',
  id: 'NJ',
  greeting: function () {
    console.log('hi there');
  }
};

var obj2 = new Object({
  name: 'Object 2',
  id: 1,
  greeting: function () {
    console.log('hi there');
  },
});
```

```
for (var key in obj3) {
  console.log(`key: ${key},
  obj3[key]:`, obj3[key]);
}

// Convert object keys, values, or both into an array that can be iterated
// With this, we can use for ... of ..., .forEach();
console.log("Object.keys(obj3) =", Object.keys(obj3));
console.log("Object.values(obj3) =", Object.values(obj3));
console.log("Object.entries(obj3) =", Object.entries(obj3));
```

PRIMITIVE VS REFERENCE TYPES

- Primitive types
  - size in memory is fixed
  - stored on the stack
  - manipulate the **actual value** stored in a variable

- Reference
  - size in memory is dynamic
  - stored on the heap
  - manipulate the **reference** to that object in memory
    - ex: passing objects as arguments to a function and manipulating it will affect the original object

- **Shallow copy**: storing the reference of an object in another variable (both old and new point to same object in memory)

  Deep copy: each of the old object's properties are individually copied into a new object (different objects in memory)

FUNCTION

- **Function**: a special object that can be invoked to execute code that performs a specific task.
  - **return**: unary operator that ends execution of the function and gives a specified value or undefined back to the function caller
    - If no explicit "return", it returns undefined.

- 3 Ways to Declare a Function
  - Function declaration
  - Function expression
  - Arrow function (ES6)

```
// keyword function
function f1() { }
// function expression
var f2 = function () { };
// arrow function (ES6)
var f3 = () => { };
```

```
// One argument, no ()
var f5 = input => {
  return input * 10;
};
// Function body is an expression, no { return ... }
var f3 = input => input * 10;
// Multiple arguments requires ()
var f4 = (input1, input2) => input1 + input2;
```

- **Anonymous function**: function without a name

  First-class function: function that is treated like other variables
  - assigned to variables, passed as arguments to a function, returned from a function

  Higher-order function: function that takes in functions as arguments or returns functions

  Pure function: function that only depends on its inputs and does not change anything outside of its scope
  - If given the same inputs, it will always have the same result.

ARRAY

- **Array**: a special object that stores an ordered list of values that are accessed by a 0-based index and has a length property
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

- Create an array: **[**item1, item2, item3, …**]**
  - **new Array(**item1, item2, item3, …**)**
  Access elements: arr**[**index**]**

- arr**.push()**: add element to the end, return the number of elements in the new array
  arr**.unshift()**: add element to the beginning, return the number of elements in the new array
  arr**.pop()**: remove element from the end, return the element removed
  arr**.shift()**: remove element from the beginning, return the element removed
  arr**.slice(start?, end?)**: return a new subarray from the starting to the ending index of the original array
  arr**.splice(start, deleteCount?, item1?, …, itemN?)**: remove a specified number of elements from the original array starting at some index and add item1 … itemN, return the removed subarray
  arr**.join(separator?)**: return a string that is the concatenation of all elements, separated by the separator or comma if none specified
  arr**.includes()**: return true/false based on whether the arr has the specified element

ARRAY METHODS

- These are all higher-order functions that take a function as an input. This input function specifies how to operate on every element.

- arr.**map()**: return a new array of transformed elements using the original array
  arr.**filter()**: return a new array of elements that match the filter conditions
  arr.**sort()**: sorts the original array in place and returns a reference to it
    - localeCompare() for sorting strings
  arr.**find()**: returns the first element that matches the search conditions
  arr.**reduce()**: returns a single value that accumulated
  arr.**some()**: returns true/false if one element matches the conditions
  arr.**every()**: returns true/false if all elements match the conditions

ITERATING THROUGH ARRAYS

