Web Application Programming

JavaScript

Part 1

INTRODUCTION

What is JavaScript

- JavaScript (JS) is a lightweight interpreted or JIT-compiled programming language with first-class functions.

- While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as node.js and Apache CouchDB.

Why Study JavaScript?

- JavaScript is one of the 3 languages all web developers must learn:

1. HTML to define the content of web pages
2. CSS to specify the layout of web pages
3. JavaScript to program the behavior of web pages

Part 2

INCLUDING JAVASCRIPT IN HTML

The <script> Tag

- In HTML, JavaScript code must be inserted between <script> and </script> tags.

```
<script>
document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

JavaScript in <head> or <body>

- You can place any number of scripts in an HTML document.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.

JavaScript in <head>

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>
</head>

<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JavaScript in <body>

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

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try it</button>

<script>
function myFunction() {
   document.getElementById("demo").innerHTML = "Paragraph changed.";
}
</script>

</body>
</html>
```

Note

- Placing scripts at the bottom of the `<body>` element improves the display speed, because script compilation slows down the display.
- Old JavaScript examples may use a type attribute:

  `<script type="text/javascript">`.
- The type attribute is not required. JavaScript is the default scripting language in HTML.

External JavaScript

- Scripts can also be placed in external files:
- External scripts are practical when the same code is used in many different web pages.
- JavaScript files have the file extension .js.
- To use an external script, put the name of the script file in the src (source) attribute of a <script> tag.

External JavaScript

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

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

</body>
</html>
```

External JavaScript

- You can place an external script reference in `<head>` or `<body>` as you like.
- The script will behave as if it was located exactly where the `<script>` tag is located.
- External scripts cannot contain `<script>` tags.

External JavaScript: Advantages

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

External References

- External scripts can be referenced with a full URL or with a path relative to the current web page.
- This example uses a full URL to link to a script:

```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

- To add several script files to one page - use several script tags:

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

Part 3

JAVASCRIPT FUNCTIONS

JavaScript Function

- A JavaScript function is a block of code designed to perform a particular task.

- A JavaScript function is executed when "something" invokes it (calls it).

```
function myFunction(p1, p2) {
    return p1 * p2;
}
```

JavaScript Function: Syntax

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

- Function parameters are the names listed in the function definition.

- Function arguments are the real values received by the function when it is invoked.

JavaScript Function: Invocation

- The code inside the function will execute when "something" invokes (calls) the function:

  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

JavaScript Function: Return

- When JavaScript reaches a return statement, the function will stop executing.

- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

- Functions often compute a return value. The return value is "returned" back to the "caller":

JavaScript Function: Return

```
var x = myFunction(4, 3);        // Function is called, return value will end up
in x

function myFunction(a, b) {
    return a * b;                // Function returns the product of a and b
}
```

JavaScript Function: The () Operator

- Using the example below, `toCelsius` refers to the function object, and `toCelsius()` refers to the function result.
- Accessing a function without `()` will return the function definition instead of the function result:

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
document.getElementById("demo").innerHTML = toCelsius;
```

JavaScript Function: Self-Invoking

- Function expressions can be made "`self-invoking`".
- A self-invoking expression is invoked (started) automatically, without being called.
- Function expressions will execute automatically if the expression is followed by `()`.
- You cannot self-invoke a function declaration.

JavaScript Function: Self-Invoking

- You have to add parentheses around the function to indicate that it is a function expression:

