Lecture

JavaScript – Basics

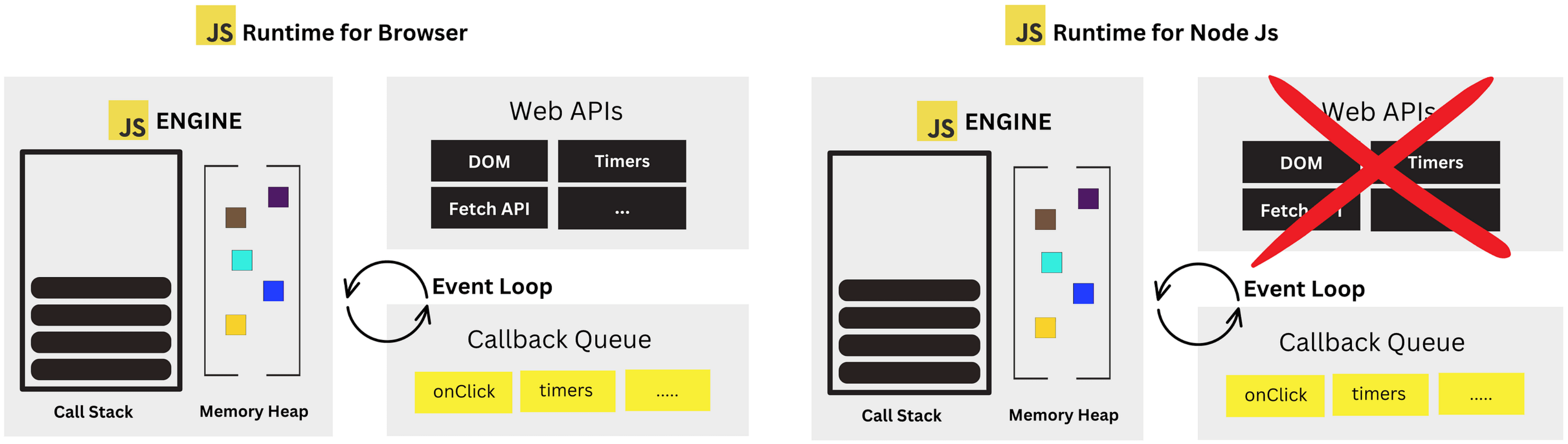
What are We Talking About Today?

- **JavaScript Overview**
  - History of JavaScript
  - Interpreted, versatile, object-oriented language
  - Runs on both client and server sides
- **HTML Integration**
  - Embed JavaScript in <body> or <head>
  - Control web page content and user interactions
- **Debugging Tools**
  - Available in Firefox, Chrome, Safari, Edge
  - Essential for optimizing and debugging code

What is JavaScript

- JavaScript is a "simple", interpreted, programming language with prototype OOP inheritance

- JavaScript Runtime usually consists of
  - JavaScript Engine & Event Loop themselves
  - Web APIs (for browsers to interact with page)



https://srivastavaankita080.medium.com/javascript-engine-runtime-ae9d392c170a

JavaScript Engines

- Typically, JS engine provides the following:
  - JS Interpreter + JIT compiler (-s)
  - Sandboxing
  - WebAssembly support
  - API bridge (to interact with DOM API, OS, etc.)

- There are numerous JavaScript Engines

Engine	Frontend usage	Backend usage
V8	Google Chrome, Chromium, Edge	Node.js, Deno, Couchbase
SpiderMonkey	Mozilla Firefox, Linux GNOME shell	Adobe Acrobat, MongoDB
JavaScriptCore	Apple Safari	Bun
Chakra	MS IE, early Edge	

JavaScript History

- 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
- Language was renamed **LiveScript**, then **JavaScript**
- **LiveWire** → *server-side JavaScript* environment
- **Netscape Enterprise Server / Netscape Application Server** → the actual servers that hosted it
- See "A Short History of JavaScript": https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript

LiveWire vs. Node.js (Server-Side JavaScript)

Dimension	LiveWire (1996-1999)	Node.js (2009-)
Execution model	Embedded in Netscape server	Standalone runtime
JS engine	Interpreted Netscape JS	V8 (JIT compiled)
Concurrency	Thread-per-request	Event loop
I/O model	Blocking	Non-blocking async
Performance	Low throughput	High throughput
Deployment	Proprietary server	OS-level process
Ecosystem	Closed, minimal	npm, open source
Scaling	Vertical	Horizontal
Use cases	HTML-centric apps	APIs, services, realtime

Why Java Killed LiveWire

- Enterprise credibility
  - Strong typing and tooling
  - Backed by Sun, IBM, BEA
- Superior performance
  - True multithreading
  - Better CPU utilization
- Mature server stack
  - Servlets and JSP
  - Later full Java EE
- Perception of JavaScript
  - Seen as a toy language
  - Associated with browser hacks
- Vendor strategy shift
  - Netscape pivoted to Java
  - LiveWire deprioritized

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>
```

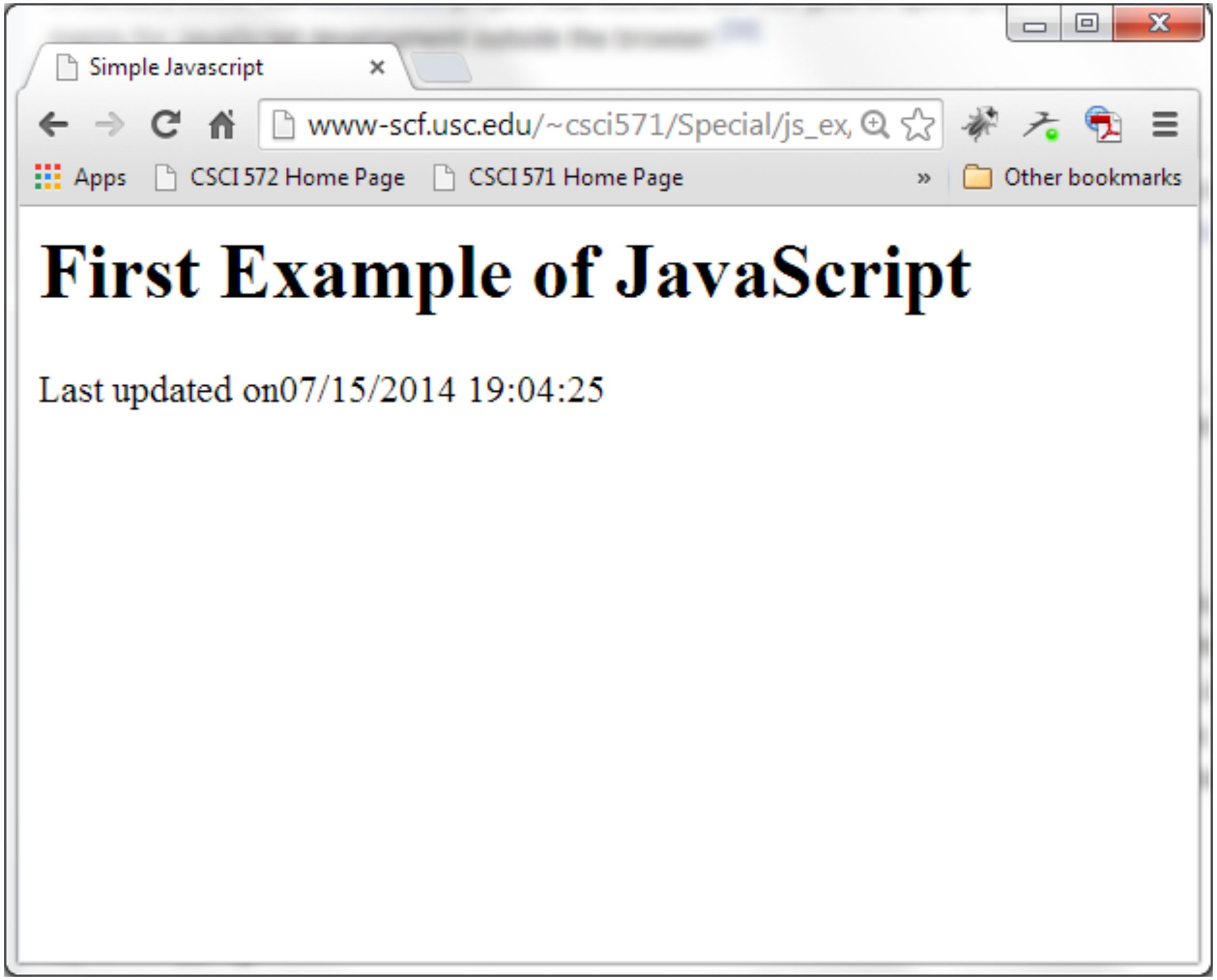
- **or in the <head> as a deferred script**

