dotAstronomy 7 Day Zero: JavaScript Tutorial

Demitri Muna • Ohio State University
@demitrimuna

3 November 2015

GitHub Repository

This repository contains this presentation and a few sample HTML files.

```
git clone git@github.com:demitri/dotastro7dayzero.git
```

https://github.com/demitri/dotastro7dayzero

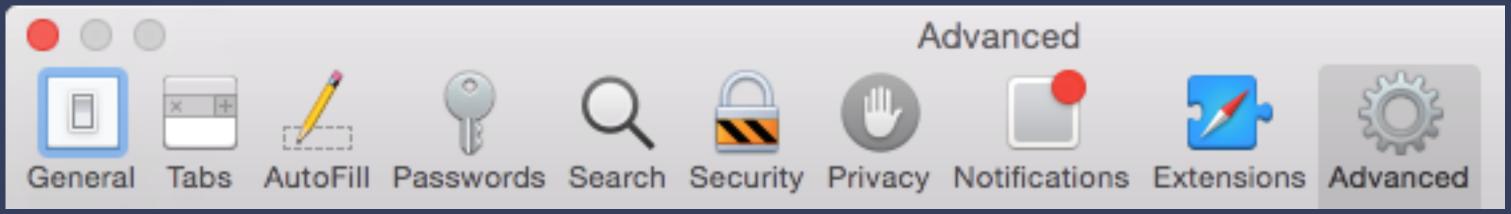
What is JavaScript?

- Computer language (interpreted, like Python)
- Allows for user interaction without returning to server
- Most typical use has code embedded (or called from) web pages
- Standalone language, command line, server side (see: Node.js)

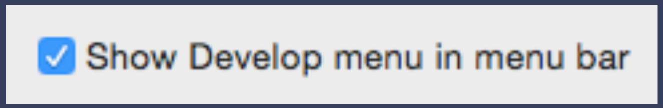
The Browser Console

We'll use JavaScript in the browser. But where's the command line?

