Lecture

JavaScript – Basics

What is JavaScript

- JavaScript is a "simple", interpreted, programming language with elementary object-oriented capabilities

- JavaScript has two distinct systems
  - client-side JavaScript runs on Web browsers
  - server-side JavaScript runs on Web servers

- JavaScript syntax resembles C, C++, and Java

- Developed in 10 days by **Brendan Eich** at Netscape in **May 1995** (now CEO at Brave Software after serving as cofounder and CTO at Mozilla)

- The original name was **Mocha,** chosen by Marc Andreessen, founder of Netscape

- Was renamed as **LiveScript**, then **JavaScript**

- See "A Short History of JavaScript":
  https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

JavaScript is Embedded in HTML

- **In the <body>**
```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
//the Javascript here produces content for the BODY on
   loading
</SCRIPT>
</BODY>
</HTML>
```
- **or in the <head> as a deferred script**
```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
//the Javascript here creates functions for later use
</SCRIPT>
</HEAD>
<BODY>
</BODY></HTML>
```

A Simple Example

```
<HTML>
<HEAD>
<TITLE>Simple Javascript</TITLE>
</HEAD>
<BODY>
<H1>First Example of JavaScript</H1>
<SCRIPT LANGUAGE="JavaScript">
document.write("Last updated on " + document.lastModified + ". ")
</SCRIPT>
</BODY>
</HTML>
```

Example 1: Browser Output

Another Example
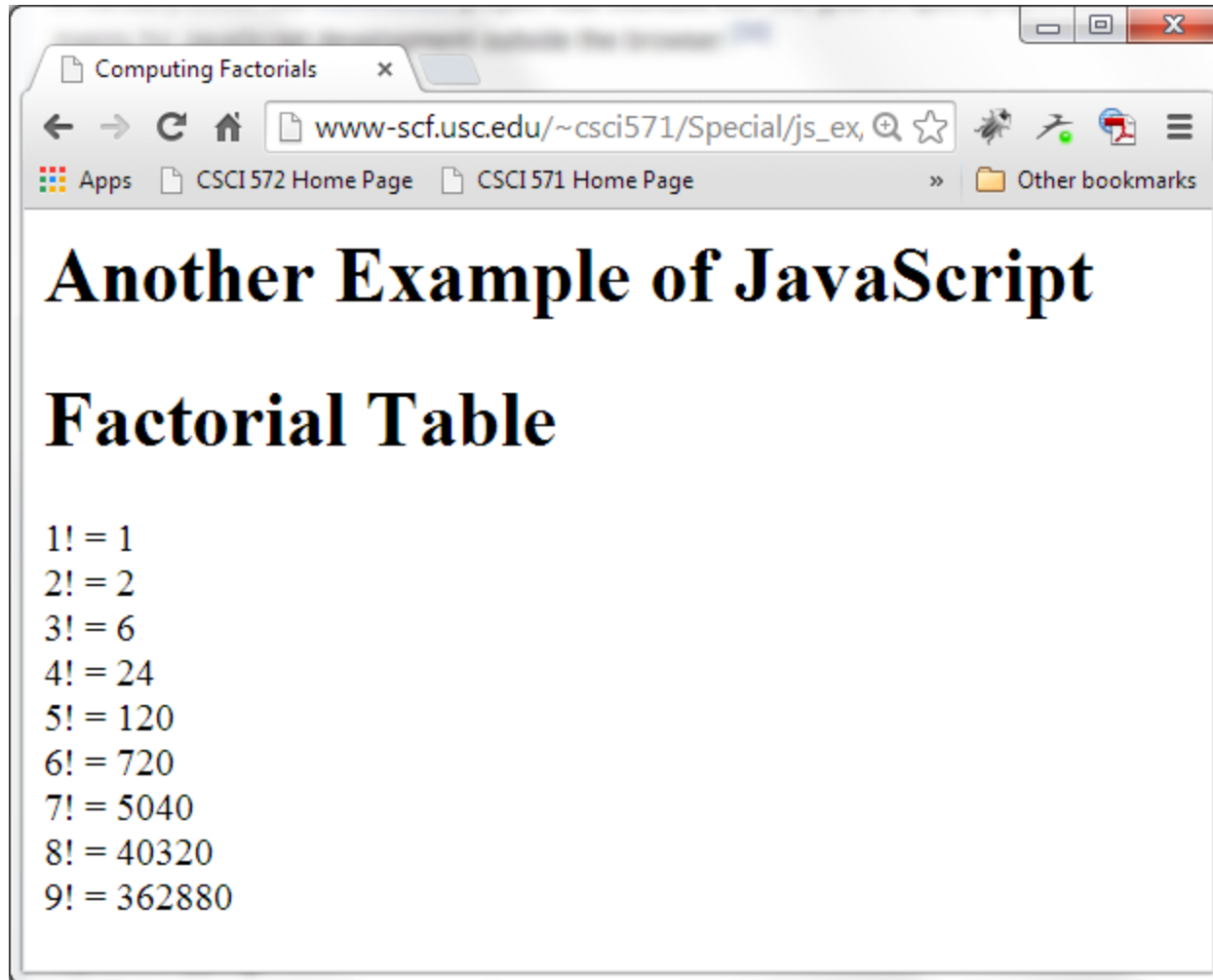
```
<HTML>

<HEAD><TITLE>Computing Factorials</TITLE></HEAD>

<BODY>

<H1>Another Example of JavaScript</H1>

<SCRIPT LANGUAGE="JavaScript">

document.write("<H1>Factorial Table</H1>");

for ( i = 1, fact = 1; i < 10; i++, fact = fact * i) {

   document.write(i + "! = " + fact);

   document.write("<BR>");

}

</SCRIPT>
</BODY>

</HTML>
```