```
<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



First Example of JavaScript

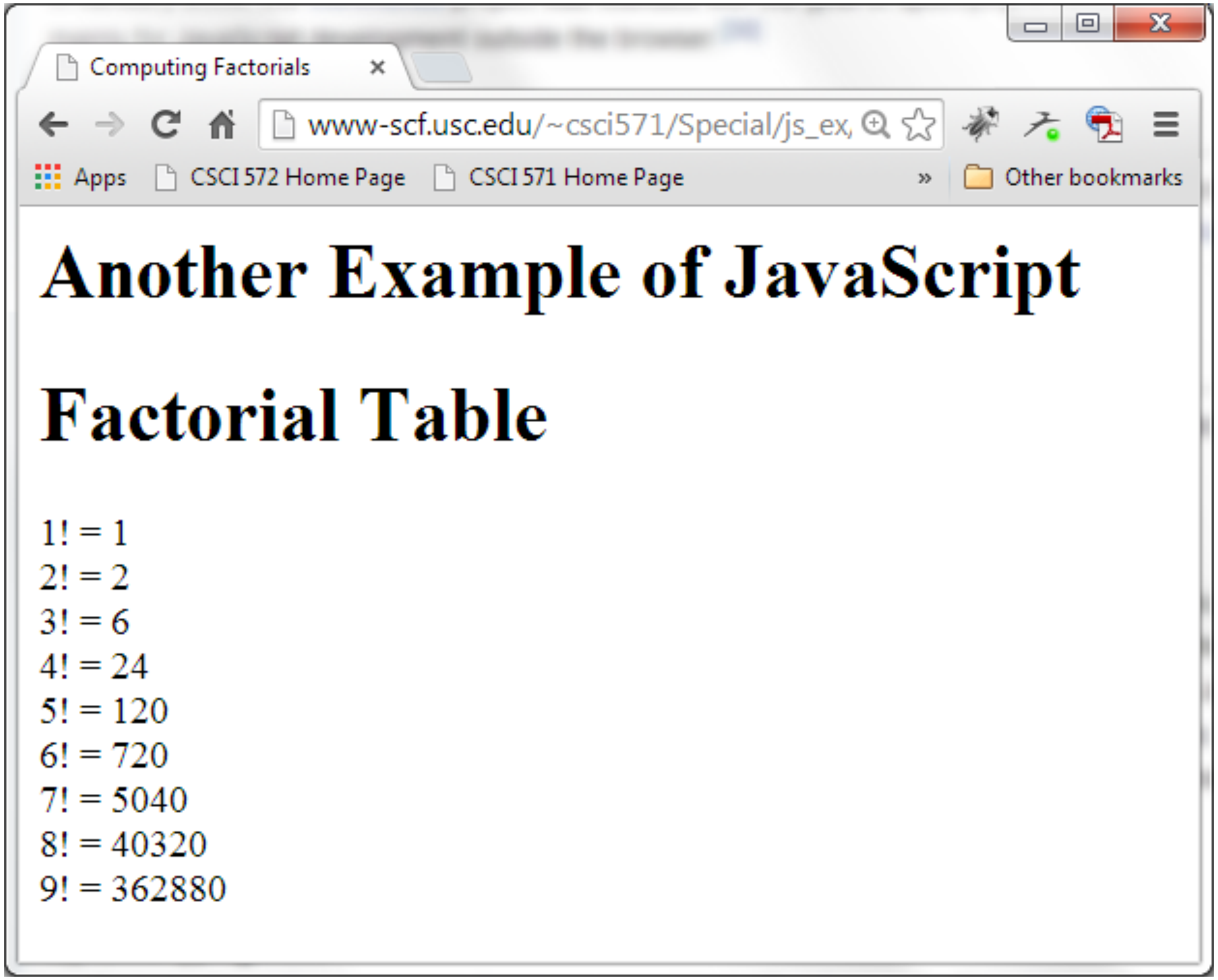
Last updated on07/15/2014 19:04:25

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



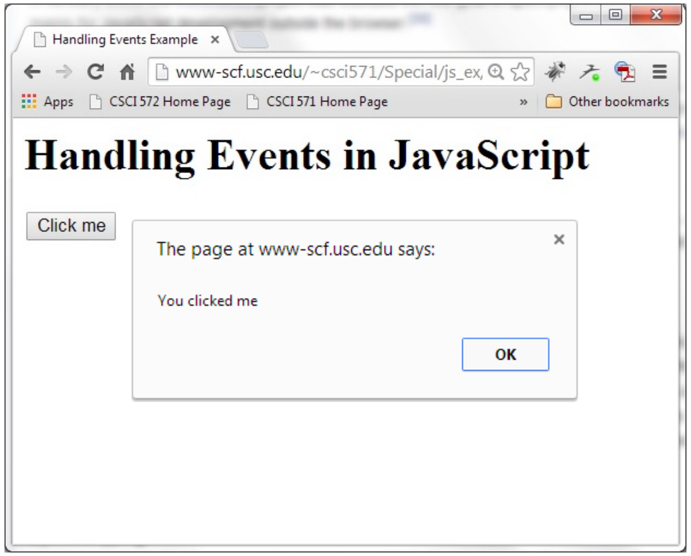
Another Example of JavaScript

Factorial Table

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
```

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>
```



Handling Events in JavaScript

The page at www-scf.usc.edu says:

You clicked me

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

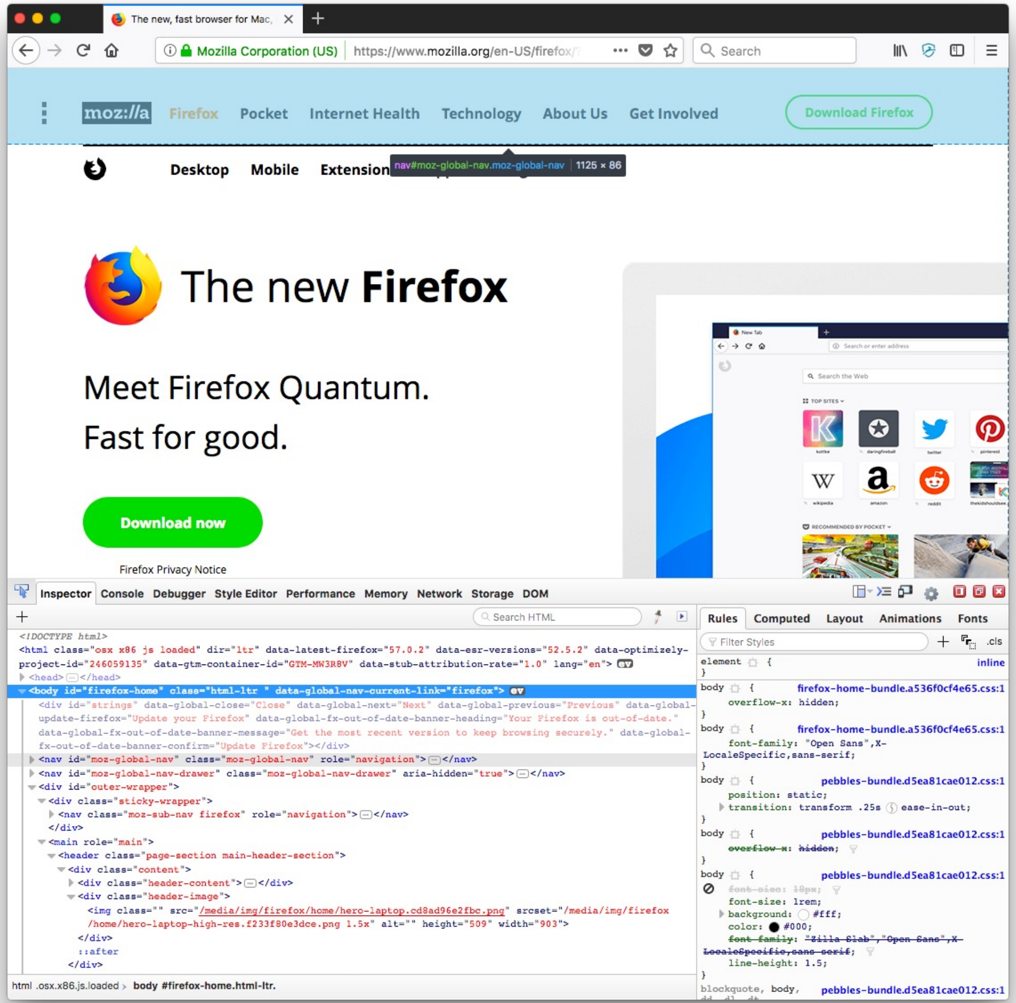
- **Keyboard Events**

– 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



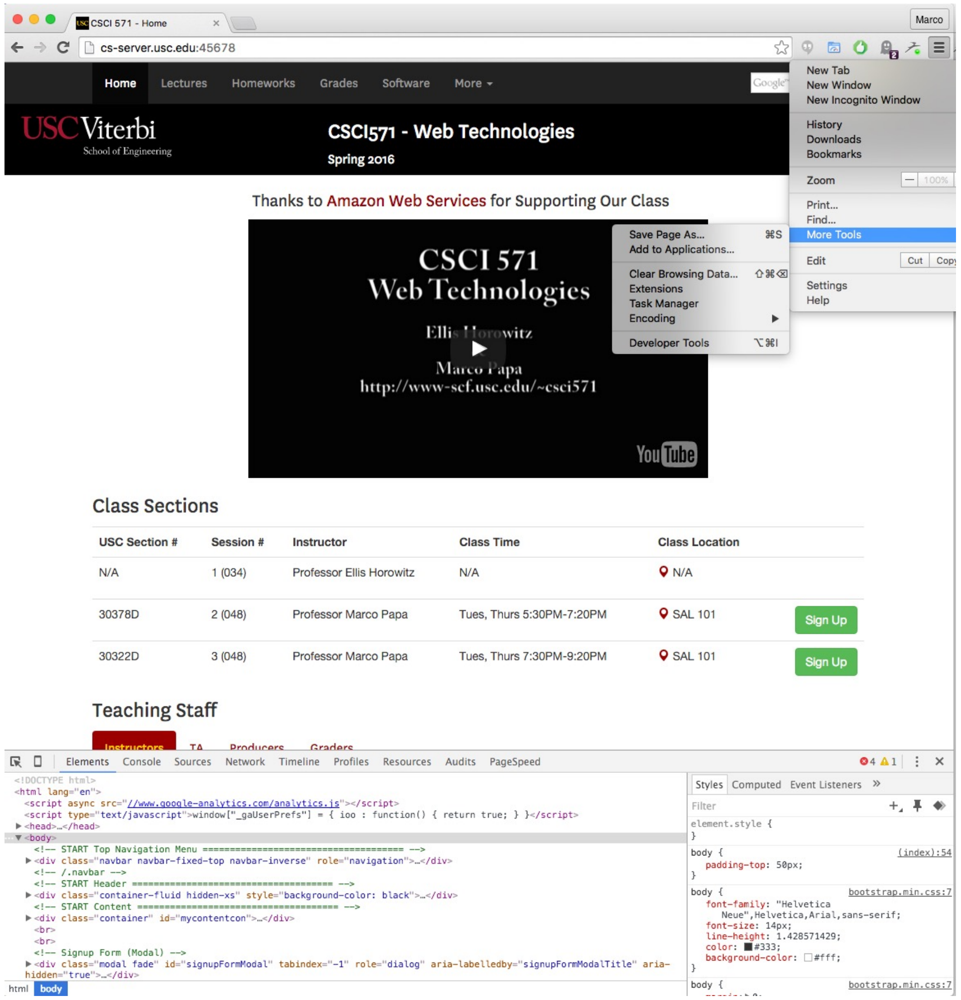
```
<!DOCTYPE html>
<html class="osx x86 js loaded" dir="ltr" data-latest-firefox="57.0.2" data-esr-versions="52.5.2" data-optimizely-
project-id="246059135" data-gtm-container-id="GTM-MW3R8V" data-stub-attribution-rate="1.0" lang="en"> ev
 <head>…</head>
 <body id="firefox-home" class="html-ltr " data-global-nav-current-link="firefox"> ev
   <div id="strings" data-global-close="Close" data-global-next="Next" data-global-previous="Previous" data-global-
   update-firefox="Update your Firefox" data-global-fx-out-of-date-banner-heading="Your Firefox is out-of-date."
   data-global-fx-out-of-date-banner-message="Get the most recent version to keep browsing securely." data-global-
   fx-out-of-date-banner-confirm="Update Firefox"></div>
  <nav id="moz-global-nav" class="moz-global-nav" role="navigation">…</nav>
  <nav id="moz-global-nav-drawer" class="moz-global-nav-drawer" aria-hidden="true">…</nav>
  <div id="outer-wrapper">
   <div class="sticky-wrapper">
    <nav class="moz-sub-nav firefox" role="navigation">…</nav>
   </div>
   <main role="main">
    <header class="page-section main-header-section">
     <div class="content">
      <div class="header-content">…</div>
      <div class="header-image">
        <img class="" src="/media/img/firefox/home/hero-laptop.cd8ad96e2fbc.png" srcset="/media/img/firefox
        /home/hero-laptop-high-res.f233f80e3dce.png 1.5x" alt="" height="509" width="903">
      </div>
      ::after
     </div>
```

html .osx.x86.js.loaded > body #firefox-home.html-ltr.

```
element {                                                  inline
}
body {                      firefox-home-bundle.a536f0cf4e65.css:1
    overflow-x: hidden;
}
body {                      firefox-home-bundle.a536f0cf4e65.css:1
    font-family: "Open Sans",X-
LocaleSpecific,sans-serif;
}
body {                          pebbles-bundle.d5ea81cae012.css:1
    position: static;
    transition: transform .25s ease-in-out;
}
body {                          pebbles-bundle.d5ea81cae012.css:1
    ~~overflow-x: hidden;~~
}
body {                          pebbles-bundle.d5ea81cae012.css:1
    ~~font-size: 18px;~~
    font-size: 1rem;
    background: #fff;
    color: #000;
    ~~font-family: "Zilla Slab","Open Sans",X-~~
~~LocaleSpecific,sans-serif;~~
    line-height: 1.5;
}
blockquote, body,               pebbles-bundle.d5ea81cae012.css:1
```