```
(function () {
    var x = "Hello!!";      // I will invoke myself
})();
```

- The function above is actually an anonymous self-invoking function (function without name).

Part 4

HTML DOM

What is the HTML DOM?

The HTML DOM is a standard object model and programming interface for HTML. It defines:

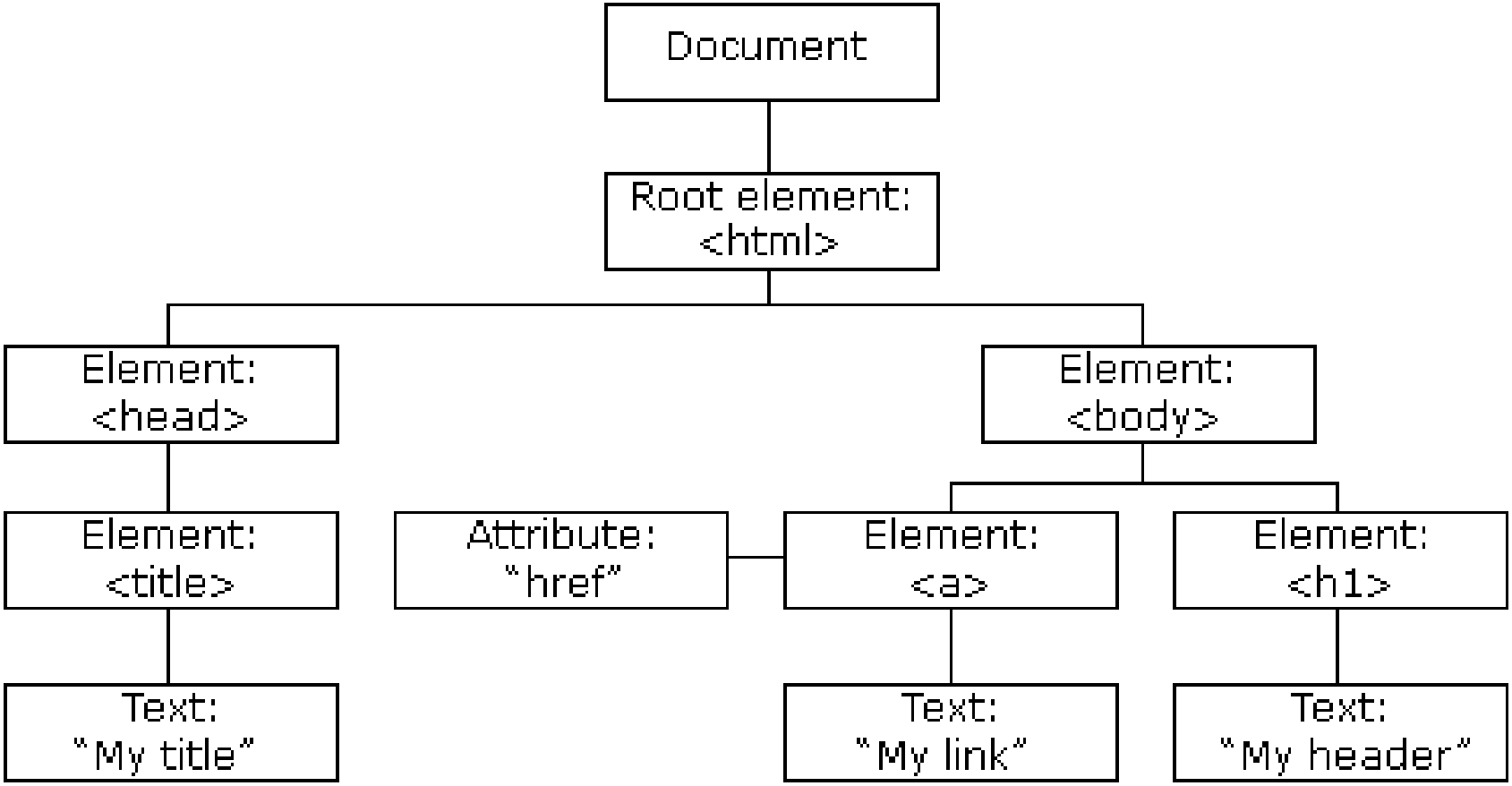
- The HTML elements as objects
- The properties of all HTML elements
- The methods to access all HTML elements
- The events for all HTML elements

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

The HTML Document Object Model

- When a web page is loaded, the browser creates a Document Object Model of the page.
- The HTML DOM model is constructed as a tree of Objects:
- With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM Tree of Objects



With the DOM, JavaScript can

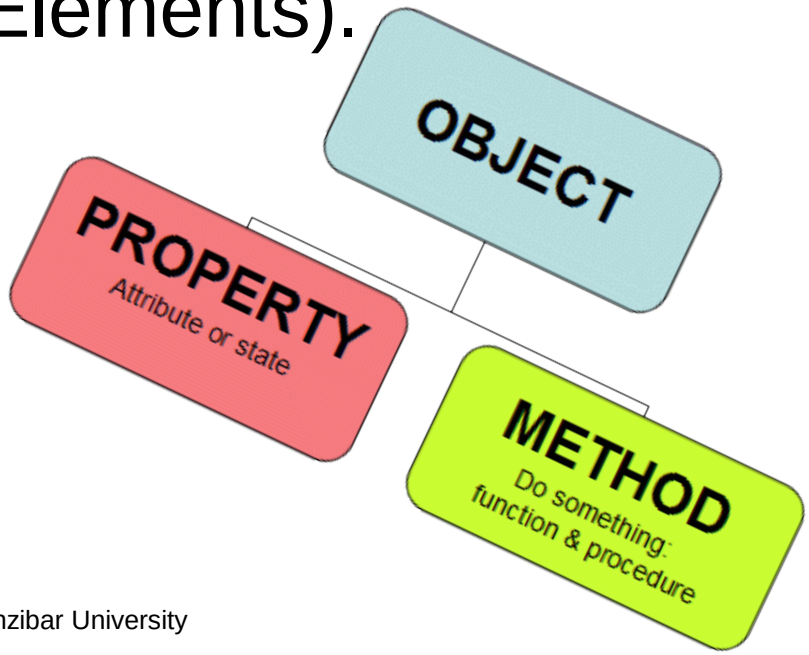
- Change all the HTML elements in the page
- Change all the HTML attributes in the page
- Change all the CSS styles in the page
- Remove existing HTML elements and attributes
- Add new HTML elements and attributes
- React to all existing HTML events in the page
- Create new HTML events in the page

What You Will Learn

- How to change the content of HTML elements
- How to change the style (CSS) of HTML elements
- How to react to HTML DOM events
- How to add and delete HTML elements

HTML DOM Methods

- HTML DOM properties are values (of HTML Elements) that you can set or change.
- HTML DOM methods are actions you can perform (on HTML Elements).



HTML DOM Document

- The document object is the owner of all other objects in your web page.
- The document object represents your web page.
- If you want to access any element in an HTML page, you always start with accessing the document object.

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

```
var x = document.querySelectorAll("p.intro");
```

Changing HTML Elements

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

HTML DOM Events

- HTML DOM allows JavaScript to react to HTML events.

- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

Examples of HTML events:

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

Assigning Events to HTML Elements

- HTML Event Attributes
- Assign Events Using the HTML DOM

Assign Events Using HTML Event Attributes

- To assign events to HTML elements you can use event attributes.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute:

```
onclick=JavaScript
```

HTML events: Example

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

<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>

</body>
</html>
```

HTML events: Example

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

<h1 onclick="changeText(this)">Click on this text!</h1>

<script>
function changeText(id) {
    id.innerHTML = "Ooops!";
}
</script>

</body>
</html>
```

Assign Events Using the HTML DOM

- The HTML DOM allows you to assign events to HTML elements using JavaScript:

```
<script>
document.getElementById("myBtn").onclick = displayDate;
</script>
```

Mouse Events

Attribute	Value	Description
onclick	*script*	Fires on a mouse click on the element
ondblclick	*script*	Fires on a mouse double-click on the element
onmousedown	*script*	Fires when a mouse button is pressed down on an element
onmousemove	*script*	Fires when the mouse pointer is moving while it is over an element
onmouseout	*script*	Fires when the mouse pointer moves out of an element
onmouseover	*script*	Fires when the mouse pointer moves over an element
onmouseup	*script*	Fires when a mouse button is released over an element
onmousewheel	*script*	Deprecated. Use the onwheel attribute instead
onwheel	*script*	Fires when the mouse wheel rolls up or down over an element

HTML DOM Events

- For full list of HTML DOM Events follow this link.

HTML DOM EventListener

- Events can be listed in DOM using the JavaScript `addEventListner()` method.

- The `addEventListener()` method attaches an event handler to the specified element.

- It attaches an event handler to an element without overwriting existing event handlers.

HTML DOM EventListener: You can…

- Add many event handlers to one element.
- Add many event handlers of the same type to one element, i.e two "click" events.
- Add event listeners to any DOM object not only HTML elements. i.e the window object.

- The `addEventListener()` method makes it easier to control how the event reacts to bubbling.

HTML DOM EventListener

- When using the `addEventListener()` method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

- You can easily remove an event listener by using the `removeEventListener()` method.

HTML DOM EventListener: Syntax

```
element.addEventListener(event, function, useCapture);
```

- The first parameter is the type of the event
- The second parameter is the function we want to call when the event occurs.
- The third parameter is a `boolean` value specifying whether to use event bubbling or event capturing. This parameter is optional.

HTML DOM EventListener: Adding Events

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

- Note that you don't use the "on" prefix for the event; use "`click`" instead of "`onclick`".

```
element.addEventListener("click", myFunction);

function myFunction() {
    alert ("Hello World!");
}
```

HTML DOM EventListener: Adding Many Events

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

Event Bubbling or Event Capturing?

- There are two ways of event propagation in the HTML DOM, bubbling and capturing.

- Event propagation is a way of defining the element order when an event occurs.
- If you have a `<p>` element inside a `<div>` element, and the user clicks on the `<p>` element, which element's "`click`" event should be handled first?

Event Bubbling

- In bubbling the inner most element's event is handled first and then the outer:
- the `<p>` element's click event is handled first, then the `<div>` element's click event.

Event Capturing

- In capturing the outer most element's event is handled first and then the inner:
- the element's click event will be handled first, then the <p> element's click event.

- Examples
  - Try It Your Self Page

The End

- References
  - https://www.w3schools.com/js/default.asp
- Exercises
  - https://www.w3schools.com/js/exercise.asp?filename=exercise_dom_elements2