Classical C for statement

Example 2: Browser Output

JavaScript has Event Handlers
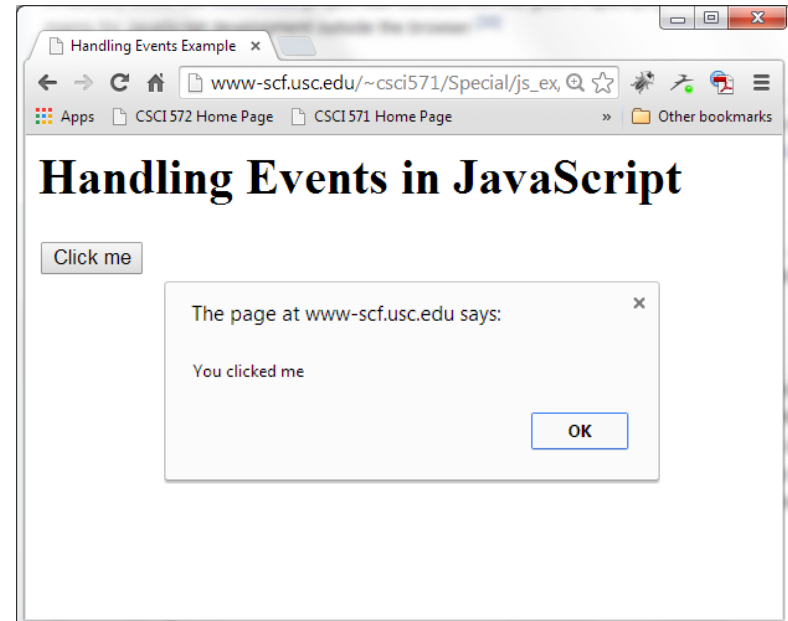
```
<HTML>

<HEAD><TITLE>Handling Events Example</TITLE></HEAD>

<BODY>

<H1>Handling Events in JavaScript</H1>

<INPUT TYPE="button" VALUE="Click me"

    onClick="alert('You clicked me')" >

</BODY>

</HTML>
```

Some Common Events

- **Mouse Events**
  - onclick   user clicks an HTML element
  - ondblclick   user double-clicks an element
  - onmouseover   user moves the mouse over an HTML element
  - onmouseout   user moves the mouse away from an HTML element
- **Keyboard Events**
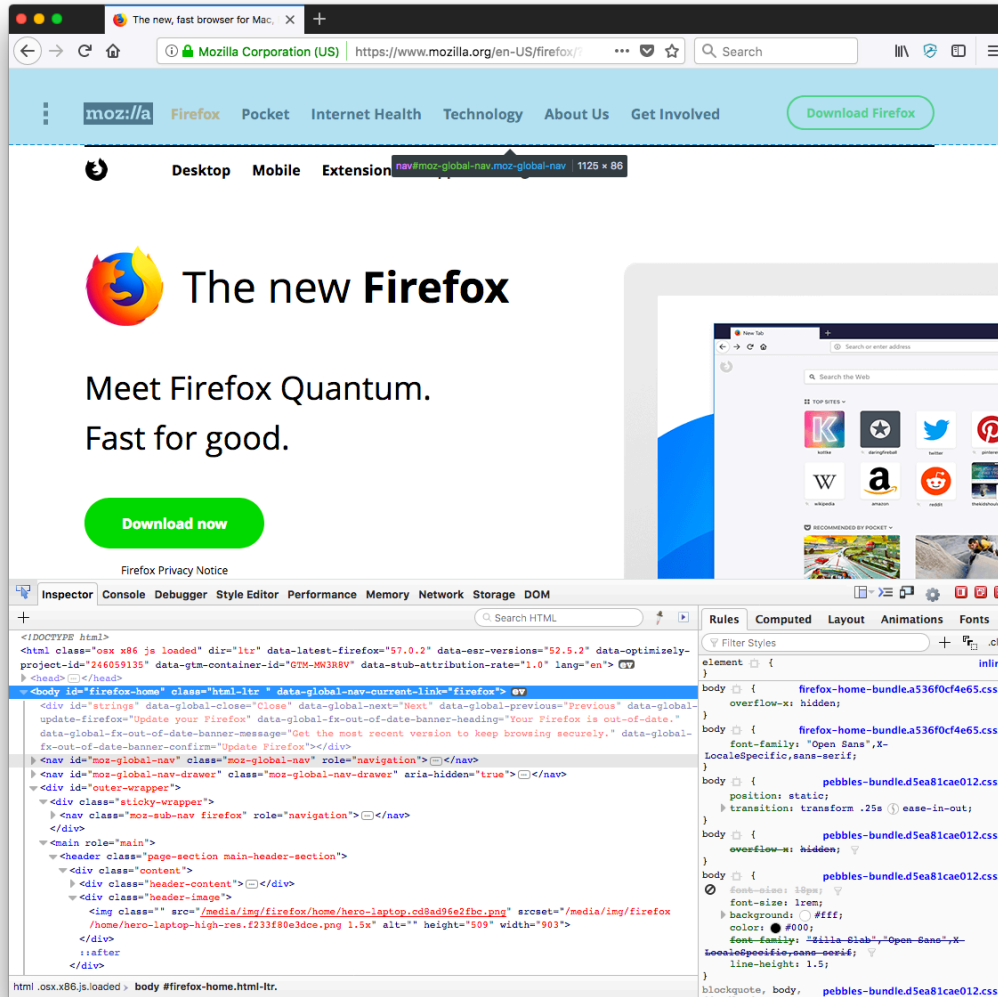 - onkeydown   user presses a key
  - onkeyup   user releases a key
- **Object Events**
  - onload   browser has finished loading the page
  - onunload   a page has unloaded
  - onresize   a document view is resized
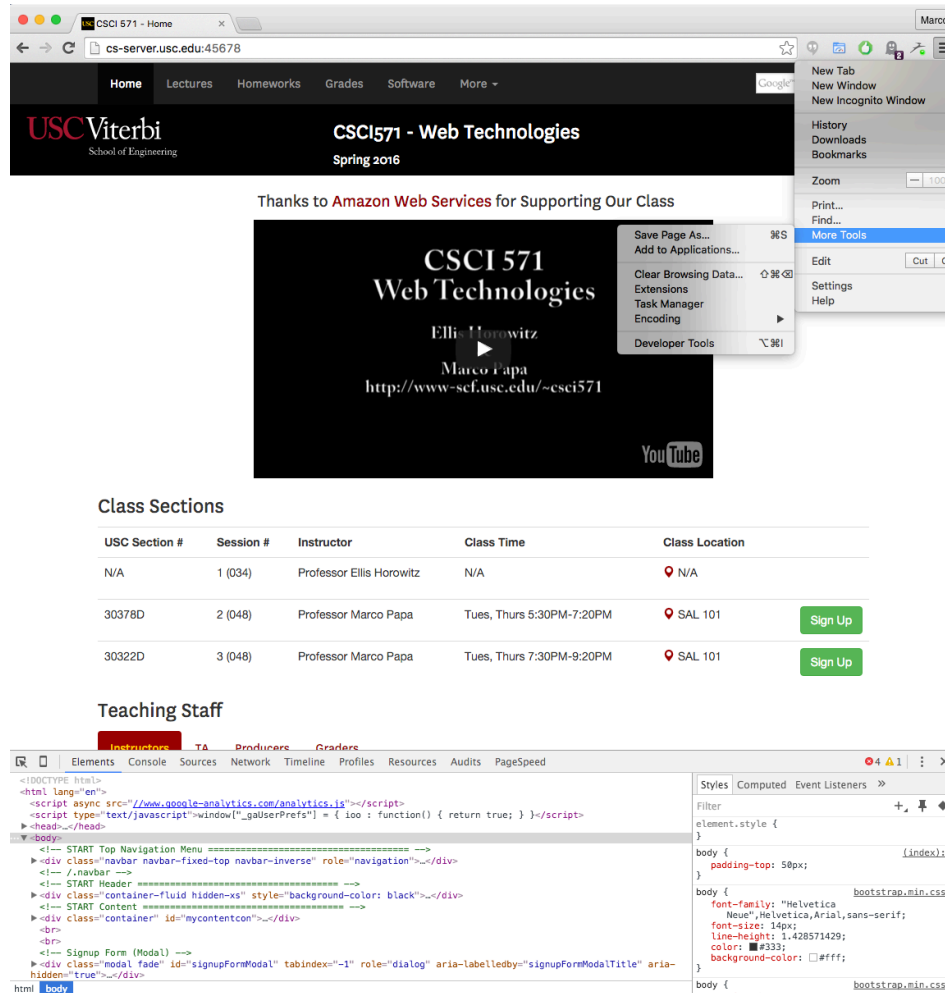 - onscroll   a document view is scrolled