Safari

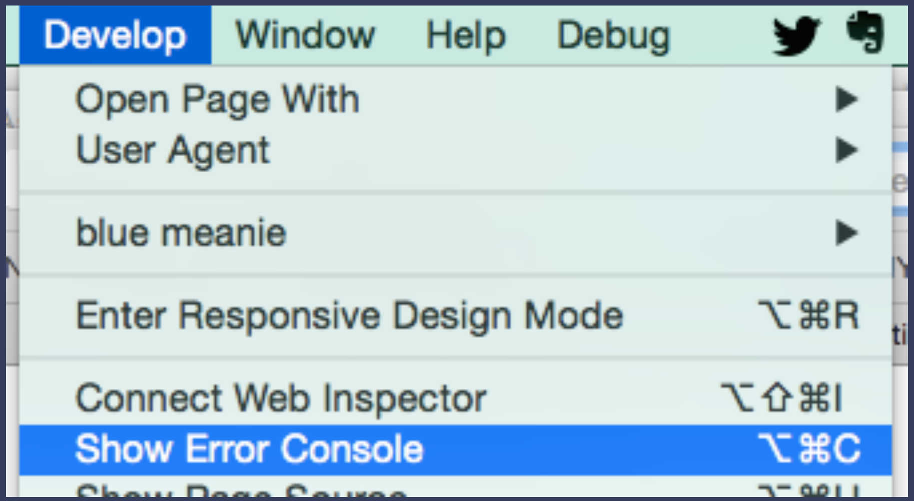


1



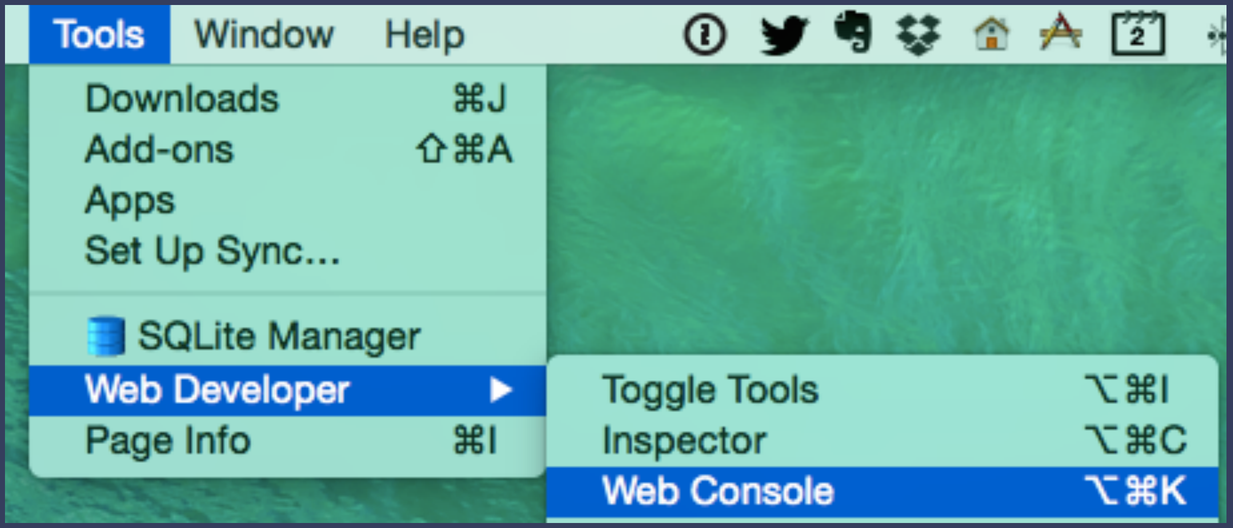
☑ Show Develop menu in menu bar

2

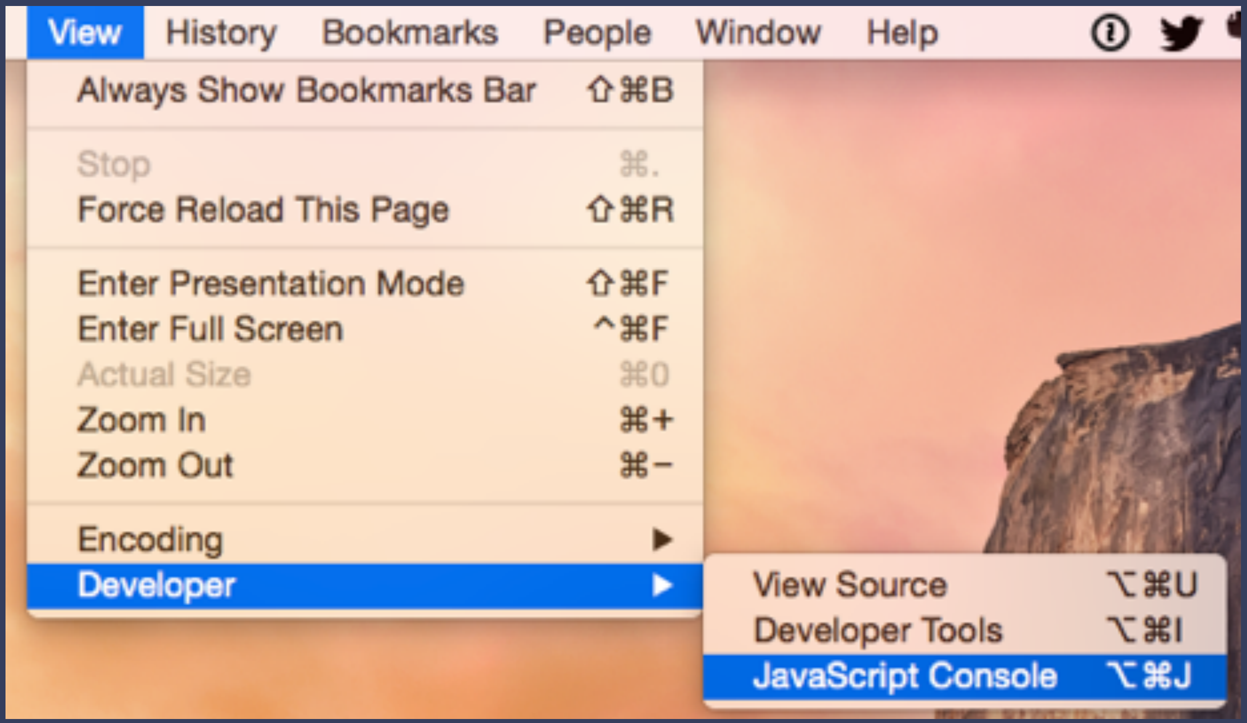


Firefox

(open a new window first)



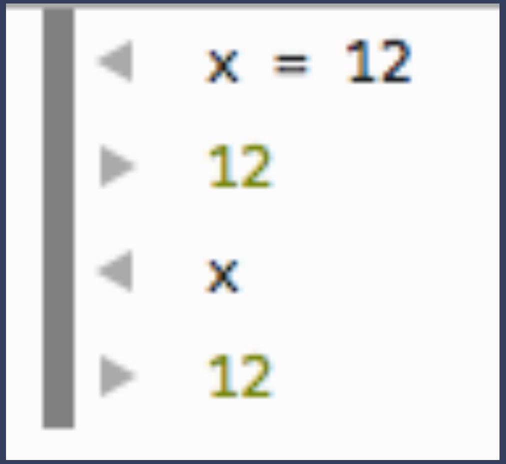
Chrome



The Browser Console

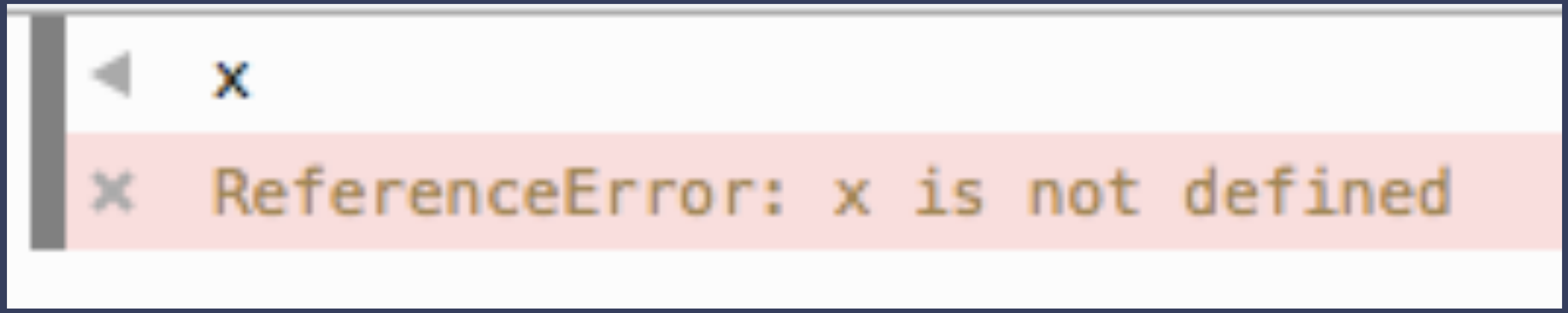
JavaScript commands can be entered directly into the console as you would in the Python interactive interpreter.

Anything defined is local to the particular web page (window) you are working in – a new window is a new environment.



```
x = 12
12
x
12
```

define, print new variable



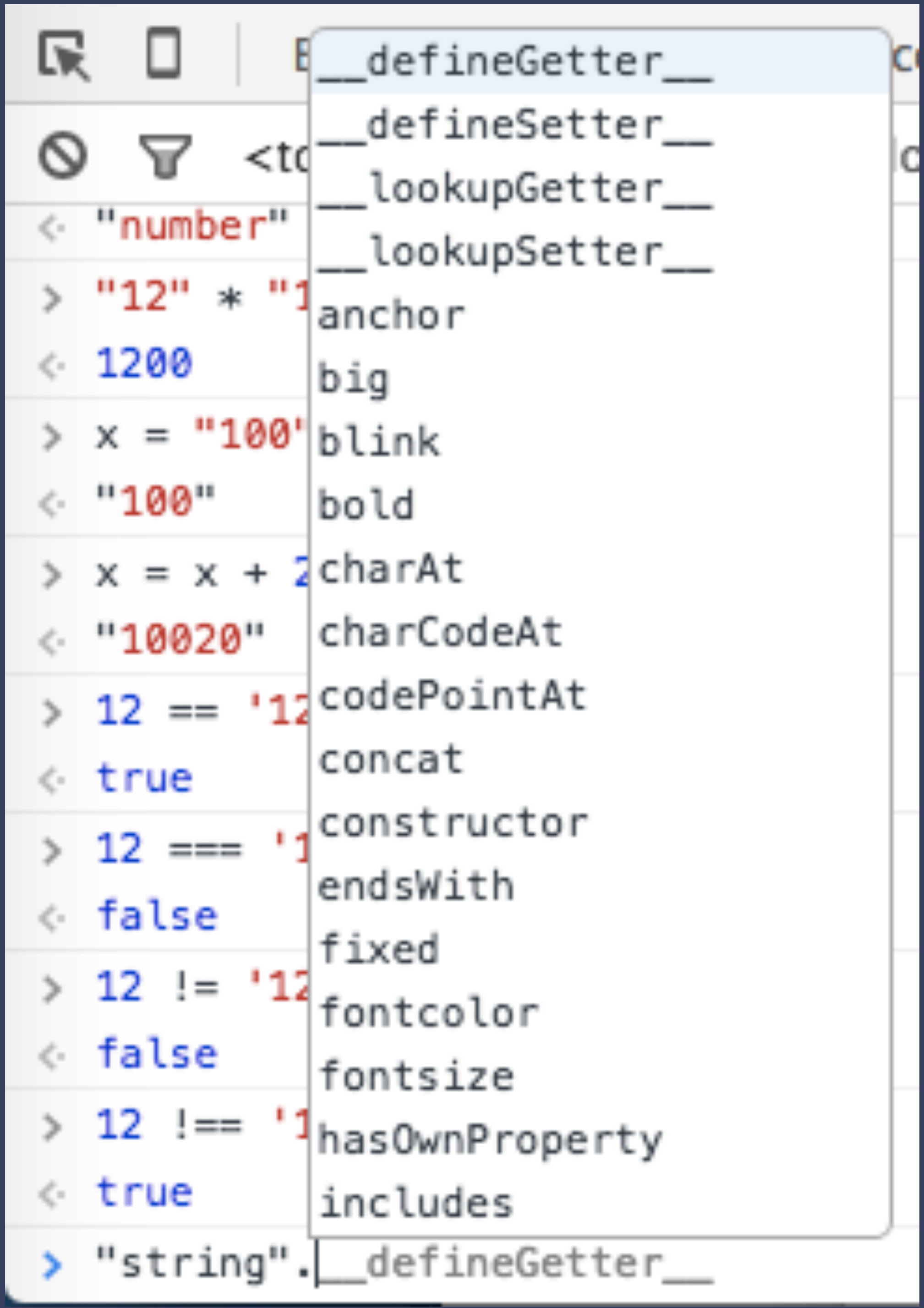
```
x
ReferenceError: x is not defined
```

another window

Keep a browser window with the console open to try out commands. The console is always the first place to look for warnings or errors.

JavaScript Console Autocomplete

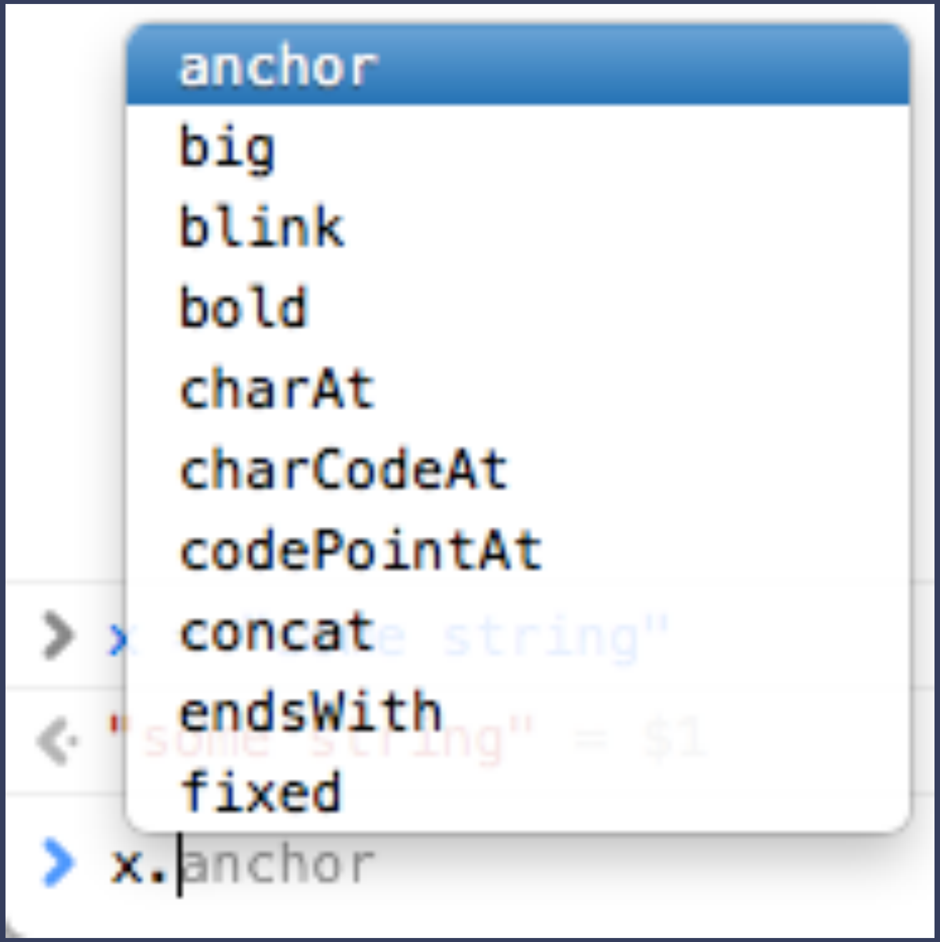
Chrome



```
> "12" * "1
< 1200
> x = "100"
< "100"
> x = x + 2
< "10020"
> 12 == '12
< true
> 12 === '1
< false
> 12 != '12
< false
> 12 !== '1
< true
> "string".__defineGetter__
```

```
__defineGetter__
__defineSetter__
__lookupGetter__
__lookupSetter__
anchor
big
blink
bold
charAt
charCodeAt
codePointAt
concat
constructor
endsWith
fixed
fontcolor
fontsize
hasOwnProperty
includes
```

Safari



```
anchor
big
blink
bold
charAt
charCodeAt
codePointAt
concat
endsWith
fixed
```

```
> x.anchor
```

Trigger autocomplete with the <tab> key to view properties.

JavaScript Basics

Comments begin with "**`//`**".

End all commands with semicolons. Technically this is optional, but it can get you into trouble as the compiler can make (silent!) mistakes.

Use the "**`console.log`**" command to print values to the console (invaluable for debugging!).

```
// this is a comment in JavaScript

// equivalent to print statement in Python
console.log(x);
```

JavaScript Variable Scope

```
// create variables as needed
x = 12; // this is global

function foo() {
    return x; // valid! x defined above
}

// use 'var' to make a variable local
function foo() {
    var x = 42;
    return x; // returns 42, not 12
}

// careful!
function badFoo() {
    x = 99; // new global variable declared!
    return x;
}
```

Variables are global by default. Get into the habit of using **`var`** for everything.

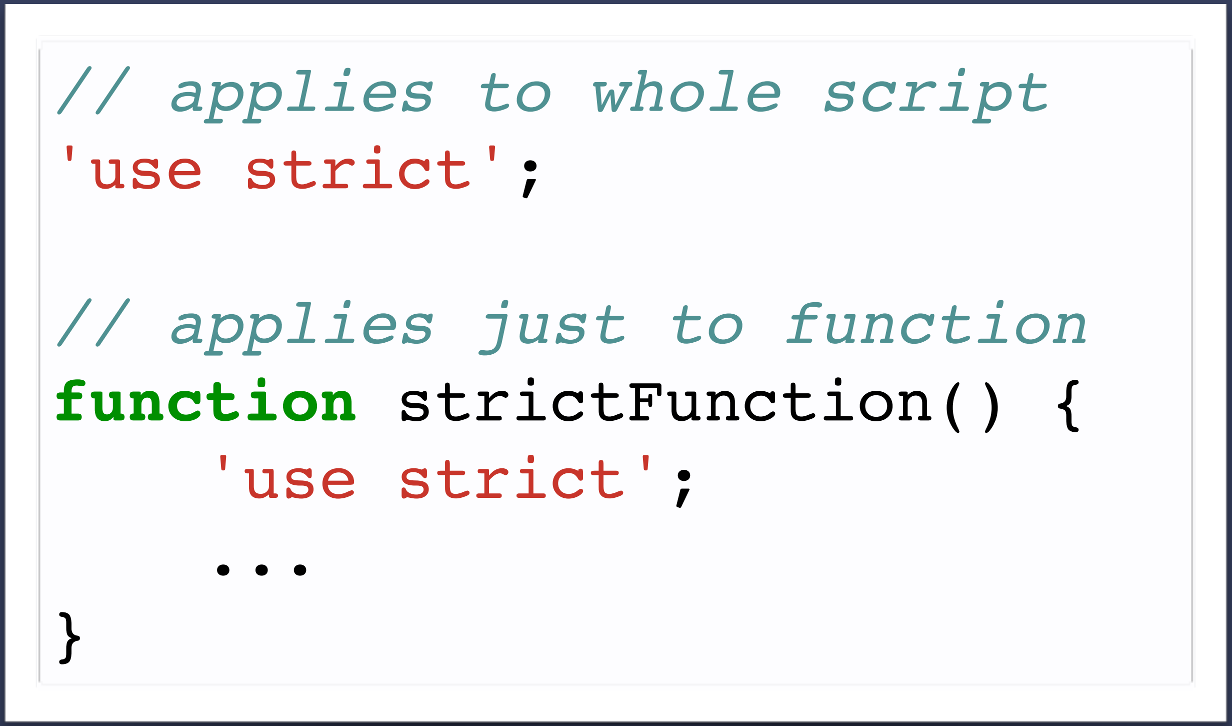
Variable is local to *function* wherever it's called, not brackets. Javascript does not scope to brackets.

http://stackoverflow.com/questions/500431/what-is-the-scope-of-variables-in-javascript

JavaScript Variables

First character of a variable name can be any Unicode letter, ‘$’, or ‘_’. However, browser command lines may not necessarily support all languages, so stick with English letters.

Note that ‘$’ is used to represent the JQuery library, so possibly better to avoid it’s use as well.



```
// applies to whole script
'use strict';

// applies just to function
function strictFunction() {
    'use strict';
    ...
}
```

JavaScript has a “strict” mode that fixes past sins. For example, it forces you to explicitly declare all variables. Variables must also be declared at the top of functions. Recommend to always use.

JavaScript Data Types

JavaScript has only six data types:

Undefined, Null, Boolean, String, Number, Object

```
var x; // x is "undefined"
console.log(x === undefined) // prints true

var empty = null; // 'null' value
console.log(x === null) // prints true

var indiepopIsAwesome = true; // boolean

var name = 'Zaphod'; // single or
var name = "Arthur"; // double quotes

var answer = 42; // all numbers are double,
                 // 64-bit floats

// Use "typeof" to determine variable type
// on console
> typeof name
```

These values are *primitives*. Primitive values are *immutable*.

Everything else is an “object”.

JavaScript Objects

Most everything in JavaScript is an “object”. You can think of an object as a dictionary with key/pairs. The value can also be an object… which could be a function! Consequently, functions can be passed around as variables.

```
var fruitColors = {apple:"red", banana:"yellow",
                   grape:"purple", orange:"orange"};

// Similar to a Python dictionary. Note that the
// properties (Python: keys) are bare, not strings.

// Use dot notation to access values:
> console.log(fruitColors.apple, fruitColors.grape)

a = [1,2,3,4,5]
console.log(a.length) // length is a property (number)

> a.toString
"1,2,3,4,5"

> typeof a.toString // function toString() { [native code] }
```

JavaScript Objects

New properties can be added to an object just by assigning a value to it (Python works in the same way).

```
var fruitColors = {apple:"red", banana:"yellow",
                   grape:"purple", orange:"orange"};

// add new property
fruitColors.grapefruit = 'pink'

// create an empty object - these are equivalent
a = {}
b = new Object()

// accessing individual elements
fruitColors.apple     // dot notation when you know the key
fruitColors["apple"] // key as string
```

JavaScript Loops

```
// array
for (var i=0; i < a.length-1; i++) {
    // don't forget the "var"!
    console.log(a[i]);
}

// dictionary (i.e. Object)
for (var key in fruitColors) {
    // note we can't use the dot notation here
    console.log(fruitColors[key]);
}
```

Probably a good idea not to use spaces in property names…

JavaScript Type Coercion

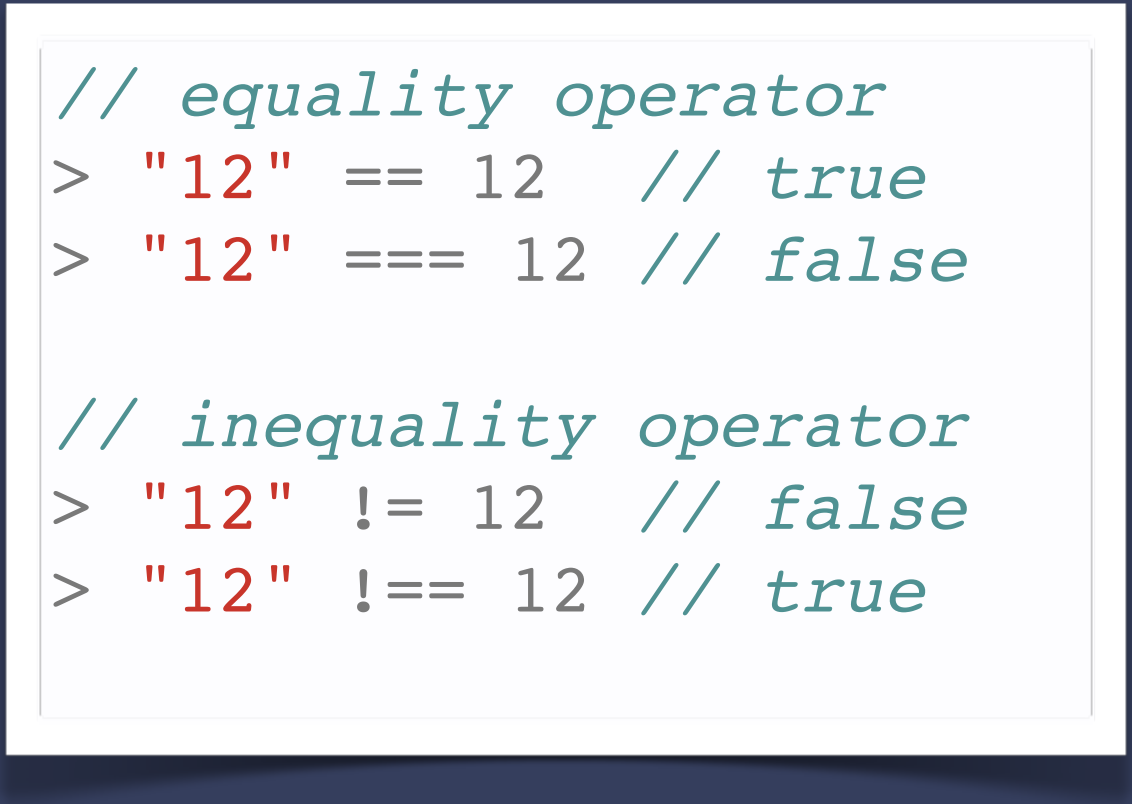
JavaScript will silently convert data types as needed. Be careful of this – this can hide bugs!

```javascript
a = '42' * '100'; // valid, result is 42000
> typeof a // number

a = 50 + 'ways to leave your lover'
console.log(a)
> "50 ways to leave your lover"

// Example:
x = "100"       // e.g. you got this from a web form
x = x + 50      // add 50 to that value
> console.log(x)
10050           // probably not what was expected

// coerce values to be safe
x = Number("100")
```

JavaScript Type Coercion

How could this possibly go wrong?

There is a "==" operator; *never* use it.



```
// equality operator
> "12" == 12   // true
> "12" === 12 // false

// inequality operator
> "12" != 12   // false
> "12" !== 12 // true
```

JavaScript Odds & Ends

Boolean operators

```
// Boolean operators
x || y    // OR
x && y    // AND
!x        // NOT
```

JavaScript functions

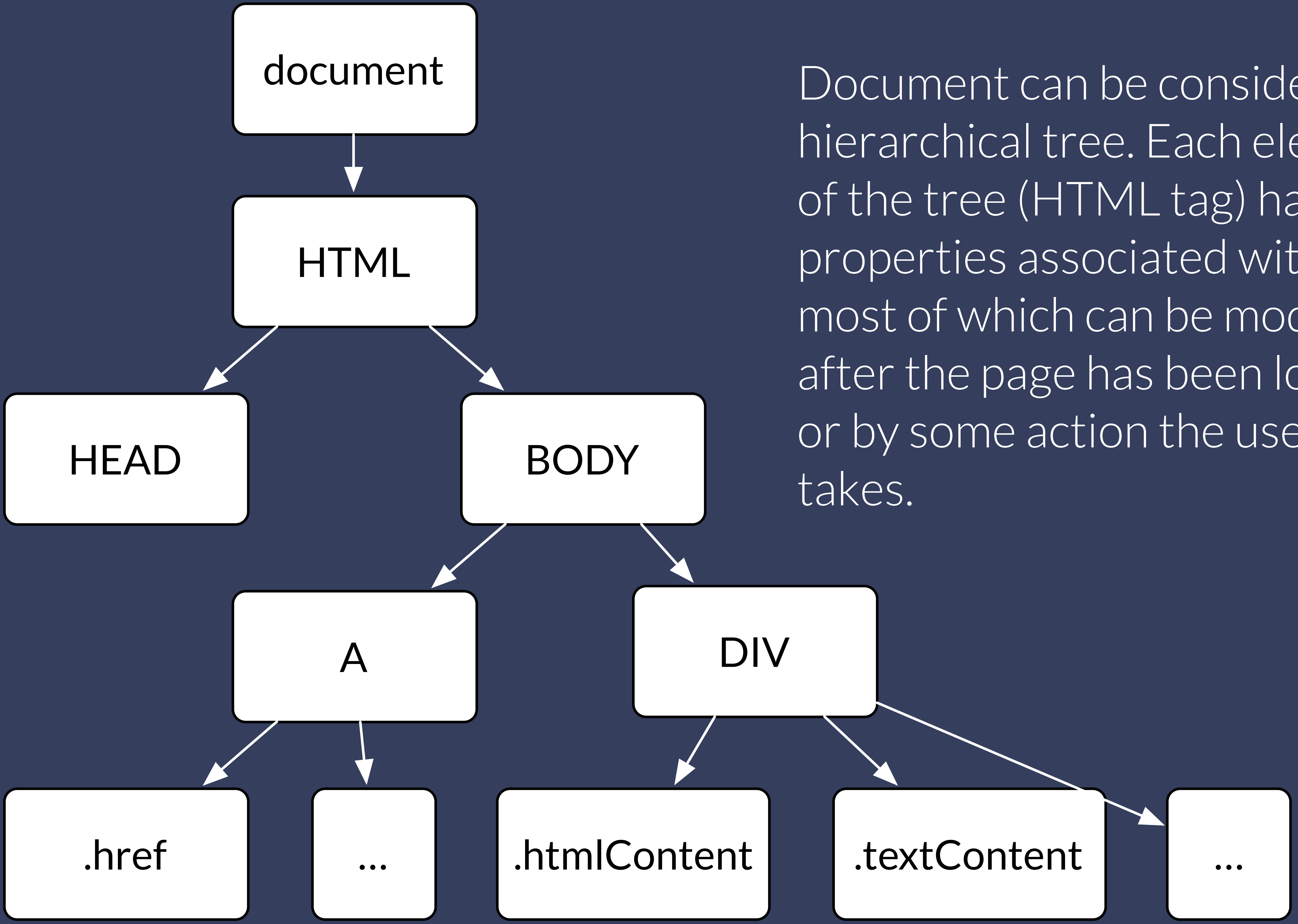
```
function foo(x) {
    // use to set a default value
    x = x || "set x to this if x was not defined"
}
```

HTML

Consider an extremely simple HTML page.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- content of head goes here -->
</head>

<body>
   <!-- content of body goes here -->
    <a href="http://dotastronomy.com">
        Welcome to Day Zero!
    </a>
    <div>Text in div.</div>
</body>

<html>
```

Document Object Model (DOM)



Document can be considered a hierarchical tree. Each element of the tree (HTML tag) has properties associated with it, most of which can be modified after the page has been loaded or by some action the user takes.

Where is JavaScript Code Placed?

The JavaScript code can be added in one of two places:

- inline in the HTML page
- called from an external file

Which option you use is up to you. Small pieces of code that are specific to a page might be best to place on the page itself, but anything that will be re-used by other pages you write (or others!) should be put in an external file.

Good coding practice always suggests to separate your user interface (view) from your code (controller).

External JavaScript Code

Loading an external JavaScript library from a remote source:

```
<script src="//code.jquery.com/jquery-2.1.4.min.js"></script>
```

Loading an external JavaScript library from a local file:

```
<script type="text/javascript" src="myscript.js"></script>
```

Place these lines in the HEAD tag if you need the routines before the page loads, or right before the close of the BODY tag if you can wait until after the pages loads.

Inline JavaScript Code

JavaScript in an HTML page looks like this:

```
<script type="text/javascript">
    // JavaScript code goes here
    // This is the style for comments in this block
    console.log("Hello world!");
</script>
```

We will place this right before the closing BODY tag.

Load the page "dayzero_2.html", and open the console in your browser. You will see the message above indicating the code was executed!

JavaScript Events

We probably don't want all of our code executed when we load the page, but instead to be called when the user triggers some event. An event can be clicking on a link or some text, moving the mouse over the cell of a table or an image, resizing the web page, etc. There are many!

When we want an event to trigger some action, it's best to put that action in a function.

Exercise: Change the background color of our **`<div>`** element when the mouse is over the text.

Element IDs

To access an element on the web page (in the DOM), it's easiest if we give the element a unique ID and then ask the DOM for the object using that ID.

```
<div id="mydivID">Text in div.</div>
```

Get the HTML element as a variable using the ID reference:

```
<script type="text/javascript">
    myDiv = document.getElementById("mydivID");
</script>
```

Changing Properties via Events

```
<script type="text/javascript">
    myDiv = document.getElementById("mydivID");
    myDiv.onmouseover = function changeBackground() {
        myDiv.style.backgroundColor = 'blue';
    };
    myDiv.onmouseout = function changeBackgroundBack() {
        myDiv.style.backgroundColor = 'white';
    };
</script>
```

First, we get a reference to the DIV. Functions can then be attached to actions that the element responds to; here, "**`onmouseover`**" and "**`onmouseout`**". When these events happen, we change the property "**`backgroundColor`**" of the DIV.

Open "dayzero_3.html" to see this in action.

Create New Elements

It's a new way to think of web pages, but consider HTML elements as variables that you can modify and create. Their properties (color, style, etc.) can be modified as you like, as well as location on the page, etc.

Exercise: Attach a function to a button to create a link inside an already existing **`<div>`** tag.

Create New Elements

Let’s do this directly from the console this time.

Open the page "dayzero_4.html", then open the console.

```
// get the div
theDiv = document.getElementById("mydivID");

// create a new HTML element, here a link
newLink = document.createElement('a'); // produces <a></a>
newLink.href = 'http://the-toast.net/2015/10/06/two-monks/';
newLink.textContent = "Two Monks Discover How Tall Women And
Horses Are"; // text between tags

theDiv.appendChild(newLink);
```

The file "dayzero_5.html" implements the same as an inline script.

Where To Go From Here

- JQuery - Most commonly used JavaScript library. Useful enough to always include in your code. Smooths the differences between browsers.
- CSS - Means of defining the display style of HTML elements (e.g. color, borders, etc.).
- StackOverFlow/Google - Web development has a *huge* community (of course). Search for sample code before inventing your own.
- Emscripten - Compile C/C++/Fortran code into a JavaScript library.