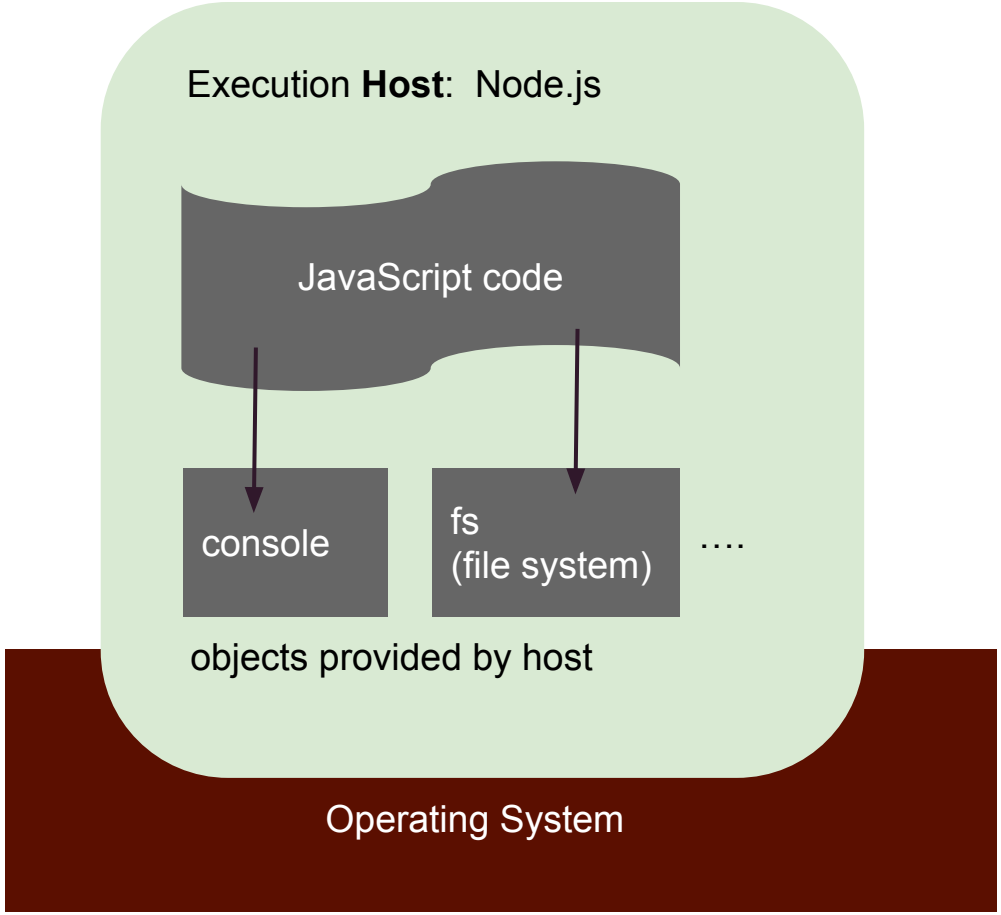
Client-Side JavaScript

Lecture 12

Reading

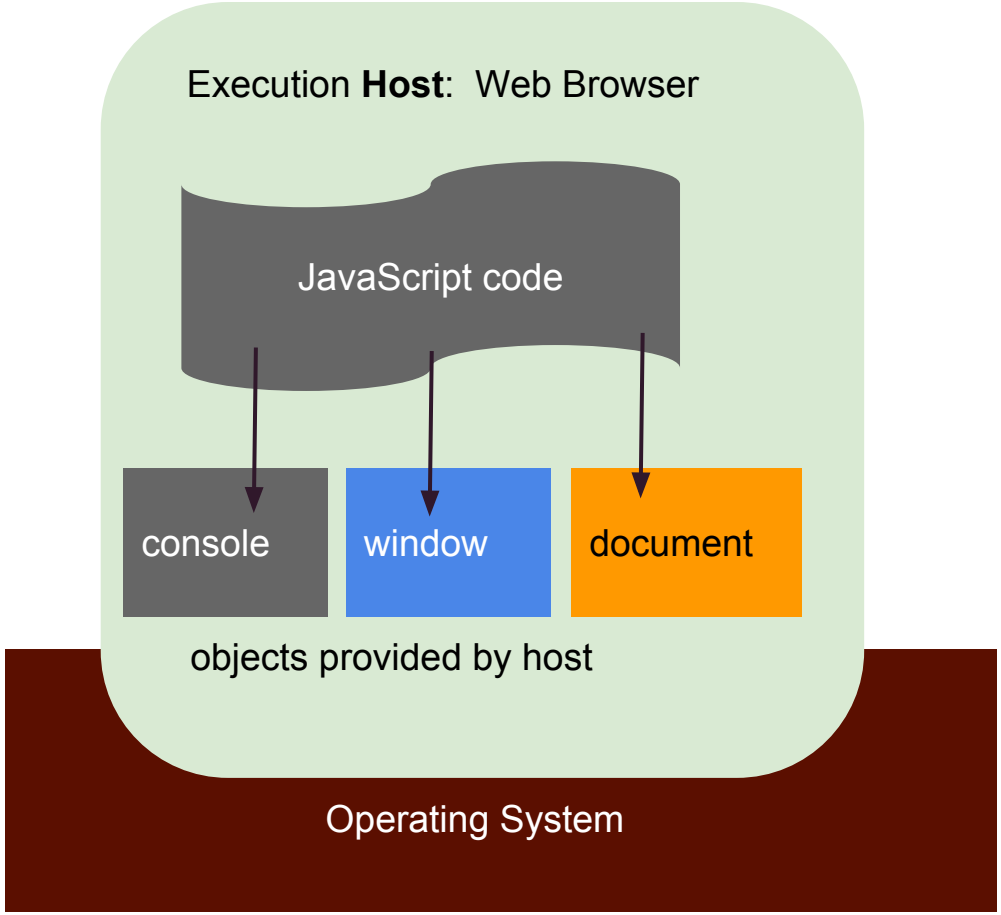
- The HTML book contains a nice introduction of JavaScript and specifically client-side scripting in chapter 10
- We've already seen the core language - so much will be review
- The JavaScript text has very detailed information in Chapter 13-19

Execution Environment



- So far Node has provided our execution environment
- It has provided access to the console, the file system, and has many more modules that we haven't yet seen.
- Node is new on the block… the browser was the original host.

Execution Environment



- **window** represents the browser itself
  - Web history
  - Timers
  - Location (loaded URL)
  - Information about platform
  - Dialog boxes

- document represents the **page** loaded
  - Access to HTML element
  - Event registration

Client-side JavaScript

- Using the window and document objects, JavaScript can:
  - Manipulate the document by add/remove/edit HTML elements, modifying attributes, and applying CSS
  - Respond to events (clicking on a button)
  - Change the URL to load another page
  - Talk to the user (~~alert, prompt~~)
  - Create animations (using timers)
  - And even talk to the server…

Adding JavaScript to your page

- JavaScript appears in two possible places in your HTML:
- Inside a script element
  - Directly embedded
  - External files
- Inside an attribute value (events)

Note: we are going to see some *bad* examples along with good. Please take notes so you understand what patterns are recommended, and which style are “old” and no longer recommended!

The script element

- When loading an HTML page, the browser **renders** each element as it sees it…
  - The **rendering** of a **script** element actually means *executing* the code within it!

```
<script>
    alert("hello world… obnoxiously!");
    console.log("hello world… discretely!");
    document.write("<p>hello world…</p>");
</script>
```

- Recall - in Node.js, all JavaScript outside of functions resided at "global scope" and were executed immediately. The same holds for browsers!

The alert box

- Any variable or function not explicitly scoped is actually part of the **window** object

```
alert("...") == window.alert("...")
```

- Alert lets you invoke a **modal** dialog with a message.
- The user must dismiss the dialog before execution or rendering will continue!
- Alerts should really **never** be used anymore…
  - They are ugly, can't be styled, and they annoy people.