- Built into Firefox
- To invoke, select Firefox "burger" menu > More tools > Web Developer Tools > Debugger

Debugging JavaScript
Chrome Developer Tools



Class Sections

USC Section #	Session #	Instructor	Class Time	Class Location	
N/A	1 (034)	Professor Ellis Horowitz	N/A	N/A	
30378D	2 (048)	Professor Marco Papa	Tues, Thurs 5:30PM-7:20PM	SAL 101	Sign Up
30322D	3 (048)	Professor Marco Papa	Tues, Thurs 7:30PM-9:20PM	SAL 101	Sign Up

```
<!DOCTYPE html>
<html lang="en">
  <script async src="//www.google-analytics.com/analytics.js"></script>
  <script type="text/javascript">window["_gaUserPrefs"] = { ioo : function() { return true; } }</script>
▶<head>…</head>
▼<body>
    <!-- START Top Navigation Menu ==================================== -->
  ▶<div class="navbar navbar-fixed-top navbar-inverse" role="navigation">…</div>
    <!-- /.navbar -->
    <!-- START Header ====================================== -->
  ▶<div class="container-fluid hidden-xs" style="background-color: black">…</div>
    <!-- START Content ====================================== -->
  ▶<div class="container" id="mycontentcon">…</div>
    <br>
    <br>
    <!-- Signup Form (Modal) -->
  ▶<div class="modal fade" id="signupFormModal" tabindex="-1" role="dialog" aria-labelledby="signupFormModalTitle" aria-
  hidden="true">…</div>
```

```
element.style {
}
body {                                  (index):54
    padding-top: 50px;
}
body {                       bootstrap.min.css:7
    font-family: "Helvetica
       Neue",Helvetica,Arial,sans-serif;
    font-size: 14px;
    line-height: 1.428571429;
    color: ■#333;
    background-color: □#fff;
}
body {                       bootstrap.min.css:7
```

- The Chrome Developer Tools are built into Google Chrome.

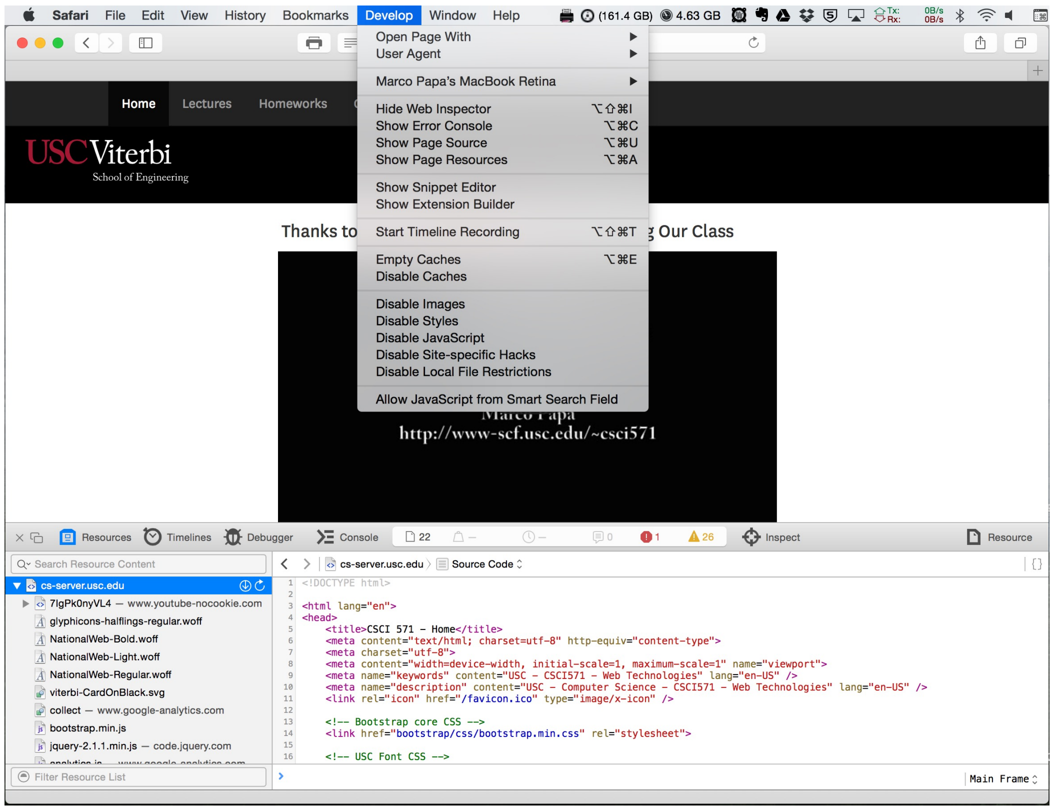
- They provide deep access into the internals of the browser and their web application.

Use the DevTools to

- track down layout issues
- set JavaScript breakpoints
- get insights for code optimization.

- Go to Customize and Control Chrome > More Tools > Developer Tools

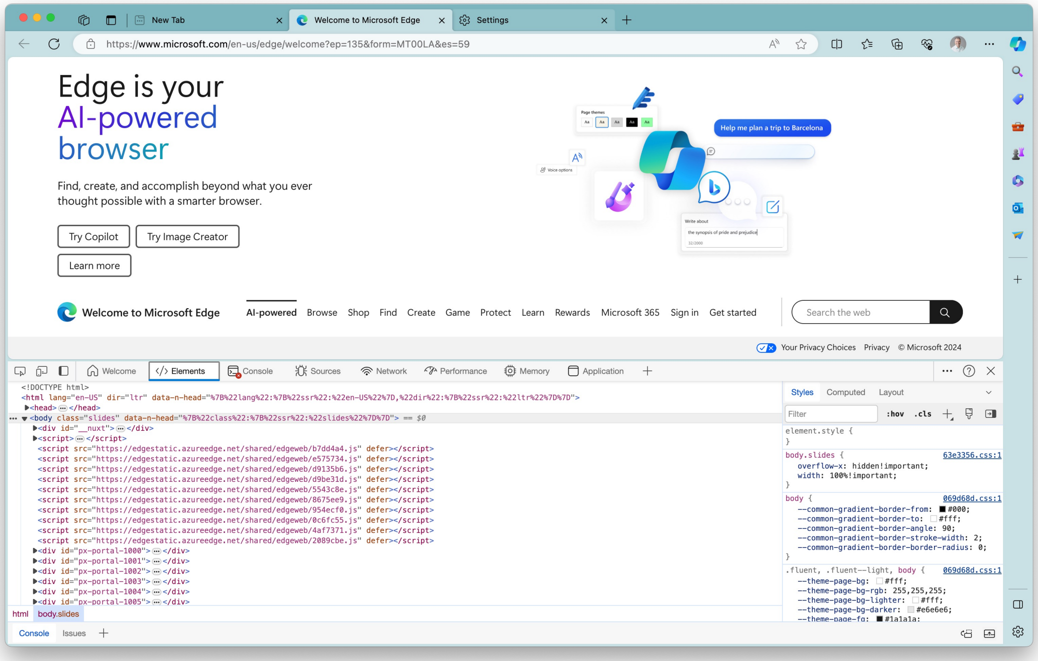
Debugging JavaScript Safari Developer Tools



```
<!DOCTYPE html>

<html lang="en">
<head>
    <title>CSCI 571 – Home</title>
    <meta content="text/html; charset=utf-8" http-equiv="content-type">
    <meta charset="utf-8">
    <meta content="width=device-width, initial-scale=1, maximum-scale=1" name="viewport">
    <meta name="keywords" content="USC – CSCI571 – Web Technologies" lang="en-US" />
    <meta name="description" content="USC – Computer Science – CSCI571 – Web Technologies" lang="en-US" />
    <link rel="icon" href="/favicon.ico" type="image/x-icon" />

    <!-- Bootstrap core CSS -->
    <link href="bootstrap/css/bootstrap.min.css" rel="stylesheet">

    <!-- USC Font CSS -->
```

- 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" > Web Inspector > Sources > Scripts

Debugging JavaScript Edge Developer Tools



Edge is your AI-powered browser

Find, create, and accomplish beyond what you ever thought possible with a smarter browser.

```
<!DOCTYPE html>
<html lang="en-US" dir="ltr" data-n-head="%7B%22lang%22:%7B%22ssr%22:%22en-US%22%7D,%22dir%22:%7B%22ssr%22:%22ltr%22%7D%7D">
 ▶<head> … </head>
… ▼<body class="slides" data-n-head="%7B%22class%22:%7B%22ssr%22:%22slides%22%7D%7D"> == $0
   ▶<div id="__nuxt"> … </div>
   ▶<script> … </script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/b7dd4a4.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/e575734.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/d9135b6.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/d9be31d.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/5543c8e.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/8675ee9.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/954ecf0.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/0c6fc55.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/4af7371.js" defer></script>
    <script src="https://edgestatic.azureedge.net/shared/edgeweb/2089cbe.js" defer></script>
   ▶<div id="px-portal-1000"> … </div>
   ▶<div id="px-portal-1001"> … </div>
   ▶<div id="px-portal-1002"> … </div>
   ▶<div id="px-portal-1003"> … </div>
   ▶<div id="px-portal-1004"> … </div>
   ▶<div id="px-portal-1005"> … </div>
```

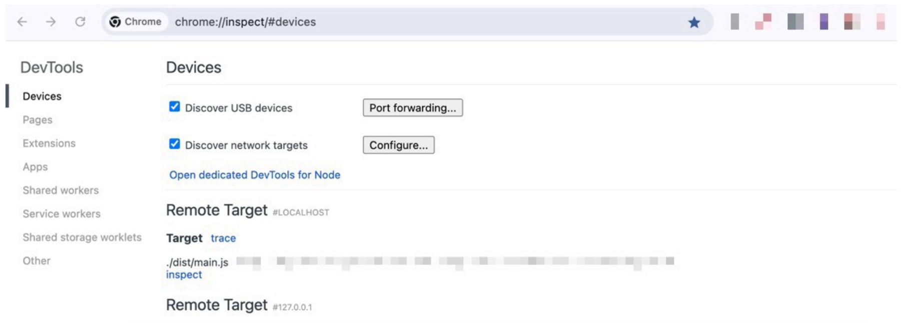
```
element.style {
}
body.slides {                                  63e3356.css:1
    overflow-x: hidden!important;
    width: 100%!important;
}
body {                                         069d68d.css:1
    --common-gradient-border-from: #000;
    --common-gradient-border-to: #fff;
    --common-gradient-border-angle: 90;
    --common-gradient-border-stroke-width: 2;
    --common-gradient-border-border-radius: 0;
}
.fluent, .fluent--light, body {                069d68d.css:1
    --theme-page-bg: #fff;
    --theme-page-bg-rgb: 255,255,255;
    --theme-page-bg-lighter: #fff;
    --theme-page-bg-darker: #e6e6e6;
    --theme-page-fg: #1a1a1a;
```

Select:

- Settings and more -> More Tools -> Developer Tools

Debugging Backend JavaScript

You can usually also use a browser debugger for your backend runtimes that use the same JS engine!



Devices

☑ Discover USB devices — Port forwarding...

☑ Discover network targets — Configure...

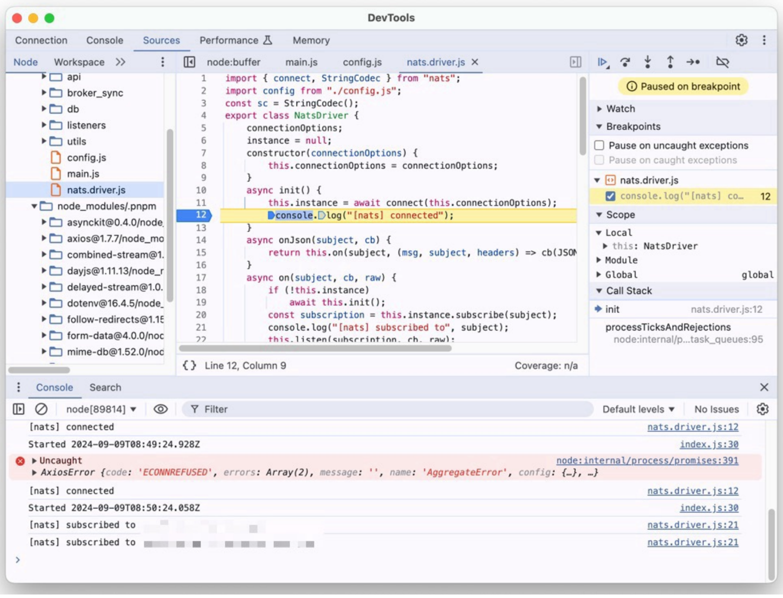
Open dedicated DevTools for Node

Remote Target #LOCALHOST

Target trace

./dist/main.js [redacted]
inspect

Remote Target #127.0.0.1



```
import { connect, StringCodec } from "nats";
import config from "./config.js";
const sc = StringCodec();
export class NatsDriver {
    connectionOptions;
    instance = null;
    constructor(connectionOptions) {
        this.connectionOptions = connectionOptions;
    }
    async init() {
        this.instance = await connect(this.connectionOptions);
        console.log("[nats] connected");
    }
    async onJson(subject, cb) {
        return this.on(subject, (msg, subject, headers) => cb(JSON
    }
    async on(subject, cb, raw) {
        if (!this.instance)
            await this.init();
        const subscription = this.instance.subscribe(subject);
        console.log("[nats] subscribed to", subject);
        this.listen(subscription, cb, raw);
```

```
[nats] connected                                                          nats.driver.js:12
Started 2024-09-09T08:49:24.928Z                                              index.js:30
▸ Uncaught                                          node:internal/process/promises:391
▸ AxiosError {code: 'ECONNREFUSED', errors: Array(2), message: '', name: 'AggregateError', config: {…}, …}
[nats] connected                                                          nats.driver.js:12
Started 2024-09-09T08:50:24.058Z                                              index.js:30
[nats] subscribed to [redacted]                                           nats.driver.js:21
[nats] subscribed to [redacted]                                           nats.driver.js:21
>
```

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 draw graphics.
  - **This has been dramatically improved**
- [**Was**] No access to the underlying file system or operating system
  - **File System API**
  - https://developer.mozilla.org/en-US/docs/Web/API/File_System_API
- [**Was**] Unable to open arbitrary network connections
  - Partially allowed with **CORS**
- [**Was**] No support for multithreading
  - WebWorkers/ServiceWorkers/SharedWorkers
  - WASM (Web Assembly)
- [**Was**] Not suitable for computationally intensive applications
  - **This has been dramatically improved**

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 Primitives

Type	typeof	Object wrapper
null	"object"	N/A
undefined	"undefined"	N/A
boolean	"boolean"	Boolean
number*	"number"	Number
bigint	"bigint"	BigInt
string	"string"	String
symbol	"symbol"	Symbol

Everything else is an Object – Arrays, Functions, Sets, Maps, Promises, Dates, etc.
The language heavily uses prototype inheritance – you can call Object's methods on Arrays, add object key-values on functions, etc…

*Numbers can store both
- floats (64-bit)
- integers (safe from $-(2^{53} - 1)$ to $(2^{53} - 1)$, outside of this range they are represented by floats)

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.

JavaScript String building

- **Concatenation** (less efficient - strings are immutable), one can use <u>single or double quotes</u>.

```
let newWebPage = ""
newWebPage += "<HTML><HEAD>"
newWebPage += "<TITLE>A Sample Page</TITLE></HEAD>"
newWebPage += "<BODY>My Home Page</BODY>"
newWebPage += "</HTML>”
```

- **Template literals** (<u>backticks</u> - preserve formatting, spaces and new lines and can embed variable values)

```
const title = "A Sample Page"
const newWebPage = `<HTML><HEAD>
<TITLE>${title}</TITLE></HEAD>
<BODY>Rendered at ${(new Date()).toDateString()}</BODY>
</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 mbetween 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” ”let” “const” are ***local*** to the function
    - if a keyword 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 Variable Declaration

Var

- pre-ES6 way of variable declaration
- scoped to function when used inside a function, global otherwise
- can be redeclared and updated
- initialized as *undefined* before declaration – exists even before declaration!

Let

- added in ES6
- block-scoped (bounded by {}, etc)
- can be updated, but not redeclared
- reference before declaration throws an error

Const

- same as Let, but
- can not be updated (is a constant)

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

- Arrays could be created from any Iterable using Array.from
- JavaScript does NOT provide a way to declare the size (dimension) of an array, but we can add one

```
Array.dim = (dimension, initial) =>
               Array.from({ length: dimension }, () => initial);
                         // ^ object that mimics an array of size dimension

var myArray = Array.dim(10, 0); //makes an array of ten zeros
```

More on Arrays

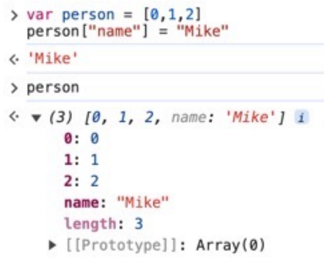
- There are many ways to iterate over an array
  - for loop; **`for (i=0; i < len; i++) {. . . }`**
  - for in loop; **`for (x in person) { . . . }`**
  - while loop; **`while (condition) {. . . }`**
- There are many built-in methods for working with arrays, here are just a few:
  - concat(), joins two 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 *JavaScript Guide*:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

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; however, the following is possible



```
> var person = [0,1,2]
  person["name"] = "Mike"
< 'Mike'
> person
< ▼ (3) [0, 1, 2, name: 'Mike']
      0: 0
      1: 1
      2: 2
      name: "Mike"
      length: 3
    ▶ [[Prototype]]: Array(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 dot notation; and must be accessed using bracket notation.

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"};
```

- Object **properties** are like JavaScript **variables**
- 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) }
}
class Cat {
  constructor(name, meow) { this.name = name; }
  talk() { alert(this.name + " say " + meow) }
}
cat1 = new cat("felix", "purr");
cat1.talk();
cat2 = new Cat("ginger", "hiss");
cat2.talk();
```

- cat() is an object constructor with properties and methods declared inside
- Cat is a class (ES6) with properties and methods declared inside

Predefined JavaScript Objects

- There are a set of **standard built-in 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 11, 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**)**

More predefined JavaScript Objects

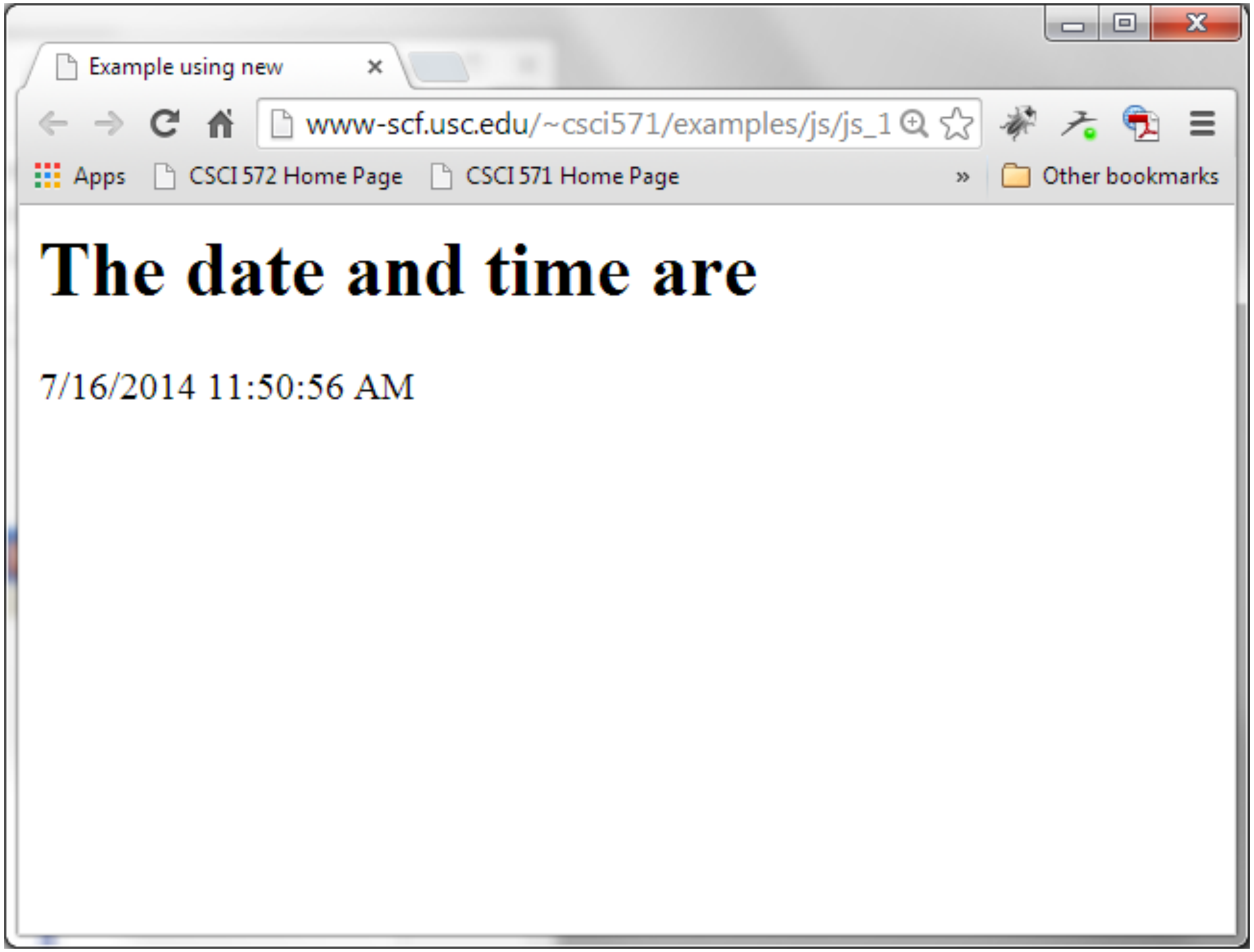
- **Map** (represents HashMap)
- **Set** (represents HashSet)
- **WeakMap / WeakSet** (do not increase GC references)
- **Symbol** (generates unique value for the app lifetime)
- **Promise** (async-await primitive)
- **Proxy**, **Reflect**
- **Atomics**
- **ArrayBuffer** (byte arrays), **DataView**, **SharedArrayBuffer**
- **TypedArray**
  - Uint8Array, Int8Array, Uint16Array, Int16Array, Uint32Array, Int32Array
  - Uint8ClampedArray
  - Float16Array, Float32Array, Float64Array
  - BigUint64Array, BigInt64Array
- **FinalizationRegistry** (GC callbacks)

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



The date and time are

```
7/16/2014 11:50:56 AM
```

Functions

- Couple way to define functions:
  - Function declaration

```
function square(number) {
  return number * number;
}
console.log(square(2)); // 4
```

  - Function expression

```
const cube = function (number) {
  return number * number * number;
};
console.log(cube(2)); // 8
```

  - Arrow function (aka lambda – uses parent context)

```
const squareArrow = (number) => number * number;
console.log(squareArrow(2)); // 4
```

- Functions can take unknown number of arguments

```
const max = (...args) => { // args – array of arguments };
```

- Functions can take less aguments, rest will be undefined
- Functions can be passed to other fucntions (aka callbacks)

JavaScript Popup Boxes

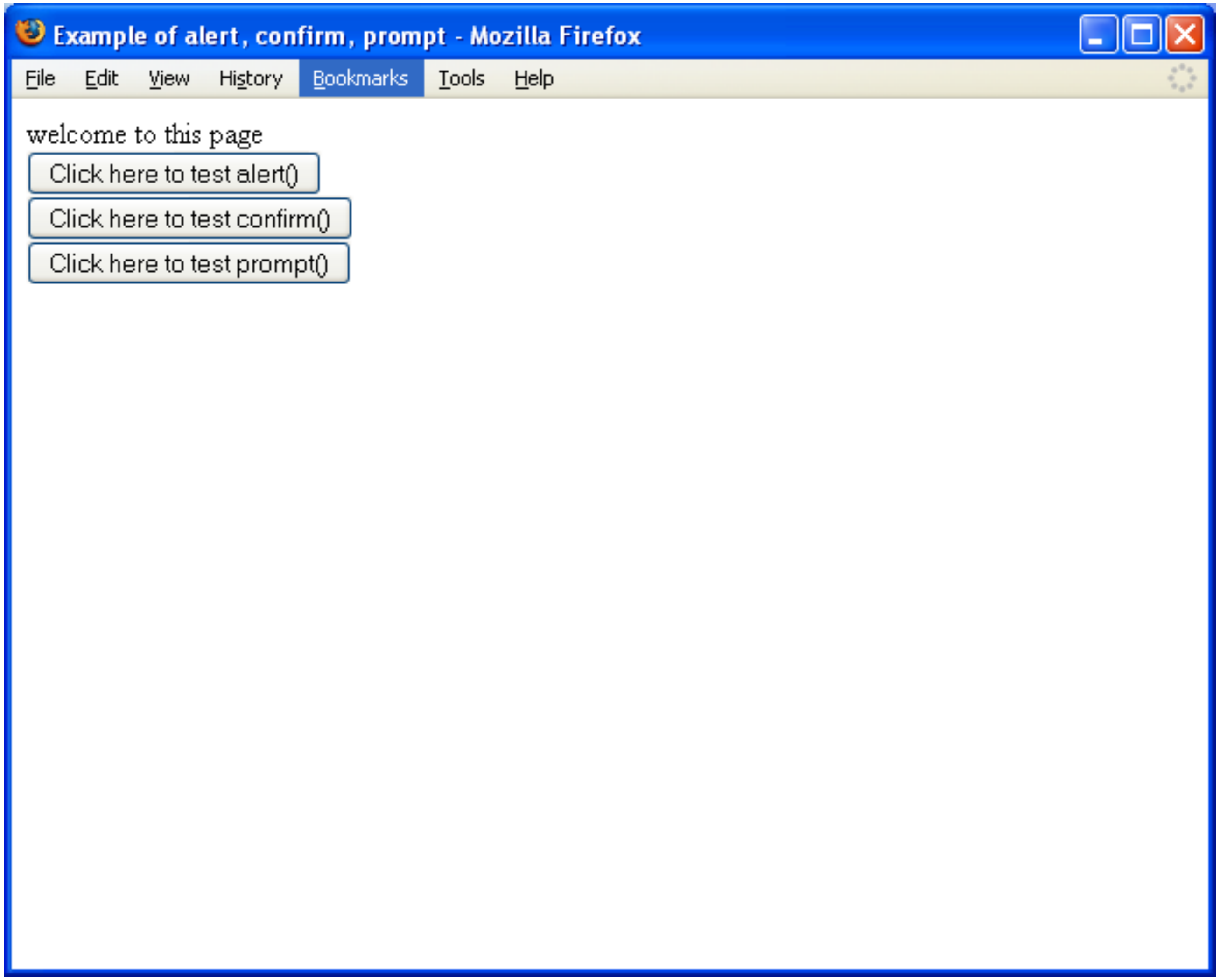
alert(), confirm(), and prompt()

```
<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>
```

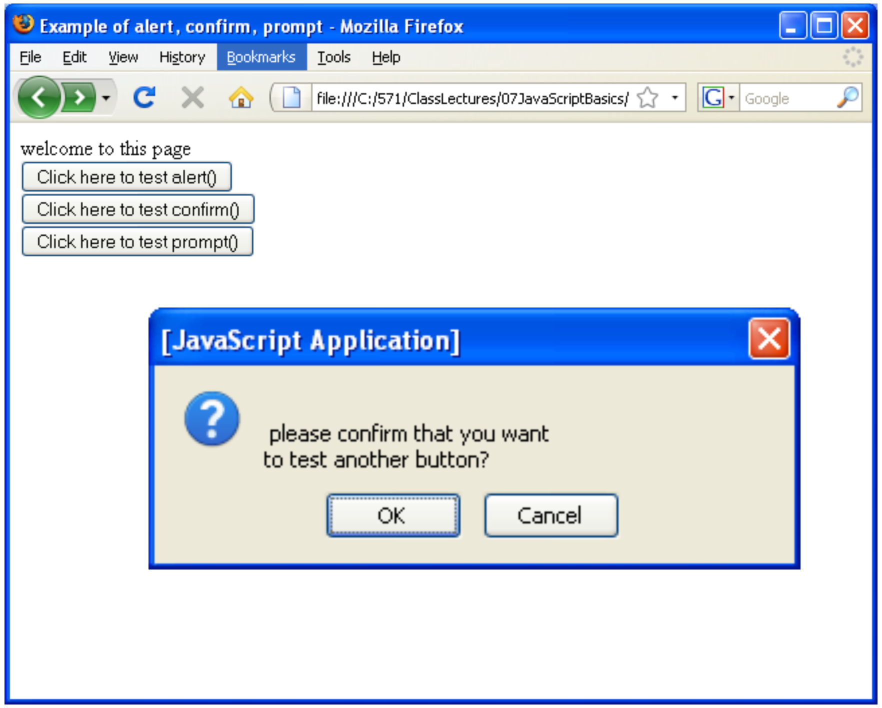
Example 5: Browser Output


Example of alert, confirm, prompt - Mozilla Firefox
File Edit View History Bookmarks Tools Help

welcome to this page

Click here to test alert()
Click here to test confirm()
Click here to test prompt()

