JavaScript and HTML Documents

Objectives

- How HTML documents are represented
- How we can reference HTML elements from JavaScript, create them, and modify them
- How to handle events that occur in a web page
- How to use the navigator object and the canvas element

Examples

Yahtzee Dice [link]

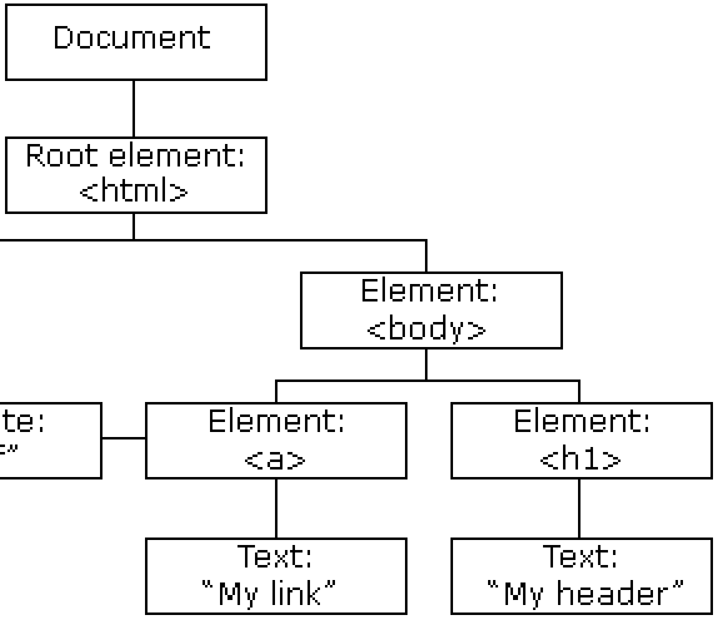
Yahtzee Dice with External JS [link]

Document Object Model

The Document Object Model

The ***Document Object Model (DOM)*** is an abstract model that defines the interface between HTML documents and application programs—an API

Documents in the DOM have a treelike structure



The Document Object Model

HTML elements → JavaScript objects

HTML attributes → JavaScript properties

Example:

```
<input type = "text" name = "address">
```

would be represented as an object with two properties, `type` and `name`, with the values "`text`" and "`address`"

Note: Chrome offers DevTools that can show the tree of a document and other useful information [link]

Examples

Let’s see an example:

table2.html [link]

JavaScript Execution Environment

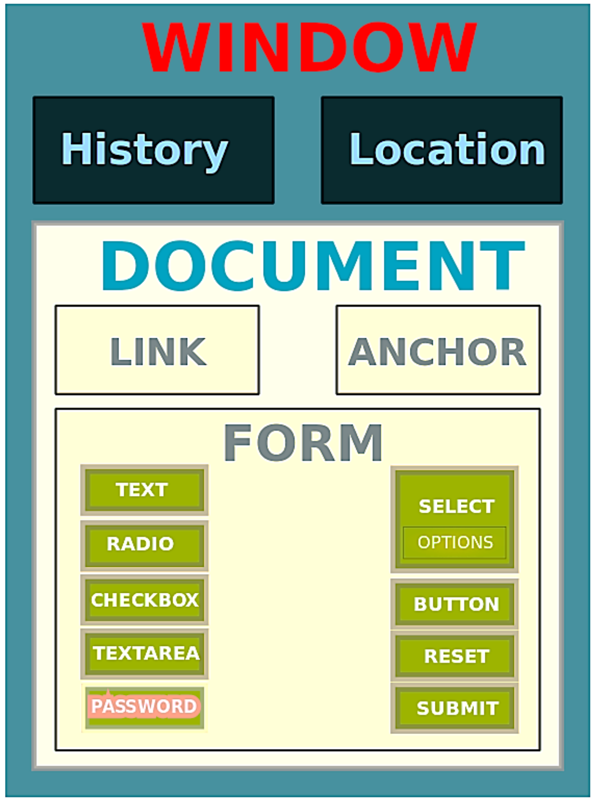
`Window` object represents the window in which the browser displays documents

- provides the largest enclosing referencing environment for scripts
- all global variables are properties of `Window`
- Has some implicitly defined properties:

`document` - a reference to the `Document` object that the window displays

`history` - reference to browser history

`location` - reference to current URL



JavaScript Execution Environment

Every **`Document`** object has:

- `anchors`
- `links`
- `images`
- `forms` - an array of references to the forms of the document
  - Each `Form` object has an `elements` array, which has references to the form's elements

Element Access in JavaScript

Example (one form and one widget):

```
<form name = "myForm" action = "">
  <input type = "button" id = "pushMe"
name = "pushMe" />
</form>
```

There are several ways to do it:

1. **Using the DOM address:**
   `document.forms[0].element[0]`

Problem: if document changes, indexes will be wrong

Element Access in JavaScript

Example (one form and one widget):

```
<form name = "myForm" action = "">
  <input type = "button" id = "pushMe"
name = "pushMe" />
</form>
```

2. **Using Element names** – requires the element and all of its ancestors (except `body`) to have `name` attributes:
   `document.myForm.pushMe`

Element Access in JavaScript

Example (one form and one widget):

```
<form name = "myForm" action = "">
  <input type = "button" id = "pushMe" name = "pushMe" />
</form>
```

3. **Using the** `getElementById` **method:**

   ```
   document.getElementById("pushMe")
   ```

Element Access

Checkboxes and radio button can be accessed through an implicit array, which has their name, e.g.:

```
 <form id = "topGroup">
   <input type = "checkbox"  name = "toppings"
          value = "olives" />
   ...
   <input type = "checkbox"  name = "toppings"
          value = "tomatoes" />
</form>
...
var numChecked = 0;
var dom = document.getElementById("topGroup");
for (index = 0; index < dom.toppings.length;
index++)
  if (dom.toppings[index].checked]
    numChecked++;
```

Event Handling

Events and Event Handling

An ***event*** is a notification that something specific has occurred, either with the browser or an action of the browser user

An ***event handler*** is a script that is implicitly executed in response to the appearance of an event

The process of connecting an event handler to an event is called ***registration***

Webpage Events

Events that occur in a web page have an ***event name*** and a corresponding ***tag attribute***

For example:

`click` is the name of an event that occurs when the user presses his mouse button onto a HTML element

`onclick` is the name of the tag attribute associated with that event

Webpage Events

Event	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

Event handler registration

There are several ways of **assigning handlers for an event:**

One way is to assign an event handler script to an event tag attribute, e.g.:

```
   onclick = "alert('Mouse click!');"
```

Or assign a function to an event tag attribute:

```
   onclick = "myHandler();"
```

Alternatively, you can assign a function name using JavaScript:

```
document.getElementById("mybutton").onclick
= myHandler;
```

Note the syntax: no quotes or parameter list

Notes on writing event handlers

Don't use `document.write` in an event handler, because the output may go on top of the display

The same attribute can appear in several different tags

Example: The `onclick` attribute can be in `<a>` and `<input>`

Event Handlers for Form Elements

The `load` and `unload` Events

Many times, you may want to do some **initialization** when a page loads or **cleanup** after the page unloads

This can be done by registering load and unload event handlers

The **`load` event** is triggered when the loading of a document is completed

The **`unload` event** is typically used to do some cleanup before a document is unloaded

It is common to set the handler by setting the `onload` tag attribute of the **body element**

Examples

load.html [link]

Handling Events from Button Elements

To handle events from (plain) buttons:

just use the `onclick` property

e.g. `<input type=button value="Press me" onclick = "dostuff()">`

some function defined in the <head>

Handling Events from Textbox and Password Elements

Useful textbox and password element events:

`blur`, **`focus`**, **`change`**, and **`select`**

The **`focus`** and **`blur`** events occur when the element acquires or loses focus, respectively.

One use of `focus`: prevent illicit changes to a text box, e.g.:

```
onfocus = "this.blur();"
```

Examples

nochange.html [link]

Handling Events from Textbox and Password Elements

onchange event: detect when user enters text into the textbox

- Can check for proper formatting
- `value` is the string property containing the text in the textbox

The handler can check the format using a regular expression, e.g.:

```
var myPhone = document.getElementById(“phone”);
var pos = myPhone.value.search(/^\d{3}-\d{3}-\d{4}$/);
if (pos != 0) {
  alert(“Wrong format! Correct form is: ddd-ddd-dddd”);
  return false;
} else
  return true;
}
```

Handling Events from Textbox and Password Elements

Checking form input is a good use of JavaScript, because it **finds errors in form input before it is sent to the server for processing**

This saves both Server time and Internet time

Things that must be done:

- **Detect** the error and produce an `alert` message
- **Inform** the user of the error and present the correct format
- NOTE: To keep the form active after the event handler is finished, the handler must return `false`

Examples

pswd_chk.html [link]

validator.html [link]

Checking Input Format in HTML5

HTML5 made validation easier.

