LECTURE11: JAVASCRIPT 2

CS418/518

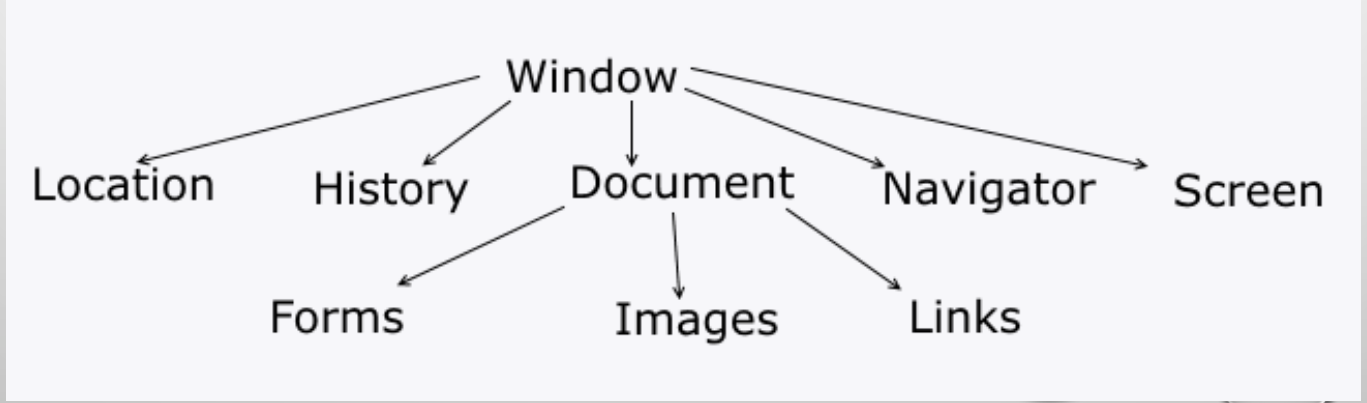
JIAN WU

Recall last class

- Introduced JavaScript
- How to access HTML elements
- How to create a click event function
- Variables, data types, loops, operators, control statements,
- How to display JavaScript data
- Writing into an alert box, using `window.alert()`.
  - The window object is the global scope object, that means that variables, properties, and methods by default belong to the window object.

The six global DOM objects

Name	Description
document	current HTML page and its content
history	list of pages the user has visited
location	URL of the current HTML page
navigator	info about the web browser you are using
screen	info about the screen area occupied by the browser
window	the browser window



The window object

- ***The entire browser window; the top-level object in DOM hierarchy***
- Technically, all global code and variables become part of the window object properties:
  - document, history, location, name
- Methods:
  - alert, confirm, prompt (popup boxes)
  - `setInterval, setTimeout clearInterval, clearTimeout` (timers)
  - `open, close` (popping up new browser windows)
  - `blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo`

For a complete list, see: https://www.w3schools.com/jsref/obj_window.asp

The document object

- ***The current web page and the elements inside it***
- Properties:
  - `anchors, body, cookie, domain, forms, images, links, referrer, title, URL`
- Methods:
  - `getElementById`
  - `getElementsByName`
  - `getElementsByTagName`
  - `close, open, write, writeln`

For a complete list, see:
https://www.w3schools.com/jsref/dom_obj_document.asp

The location object

- ***The URL of the current web page***
- Properties:
  - `host, hostname, href, pathname, port, protocol, search`
- Methods:
  - `assign, reload, replace`
- Example:

```
var x = location.host;
location.reload();
```

For a complete list, see: https://www.w3schools.com/jsref/obj_location.asp

The navigator object

- *Information about the web browser application*
- Properties:
  - `appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent`
- Some web programmers examine the navigator object to see what browser is being used, and write browser-specific scripts and hacks:

```
if (navigator.appName === "Microsoft Internet Explorer") { ...
```
JS

The screen object

- *Information about the client's display screen*
- Properties:
  - `availHeight, availWidth, colorDepth, height, pixelDepth, width`
- Example:

```
var x = "Total Width: " + screen.width;
```