The console object

- The console object was not originally part of the standard client-side toolset
  - In the dark ages, alert boxes were the "go to" debugging technique… and people were sad.
  - Firefox… and then Chrome added it.
  - Shockingly, Internet Explorer did not.

```
if (window.console) {
    console.log('message');
}
```

  - In IE9, console works, but only when the developer tools (F12) are open!!!
  - console is available in IE10+

Writing right to the document!

The last "hello world" we saw previously was writing directly into the document

```
document.write("<p>hello world...</p>");
```

- While this is very rarely used like this anymore, this code synchronously writes HTML into the document (as it is being rendered)
- Note - the HTML elements embedded in the string. The write function injects pure text.

document.write

```
<!doctype html>
<html>
    <body>
        <ul>
        <script>
            for ( i = 0; i < 5; i++ ) {
                document.write ("<li>hello</li>");
            }
        </script>
        </ul>
    </body>
</html>
```

Dev tools v.s. view source

- hello
- hello
- hello
- hello
- hello

```
<!doctype html>
<html>
        <body>
                <ul>
                <script>
                        for ( i = 0; i < 5; i++ ) {
                                document.write ("<li>hello</li>");
                        }
                </script>
                </ul>
        </body>
</html>
```

View source issues a new request to the server, and displays the raw source for you

Elements Resources Network Sources Timeline Profiles Audits Co

```
<!DOCTYPE html>
▼<html>
  <head></head>
  ▼<body>
    ▼<ul>
      <script>
                for ( i = 0; i < 5; i++ ) {
                    document.write ("<li>hello</li>");
                }
            </script>
      <li>hello</li>
      <li>hello</li>
      <li>hello</li>
      <li>hello</li>
      <li>hello</li>
    </ul>
  </body>
</html>
```

Developer tools (F12 on Chrome) shows you the **currently loaded document**.

Where to put your JavaScript

- Its common to have JavaScript directly in `<script>` elements
  - Typically in the <head> element, not in the body
  - Can be anywhere though…
- When you have a lot of JavaScript, it's a lot better to put it in an external file

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

Important: the script element cannot be "empty"... always `<script ..></script>`, not `<script ... />`