-it introduced ***self-validating input types***, e.g.:

```
<input type="email" placeholder=name@domain.com />
```

- Format is checked whenever the user presses the Submit button
- This eliminates the need for JavaScript input validation in most cases
- NOTE: **Not all browsers support the feature**

Handling Events from Radio Buttons

For **radio buttons**, the easy way is to

register the handler in the markup

(use a parameter to differentiate which button was set)

e.g., if `planeChoice` is the name of the handler and the value of a button is 172, then

```
      onclick = "planeChoice(172)"
```

Handling Events from Radio Buttons

If the handler is registered in the JavaScript:

iterate through the button array and determine the checked value.

This is a multistep process:

1. Assign the address of the handler function to the event property of the JavaScript object associated with the HTML element, e.g.:

If the name of the buttons is `planeButton`, then we write

```
var dom = document.getElementById("myForm")
dom.planeButton[0].onclick = planeChoice;
dom.planeButton[1].onclick = planeChoice;
```

Note:

This registration **must follow both the handler function and the HTML form.**

If this is done for a radio button group, **each element of the array must be assigned.**

Handling Events from Radio Buttons

2. We can then implement the `planeChoice` function to determine whether which button is clicked by examining the checked property of each radio button object, e.g.:

```
var dom = document.getElementById("myForm");
for (var index = 0;
  index < dom.planeButton.length; index++) {
  if (dom.planeButton[index].checked) {
    plane = dom.planeButton[index].value;
    break;
  }
}
```

Handling Events from Radio Buttons

The disadvantage of specifying handlers by assigning them to event properties is that there is no way to use parameters

So why do this?

1. It is good to keep HTML and JavaScript separate
2. The handler could be changed during use

Examples

radio_click.html [link]

radio_click2.html [link]

Start Session 17

Course Number: CPSC-24700

Instructor: Eric Pogue

The navigator object

Since different browsers support different features, we need to be able to detect the browser used by the user

The browser used can be accessed through the **`navigator` object**

2 useful properties:

- The `appName` property has the browser's name
- The `appVersion` property has the version #

Note: the `addVersion` may not tell you exactly what you need

- Microsoft has chosen to set the `appVersion` of IE9 to 5 (?)
- Firefox has chosen to set the `appVersion` of Firefox to 5.0 (?) and the name to Netscape (?)

Examples

`navigate.html`

The Canvas Element

The canvas Element

The **`canvas` element** is a new element introduced by HTML5 to support graphics and animations.

It creates a rectangle into which bit-mapped graphics can be drawn using JavaScript.

It's optional attributes are `height`, `width`, and `id`

The `id` attribute is necessary if something will be drawn

Example:

```
<canvas id = "myCanvas" height = "200" width = "400">
    Your browser does not support the canvas element
</canvas>
```

Examples

```
circles.html
parallel.html
rects.html
```

DOM Traversal

DOM Tree Traversal and Modification

The Document Object Model is a hierarchical model, i.e., it **has a tree structure**

You can traverse this tree by accessing different object properties:

- `parentNode`
- `previousSibling`
- `nextSibling`
- `firstChild`
- `childNodes`
- `lastChild`

DOM Tree Traversal and Modification

For example, if there is an unordered list with the id `myList`, the number of list items in the list can be displayed with:

```
var dom = document.getElementById("myList");
var listItems = dom.childNodes.length;
document.write("Number of list items is: " +
                listItems);
```

The tree can also be modified using different methods:

`insertBefore`, `replaceChild`, `removeChild`, `appendChild`

DOM 2 Event Model

DOM 2 Event Model

Document Object Model version 2 (DOM2) introduced a new way of handling events

In DOM2, events ***propagate*** through the document tree from the root of the tree to the target element and back

At each point, events can be **captured** and **handled** or **canceled**.

DOM 2 Event Model

Browser automatically pass an **`Event`** **object** to each event handler

The Event object provides properties associated with the event that occurred, e.g.:

- Which mouse button was clicked?
- What are the screen coordinates of the cursor?