For more methods see: https://www.w3schools.com/jsref/obj_screen.asp

The history object

- The list of sites the browser has visited in this window
- Properties:
  - `length`
- Methods:
  - `back, forward, go`
- Sometimes the browser won't let scripts view `history` properties, for security

let vs var

- Variables declared by let have their scope in the block for which they are declared, as well as in any contained sub-blocks.
- In this way, let works very much like var. The main difference is that the scope of a var variable is the entire enclosing function:

```
function varTest() {
  var x = 1;
  {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

Obtrusive vs. Unobtrusive JavaScript

- Mixing JavaScript with HTML was *obtrusive (example below)*, in the HTML; this is bad style
- Now we'll see how to write *unobtrusive* JavaScript code
  - HTML with minimal JavaScript inside
  - Uses the DOM to attach and execute all JavaScript functions
- Allows separation of web site into 3 major categories:
  - **Content** (HTML) - what is it?
  - **Presentation** (CSS) - how does it look?
  - **Behavior** (JavaScript) - how does it respond to user interaction?

```
<!DOCTYPE html>
<html>
<body>
<h1>The Element Object</h1>
<h2>The innerHTML Property</h2>

<p id="demo" onclick="myFunction()">Click me to change my HTML content (innerHTML).</p>

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

</body>
</html>
```

Obtrusive Event Handlers (bad)

```
<button id="ok" onclick="okayClick();">OK</button>
```
HTML

```
// called when OK button is clicked
function okayClick() {
    alert("booyah");
}
```
JS

- This is bad style (HTML is cluttered with JS code)
- Goal: remove all JavaScript code from the HTML body

Attaching an event handler to DOM Elements in JavaScript

general

```
// where element is a DOM element object
element.event = function;
```
JS

example

```
$("ok").onclick = okayClick;
```
JS

DOM object event handler function

- It is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - Notice that you do not put parentheses after the function's name
- This is better style than attaching them in the HTML
- Where should we put the above code?

When does my code run?

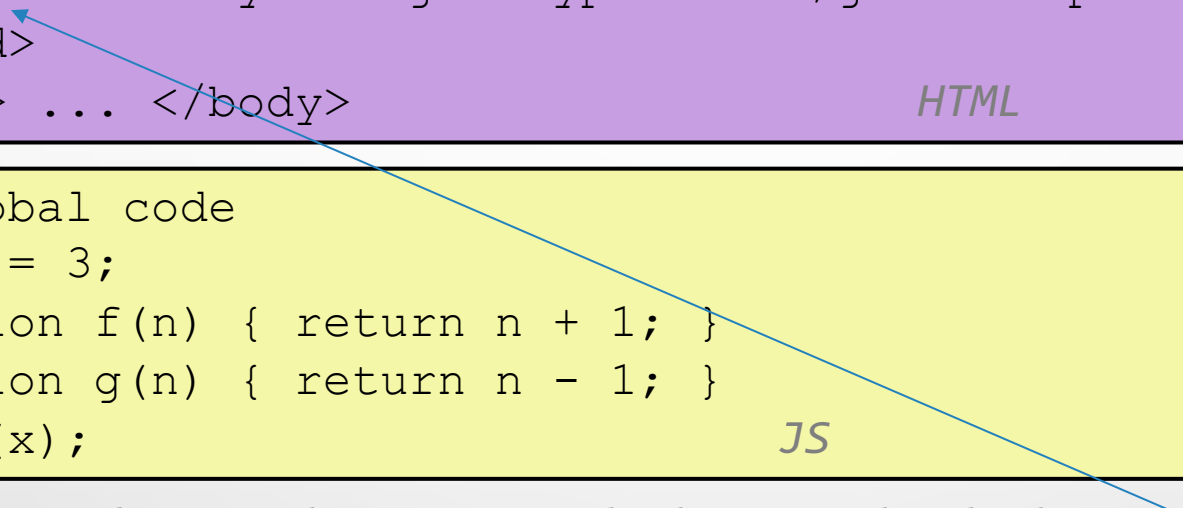
```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>
```

HTML

myfile.js

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```

JS



- Your file's JS code runs the moment the browser loads the script tag
  - Any variables are declared immediately
  - Any functions are declared but not called, unless your global code explicitly calls them

When does my code run?

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>
```