Debugging JavaScript
Firefox Developer Tools

- Built into Firefox

- Replaces Firebug add-on

- To invoke, select Firefox menu > Web Developer

Debugging JavaScript
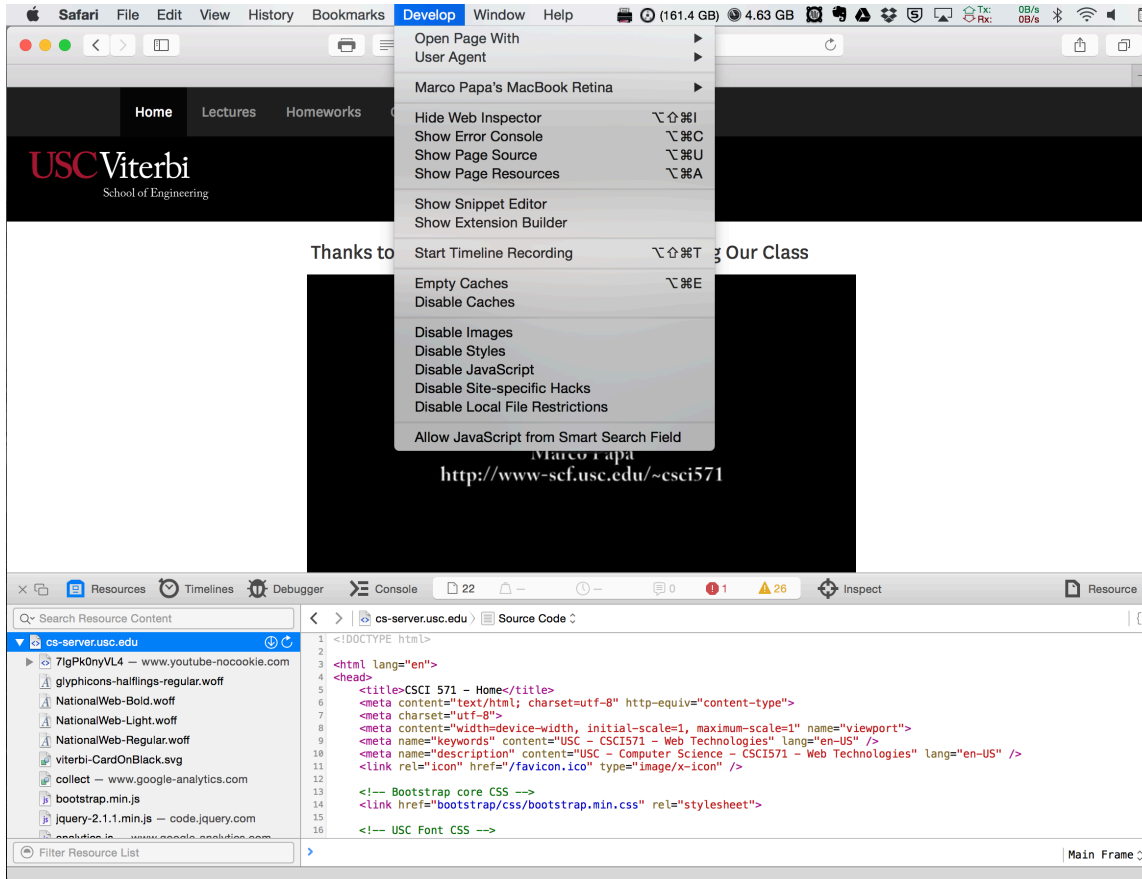# Chrome Developer Tools

- The Chrome Developer Tools are built into Google Chrome.

- They provide deep access into the internals of the browser and their web application.

Use the DevTools to
- track down layout issues
- set JavaScript breakpoints
- get insights for code optimization.