Clicking on confirm()


Example of alert, confirm, prompt - Mozilla Firefox

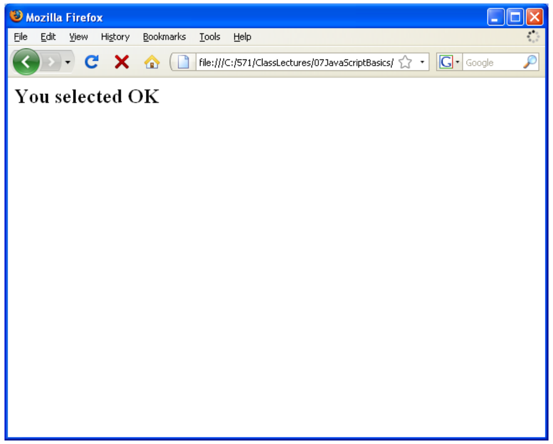
file:///C:/571/ClassLectures/07JavaScriptBasics/

welcome to this page

Click here to test alert()
Click here to test confirm()
Click here to test prompt()

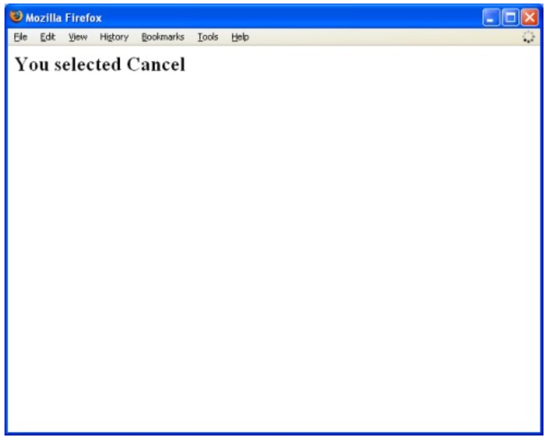
[JavaScript Application]

please confirm that you want
to test another button?


Mozilla Firefox

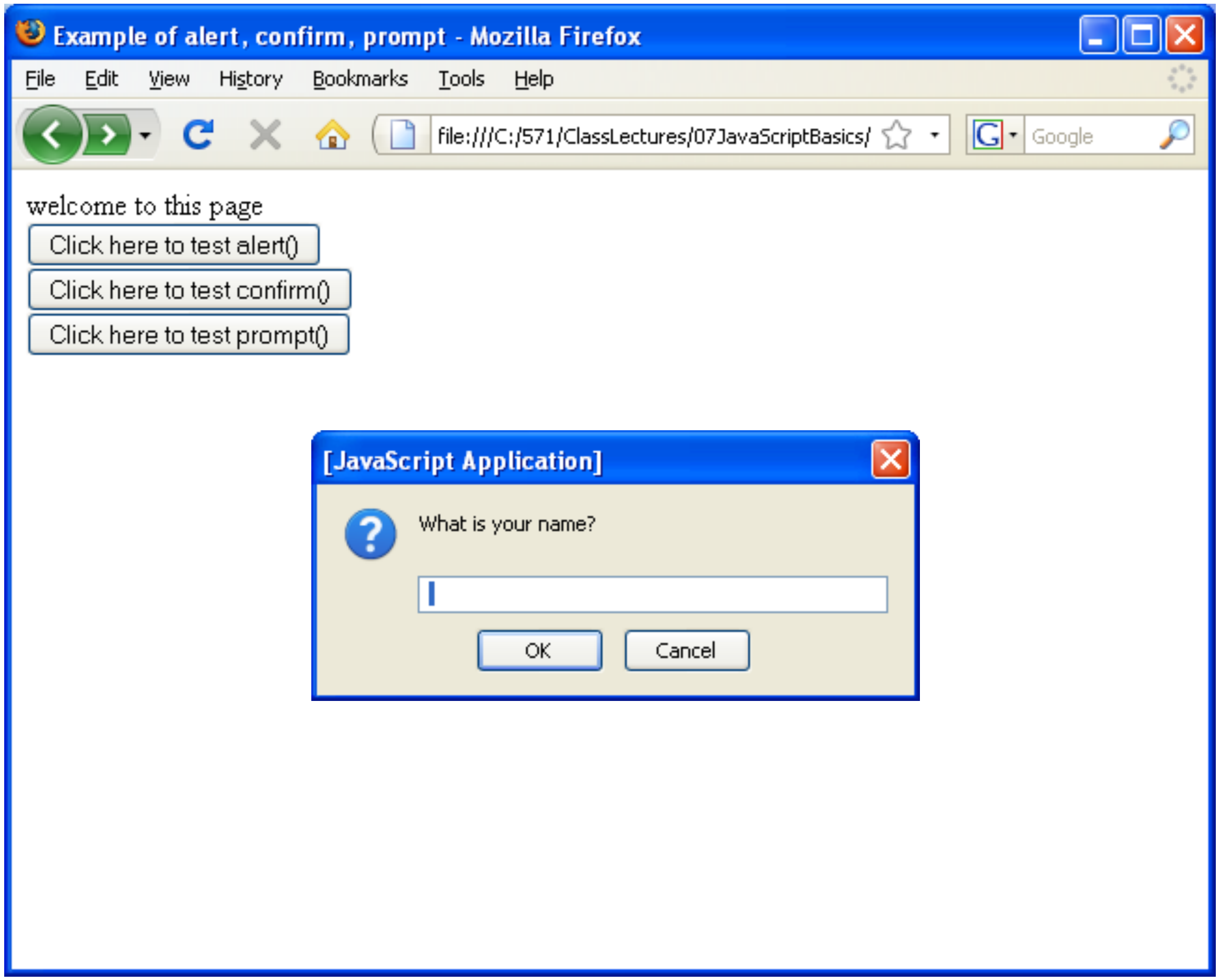
file:///C:/571/ClassLectures/07JavaScriptBasics/

You selected OK


Mozilla Firefox

You selected Cancel

Clicking on prompt()


Example of alert, confirm, prompt - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/571/ClassLectures/07JavaScriptBasics/
Google

welcome to this page

Click here to test alert()
Click here to test confirm()
Click here to test prompt()

[JavaScript Application]

What is your name?

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/let/const**, 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 17-140+	6 – N/A	ECMAScript 5.1 – ECMScript 2015 (**ES6**) – ECMAScript 2018

MS IE/Edge 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 (**ES6**)
Edge	current	Google V8 JS engine

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, 15th Edition, June 2024, ECMAScript © 2024**
- ECMA-262 15th Ed. language specification found at:
  - https://www.ecma-international.org/publications-and-standards/standards/ecma-262/
- Mozilla JavaScript Docs
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript

Helpful Links to Play with

- As usual you can use **CodePen, JSFiddle, JS Bin** etc. to learn JavaScript

  https://codepen.io

  https://jsfiddle.net/

  https://jsbin.com/
- “JavaScript vs Other Popular Languages of the 21st Century”

  https://dzone.com/articles/javascript-vs-other-popular-languages-of-21st-cent
- “JavaScript 20 Years Old Today”

  https://www.i-programmer.info/news/167-javascript/8595-javascript-20-years-old-today.html
- Free software at “wtfjs” ☺

  https://wtfjs.com/
- JavaScript WTF

  https://javascriptwtf.com/