```
for (var i = 0; i < arr3.length; i++) {
  console.log(arr3[i]);
}

for (var num of arr3) {
  console.log(num);
}

// SYNTAX: arr.forEach((num, index, arr) => {})
arr3.forEach((num, index) => console.log(`${index}th element is ${num}.`));

// Not what you're expecting, so dont use "in" to iterate arrays, only for objects
for (var num in arr3) {
  console.log(num);
}
```

MAP

- **Map**: a special object that lets you store unique key-value pairs and remembers the insertion order of your keys

- Create: new Map()
  Access: map.get(key)

- map.**size**: returns the size
  map.**set(key, value)**: sets a new key-value pair, returns the updated map
  map.**has(key)**: returns true/false based on whether the key exists in the map
  map.**delete(key)**: deletes a key-value pair based on the key, returns true if it did or false if it didn't exist
  map.**forEach((value, key, …) => {})**: iterate through the map key/values

OBJECT VS MAP

	Object	Map
Default keys?	Yes (prototype)	No
Key type	string symbol	Anything
Ordered?	No	Based on insertion
Finding size	Must be iterated	.size
Iterable?	No	Yes
Serialization/ Parsing	JSON.stringify JSON.parse	No

OBJECT VS MAP ORDER

```
63    var obj = {};
64    obj['3'] = "temp";
65    obj['2'] = "temp";
66    obj['1'] = "temp";
67
68    console.log("obj =", obj);
69    console.log("Object.entries(obj) =", Object.entries(obj));
70
71    var map = new Map();
72    map.set("3", "temp");
73    map.set("2", "temp");
74    map.set("1", "temp");
75
76    console.log("map =", map);
77    console.log("map.entries() =", map.entries());
78
```

PROBLEMS OUTPUT GITLENS DEBUG CONSOLE TERMINAL