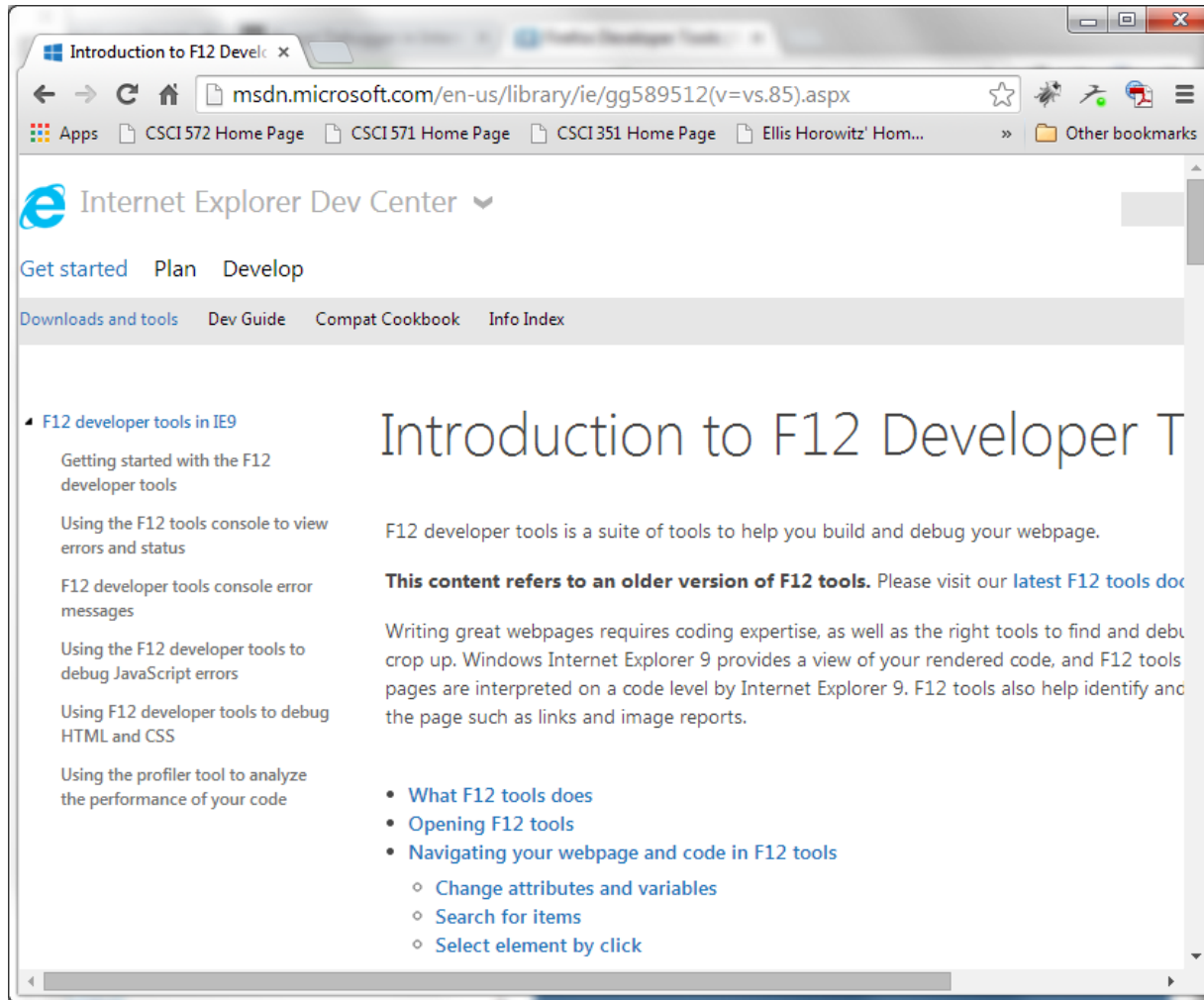- Go to Tools > More Tools > Developer Tools

Debugging JavaScript
Safari Developer Tools

- The Safari Developer Tools are built into Safari.

- Developer Tools include
  - Web Inspector
  - Error Console
  - Page Source
  - Page Resources
  - Snippet Editor
  - Extension Builder
  - Debugger
  - Timelines
  - Timeline recording

- Turn on Develop menu: Preferences > Advanced > check "Show Develop menu in menu bar"

Debugging JavaScript
Internet Explorer F12 Developer Tools

A similar set of capabilities as Firebug and Chrome Developer Tools.

Available in IE9, IE10 and IE11.

Debugging JavaScript
Edge F12 Developer Tools

Includes seven tools:

- DOM Explorer tool
- Console tool
- Debugger tool
- Network tool
- Performance tool
- Memory tool
- Emulation tool

Documentation available at: https://docs.microsoft.com/en-us/microsoft-edge/f12-devtools-guide

What JavaScript Programs Can Do

• Write programs to perform any computation; it is equivalent in power to a general-purpose programming language

• But it is specifically designed for **manipulating web pages**

  – Control Web page appearance and content (this is its intended use)

  – Control the Web browser, open windows, test for browser properties

  – Interact with document content

  – Retrieve and manipulate all hyperlinks

  – Interact with the user, sensing mouse clicks, mouse moves, keyboard actions

  – Read/write client state with cookies

Limitations of Client-Side JavaScript

- [**Was**] Extremely difficult to explicitly draw graphics.
  - **This has been dramatically improved in the latest versions**
- No access to the underlying file system or operating system
- Unable to open and use arbitrary network connections
- No support for multithreading
- [**Was**] Not suitable for computationally intensive applications
  - **This has been dramatically improved in the latest versions**

JavaScript – Basics of the Language

- JavaScript is **case-sensitive**
  - sum, SUM and Sum are 3 different identifiers
  - HTML is NOT case-sensitive
- JavaScript **ignores** spaces, tabs, newlines
  - So it can be **minified**
- Semicolon is optional
  - but multiple statements on a line require a semicolon

  ```
  i = 1; j = 2
  ```

- C and C++ style comments are supported

```
//comment to end of line
```

```
/* this can be a
multiple line comment */
```

JavaScript Literals

- **literals** are fixed values, (not variables) that you literally provide in your script
  - **numbers,** 42, -5, 3.14159, -7.5E6
    - All numbers are treated as floating point
    - Octal (begin with a zero), 01234
    - Hexadecimal (begin with zero and x), 0xFF
  - **boolean,** true, false, (also null and undefined)
  - **strings,** any sequence of zero or more characters enclosed within single or double quotes
    - Examples
      - 'a single quoted string'
      - "a double quoted string"
      - ""
      - "alert('Please Click OK')"

JavaScript Strings

- **Strings** are <u>**immutable**</u>, once created they can never be changed
- You can search a string and extract substrings, but you cannot modify a string
- "Immutable" means that once you instantiate the object, you can't change its properties.
- So, when calling methods on a string, JavaScript will return the modified string, but it won't change the initial string
- Now this doesn't mean that you can't assign a new string object to the str variable. You just can't change the current object that str references.
- One generally builds a string through concatenation

```
var newWebPage = ""

newWebPage += "<HTML><HEAD>"

newWebPage += "<TITLE>A Sample Page</TITLE></HEAD>"

newWebPage += "<BODY>My Home Page</BODY>"

newWebPage += "</HTML>"
```

Properties of Strings

• Strings have a length property

      "Lincoln".length // result = 7

      "Four score".length //result = 10

      "One\ntwo".length // result = 7

      "".length // result = 0

• Some String methods

string.**tolowerCase**(); string.**toupperCase**()

string.**indexOf**(searchstring [, startindex])   //returns index value
                          of char within string where searchString begins

string.**charAt**(index)   //returns the one char at position index

string.**substring**(indexA, indexB)   //returns characters of string
                          between indexA and indexB

JavaScript Escape Notation

- Escape sequences are used to embed special characters in a string

  \b backspace   \t   tab

  \f form feed   \'   single quote

  \n newline   \"   double quote

  \r carriage return   \\   backslash

- Example of escape characters in strings

msg = 'You\'re using an embedded single quote here.'

msg = "This is on the first line \n and this is on the second line."

msg = document.title + "\n" + document.links.length + "links present"

JavaScript Reserved Words

JavaScript identifiers start with a letter, $, or underscore followed by zero or more letters or digits;

JavaScript reserved words; you cannot use these reserved words as variables, labels, or function names

abstract	arguments	boolean	break	byte
case	catch	char	class*	const
continue	debugger	default	delete	do
double	else	enum*	eval	export*
extends*	false	final	finally	float
for	function	goto	if	implements
import*	in	instanceof	int	interface
let	long	native	new	null
package	private	protected	public	return
short	static	super*	switch	synchronized
this	throw	throws	transient	true
try	typeof	var	void	volatile
while	with	yield		

More JavaScript Words to Avoid

You should also avoid using the name of JavaScript built-in
objects, properties, and methods including:

Array	Date	eval	function	hasOwnProperty
Infinity	isFinite	isNaN	isPrototypeOf	length
Math	NaN	name	Number	Object
prototype	String	toString	undefined	valueOf

Some notes:
1. The **NaN** property represents "Not-a-Number" value.
This property indicates that a value is not a legal number
2. **Infinity** is a numeric value that represents positive infinity.
-Infinity is a numeric value that represents negative infinity.
3. The **valueOf()** method returns the primitive value of the specified object.