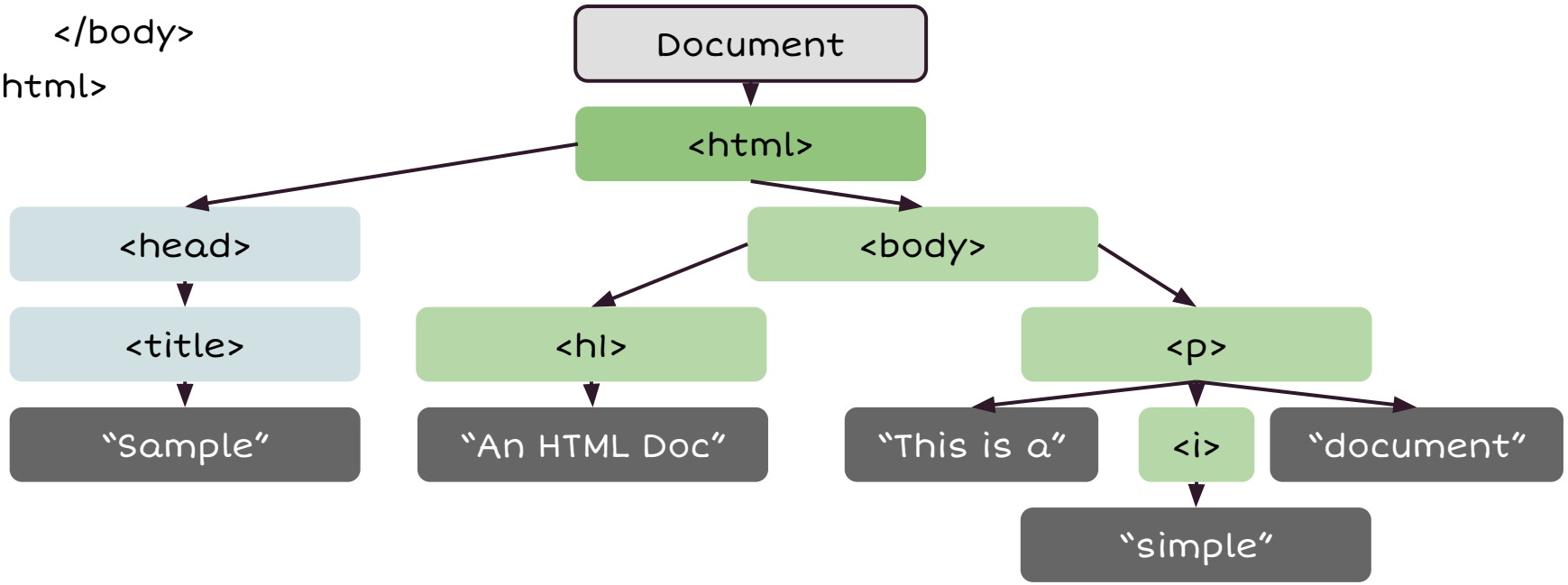
The document model

- Writing HTML manually through strings is ineffective and incredibly error prone.
- We will soon learn about the "document object model" - or DOM.
- The DOM allows you to work with JavaScript objects representing elements of HTML
- This is much easier, powerful, and effective!

DOM manipulation

```
<html>
    <head>
        <title>Sample</title>
    </head>
    <body>
        <h1>An HTML Doc</h1>
        <p>This is a <i>simple</i> document</p>
    </body>
</html>
```

Each element is a **node**
Each text part is a **node also**



Getting to elements

You need to get an element before manipulating it… this is called document query

```
document.body
document.getElementById(“id”)
document.getElementsByName(“name”)
var spans = document.getElementsByTagName(“span”)
for ( int i = 0; i < spans.length; i++ ) {
    console.write(“span”);
}
```

Timing

Careful - your javascript could execute before the elements you want to manipulate are loaded!

```
<html>
    <head>
        <script>
            var spans = document.getElementsByTagName("span");
            console.log(spans.length); // will print 0
        </script>
    </head>
    <body>
        <span>Hello</span>
        <span>World</span>
        <script>
            var spans = document.getElementsByTagName("span");
            console.log(spans.length); // will print 2
        </script>
    </body>
</html>
```

We'll see better ways of dealing with this problem...

Getting to elements

You can move from node-to-node

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

Node properties

Standard HTML attributes can be directly accessed and changed

```
img.src = “new.jpg”
console.log(form.action)
form.method = “post”
```

Other attributes available through get and set calls

```
span.getAttribute(“class”);
span.setAttribute(“data-goto”, “here”);
```

Node content

Remember, we are not talking about Node.js!

```
<div id="mydiv"><p>This is my div</p></div>
<script>
    var div = document.getElementById("mydiv");
    console.log(div.outerHTML);
    console.log(div.innerHTML);
    console.log(div.textContent);
    div.innerHTML = "<p>My change</p>";
</script>
```

Modifying DOM

You create a new node using the document

```
var newNode = document.createElement("div");
var newText = document.createTextNode("my text");
```

Note - this isn't attaching the node to the DOM!

```
parent.appendChild(newNode)
parent.insertBefore(newNode, aChildNode)
```

Modifying the DOM

`removeChild(node)` removes a node.

It is called on the ***parent***

```
parent.removeChild(node)
```

To remove a node, you could call

```
node.parentNode.removeChild(node);
```

Events

- Instead of always executing code immediately, we can set things up so JavaScript runs only when some event occurs.