DOM 2 - Event Propagation

The node of the document tree where the event is created is called the ***target node***

The ***capturing phase*** (1st phase)

- Events begin at the root and move toward the target node
- Registered and enabled event handlers at nodes along the way are run

The ***target node phase*** (2nd phase)

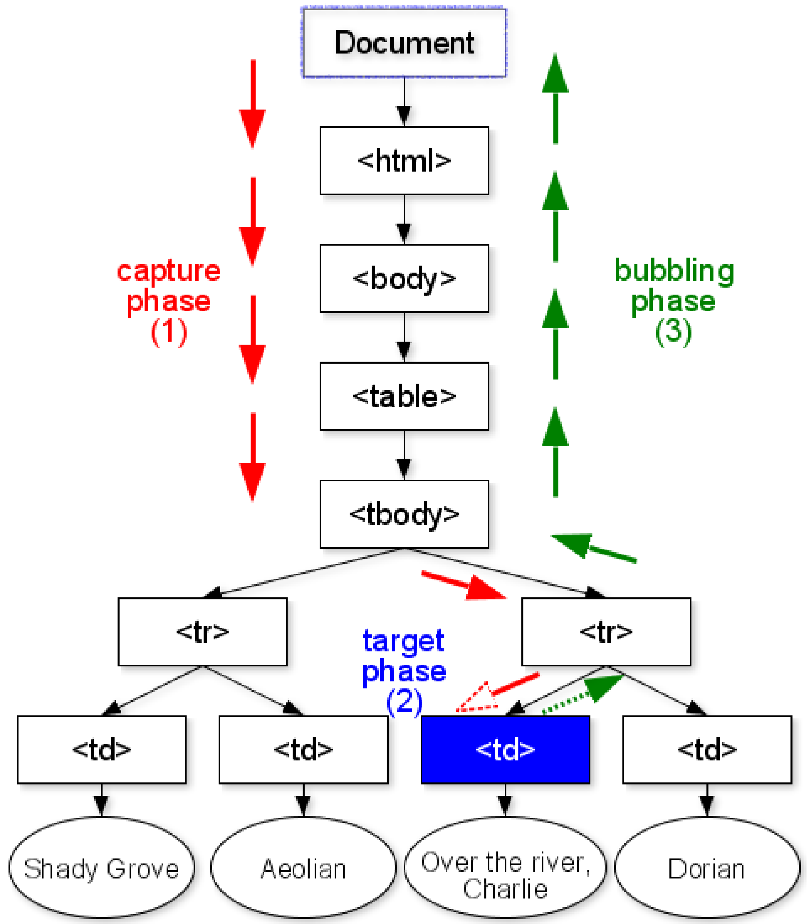
- If there are registered but not enabled handlers there for the event, they are run

The ***bubbling phase*** (3rd phase)

- Event goes back to the root; all encountered registered but not enabled handlers are run

DOM 2 - Event Propagation

Event propagation example:



DOM 2 - Event Model

A few notes about DOM 2 events handling:

- **Not all events bubble** (e.g., `load` and `unload`)
- Any handler can stop further event propagation by calling the **`stopPropagation`** method of the `Event` object
- DOM 2 model uses the `Event` object method **`preventDefault`** to stop default operations, such as submission of a form, if an error has been detected

DOM 2 Event Handlers

Event handler registration is done with the **`addEventListener`** method

3 parameters:

1. Name of the event, as a string literal
2. The handler function
3. A Boolean value that specifies whether the event is enabled during the capturing phase

Example code:

```
node.addEventListener("change", chkName, false);
```

DOM 2 Event Handlers

Some useful tips:

- A temporary handler can be created by registering it and then unregistering it with **`removeEventListener`**
- The **`currentTarget`** property of `Event` always references the object on which the handler is being executed
- `MouseEvent`s have two properties, **`clientX`** and **`clientY`**, that have the *x* and *y* coordinates of the mouse cursor, relative to the upper left corner of the browser window

Examples

`validator2.html`

Summary

- HTML pages are represented by the DOM, which has a tree structure
- HTML elements can be accessed by the DOM address, the element name, or the getElementById method
- Events that occur in the web page can be handled by assigning a function to either a corresponding tag attribute or using the addEventListener method.
- DOM elements can be traversed, added, modified, or removed using appropriate object methods
- In DOM2, events propagate through a three stage process of capturing, target node, and bubbling phases
- The navigator object can be used to get information about user's browser
- The canvas element can be used to draw to the screen through JavaScript