JavaScript Variables

- Variables should be declared, but not their type

```
var i, sum;       //declaration
var zero = 0;     //declaration and initialization
var myName = "Ellis"
```

- The **type** of value a variable can hold during execution may change.

- **Scope**

  – Any variable outside a function is a *__global__* __variable__ and can be referenced by any statement in the document

  – Variables declared in a function as "var" are *__local__* __to the function__

    – if var is omitted, the variable becomes global

- In a multi-frame or multi-window set up of the browser, scripts can access global variables from any other document currently loaded

JavaScript Data Types

JavaScript has a limited set, with some notable ones lacking, e.g. integer, and some notable ones not usually found in programming languages, e.g. function

Type	Example	Description
String	"a string"	A series of characters inside quote marks
Number	123.45	Any number not inside quote marks
Boolean	true	A logical true and false
Null	null	Completely devoid of any value, not a number, not a string, different than 0 in C/C++
Object		All properties and methods belonging to the object
Function		A function

JavaScript Arrays

- Though not an official data type, arrays are included in the language using a traditional array notation, i.e. **square brackets**

- However they differ from conventional arrays in many ways

- **array properties**

  - one dimensional, indexed from zero

  - array elements can contain any type of data including references to other arrays, to objects, to functions

  - array elements can have different types

- An **array literal** is a list of zero or more expressions, each of which represents an array element, enclosed in square brackets ([]), e.g.

  - var coffees = ["French Roast", "Columbian", "Kona"];

  - var fish = ["Tuna", , "Cod"]; (one empty element)

  - var myArray = ["Richard", 10, getPhoto()]; (string, number, function)

  - var items = [[1,2],[3,4],[5,6]]; (two-D array, items[0][0] is 1)

More on Arrays

- Every array has a length property
- The length property is the largest integer property name in the array plus one

```
var myArray = [];
myArray.length                          //0
myArray[100000] = true;
myArray.length                          //100001
```

- Arrays are **sparse,** in the above example only one index is allocated

- JavaScript does NOT provide a way to declare the size (dimension) of an array, but we can add one

```
Array.dim = function (dimension, initial) {
        var a = [], i;
        for (i = 0; i < dimension; i += 1) {
                a[i] = initial; }
        return a; };
var myArray = Array.dim(10,0); //makes an array of ten zeros
```

More on Arrays

- There are many ways to iterate over an array
  - for loop; **for (i=0; i < len; i++) {. . . }**
  - forin loop; **for (x in person) { . . . }**
  - while loop; **while (condition) {. . . }**

- There are many built-in methods for working with arrays, here are just a few:
  - concat(), joins two or more arrays
  - indexOf(), search the array for an element and return its position
  - pop(), remove the last element
  - push() add a new element at the end
  - reverse(), reverses the order of elements

- See also

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Predefined_Core_Objects

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

Arrays and Objects are Semantically Identical

- The **typeof()** function returns a string which is the type of its argument ("number", "string", "boolean", "object", "function", "undefined")

- In JavaScript objects and arrays are really identical, **typeof(array) = typeof(object) = object**

- JavaScript does **NOT** support associative arrays, e.g.

```
var person = new Array();
person["firstname"] = "John";
person["age"] = 41;
```

- person["firstName"] returns "John", but person[0] returns undefined and person.length returns 0

- Array elements are object properties in the same way that toString is a property, but trying to access an element of an array as follows throws a syntax error, because the property name is not valid:

```
    console.log(arr.0); // a syntax error
```

- There is nothing special about JavaScript arrays and the properties that cause this. JavaScript properties that begin with a digit cannot be referenced with <u>dot notation</u>; and must be accessed using <u>bracket notation</u>.

Objects

- An object literal is a list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}), e.g.

    ```
    var person = {firstName:"John",
                    lastName:"Doe",
                    age:50,
                    eyeColor:"blue"};
    ```

    – the "dot" operator is used to access the value of an object's property or to assign it a value, e.g.

    ```
    lname = person.lastName        //  returns "Doe"
    person.lastName = "Smith";
    ```

- Objects can be nested within objects, e.g.

    ```
    var myHonda = {color: "red",
            wheels: 4,
            engine: {cylinders: 4,
                      size: 2.2}
            };
    ```

Object Constructors

- It is often useful to define an "object type" that can be used multiple times to create object instances having the same structure

- To do this one creates an object constructor, which is a JavaScript function that is called with the *new* operator, e.g.

```
function cat(name, meow) { this.name = name;
  this.talk = function() { alert(this.name + " say " + meow)}}
cat1 = new cat("felix", "purr");
cat1.talk();
cat2 = new cat("ginger", "hisss");
cat2.talk();
```

- cat() is an object constructor with properties and methods declared inside

Predefined JavaScript Objects

- There are a set of predefined objects that include:
  - **Array object,** we have already seen
  - **Boolean object,** a wrapper around the primitive Boolean data type,
    - var booleanObject = new Boolean(value);
    - Any object whose value is not (undefined, null, 0, Nan or the empty string, including a Boolean object whose value is false), evaluates to true
  - **Date Object,** like Java it stores dates as the number of milliseconds since Jan. 1, 1970, 00:00:00
    - var Xmas95 = new Date("December 25, 1995");
    - Xmas95.getMonth() returns 12, Xmas95.getFullYear() returns 1995
  - **Function object**
    - Var functionObjectName = new Function([arg1, ..., argn], functionbody);
  - **Math object** includes properties and methods for mathematical constants, e.g. sin(), cos(), ceil(), floor()
  - **RegExp object** **(**discussed later**)**
  - **String object** **(**discussed later**)**