- The HTML DOM defines a collection of **intrinsic** events. In their most primitive form, they can be attached to using attributes on the HTML

```
<button onclick="alert(`clicked')">
    Click me
</button>
```

Modify DOM with buttons

- Lets let the user type some text in a text box.
- When they click “post”, write the text to the actual DOM as HTML.
- If they click “clear”, remove all their postings

Many Events

Mouse-driven events

Property	Description
onclick	The event occurs when the user clicks on an element
ondblclick	The event occurs when the user double-clicks on an element
onmousedown	The event occurs when a user presses a mouse button over an element
onmousemove	The event occurs when the pointer is moving while it is over an element
onmouseover	The event occurs when the pointer is moved onto an element
onmouseout	The event occurs when a user moves the mouse pointer out of an element
onmouseup	The event occurs when a user releases a mouse button over an element

http://www.w3schools.com/jsref/dom_obj_event.asp

```
<div ondblclick="handleDoubleClick()">Click me</div>
```

Many Events

Keyboard events

Attribute	Description
onkeydown	The event occurs when the user is pressing a key
onkeypress	The event occurs when the user presses a key
onkeyup	The event occurs when the user releases a key

http://www.w3schools.com/jsref/dom_obj_event.asp

```
<input type="text" name="f" onkeydown="dostuff()"/>
```

Many Events

Window Events

Attribute	Description
onabort	The event occurs when an image is stopped from loading before completely loaded (for <object>)
onerror	The event occurs when an image does not load properly (for <object>, <body> and <frameset>)
onload	The event occurs when a document, frameset, or <object> has been loaded
onresize	The event occurs when a document view is resized
onscroll	The event occurs when a document view is scrolled
onunload	The event occurs once a page has unloaded (for <body> and <frameset>)

http://www.w3schools.com/jsref/dom_obj_event.asp

```
<body onload="initialize_stuff()"/>
```

Many Events

Form events

Attribute	Description
onblur	The event occurs when a form element loses focus
onchange	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)
onfocus	The event occurs when an element gets focus (for <label>, <input>, <select>, textarea>, and <button>)
onreset	The event occurs when a form is reset
onselect	The event occurs when a user selects some text (for <input> and <textarea>)
onsubmit	The event occurs when a form is submitted

http://www.w3schools.com/jsref/dom_obj_event.asp

```
<input type="text" onchange="validate()"/>
```

Change the browser location

What if we didn't have <a> elements?

- CSS class: hover
- Set an attribute on the element ("data-goto")
- When clicked, set the browser location!

Side note: Starting in HTML5, it is legal to add any attribute to an element, as long as it starts with "data-"

Eventually browser support will catch up to HTML5 and let you access these attributes using an object (element.dataset.goto)....
….*but for now support is spotty.*

Note - we'll do this using the primitive JavaScript API, when we do it with jQuery it will be far better….

More on timing

- A browser is **single threaded**.
- HTML5 does introduce Web Workers - but that's not in scope here...

- If the browser is executing JavaScript, it is **not**
  - rendering HTML
  - responding to user input

- Long running scripts make your website unresponsive.
- This is **totally** unacceptable.

Stopping Events

Events often have default actions.

You can prevent them from occurring

```
<form onsubmit="validate(e)">
    ....
</form>
```

```
function validate(event) {
    if ( bad() )  event.preventDefault();
}
```

Timers

The window.setInterval function takes two arguments

1. A function to call
2. Number of milliseconds between calls.

The function is called repeatedly with given interval

```
<body onload="setInterval(doStuff, 1000)">
```

Don't forget… **milliseconds**

`setTimeout` works similarly, but is only called once

Guessing Game – 100% client side?

Lets compute a secret number on load

After each guess, we'll just do a document.write to create the output

Too high, too low

And we'll clear the text box after each bad guess

Next – jQuery

- Very few people work only with the JavaScript core API
  - Its a bit verbose
  - Browser support is spotty in some areas
- To enhance reliability, productivity, and to create cleaner code, most sane people rely on frameworks

  - jQuery
  - Prototype
  - AngularJS
  - React