HTML

myfile.js

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```

JS

- At this point in time, the browser has not yet read your page's body
  - None of the DOM objects for tags on the page have been created

A failed attempt at being unobtrusive

```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body>
<div><button id="ok">OK</button></div>
```

HTML

```
// global code
$("ok").onclick = okayClick; // error: $("ok") is null
```

JS

- Problem: global JS code runs the moment the script is loaded
- Script in head is processed before page's body has loaded
  - No elements are available yet or can be accessed yet via the DOM

The `window.onload` event

```
// this will run once the page has finished loading
function functionName() {
    element.event = functionName;
    element.event = functionName;
...
}
window.onload = functionName; // global code                    JS
```

- We want to attach our event handlers right after the page is done loading
  - There is a global event called `window.onload` event that occurs at that moment
- For the window object, the load event is fired when the whole webpage (HTML) has loaded fully, including all dependent resources, including JavaScript files, CSS files, and images.

Add two functions to window.onload:
https://www.htmlgoodies.com/beyond/javascript/article.php/3724571/using-multiple-javascript-onload-functions.htm

An unobtrusive event handler

```
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>
```
HTML

```
window.onload = function() {
    var okButton = document.getElementById("ok");
    okButton.onclick = okayClick;
};
function okayClick() {
    alert("booyah");
}
```

Common unobtrusive JS errors

```
window.onload = pageLoad();
window.onload = pageLoad;
okButton.onclick = okayClick();
okButton.onclick = okayClick;                                    JS
```

- EVENT NAMES ARE ALL LOWERCASE, NOT CAPITALIZED LIKE MOST VARIABLES

```
window.onLoad = pageLoad;
window.onload = pageLoad;                                        JS
```

Anonymous functions

```
function(parameters) {
    statements;
}
```
JS

- JavaScript allows you to declare anonymous functions (first class functions)
- Quickly creates a function without giving it a name
- Can be stored as a variable, attached as an event handler, etc.

first-class function

Anonymous function example

```
window.onload = function() {
    var okButton = document.getElementById("ok");
    okButton.onclick = okayClick;
};
function okayClick() {
    alert("booyah");
}
```

JS

The keyword `this`

```
this.fieldName // access field
this.fieldName = value; // modify field
this.methodName(parameters); // call method
```
JS

- All JavaScript code actually runs inside of an object
- By default, code runs inside the global window object
  - All global variables and functions you declare become part of window
- The `this` keyword refers to the current object

The keyword `this`

```
window.onload = function() {
    var okButton = document.getElementById("ok");
    okButton.onclick = okayClick;
};
function okayClick() {
    this.innerHTML = "booyah";
}
```

- Event handlers attached unobtrusively are **bound** to the element
- Inside the handler, that element becomes `this` (rather than the window)

More about events

abort	blur	change	click	dblclick	error	focus
keydown	keypress	keyup	load	mousedown	mousemove	mouseout
mouseover	mouseup	reset	resize	select	submit	unload

- The click event (onclick) is just one of many events that can be handled
- **Problem**: events are tricky and have incompatibilities across browsers
  - Reasons: fuzzy W3C event specs; IE disobeying web standards; etc.
- **Solution**: Prototype includes many event-related features and fixes

The $ function

```
$("id")
```
JS

- Returns the DOM object representing the element with the given id
- Short for document.getElementById("id")
- Often used to write more concise DOM code:

```
$("footer").innerHTML = $("username").value.toUpperCase();
```
JS

The $$ function

```
var arrayName = $$("CSS selector");
```
JS

```
// hide all "announcement" paragraphs in the "news"
//section
var paragraphs = $$("div#news p.announcement");
for (var i = 0; i < paragraphs.length; i++) {
    paragraphs[i].hide();
}
```
JS

- $$ returns an array of DOM elements that match the given CSS selector
  - Like $ but returns an array instead of a single DOM object
  - A shorthand for document.select
- Useful for applying an operation each one of a set of elements

Common issues with $$

```
// get all buttons with a class of "control"
var gameButtons = $$("control");
var gameButtons = $$(".control");
```
JS

```
// set all buttons with a class of "control" to have red text
$$(".control").style.color = "red";
var gameButtons = $$(".control");
for (var I = 0; i < gameButtons.length; i++) {
    gameButtons[i].style.color = "red";
}
```
JS

- Q: Can I still select a group of elements using $$ even if my CSS file doesn't have any style rule for that same group? (A: Yes!)

Attaching multiple event handlers with $$

```
// listen to clicks on all buttons with class "control" that
// are directly inside the section with ID "game"
window.onload = function() {
    var gameButtons = $$("#game > button.control");
    for (var i = 0; i < gameButtons.length; i++) {
        gameButtons[i].observe("click", gameButtonClick);
    }
};
function gameButtonClick() { ... }
```
JS

- You can use $$ and other DOM walking methods to unobtrusively attach event handlers to a group of related elements in your window.onload code

The Event object

```
function name(event) {
// an event handler function ...
}
```
JS

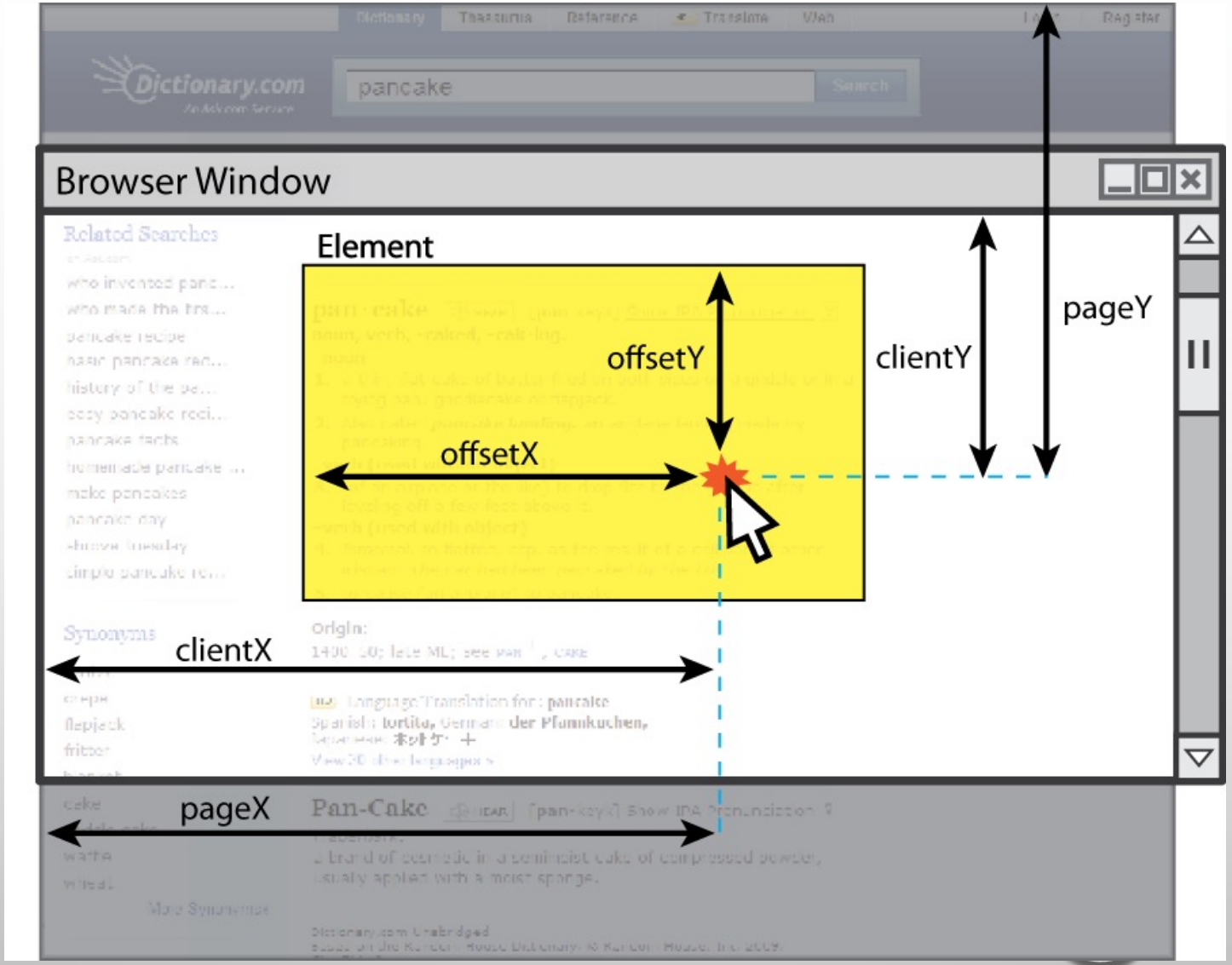
- Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties / methods:

Method / Property name	Description
type	what kind of event, such as "click" or "mousedown"
element()	the element on which the event occurred
stop()	cancels an event
stopObserving()	removes an event handler

Mouse events

click	user presses/releases mouse button on this element
dblclick	user presses/releases mouse button twice on this element
mousedown	user presses down mouse button on this element
mouseup	user releases mouse button on this element
mouseover	mouse cursor enters this element's box
mouseout	mouse cursor exits this element's box
mousemove	mouse cursor moves around within this element's box

Mouse event objects



Mouse event objects

property/method	description
clientX, clientY	coordinates in browser window
screenX, screenY	coordinates in screen
offsetX, offsetY	coordinates in element
pointerX(), pointerY() *	coordinates in entire web page
isLeftClick() **	true if left button was pressed

* replaces non-standard properties pageX and pageY
** replaces non-standard properties button and which

Example: Mouseover me

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

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
  obj.innerHTML = "Thank You"
}

function mOut(obj) {
  obj.innerHTML = "Mouse Over Me"
}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<script src="mousefun.js" type="text/javascript"></script>
</head>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

</body>
</html>
```

html

```
function mOver(obj) {
  obj.innerHTML = "Thank You"
}

function mOut(obj) {
  obj.innerHTML = "Mouse Over Me"
}
```

javascript

Examples: popup window

- http://jsfiddle.net/9RxLM/

- https://jsfiddle.net/fanchyna/j21hbezq/1/

- Every element has a default display value. However, you can override this.
- `display:block` will display elements in blocks
- `display:none` is commonly used with JavaScript to hide and show elements without deleting and recreating them.

BACKUP SLIDES BEYOND THIS POINT

CS418/518

The Event object

```
<pre id="target">Move the mouse over me!</pre>
```

HTML

```
window.onload = function() {
    $("target").observe("mousemove", showCoords);
};
function showCoords(event) {
    this.innerHTML =
    "pointer: (" + event.pointerX() + ", " +
event.pointerY() + ")\n"
    + "screen : (" + event.screenX + ", " + event.screenY
+ ")\n"
    + "client : (" + event.clientX + ", " + event.clientY +
")";
```

JS

Example: Tip Calculator

```
<h1>Tip Calculator</h1>
<div>
    $<input id="subtotal" type="text" size= "5" /> subtotal <br />
    <button id="tenpercent">10%</button>
    <button id="fifteenpercent"> 15%</button>
    <button id="eighteenpercent"> 18%</button>
    <span id="total"></span>
</div>
```

HTML

```
window.onload = function() {
    $("tenpercent").onclick = computeTip;
    }
function computeTip{
    var subtotal = parseFloat($("subtotal").value);
    var tipAmount = subtotal*0.1;//Add this code
    $("total").innerHTML = "Tip: $" + tipAmount;
}
```

JS