Example Using Date Object

```
<HTML>
<HEAD>
<TITLE>Example using new</TITLE>
<SCRIPT LANGUAGE=JavaScript>
function outputDate() {
    var d = new Date(); //creates today's date and time
    document.write(d.toLocaleString()); }
                        // converts a date to a string
</SCRIPT></HEAD>
<BODY>
<H1>The date and time are</H1>
<SCRIPT LANGUAGE=JavaScript>
 outputDate();
</SCRIPT>
</BODY>
</HTML>
```

See examples at: **https://csci571.com/examples.html#js**

Example 4: Browser Output
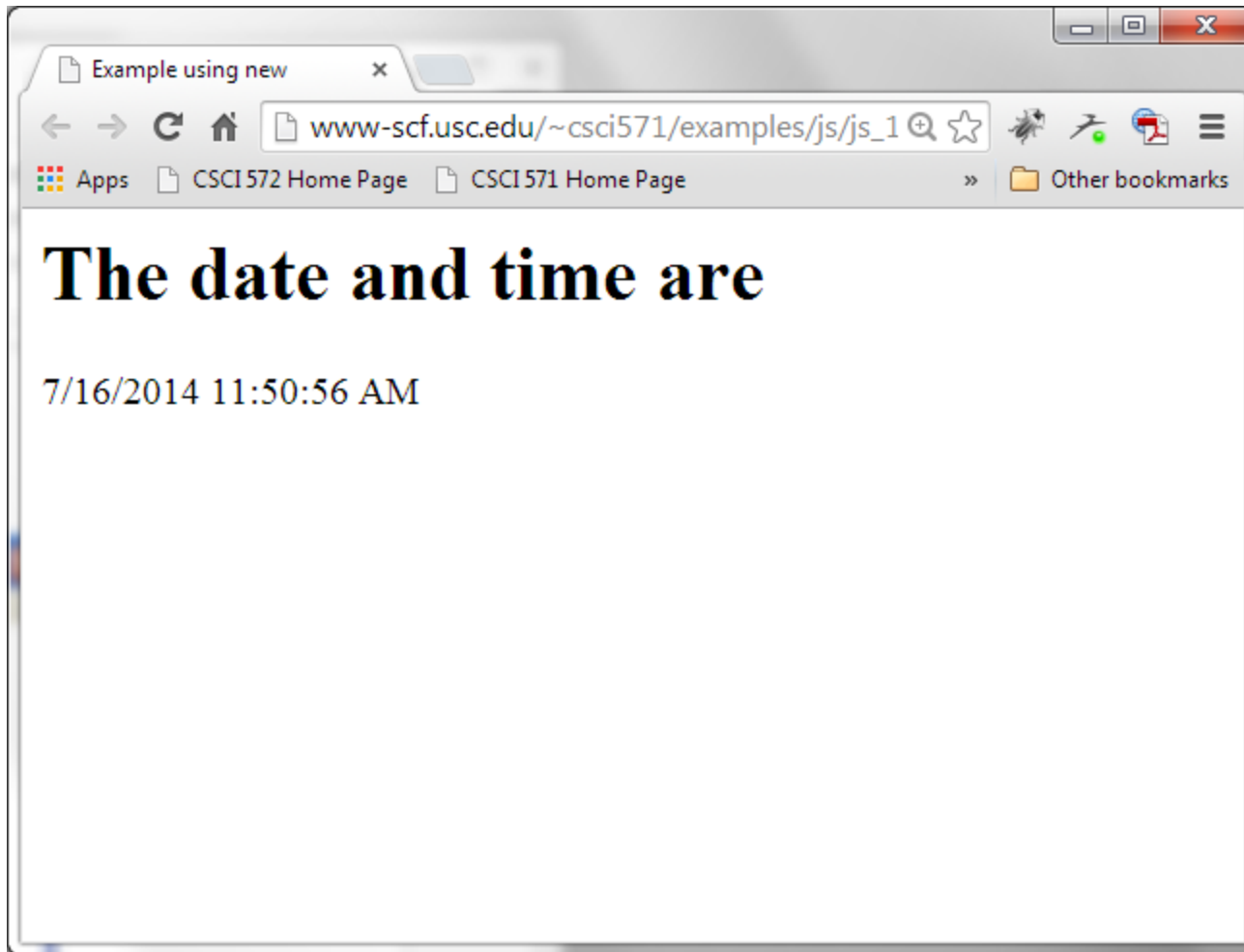
JavaScript Popup Boxes
alert(), confirm(), and prompt()

```
<HTML><HEAD>
<TITLE>Example of alert, confirm, prompt</TITLE>
<SCRIPT LANGUAGE=JavaScript>
function alertUser() {
  alert("An alert box contains an exclamation mark");}
function confirmUser() {
  var msg = "\n please confirm that you want\n" +
          "to test another button?";
