JS, JQuery, Ajax

Slides are from MIT – AITI, Marty Stepp, Jessica Miller, and Victoria Kirst, Ruth Betcher

and Ruth Christie

Based on Internet Standards

- XHTML/HTML and CSS
  - To display the data
- JavaScript (XMLHttpRequest calls)
  - To exchange data asynchronously with the server
- XML
  - To tranfer the data
- DOM (document object model)
  - To navigate the hierarchy of X/HTML elements

Variables

- A variable is a name associated with a piece of data
- Variables allow you to store and manipulate data in your programs
- Think of a variable as a mailbox which holds a specific piece of information

Variables

- In JavaScript variables are created using the keyword `var`

- Example:

```
var x = 10;

var y = 17;

var color = “red”;

var name = “Katie”;
```

Variables

- It is vitally important to distinguish between the *name* of the variable and the *value* of the variable
- For example, in the expression `var color="red"`, color is the name of the variable and red is the value. In other words, color is the name of the box while red is what is inside the box

Data Types

- Primitive Data Types
  - Numbers
  - Strings
  - Boolean (True, False)
- Composite Data Types
  - Arrays
  - Objects

Primitive Data Types

- **Numbers -** A number can be either an integer or a decimal
- **Strings -** A string is a sequence of letters or numbers enclosed in single or double quotes
- **Boolean** - True or False

Variables & Data Types

- JavaScript is *untyped*; It does not have explicit data types
- For instance, there is no way to specify that a particular variable represents an integer, string, or real number
- The same variable can have different data types in different contexts

Implicit Data Types

- Although JavaScript does not have explicit data types, it does have implicit data types
- If you have an expression which combines two numbers, it will evaluate to a number
- If you have an expression which combines a string and a number, it will evaluate to a string

Example: Variables

```
var x = 4;

var y = 11;

var z = “cat”;

var q = “17”;
```

```
Ans = x + y;
    Ans => 15

Ans = z + x;
    Ans => cat4

Ans = x + q;
    Ans => 417
```

More Examples

var x = 4;

var y = 11;

var z = “cat”;

var q = “17”;

Ans = x + y + z;

Ans => 15cat

Ans = q + x + y;

Ans => 17411

Arrays

- An array is a compound data type that stores numbered pieces of data
- Each numbered datum is called an *element* of the array and the number assigned to it is called an *index*.
- The elements of an array may be of any type. A single array can even store elements of different type.

Creating an Array

- There are several different ways to create an array in JavaScript
- Using the `Array()` constructor:
  - var a = new Array(1, 2, 3, 4, 5);
  - var b = new Array(10);
- Using array literals:
  - var c = [1, 2, 3, 4, 5];

Accessing Array Elements

- Array elements are accessed using the [ ] operator
- Example:
  - var colors = [“red”, “green”, “blue”];
  - colors[0] => red
  - colors[1] => green

Adding Elements

- To add a new element to an array, simply assign a value to it
- Example:

```
var a = new Array(10);
a[50] = 17;
```

Array Length

- All arrays created in JavaScript have a special length property that specifies how many elements the array contains
- Example:
  - var colors = [“red”, “green”, “blue”];
  - colors.length => 3

Primitive Data Types versus Composite Data Types

- Variables for primitive data types hold the actual value of the data
- Variables for composite types hold only references to the values of the composite type

Variable Names

- JavaScript is **case sensitive**
- Variable names cannot contain spaces, punctuation, or start with a digit
- Variable names cannot be reserved words

Programming Tips

- It is bad practice to change the implicit type of a variable. If a variable is initialized as a number, it should always be used as an number.
- Choose meaningful variable names

Statements

- A statement is a section of JavaScript that can be evaluated by a Web browser
- A script is simply a collection of statements

Examples:

```
Last_name = “Dunn”;
x = 10 ;
y = x*x ;
```

Programming Tips

- It is a good idea to end each program statement with a semi-colon;
  Although this is not necessary, it will prevent coding errors

- **Recommended**:

```
a = 3;
b = 4;
```

- **Acceptable:**

```
a = 3;  b = 4;
```

- **Wrong:**

```
a =
3;
```

Operators

+	Addition	= =	Equality
-	Subtraction	! =	Inequality
*	Multiplication	!	Logical NOT
/	Division	&&	Logical AND
%	Modulus	||	Logical OR
++	Increment	?	Conditional Selection
- -	Decrement		

Aggregate Assignments

- Aggregate assignments provide a shortcut by combining the assignment operator with some other operation
- The += operator performs addition and assignment
- The expression x = x + 7 is equivalent to the expression x += 7

Increment and Decrement

- Both the increment (++) and decrement (- -) operator come in two forms: prefix and postfix
- These two forms yield different results

```
x = 10;          x = 10;
y = ++ x;        z  = x ++;
```

⇒ **y = 11**

⇒ **z = 10**

⇒ **x = 11 in both cases**

Control Structures

- There are three basic types of control structures in JavaScript: the `if` statement, the `while` loop, and the `for` loop
- Each control structure manipulates a block of JavaScript expressions beginning with { and ending with }

The If Statement

- The `if` statement allows JavaScript programmers to a make decision
- Use an `if` statement whenever you come to a “fork” in the program

```
If ( x == 10)
{       y = x*x;
}
else
{       x = 0;
}
```

Repeat Loops

- A repeat loop is a group of statements that is repeated until a specified condition is met
- Repeat loops are very powerful programming tools; They allow for more efficient program design and are ideally suited for working with arrays

The While Loop

- The while loop is used to execute a block of code while a certain **condition** is true

```
count = 0;
while (count <= 10) {
    document.write(count);
    count++;
}
```

The For Loop

- The for loop is used when there is a need to have a **counter** of some kind
- The counter is initialized before the loop starts, tested after each iteration to see if it is below a target value, and finally updated at the end of the loop

Example: For Loop

```
// Print the numbers 1
   through 10

for (i=1; i<= 10; i++)
   document.write(i);
```

i=1 initializes the counter

i<=10 is the target value

i++ updates the counter at the end of the loop

Example: For Loop

```
<SCRIPT
        LANGUAGE=
        "JavaScript">
document.write("1");
document.write("2");
document.write("3");
document.write("4");
document.write("5");
</SCRIPT>
```

```
<SCRIPT
        LANGUAGE=
        "JavaScript">

for (i=1; i<=5; i++)
    document.write(i);
```

Functions

- Functions are a collection of JavaScript statement that performs a specified task
- Functions are used whenever it is necessary to repeat an operation

Functions

- Functions have inputs and outputs
- The inputs are passed into the function and are known as **arguments** or **parameters**
- Think of a function as a “black box” which performs an operation

Defining Functions

- The most common way to define a function is with the `function` statement.
- The function statement consists of the function keyword followed by the name of the function, a comma-separated list of parameter names in parentheses, and the statements which contain the body of the function enclosed in curly braces

Example: Function

```
function square(x)
{return x*x;}

z = 3;
sqr_z = square(z);
```

Name of Function: square

Input/Argument: x

Output: x*x

Example: Function

```
function sum_of_squares(num1,num2)
   {return (num1*num1) + (num2*num2);}

function sum_of_squares(num1,num2)
   {return (square(num1) + square(num2));}
```

jQuery

Slides are from Marty Stepp, Jessica Miller, and Victoria Kirst

What is jQuery?

- jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. (jQuery.com)

Why learn jQuery?

- Write less, do more:
  - *$("p.neat").addClass("ohmy").show("slow");*
- Performance
- Plugins
- It’s standard
- … and fun!

window.onload

- We cannot use the DOM before the page has been constructed. jQuery gives us a more compatibile way to do this.
  - The DOM way

```
window.onload = function() { // do stuff with the DOM }
```

  - The direct jQuery translation

```
$(document).ready(function() { // do stuff with the DOM });
```

  - The jQuery way

```
$(function() { // do stuff with the DOM });
```

Selecting groups of DOM objects

name	description
getElementById	returns array of descendents with the given tag, such as "div"
getElementsByTagName	returns array of descendents with the given tag, such as "div"
getElementsByName	returns array of descendents with the given name attribute (mostly useful for accessing form controls)
querySelector *	returns the first element that would be matched by the given CSS selector string
querySelectorAll *	returns an array of all elements that would be matched by the given CSS selector string

jQuery / DOM comparison

DOM method	jQuery equivalent
getElementById("id")	$("#id")
getElementsByTagName("tag")	$("tag")
getElementsByName("somename")	$("[name='somename']")
querySelector("selector")	$("selector")
querySelectorAll("selector")	$("selector")

The jQuery object

- The $ function always (even for ID selectors) returns an array-like object called a jQuery object.
- The jQuery object wraps the originally selected DOM objects.
- You can access the actual DOM object by accessing the elements of the jQuery object.

```
// false
document.getElementById("id") == $("#myid");
document.querySelectorAll("p") == $("p");
// true
document.getElementById("id") == $("#myid")[0];
document.getElementById("id") == $("#myid").get(0);
document.querySelectorAll("p")[0] == $("p")[0];
```

Using $ as a wrapper

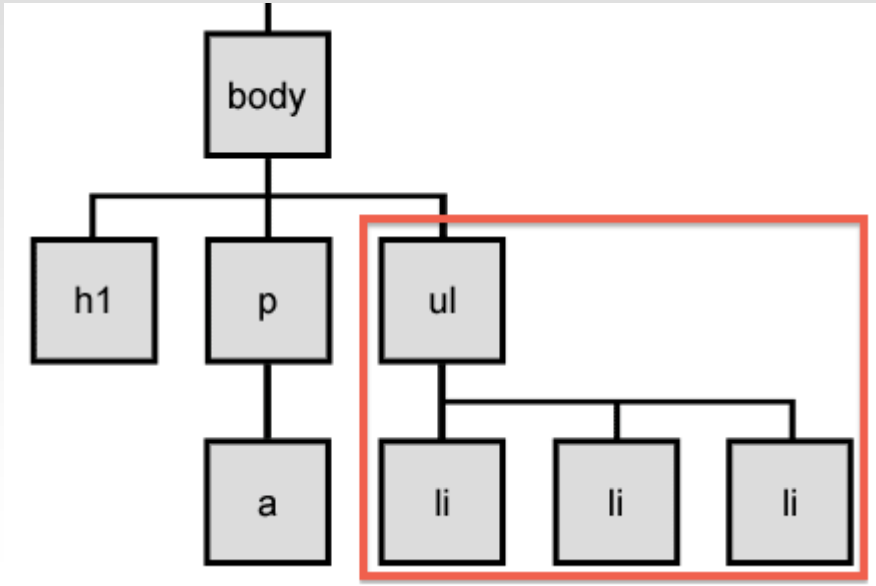
- $ adds extra functionality to DOM elements
- passing an existing DOM object to $ will give it the jQuery upgrade

```
// convert regular DOM objects to a jQuery object
var elem = document.getElementById("myelem");
elem = $(elem);
var elems = document.querySelectorAll(".special");
elems = $(elems);
```

DOM context identification

- You can use querySelectorAll() and querySelector() on any DOM object.
- When you do this, it simply searches from that part of the DOM tree downward.
- Programmatic equivalent of a CSS context selector

```
var list = document.getElementsByTagName("ul")[0];
var specials = list.querySelectorAll('li.special');
```



find / context parameter

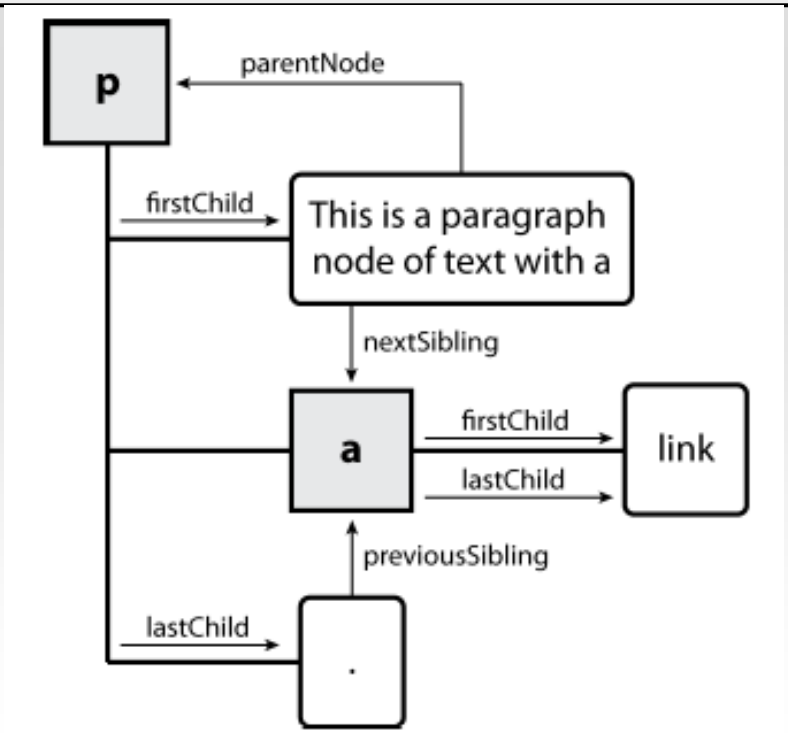
- jQuery gives two identical ways to do contextual element identification

```
var elem = $("#myid");

// These are identical
var specials = $("li.special", elem);
var specials = elem.find("li.special");
```

DOM tree traversal example

```
<p id="foo">
     This is a paragraph of text with a
     <a href="/path/to/another/page.html">link</a>.
</p>
```



jQuery traversal methods

- http://api.jquery.com/category/traversing/

jQuery tutorials

- Code Academy

  http://www.codecademy.com/courses/you-and-jquery/0?curriculum_id=4fc3018f74258b0003001f0f#!/exercises/0

- Code School:

  http://www.codeschool.com/courses/jquery-air-first-flight

Ajax/JQuery

```
$.ajax({
    type: 'POST',
    url: '/hello',
    data: {'student1': 'Mahdi' },
    dataType: 'json', // what we expect from server
    async: true, // what if the false one
    success: function(dataFromServer) {
        var result = JSON.parse(dataFromServer);
       alert('Just got back from server side!! with '+ result)
    },
    error: function() {
       alert('Something bad happened in our server !!')
    }
});
```

- `async:false` = Code **paused**. (Other code **waiting** for this to finish.)
- `async:true` = Code **continued**. (Nothing gets paused. Other code is **not waiting**.)