```
ethan@Ethans-MBP interview % node temp
obj = { '1': 'temp', '2': 'temp', '3': 'temp' }
Object.entries(obj) = [ [ '1', 'temp' ], [ '2', 'temp' ], [ '3', 'temp' ] ]
map = Map(3) { '3' => 'temp', '2' => 'temp', '1' => 'temp' }
map.entries() = [Map Entries] { [ '3', 'temp' ], [ '2', 'temp' ], [ '1', 'temp' ] }
ethan@Ethans-MBP interview %
```

SET

- Set: a special object that lets you store a unique, unordered list of values of any type.

- Create a set: **new Set()**
  Access: iterate to find it

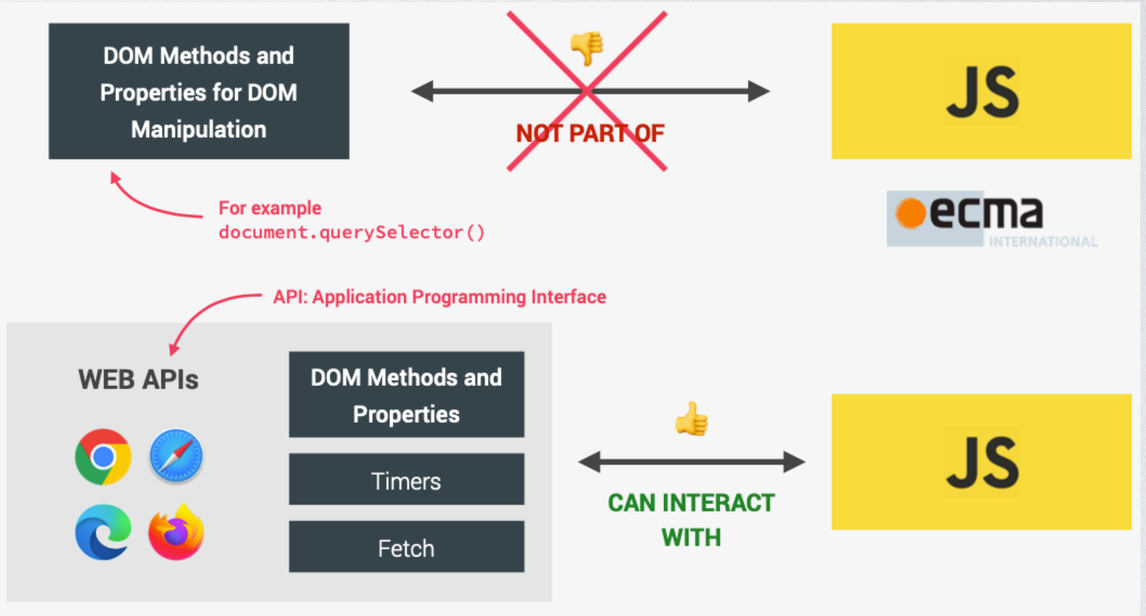
- set.**size**: returns the size
  set.**add(value)**: adds value to the end, returns the updated set
  set.**has(value)**: returns true/false based on whether the value exists in the set
  set.**delete(value)**: deletes a value, returns true if it did or false if it didn't exist
  set.**forEach(value => {})**

ARRAY VS SET

	Array	Set
Ordered?	Yes (indexed)	No
Accessing	Use index	Iterate
Unique values?	No	Yes
Lookup	.includes()	.has()
Adding	.push(), .unshift(), .splice()	.add()

WEB API

- **Web APIs**: a set of functions/objects built into a browser that you can access with JS
  - DOM
  - Fetch, XMLHttpRequest
  - Storage



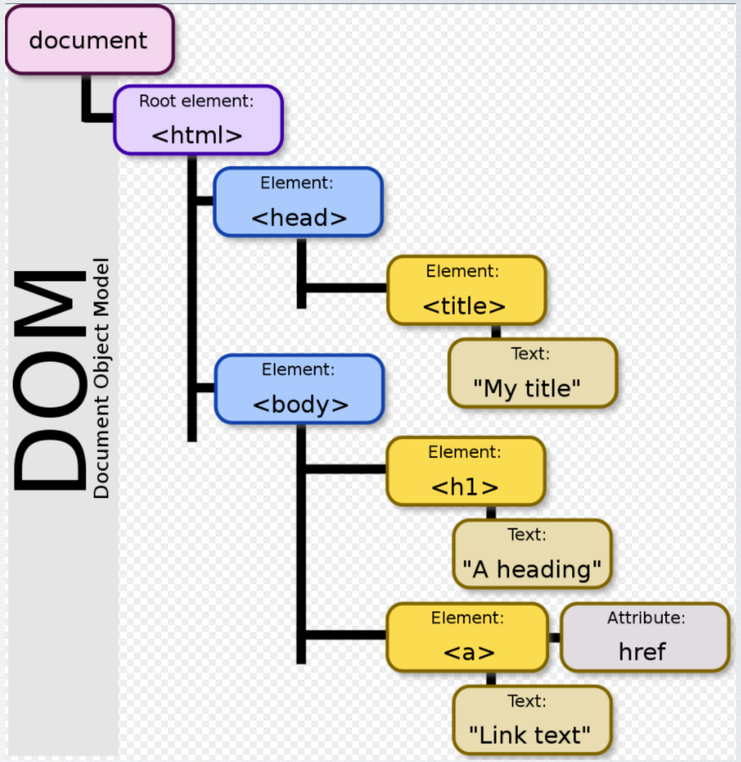
- https://developer.mozilla.org/en-US/docs/Web/API

DOM

- **Document Object Model (DOM)**: a programming interface for web documents that has a tree structure, where HTML elements are represented as nodes containing objects.

- Websites:
  - Static (HTML/CSS)
  - Dynamic (HTML/CSS/JS)

- Use JS to manipulate the DOM (modify its structure, style, and contents) based on user interactions.
  - ex: create HTML elements when the user clicks a button



DOM MANIPULATION

- Use JS to select an HTML element in the DOM.
- Register event handlers on that element (ex: user clicks).
- When the event is triggered, you have access to that element.
- Read the element properties.
  - parent, children, sibling elements, value, text contents, id, class, other attributes
- Do some logic with those properties.
  - ex: create/remove/append elements, change the contents, change the styling, hide an element

SELECTING ELEMENTS

- **Query selectors**: selects HTML elements using CSS selectors (#, .class, tag)
  - document.**querySelector()**: returns the **first** element within the document that matches the specified selector(s) or null.
  - document.**querySelectorAll()**: returns all elements that match the specified selector as a NodeList

- document.**getElementBy___**:
  - document.**getElementById()**: returns the DOM element that matches this ID
  - document.**getElementsByTagName()**, document.**getElementsByClassName()**
    - returns elements as HTMLCollection

NODE LIST VS HTML COLLECTION

- **NodeList**: static, does not recognize when new nodes are added to the DOM via JS
- **HTMLCollection**: live, will recognize when new nodes are added to the DOM via JS
- May cause bugs.

DOM EVENT

- **DOM Event**: actions or occurrences on the webpage that your code listens for and responds to using event handlers
  - element.**addEventListener(eventType, function)**
  - element.**removeEventListener(eventType, function)**

- Some event types
  - **click**: when the user clicks on an element
  - **focus**: when an element is in focus
  - **blur**: when an element is not in focus
  - **keypress**: when the user presses a key
  - **submit**: when a form is submitted

EVENT INTERFACE

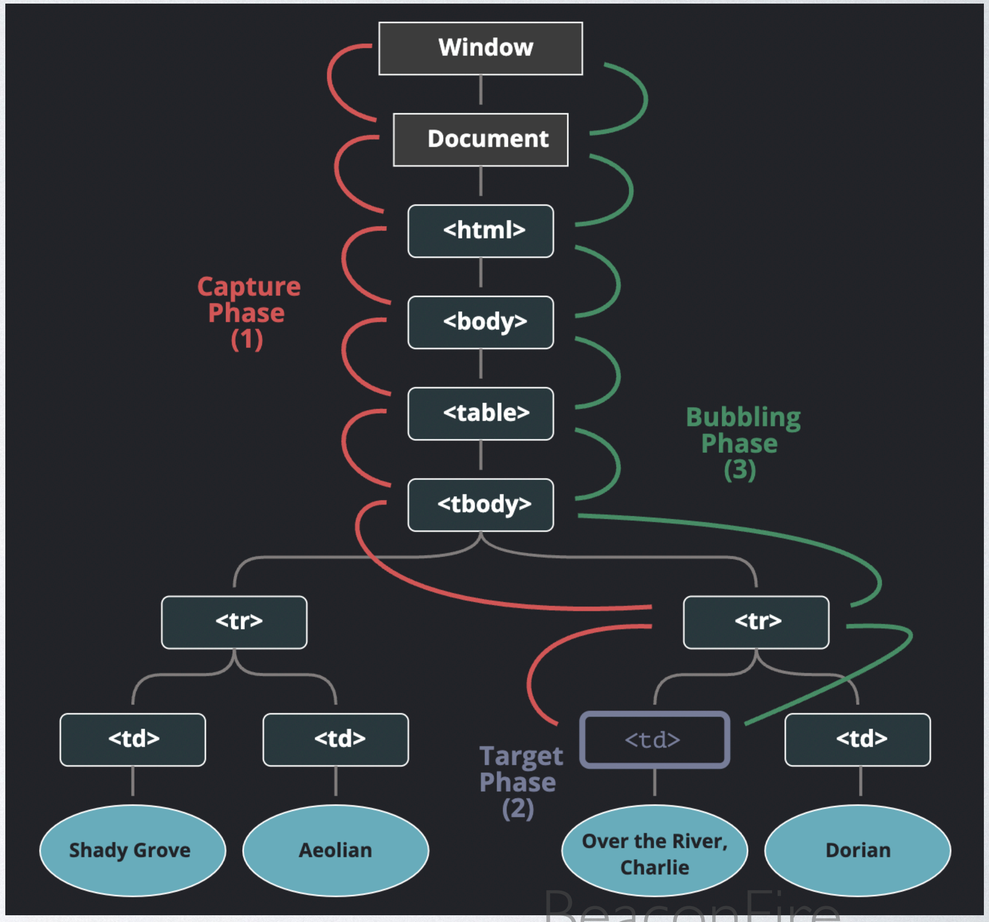
- **Event**: an object with properties that describe DOM events
  - Automatically passed to event handlers.

- event**.target**: a reference to the DOM element to which the event was originally dispatched.
  - Access this property to read or manipulate DOM elements

- **Event Propagation**: how events travel throughout the DOM and trigger event handlers.
  - Events pass through every DOM node until it reaches the final phase or is stopped by your code.
  - Phases: (1) capturing (2) target (3) bubbling

EVENT PROPAGATION

- **(1) Capturing**: event travels from the root down into the target element
- **(2) Target**: event has reached the target element
- **(3) Bubbling**: event "bubbles up" and travels outward
- Stopping propagation
  - event.**stopPropagation()**: stops current event's propagation
  - event.**stopImmediatePropagation()**: stops other handlers for the current event from being called



EVENT DELEGATION

- **Event Delegation**: an optimization technique for when you have multiple event handlers that do the same thing.
  - If many sibling elements have the same event handler logic, instead of binding the handler to each individual element, bind it once to their parent element.
  - The event handler is triggered when it bubbles up from the target element and out to the parent.
  - Check event.target in the handler.

PREVENT DEFAULT

- Some elements have default actions that occur when the event is dispatched.
  - The “submit” event on a <form> refreshes the page, resetting all of your DOM changes.
  - The “click” event on an <input type=“checkbox"> marks it with a check.

- event**.preventDefault()**: used in the event handler to prevent the default actions
  - Not all events are cancelable. Use the cancelable property to find out if an event is cancelable.

CREATE AND REMOVE ELEMENTS

- document**.createElement(tagName, options?)**: creates the HTML element specified by tagName
  document**.createTextNode()**: creates a Text node with the specified text.
  Node**.appendChild()**: appends a node as the last child to some node.

- Node**.removeChild(child)**: removes a specified child node from the selected element.
  - If there is any reference to the removed child, it will still exist in memory.
  Node.remove(): remove the targeted element

EXERCISE

- Given an array ['Apple', 'Banana', 'Orange'], display its elements in an unordered list on the webpage.

- Implement an "Add" feature, where users can type a fruit name and click a button to add it to the list.
  - The fruit name is from the text <input>.
  - The button says "Add" and has click functionality.

JQUERY

- **jQuery**: a fast, lightweight, feature-rich JavaScript library. It shortens the syntax needed to implement things like selecting/manipulating DOM elements, registering event handlers, creating animations, and making AJAX requests.
  - https://jquery.com/
  - Import jQuery from https://cdnjs.com/libraries/jquery

- The jQuery API is simple and works the same across all major browsers. It follows the format: **$(selector).action()**
  - $: defines or accesses jQuery
  - (selector): queries or finds HTML elements, uses CSS selector
  - .action(): performs a jQuery action on the element

$(DOCUMENT).READY()

- All jQuery methods should be inside a document.ready event. This ensures that the document is finished loading before any jQuery code is executed.
  - We want to avoid issues like selecting elements that were not yet created.

```
// deprecated
$(document).ready(function () {
  alert("hi");
});

// recommended
$(function () {
  alert("ready");
});
```

ANY QUESTIONS?

DOM ELEMENT VS JQUERY OBJECT

- **DOM element**: the objects that the browser uses to render elements on the web page.
  - document.getElementById('div')

- **jQuery Object**: wrapper objects around a set of DOM elements.
  - $('#div')

JQUERY SELECTORS

- Basic Selectors
    - all elements: $('*')
    - id: $('#id')
    - class $('.class')
    - tag: $('tag')
    - Union: $('div, p, li') // all <h1> and <p> elements
    - Intersection: $('div.redClass')

JQUERY SELECTOR

parent > child	$("div > p")	All <p> elements that are a direct child of a element
parent descendant	$("div p")	All <p> elements that are descendants of a element
element + next	$("div + p")	The <p> element that are next to each elements
element ~ siblings	$("div ~ p")	All <p> elements that appear after the element

:eq(*index*)	$("ul li:eq(3)")	The fourth element in a list (index starts at 0)
:gt(*no*)	$("ul li:gt(3)")	List elements with an index greater than 3
:lt(*no*)	$("ul li:lt(3)")	List elements with an index less than 3
:not(*selector*)	$("input:not(:empty)")	All input elements that are not empty

JQUERY HTML

- How do we get and set an element's contents (text, html, value)? It depends on the number of parameters.
  - *text()*: sets or returns the text content of selected elements
  - *html():* sets or returns the content of selected elements(including HTML makeup)
  - *val():* sets or returns the value of the form fields
- How do we get and set an element's CSS style?
  - *css(propertyName, value)*
  - *addClass(), hasClass(), removeClass()*

JQUERY HTML

- .attr(): could either get the value of an attribute, or it could set one or more attributes, depending on the number or type of arguments.
  - *$img.attr('alt')*
  - *$img.attr('src',"google.png")*
  - *$img.attr({'alt':'google','a':'aaa'})*
- prop(): used to check boolean attribute values (checked, selected, disabled…)

JQUERY EVENT

- *on():* used to register 1+ events
- *off():* if no params, removes all events registered on the element
- *trigger():* programmatically trigger an event (can be used for customized events)
- *this:* refers to the DOM element that a listener was attached to

JQUERY EVENT DELEGATION

- *Review:* Event delegation is when we register an event handler for a parent element because the children have similar onEvent functionality, and the handler is triggered by the bubbling phase for children elements.

- jQuery lets you specify a second selector parameter that tells the handler to listen for the specified event and check if the target element matches the second parameter.

```
$("ol").on("mouseenter", "li", function () {
  $(this).css("backgroundColor", "red");
});
```

JQUERY ANIMATION

- *.XXX([duration], [complete])*
  - duration: determines how long the animation will run, in ms
    - 400 (default, can be Number or String)
  - complete: a function to execute once the animation completes
- show: .show(), .hide(), toggle()
- slide: .slideUp(), .slideDown(), slideToggle()
- fade: .fadeIn(), .fadeOut(), fadeToggle()

JQUERY ANIMATION

- *.animate(properties, [duration], [easing], [complete])*
  - duration: determines how long the animation will run, in ms
    - 400 (default, can be Number or String)
  - easing: specifies the speed at which animations progress at different points within the animation
    - 'linear', 'swing' (default)
  - complete: a function to execute once the animation completes
  -

```
$animationDiv.animate(
  {
    left: "+1000px",
    opacity: 0.25,
  },
  5000,
  "linear"
);
```

JQUERY ANIMATION

- *.stop([,clearQueue] [, jumpToEnd])*
  - clearQueue: A boolean (default: false) indicating whether to remove the queued animation
  - jumpToEnd: A boolean (default: false) indicating whether to complete the current animation immediately

JQUERY.AJAX()

- *$.ajax(url, [settings]):* Perform an asynchronous HTTP (AJAX) request

- *$.get(url, [data], [success], [dataType]):* Load data from the server using a HTTP GET request.

```
$.get("https://jsonplaceholder.typicode.com/todos/1", function(data, status){
  console.log(data)
  console.log(status)
})
.fail(function(error){
  console.log(error)
})
```