  if (confirm(msg))
    document.write("<h2>You selected OK</h2>");
  else document.write("<h2>You selected Cancel</h2>"); }
function promptUser() {
  name1=prompt("What is your name? ", " ");
  document.write("<h2>welcome to this page " + name1 + "</h2>");
   }
</SCRIPT>
</HEAD>
```
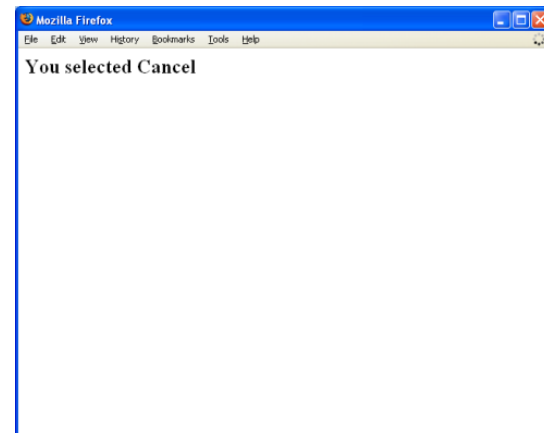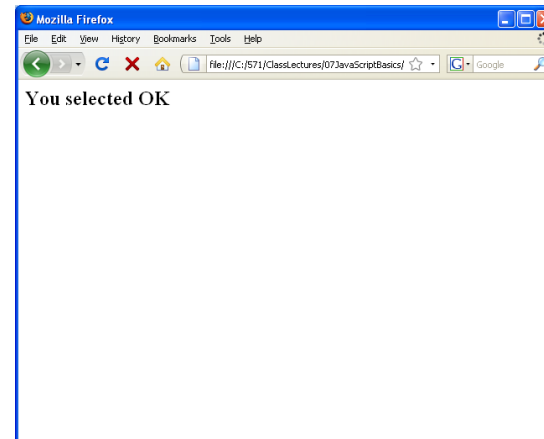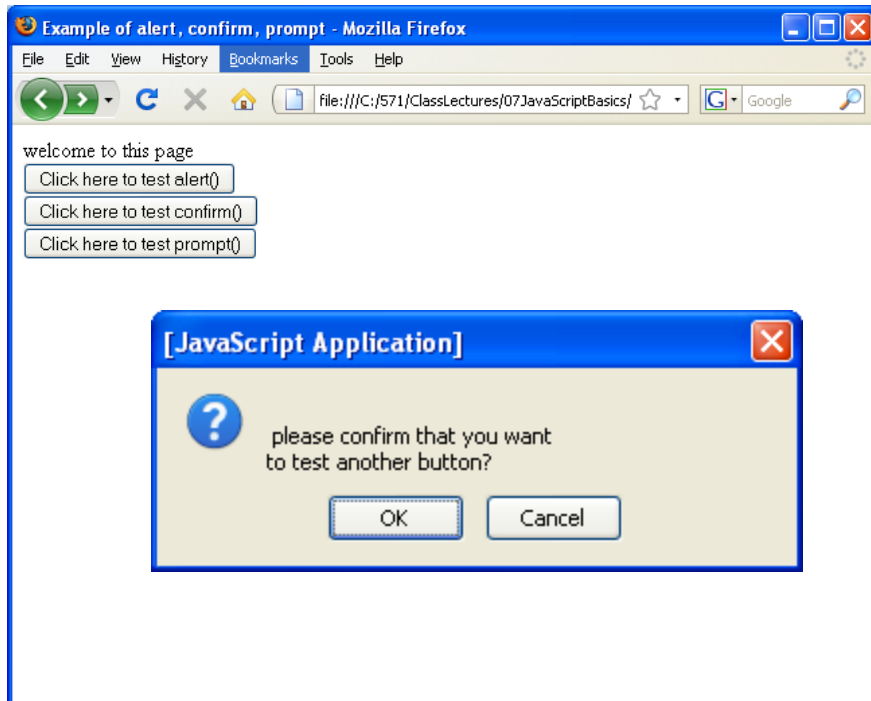
Using alert(), confirm(), and prompt()

```
<BODY>welcome to this page<br>
<FORM>
<INPUT TYPE=button VALUE="Click here to test alert()"
   onClick="alertUser()"><BR>
<INPUT TYPE=button VALUE="Click here to test confirm()"
   onClick="confirmUser()"><BR>
<INPUT TYPE=button VALUE="Click here to test prompt()"
   onClick="promptUser()">
</FORM>
</BODY>
</HTML>
```
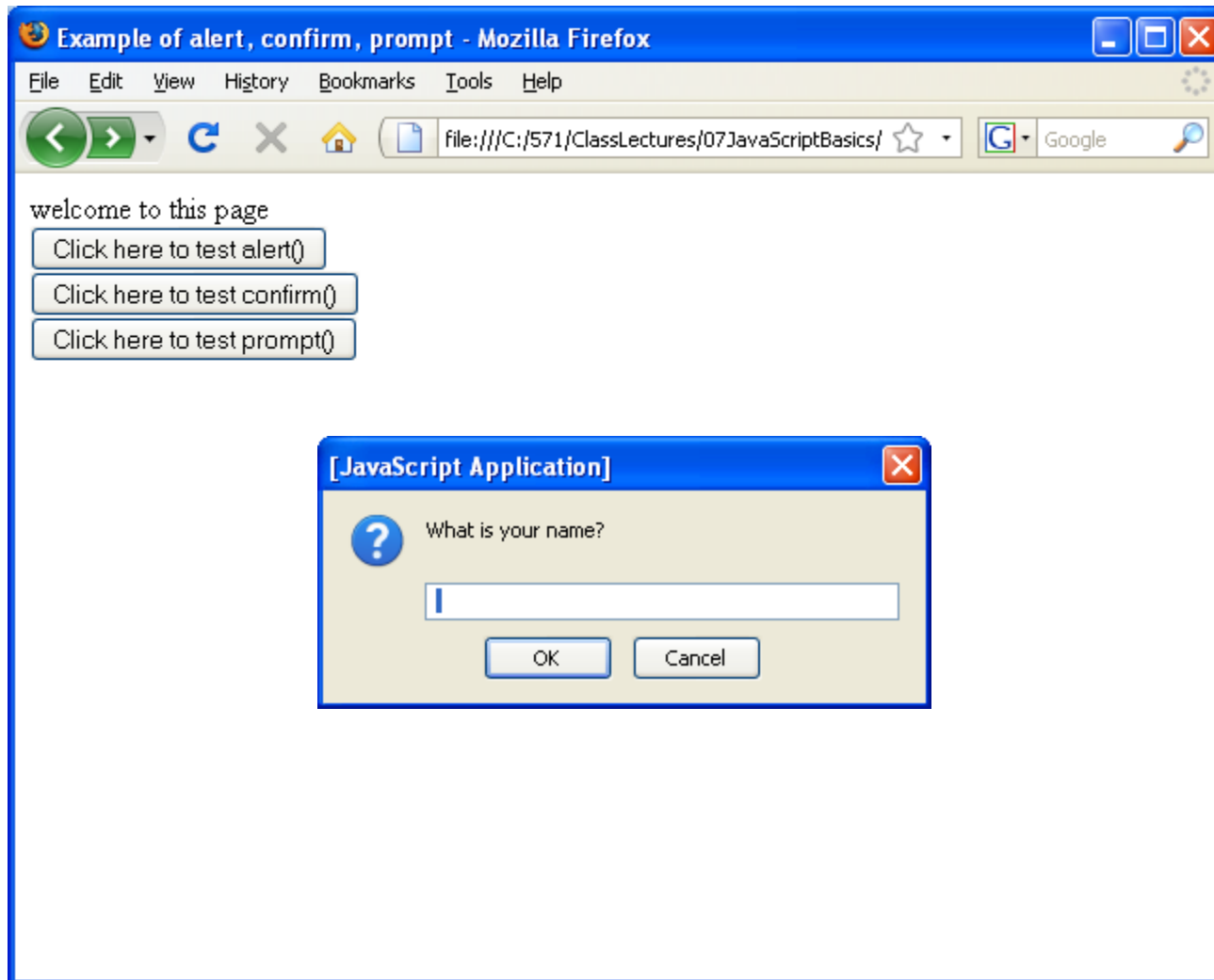
Example 5: Browser Output
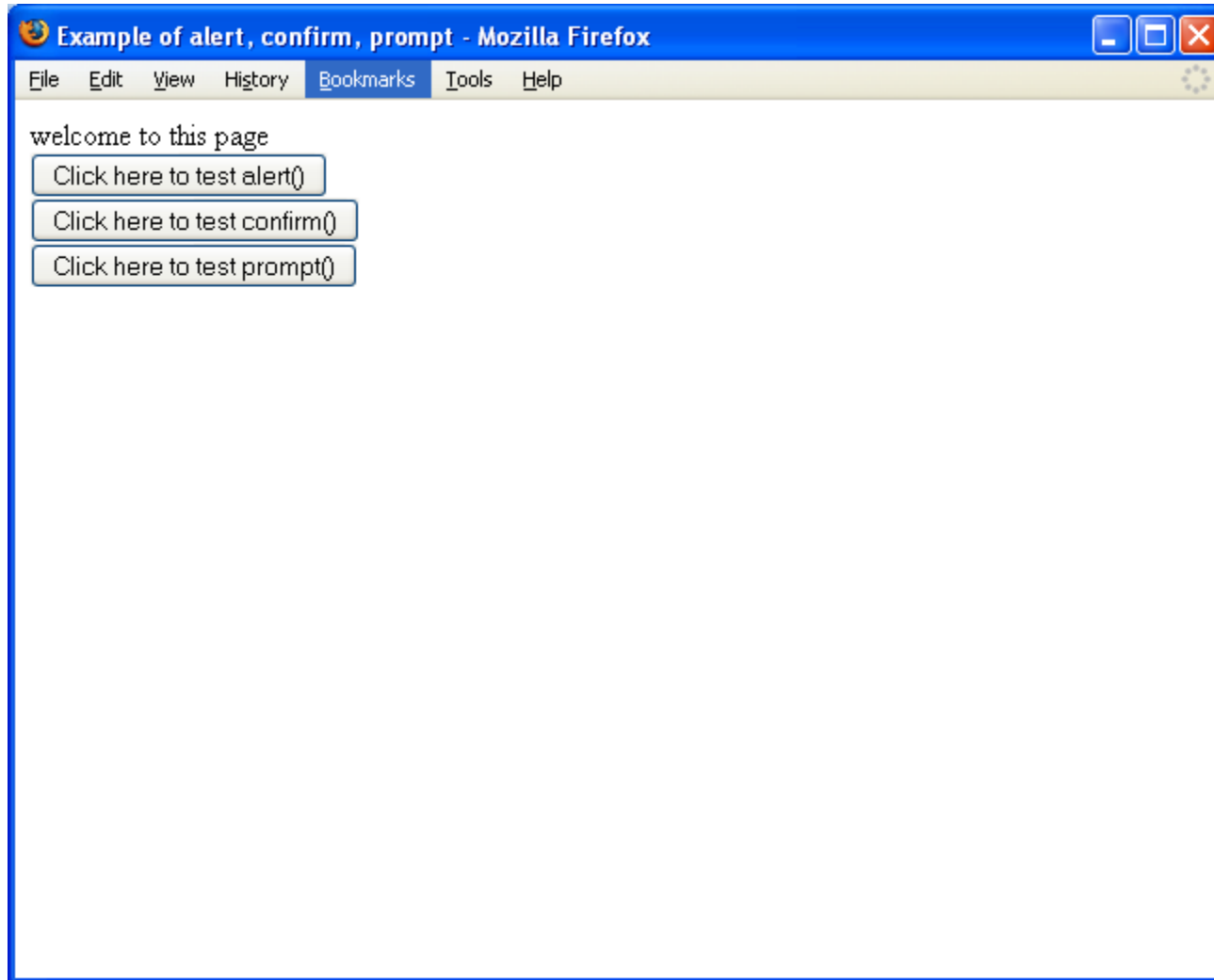
Clicking on confirm()

Clicking on prompt()

Final Thoughts - Common Mistakes

1. Undefined may not be null

- In JavaScript something that has not been assigned to is not null, but undefined. _Undefined_ is different from _null_ when using != = but not when using the weaker != because JavaScript does some implicit casting in the later case
- For details see http://javascript.about.com/od/hintsandtips/a/Null-And-Undefined.htm

2. You cannot overload a function

- If you try to define two different functions with the same name but with different arguments, assuming the proper function will be called when the number of arguments match (this is overloading), then your assumption is incorrect. JavaScript will simply use the latest-defined version of the function and call it;
- If a parameter is omitted it is undefined

3. Undeclared variables are global

- If a variable is NOT declared using var, then it is global. Two variables of the same name, both undeclared will create conflicts that are hard to debug

Netscape JavaScript Versions

Netscape Browser	JavaScript Version	Comments
2.0	1.0	a.k.a. LiveScript
3.0	1.1	Adds images, arrays, applets, plug-ins
4.0-4.05	1.2	More enhancements
4.06-4.7	1.3	
	1.4	Server only
6-7,Firefox 1.0-4	1.5 (1999)	ECMAScript-262 compliant
Firefox 1.5	1.6	ECMAScript-262 Edition 3 compliant
Firefox 2.0	1.7	Generators, iterators and let statement
Firefox 3.0-3.5	1.8-1.8.1	Some ECMAScript 4 updates, JSON codec
Firefox 4.0-6.0	1.8.5 (5)	ECMAScript 5 partial support
Firefox 17-60+	6 - 9	ECMAScript 5.1 - Partial ECMScript 2015 (ES6) - ECMAScript 2018

MSIE JavaScript Versions

MSIE Browser	JScript	Comments
3.X/1	1.0	More or less compatible with NS JavaScript 1.0
3.X/2	2.0	More or less compatible with NS JavaScript 1.1
4.0	3.0	More or less compatible with NS JavaScript 1.2
5.0	5.0	More or less compatible with NS JavaScript 1.5
5.1	5.1	Minor update
5.5	5.5	Minor update
6.0	5.6	ECMA-262 3rd edition
7.0	5.7	ECMA-262 3rd edition + ECMA-327
8.0	5.8	ECMA-262 3rd edition + ECMA-327 + JSON (RFC 4627)
9.0-11.0	10.0	Features from ECMA-262 5th edition (ES 5), .NET
Edge	1.0	ECMAScript 2015 (aka ES6)

ECMAScript

- JavaScript now controlled by the ECMA standard body
- **ECMA** stands for **European Computer Manufacturers Association**
- First language specification, ECMA-262, a.k.a. ECMAScript, approved in 1997, closely resembles Netscape JavaScript 1.1
- Current language specification is **ECMA-262, 10th Edition, June 2019, ECMAScript © 2019**
- ECMA-262 10th Ed. language specification found at:
  - http://www.ecma-international.org/ecma-262/10.0/index.html
  - Microsoft Edge Chakra JavaScript Engine open source:
  - https://blogs.windows.com/msedgedev/2016/01/13/chakracore-now-open/
- Mozilla JavaScript Docs
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript