JavaScript

Discover the JavaScript Language

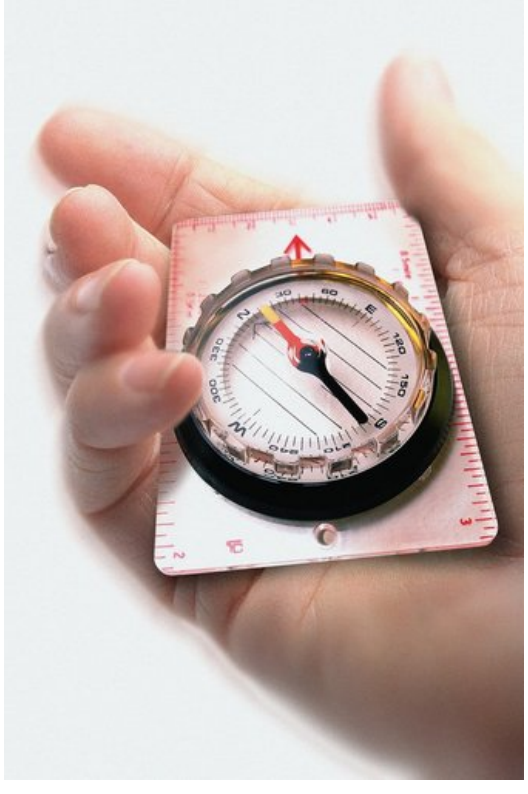
Course objectives

By completing this course, you will be able to:

- Explain origins of JavaScript
- Describe how it works
- Explain its utility
- Develop basic JavaScript procedures
- Manipulate the DOM with JavaScript

JavaScript

Course plan



- Presentation
- Basics notions
- Functions & Scope
- Events
- DOM Interactions
- Object Modeling

Discover JavaScript language

INTRODUCTION

Presentation

- JavaScript is a scripting language
- Design by **Brendan Eich** (Netscape) in 1995
- Inspired by many languages, including **Java** and **Python**

Presentation

- At first, server side language called **LiveScript**

- Then, client side version called **JavaScript**
  - Partnership between Sun Microsystems and Netscape about the name
  - "JavaScript" was a trademark of **Sun Microsystems** and now **Oracle Corp**

Presentation

- Client side Interpreted by web browser
  - Different from PHP

- Complementary to HTML and CSS
  - Add dynamism!
    - User Interactions
    - Animations
    - Navigation Help

Presentation

- As CSS, JavaScript code can be defined in:
  - HTML code
  - a separate script file (.js)

```
<script type="text/javascript">

 /* ... Some code ... */

</script>
```

Presentation

- Based on **events**:
  - onload
  - onfocus
  - onclick
  - ondblclick
  - onabort
  - onerror
  - onmouseover
  - ....
- Associated to DOM objects:

  window, document, forms, ...

Presentation

- Not a classical **OOP** language
  - Prototype-based
- No real concept of class
  - "Pseudo-classes" can be written like **collections of key/value pairs**
- Includes most of class-based OOP features

Introduction

History

1994

1995

1996

Introduction

History

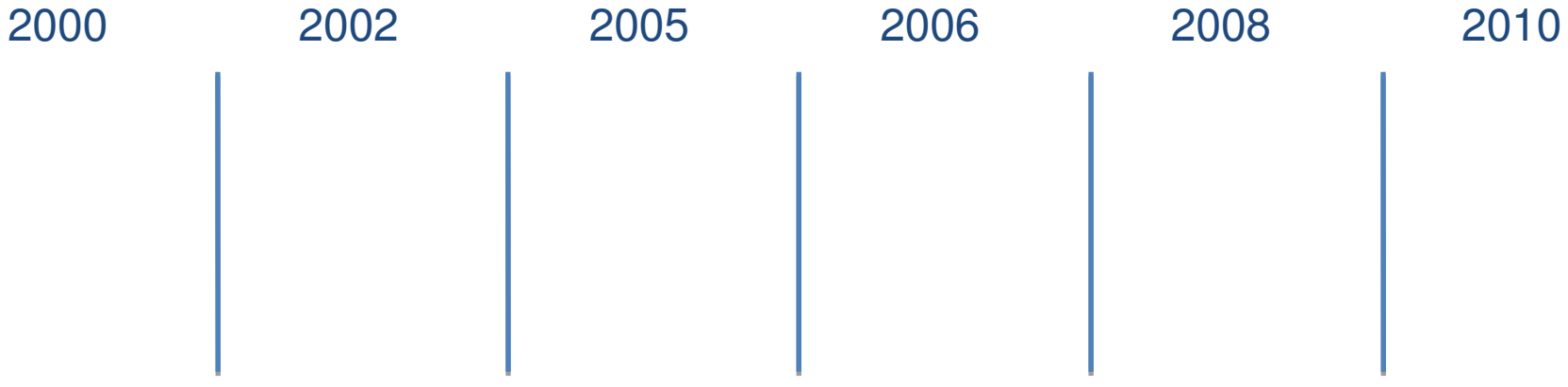
1996

1997

1999

Introduction

History



Libraries

- Many libraries :
  - jQuery
  - Ext JS
  - Prototype
  - Dojo
  - Etc…

jQuery
write less, do more.

prototype

dōjō

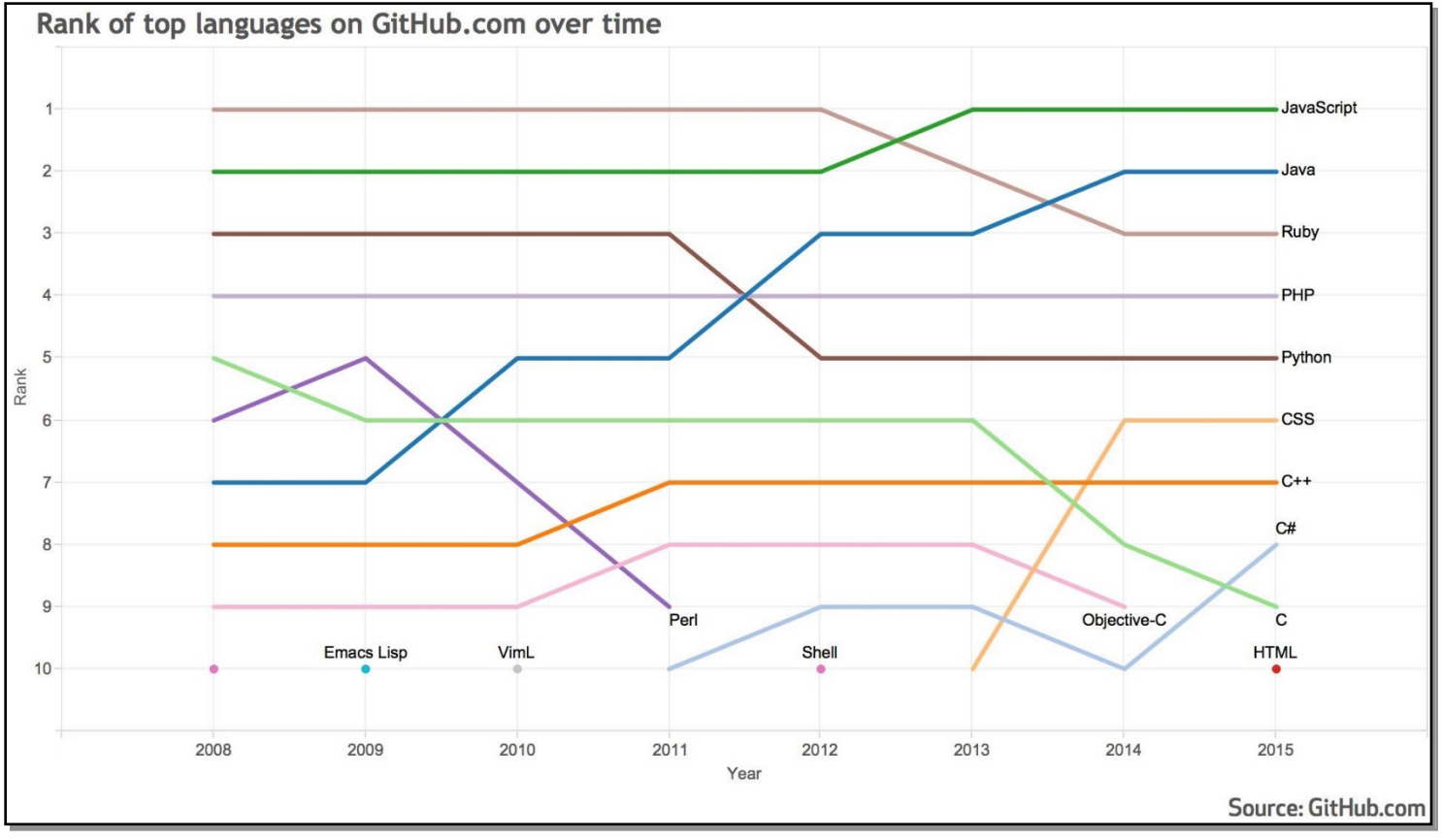
Introduction

Community

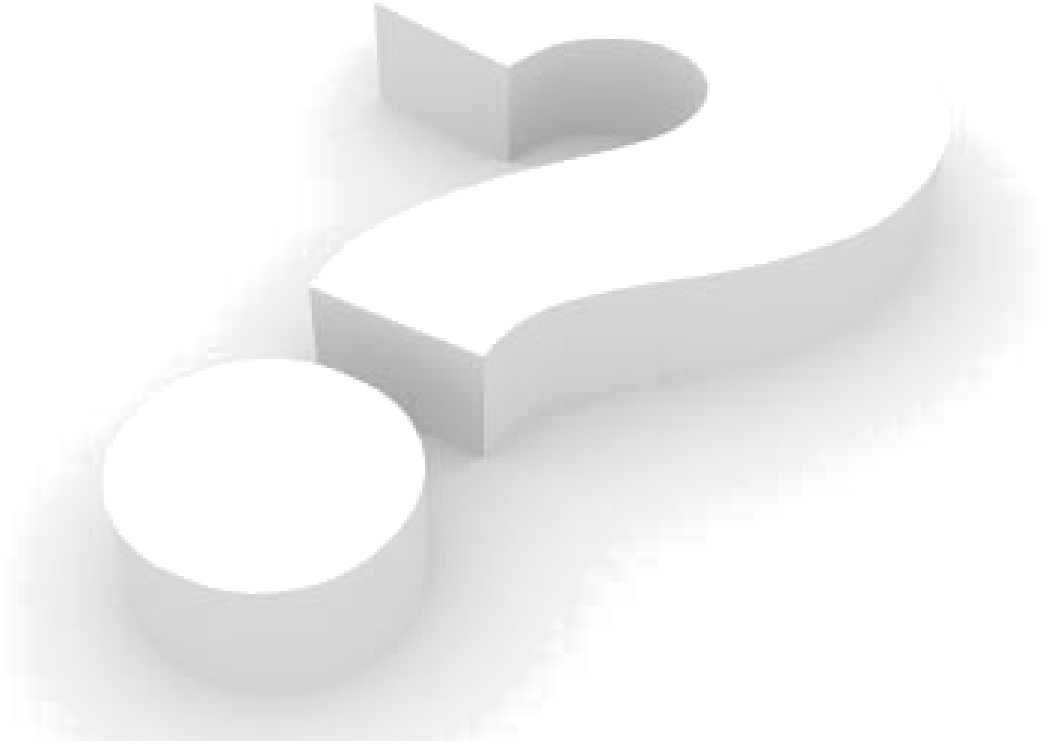
atl(js);

Introduction

Community



Questions ?



Discover JavaScript language

BASIC NOTIONS

Hello World!

- Hello world example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <script type="text/javascript">
      alert('Hello world!');
    </script>
  </body>
</html>
```

Implementation of JavaScript

- Two places to declare JavaScript code:
  - In HTML file directly between <script> markers:

```
<script type="text/javascript">
    var name = "Grima";
</script>
```

  - In an external script file (better most of the time):

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

Syntax rules and basics

- An instruction ends up properly with a semicolon:

```
<script type="text/javascript">
   Instruction_1;
   Instruction_2; Instruction_3;
</script>
```

- Possible to put on the same line several instructions

Syntax rules and basics

We create a single line comment with **//,** multi-lines between **/* and */**

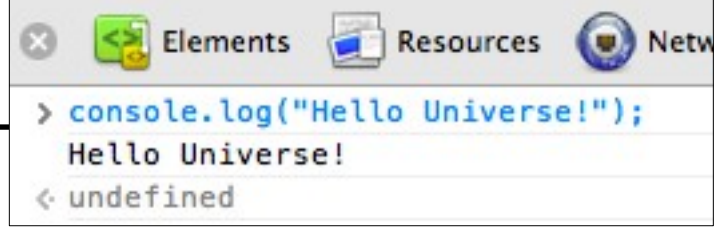
```
<script type="text/javascript">
    // Single line comment
    /*  Multi
        line
        comment  */
    </script>
```

Syntax rules and basics

To display text on the console:

console.log()

```
<script type="text/javascript">

    // Displays Hello Universe!
    console.log("Hello Universe!");

    </script>
```



Syntax rules and basics

To display text on a web page:

document.write()

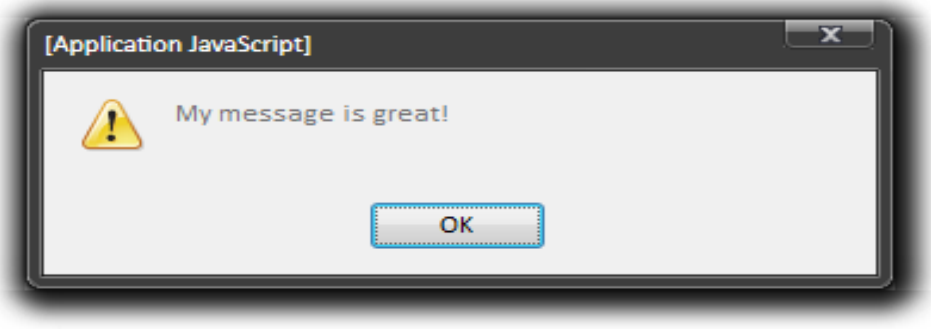
```
<script type="text/javascript">

    // Add « Hello World! » inside the page
    document.write("Hello World!");

    </script>
```

Syntax rules and basics

To display a message through a dialog box:

window.alert(*message*)

```
window.alert("My message is great!");
// Also works
alert("My message is great!");
```

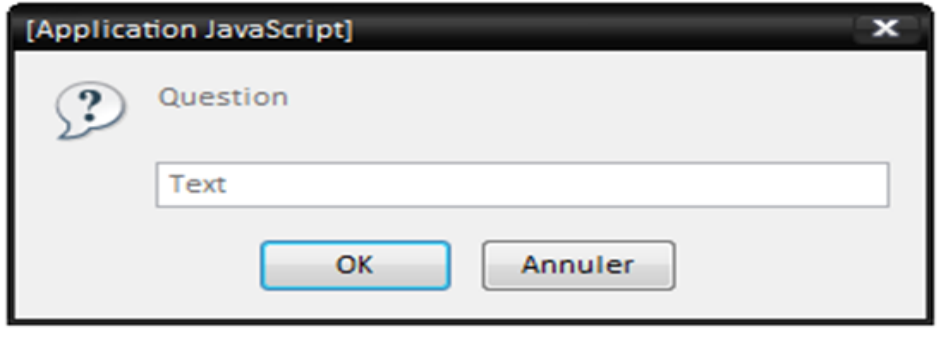


Syntax rules and basics

To get a value with a prompting box:

window.prompt(*text*, *default_value*)

```
var returnValue = window.prompt("Question", "Text");
// Also works
returnValue = prompt("Question", "Text");
```



Variables

- Case sensitive:

  myvariable ≠ *myVariable*

- We assign a value to a variable by setting its name on the left of the assignment operator

  (=), and the value on the right

Variables

- Explicit declaration with **var** keyword

```
<script type="text/javascript">
    var name = "Estelle";
</script>
```

- A variable's name:
  - Can not begin by a number
  - Must contain only alphanumerical characters.
  - Can not be a reserved word (**var**, **for**…)

Basic notions

Variables

- Implicit declaration without **var**

```
<script type="text/javascript">
    name = "Doug";
</script>
```

Variables

- Concatenation: Combine string value(s) with other types
  - Operator +

```
var max_age = 18;
var message = "Not allowed under " + max_age + " years old";
```

Variables

- Weak typing
- Type of a variable defines format of its content
- Obtain type of a variable: **typeof**

```
var myVar1 = "I am a string !";
var myVar2 = "Am I really a string ?";
myVar2 = 100;

document.write(typeof myVar1); // Will display "string"
document.write(typeof myVar2); // Will display "number"
```

Variables cast

- parseXXX: Parse from one type to another
  - parseInt
  - parseFloat

```
var number = "11";
var parsed = parseInt(number);

document.write(parsed + 1); // Will display 12
document.write(number + 1); // Will display 111
```

Operators

Mathematic operators:

Symbol	Example	Explanation
=	var salary = 2800;	Affectation
+	salary = salary + 2800	Operation or Concatenation
-	salary = salary – 2800	Substraction
*	salary = salary * 2800	Multiplication
/	salary = salary / 2800	Division
%	salary = salary % 2800	Modulo

Basic notions

Operators

Comparison operators:

(given **salary = 2800**)

Symbol	Example	Returns	Explanation
==	salary == 2800 salary == "2800"	*true* *true*	Equals
===	salary === "2800" salary === 2800	*false* *true*	Exactly equals (value and type)
!=	salary != 2800	*false*	Not equals
!==	salary !== "2800"	*true*	Not exactly equals (value and type)

Basic notions

Operators

Comparison operators:

(given **salary = 2800**)

Symbol	Example	Returns	Explanation
>	salary > 2800	*false*	Greater than
>=	salary >= 2800	*true*	Greater than or equals
<=	salary <= 2800	*true*	Lower than or equals
<	salary < 2800	*false*	Lower than

Operators

Logic operators:

Symbol	Example	Explanation
&&	age == 18 && salary > 2800	AND
||	age == 18 || salary > 2800	OR
^	age == 18 ^ salary > 2800	Exclusive OR
>>	salary >> age	Bitwise shift right
<<	salary << age	Bitwise shift left

Operators

Misc operators:

Symbol	Example	Explanation
+=	age += 18;	Addition (number) or concatenation (string)
new	var array = new Array();	Object instanciation
delete	delete array;	Object destruction

Conditional statements

Conditional test: **if** … **else if** … **else**

```
if( expression1 ) {
    // If "expression1" is evaluated to true, then this
    // block is executed
} else if ( expression2 ) {
    // Otherwise, if "expression2" is evaluated to true,
    // this block is executed
} else {
    // Otherwise, this code block is executed
}
```

Conditional statements

- Case test: **switch**

```
switch(myVar) {
    case "case1":
       // if(myVar === "case1")
       break;
    case "case2":
       // if(myVar === "case2")
       break;
    default:
       // else - Default code to execute
       break;
}
```

Arrays

- Contain several data sequences
- Many ways to create an array:
  - By creating an **Array** object
  - By using square-brackets [] (adviced)
- Support all JavaScript data types

Arrays

- Creation of Arrays

```
var fruitBasket1 = new Array("Apples", "Bananas", "Pears");
var fruitBasket2 = [ "Oranges", "Bananas", "Strawberries"];
var fruitBasket3 = [];

var apple = fruitBasket1[0];
fruitBasket3.push(apple);
```

Loops

- Conditional loop: **while**

```
// It loops 40 times
var myVariable = 40;
while( myVariable > 0 ) {
   myVariable = myVariable – 1;
}
```

Loops

- Conditional loop: **do** … **while**
  - Same as **while**
  - First test **after** the first execution of loop’s block

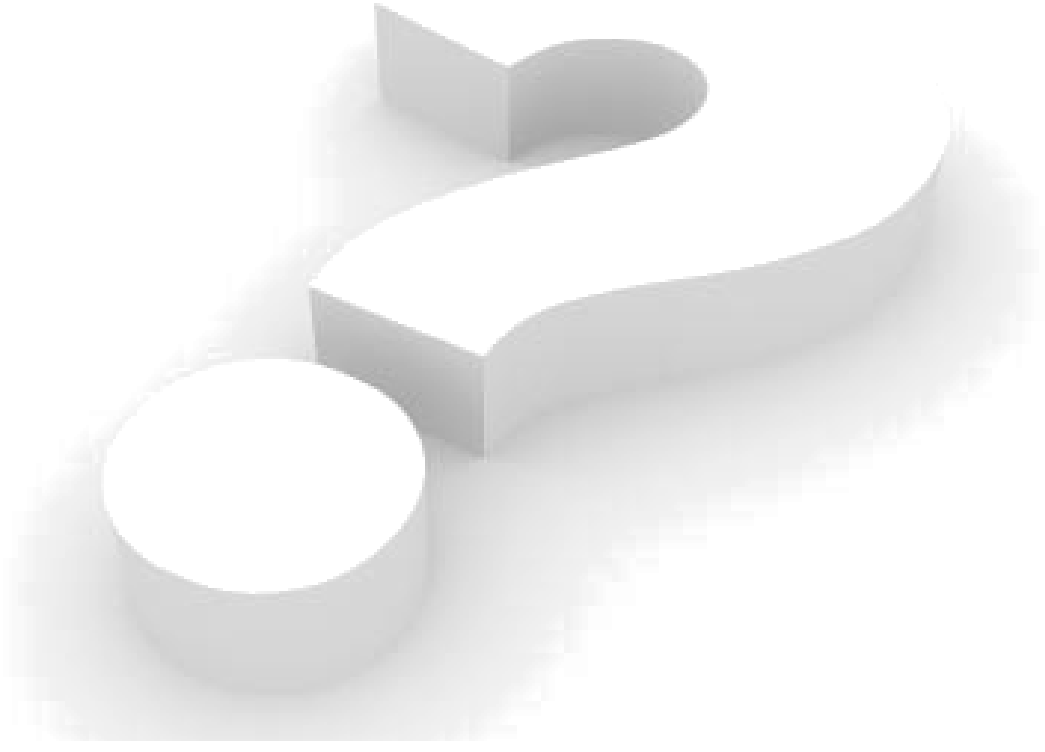
```
var myVariable = 0;
// Loop will execute once even if the test returns false
do {
    myVariable -= 1;
} while (myVariable > 0);
```

Loops

- Iterative loop: **for**
  - Specify (or not):
    - Initial state
    - Iteration condition
    - Iteration instruction

```
var a;
for (a = 0; a < 100; a += 1){
    // Loop will display Blabla 100 times.
    document.write("<p>Blabla</p>");
}
```

Questions ?



Exercise (1/3)

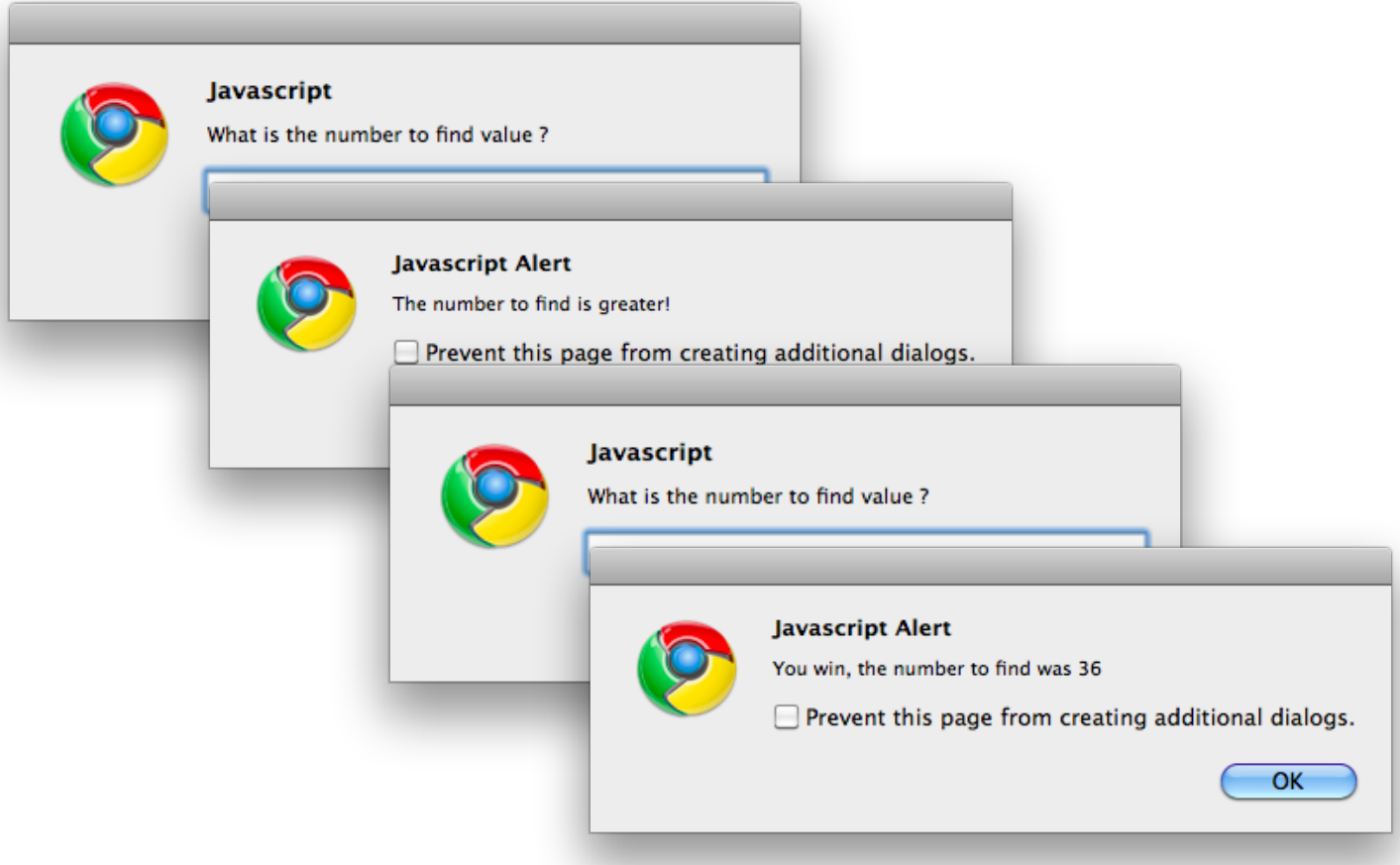
- You’re going to design your first JavaScript app:
  - **A Guess the Number Game**!
- Initialize a variable named **numberToFind** with a random number between 0 and 100*
- Initialize another variable named **remainingAttempts** with the integer value 7

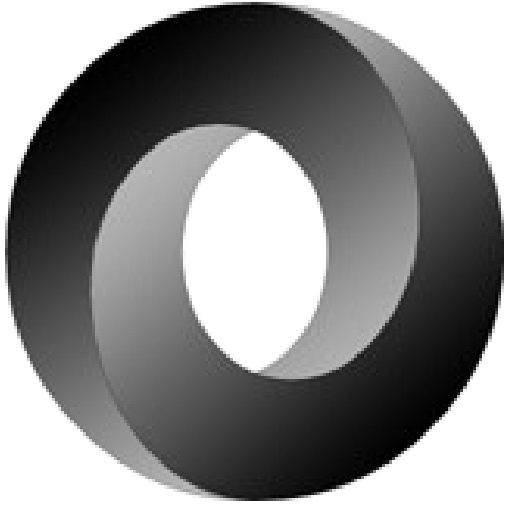
Exercise (2/3)

- Until the user doesn’t find the **numberToFind** value and has remaining attempt, ask him to choose a number
  - If the number is the **numberToFind**
    - Display a popup to the user to notify him he won
  - If the number is not the **numberToFind**
    - Display a popup to the user to notify him if the number to find is greater or not

Basic notions

Exercise (3/3)





Discover JavaScript language

FUNCTIONS & SCOPE

Functions

- Instruction unit
- Declared with **function** keyword
- Can take values or references called **arguments**

```
function myFunction(myParam1, myParam2) {
    // Some code to execute
}
```

Functions

- Called with its name followed by brackets:

functionName();

functionName(arguments);

- Can send back a value with **return** keyword
- Returned value can be use by the

Functions

- Example:

```
function howOld(year){
    var currentYear = new Date().getFullYear();
    return currentYear - year;
}

// someValue will contain 22 (if current year is 2012)
var someValue = howOld(1990);

// Will display Bryan is 42 years old (if 2012)
console.log("Bryan is " + howOld(1970) + " years old");
```

Variable scope

- Local:
  - Reachable only in the function where it's defined
- Global:
  - Reachable in the whole document
- function = scope
  - Explicit declaration inside function = local
  - Implicit declaration = global variable

Variable scope

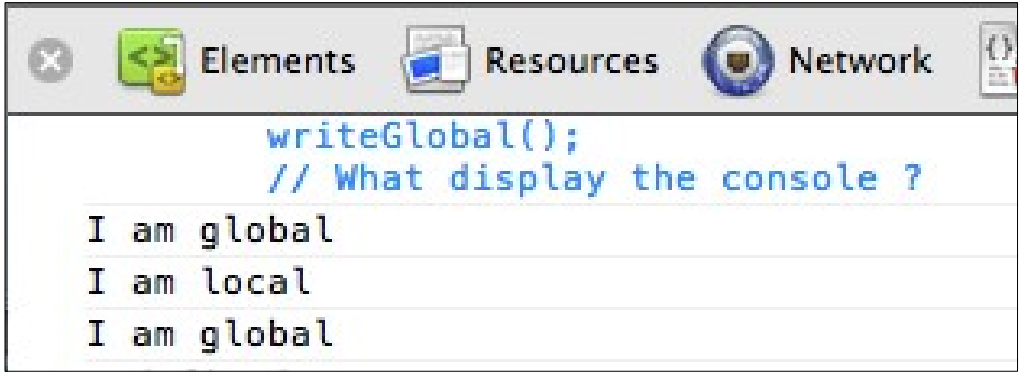
- Variables declared with **var** keyword inside a function are local variables of the function
- **Otherwise**, without **var** keyword, they are considered as global variables
- Declared variable outside function are also global variables

Quizz

```
var myVar = "I am global\n";

function writeGlobal(){
    console.log(myVar);
}

function setGlobal_writeLocal(newValue){
    myVar = newValue;
    var myVar = "I am local\n";
    console.log(myVar);
}

writeGlobal();
setGlobal_writeLocal("I am still global\n");
writeGlobal();

// What displays the console ?
```

Variable scope

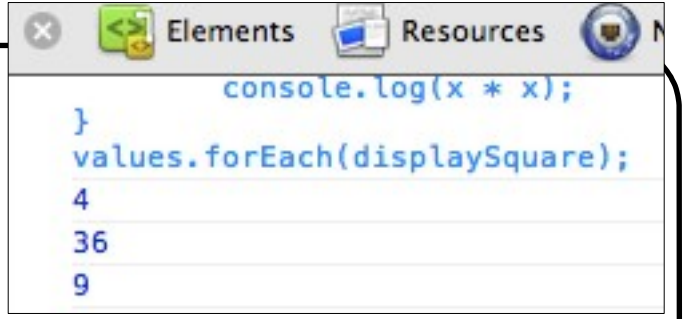
- Be careful:
  - Variable declarations with **var** are always interpreted before the function execution



Function Expressions

- JavaScript supports also function expressions
  - Functions with or without name (anonymous)
  - Can be used to contain functionality for short-term use

```
var values = [2, 6, 3];
var displaySquare = function(x) {
   console.log(x * x);
}
values.forEach(displaySquare);
```



Functional

- JavaScript is also a functional language !
- First-class functions:
  - Can be assigned to variables or stored in data structures
  - Can be passed as arguments to other functions
  - Can be returned as the values from other functions

Pass a function as parameter – Example

- Execute an operation once per array element:
  - The current element is represented by the param of the anonymous function

```
var myArray = ["Apple", "Strawberry"];
myArray.forEach( function(element) {
   console.log(element + "/");
});
```

Fn expression VS Fn declaration

- Function declarations are evaluated before any instructions in the same context

- Function expressions are evaluated after all the instructions preceding it

Functions & Scope

Example

```
function declaration() {
   console.log("I'm a function declaration");
}
var expression = function() {
   console.log("I'm a function expression");
}
declaration();
expression();
```

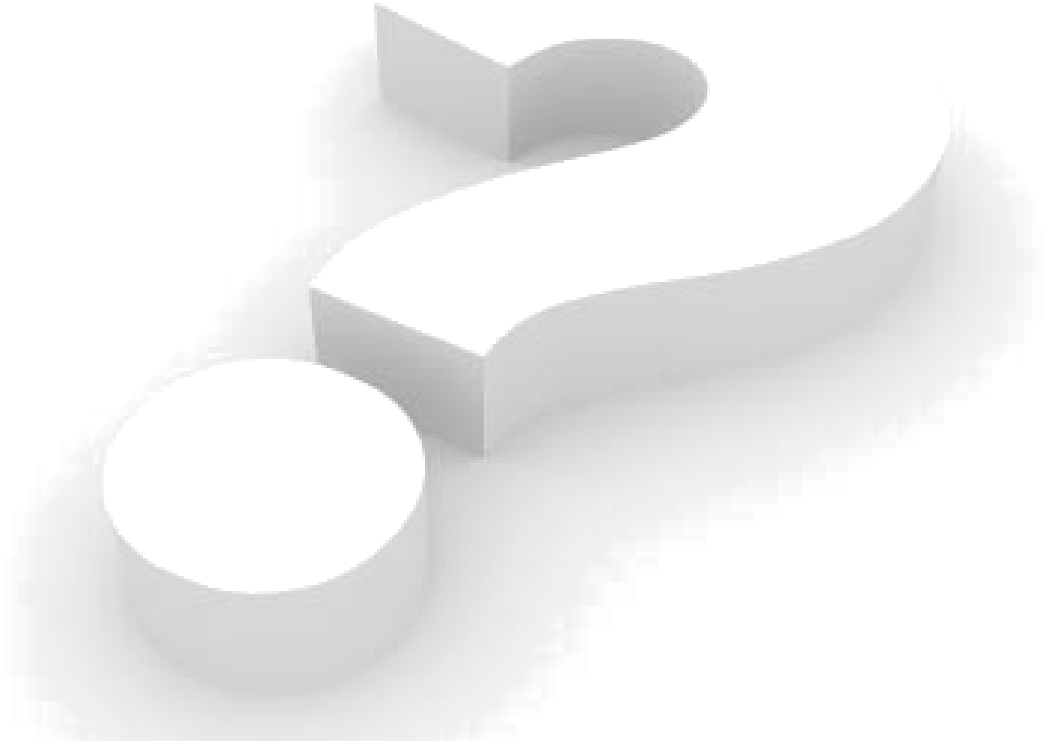
```
I'm a function declaration
I'm a function expression
>
```

Example

```
declaration();
expression();
function declaration() {
   console.log("I'm a function declaration");
}
var expression = function() {
   console.log("I'm a function expression");
}
```

```
I'm a function declaration
▶ Uncaught TypeError: expression is not a function
>
```

Questions ?



Discover JavaScript language

DOM INTERACTIONS

Introduction

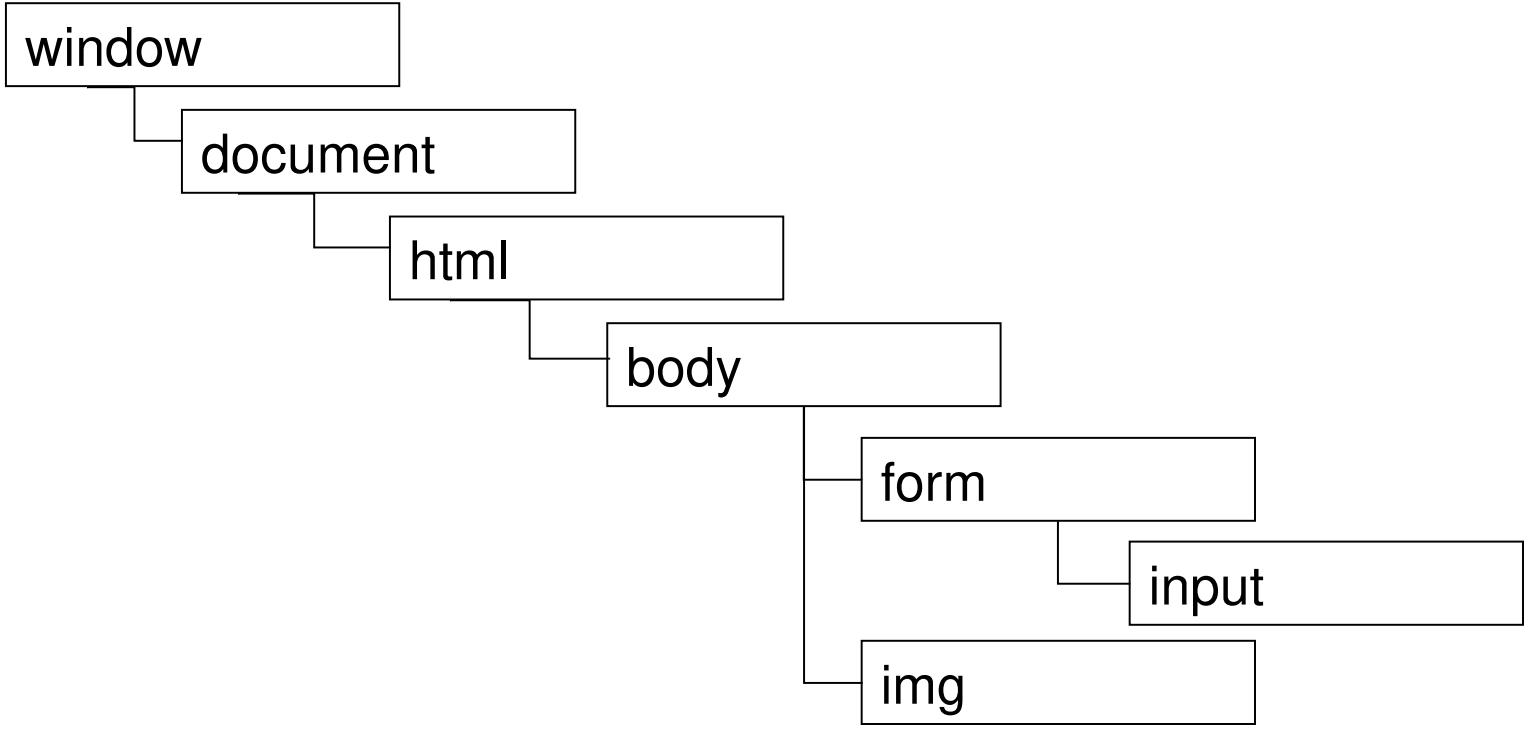
- Document Object Model

- W3C Standard
  - Whole of standardized objects for HTML
  - Standardized tools to access and manipulate HTML documents
- Independent of the language or the platform

DOM

Introduction

Simplified arborescence:



Access Elements

- Access to all the structure of an HTML page

- We will be able to dynamically:
  - Access HTML elements
  - Access, Modify and delete attributes and values
  - Create, modify and delete elements
  - Organize elements into a hierarchy

Access Elements

- Access a single element by a CSS selector:

```
document.querySelector("#myElement");
```

- Access an element list by a CSS selector:
  - Return a JavaScript array containing all homonyms

```
document.querySelectorAll("p");
```

Access Elements

- Access elements by their tag name:
  - Return a JavaScript array containing all elements with the specified tag

```
document.getElementsByTagName(tagName);
```

Access Elements

- Example:

```
<img id="img1" name="theImage" src="test.png" />
<script type="text/javascript">
  var img = document.querySelector("#img1");
  var elements = document.getElementsByName("theImage");
  console.log(elements[0] === img); // true
  elements = document.querySelectorAll("img");
  console.log(elements[0] === img); // true
</script>
```

Access Elements

- Access to all child nodes of an element:

```
element.childNodes;
```

- Access to the parent node:

```
element.parentNode;
```

Manipulate Attributes

- Access to the attributes of an element:

```
element.getAttribute("attribute");
```

- Modify the attributes of an element:

```
element.setAttribute("attribute", "value");
```

Manipulate Values

- Access to the text of an element:

```
element.firstChild.nodeValue;
```

- Modify the text of an element:

```
element.firstChild.nodeValue = "text";
```

Other manipulations…

- Create an element

```
var e = document.createElement('p');
```

- Add the element to the parent

```
parent.appendChild(e);
```

Other manipulations…

- Modify the element

```
e.style.textAlign = 'center';
```

- Delete an element

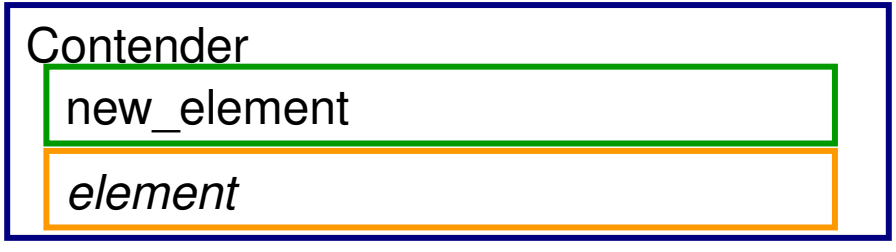
```
var e = document.querySelector("#deleteMe");
e.parentNode.removeChild(e);
```

DOM

Other manipulations…

- Add an element before an other one

```
element.parentNode.insertBefore(new_element,element);
```



Accessing Elements

```
<div id="content">
   <h1>Hello world!</h1>
   <p>
      It's my <strong>awesome</strong> page!
   </p>
</div>

<script type="text/javascript">
   var divEl = document.querySelector("#content");
   var strongEl = divEl.childNodes[1].childNodes[1];

   // Somehow easier...
   var sEl = divEl.querySelector("#content strong");
</script>
```

DOM

Cascade access

- Select an element from another

```
<p>
  <strong>Hello</strong> world!
</p>
<script>
  var p = document.querySelector("p");
  var strong = p.querySelector("strong");
  console.log(p);
  console.log(strong);
</script>
```

```
▶ <p>
…</p>
  <strong>Hello</strong>
>
```

DOM Element properties

- Useful properties:

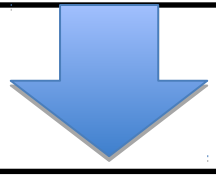
Property	Explanation
textContent	Access to the inner text inside an element
innerHTML	Access to the inner HTML content of an element
classList	Access to CSS classes
Style	Access to CSS style (including classmade)

DOM Element Properties

- Add an element before an other one

```
<p></p>
```

```
var p = document.querySelector("p");
p.textContent = "Hello World";
```



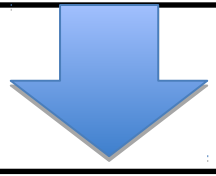
```
<p>Hello World</p>
```

DOM Element Properties

- Add an element before an other one

```
<p></p>
```

```
var p = document.querySelector("p");
p.innerHTML = "<strong>Hello World</strong>";
```



```
<p><strong>Hello World</strong></p>
```

DOM Element Properties

- Add an element before an other one

```
<p class="otherClass"></p>
```

```
var p = document.querySelector("p");
p.classList.add("myClass");
p.classList.remove("otherClass");
```

```
<p class="myClass"></p>
```

DOM Element Properties

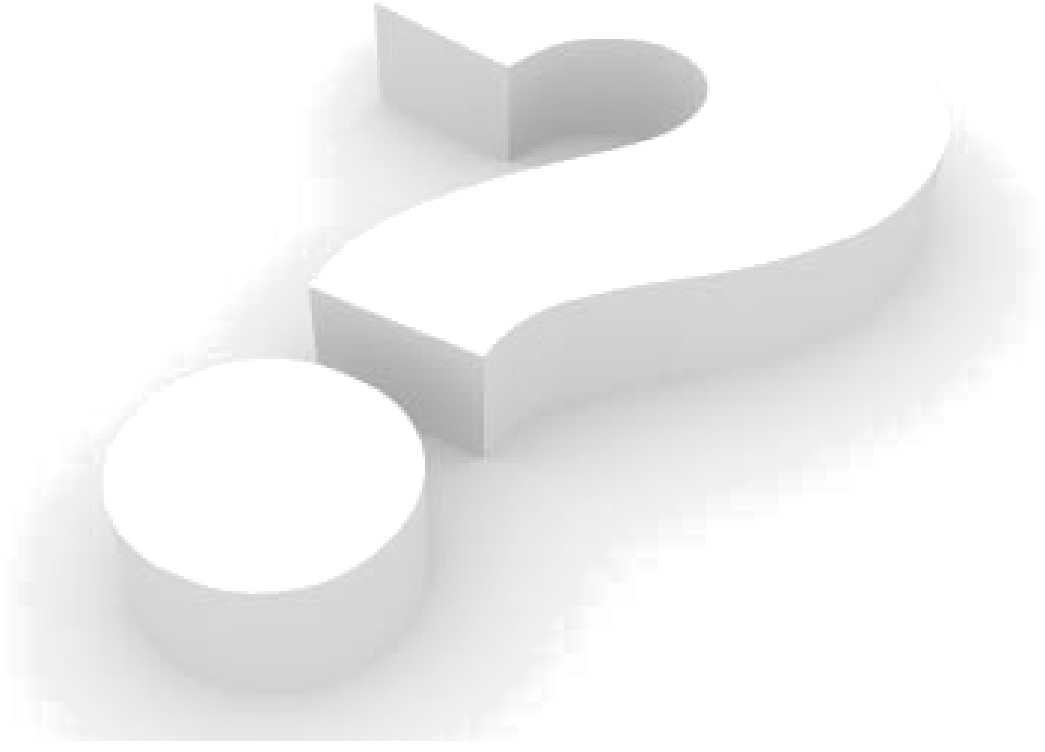
- Add an element before an other one

```
<p>This is a paragraph without style</p>
```

```
var p = document.querySelector("p");
p.style.color = "red";
p.style.fontFamily = "Calibri";
p.style.border = "1px blue solid";
p.style.textTransform = "uppercase";
p.style.width = "200px";
p.style.fontWeight = "bold";
```

THIS IS A PARAGRAPH WITHOUT STYLE

Questions ?



Exercise (1/3)

- Update your **Guess the number** page
  - We'll change alert functions to DOM nodes

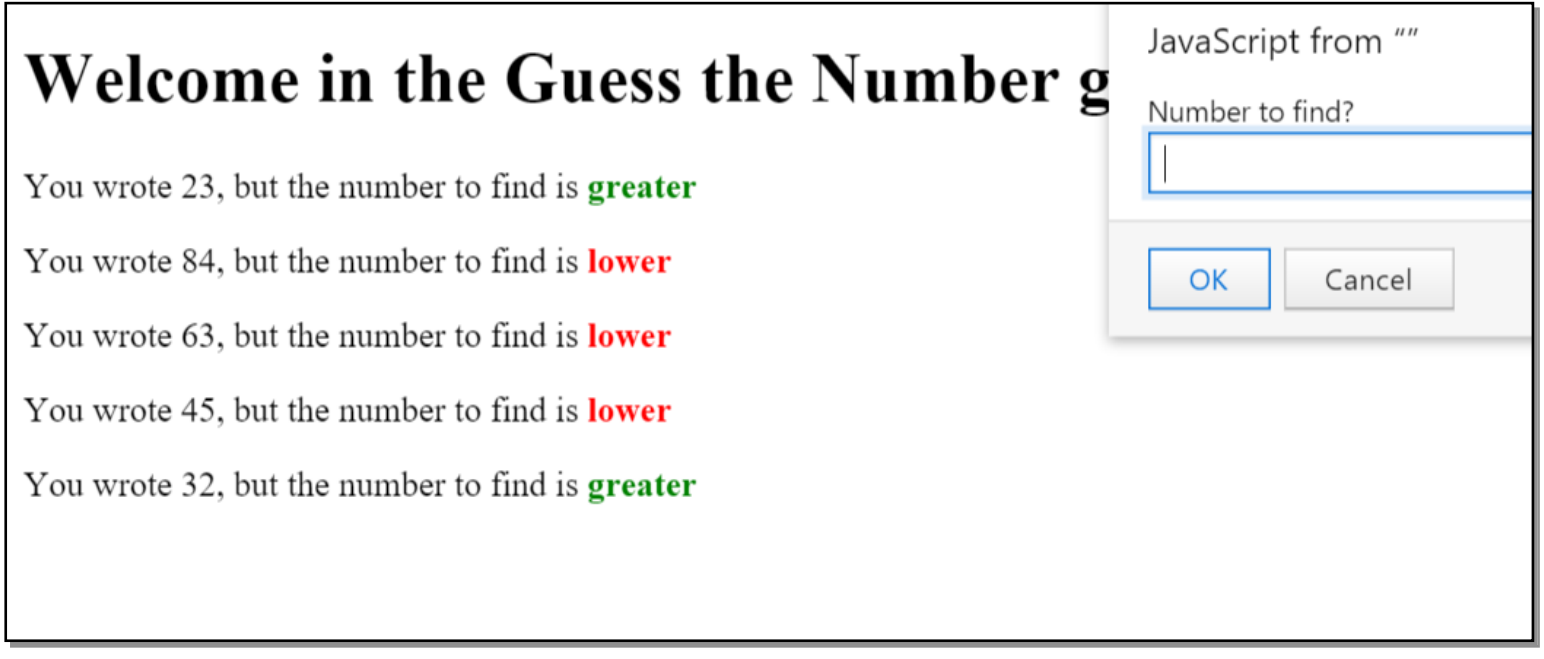
- Part one:
  - Use prompts to ask user a value
  - Create paragraphs and append them to the page instead of displaying alerts

Exercise (2/3)

- Update your **Guess the number** page
  - We'll change alert functions to DOM nodes
- Part two (use style property of your elements):
  - « greater » word should be set in green
  - « lower » word should be set in red
  - Both words should be bold

Exercise (3/3)

- Example rendering:



Discover JavaScript language

EVENTS

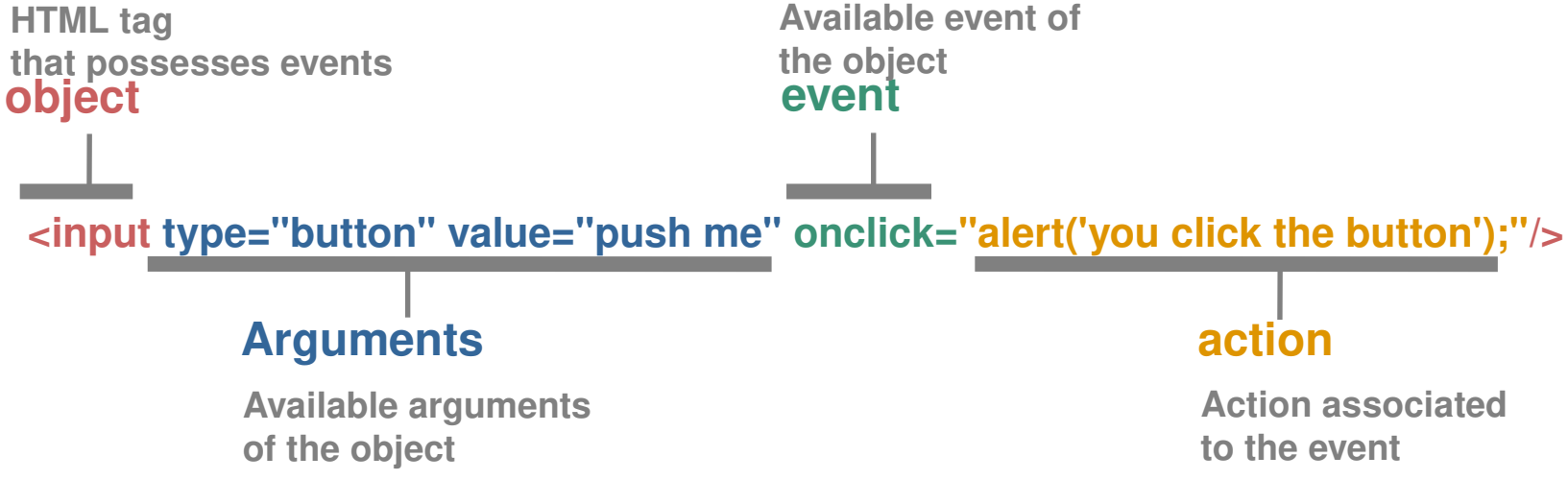
Presentation

- When occurs:
  - User interaction
  - Action in the execution context
- Properties of objects that depends on them
- May call functions

Events

Standard Event Attributes

- Attach an event on client click
  - Old way to do it: Bad practice!



Events

Standard Event Attributes

- Attach an event on client click
  - New way to do it: Good practice!

```
<input type="button" id="myButton" value="Push me!" />
```

```
var button = document.querySelector("#myButton");
button.addEventListener("onclick", function() {
    alert("You clicked the button!");
});
```

Events

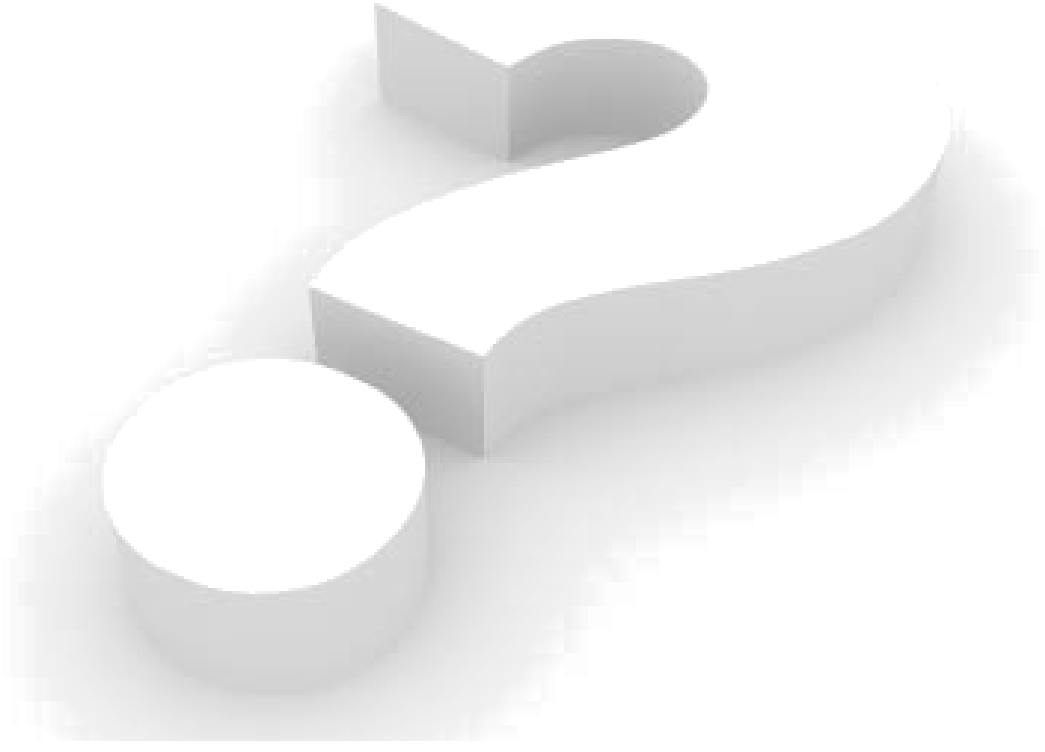
Standard Event Attributes

Event	Description
DOMContentLoaded	Page’s DOM is built (CSS, JS & Images aren’t loaded yet)
load	Element is fully loaded
unload	Browser leaves current page
click	User clicks on an element
dblclick	User double clicks on an element
mouseover	Mouse flies over an element
mouseout	Mouse leaves an element

Standard Event Attributes

Event	Description
focus	An input field gets focus
blur	An input field loses focus
change	User modifies content of an input field
select	User selects content of an input field
submit	User submits a form

Questions ?



Exercise One

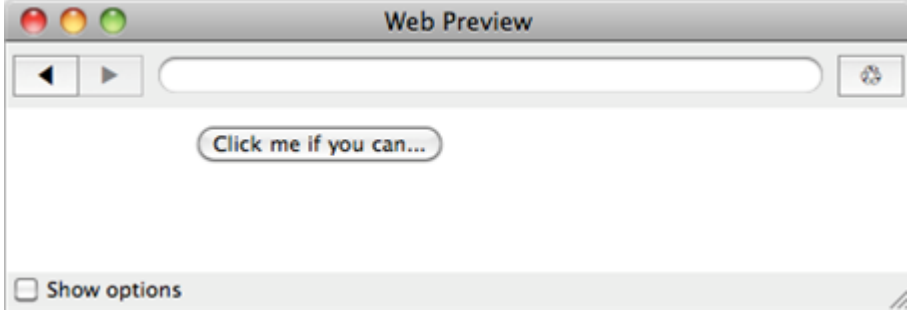
- You’re going to add JavaScript code to your contact form to validate user inputs!
- Create a new JS file.
  - Declare a function named validateForm() which validate your form inputs
    - All fields have to be filled
    - If a field has an error, set his color to red
    - Execute the function on form submit!

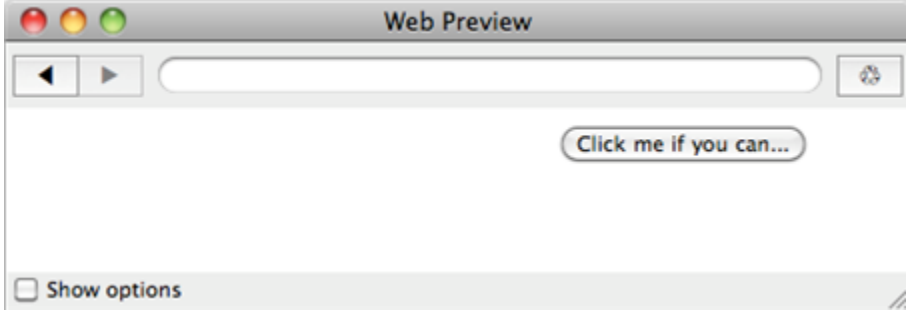
Exercise Two (1/2)

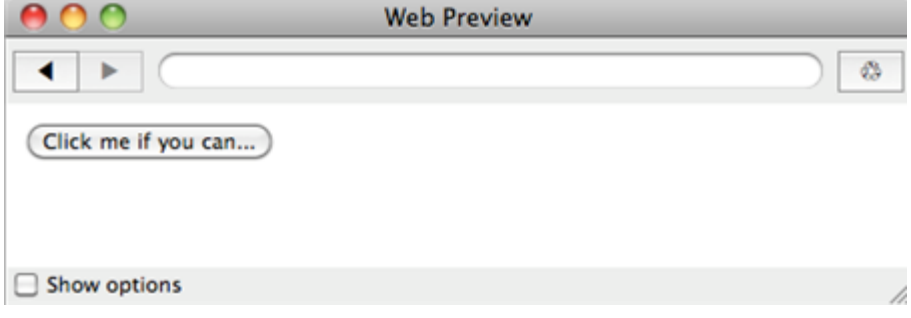
- You’re going to play with event !
- Create a page containing only one HTML button
- Create a new JS file
  - With two functions inside:
    - **goToGoogle()**: redirect the user to the google website
    - **moveButton()**: move the button inside the window without outpassing page bounds

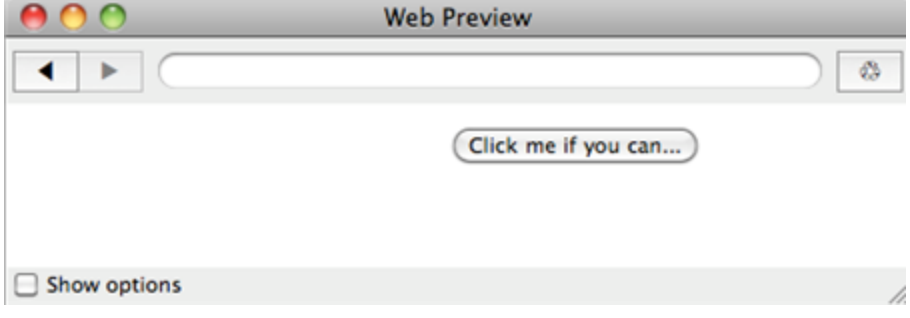
Events

Exercise Two (2/2)









Discover JavaScript language

CALLBACK CONCEPT

What are callbacks?

- Everything that calls a function passed as parameter
  - It’s a design model, not a language feature.
  - Look again at this code:

```
var button = document.querySelector("#myButton");
button.addEventListener("onclick", function() {
    alert("You clicked the button!");
});
```

Callback Concept

What are callbacks?

- In this example, you might understand that:
  - You have the targetted element (button)
  - You specify that, in case of a click event…
  - You'll do that function

```
var button = document.querySelector("#myButton");
button.addEventListener("onclick", function() {
    alert("You clicked the button!");
});
```

What are callbacks?

- This callback pattern is everywhere in JavaScript
  - Look this another example:
    - The inner anonymous function is called for every element

```
var myArray = ["Apple", "Strawberry"];
myArray.forEach( function(element) {
   console.log(element + "/");
});
```

Callback Concept

Example of forEach rewrite

```
function forEach(array, callback) { // Contains the function
  for(var i = 0; i < array.length; i++) {
    callback(i, array[i]); // Execute the function
  }
}
```

```
var fruits = ["Apple", "Orange", "Strawberry"];
forEach(fruits, function(index, value) {
   console.log("Element at index "
     + index + " is " + value);
});
```

```
Element at index 0 is Apple
Element at index 1 is Orange
Element at index 2 is Strawberry
>
```

Callback Concept

What are callbacks?

- Remember that functions can be stored in JS

```
forEach(fruits, function(index, value) {
   console.log("Element at index "  + index + " is " + value);
});
```

Is equal to

```
var callback = function(index, value) {
   console.log("Element at index " + index + " is " + value);
};
forEach(fruits, callback);
```

Why callbacks?

- Callbacks are useful to:
  - Override a logic
    - By default, nothing happens when you click on a HTML element. addEventListener allows you to create your own logic with callback. Same for forEach.
  - Do something when something else is done
    - After an asynchronous call or after a delay, you want to do a specific action. JavaScript use callbacks for that.

Callback Concept

SetTimeout

- Allows to delay an action after some time
  - Count in milliseconds (ms)
  - Uses callbacks

```
setTimeout(function() {
   alert("Hello");
}, 1000); // 1000ms = 1 second
alert("world");
```

  - Which alert will be displayed first? Why?

Callback Concept

SetInterval

- Allows to **do** an action **periodically**
  - Count in milliseconds (ms)
  - Uses callbacks

```
setInterval(function() {
   alert("Hello");
}, 1000);
// Displays Hello every second, which is quite annoying
```

Callback Concept

Clear Timeout & Interval

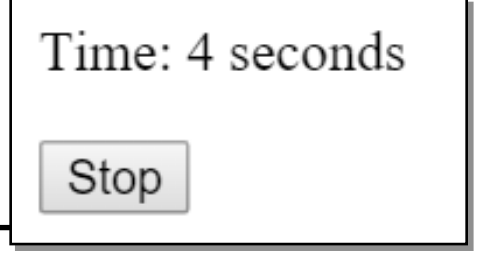
- ClearTimeout:
  - Allows to remove a setTimeout function

```
var timeoutID = setTimeout(function() {
   alert("Hello");
}, 1000);
clearTimeout(timeoutID); // Won't display the alert
```

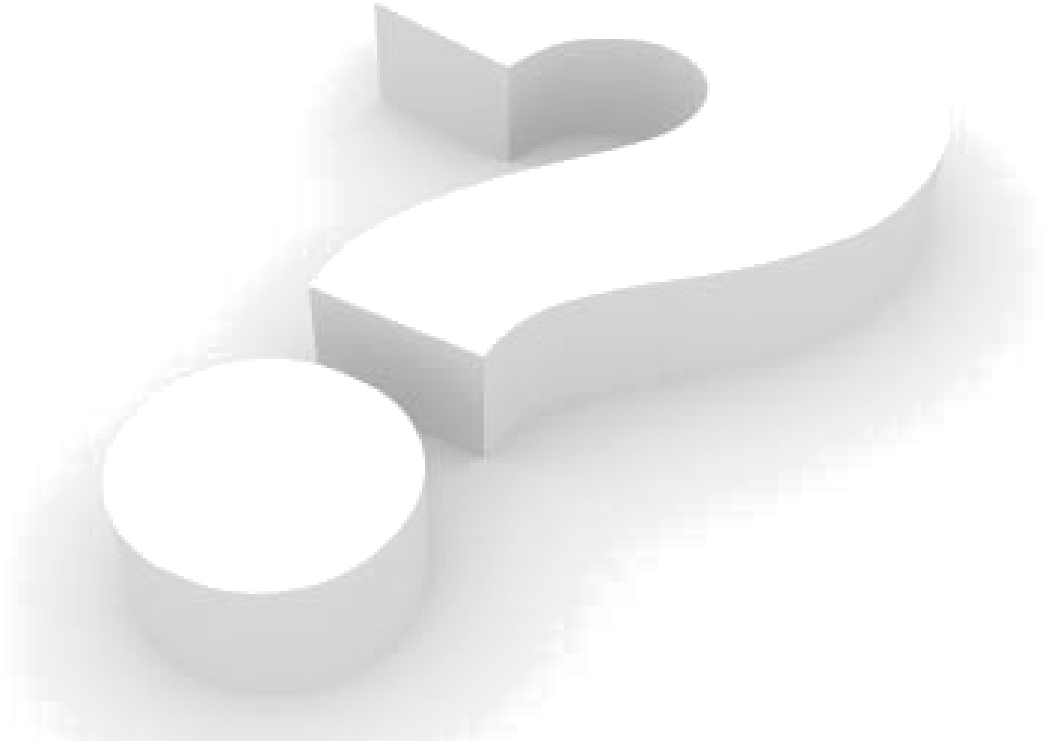
- ClearInterval: Same logic

ClearInterval example

```
<p>Time: <span>0</span> seconds</p>
<button>Stop</button>
<script>
var span = document.querySelector("span");
var button = document.querySelector("button");
var intervalID = setInterval(function() {
   var seconds = parseInt(span.textContent);
   span.textContent = ++seconds;
}, 1000);
 button.addEventListener("click", function() {
   clearInterval(intervalID);
});
</script>
```



Questions ?



Exercise (1/3)

- We’ll do a simple game
  - Goal: Click on five buttons in less that four seconds

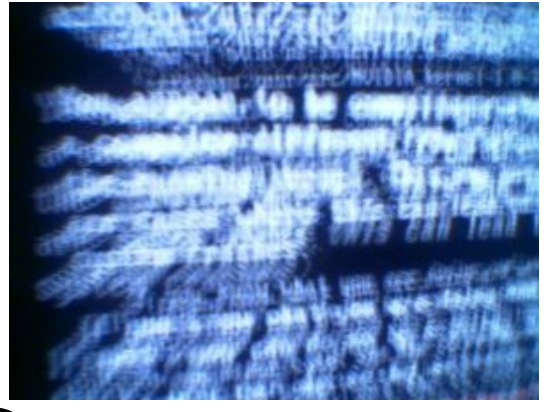
- Create a new web page with:
  - A call to the setTimeout function
    - Will display « You lost » after four seconds

Exercise (2/3)

- Five buttons
  - All buttons with the text « Click me »
  - Use a loop to assign events
  - Click on a button will:
    - deactivate it*
    - Call the checkStatus function described below

Exercise (3/3)

- A function checkStatus
  - Will get the « disabled » attribute on every button
  - If every button is disabled:
    - Display a « You won » alert
    - Clear the timeout of the « You lost » alert »



Discover JavaScript language

INTRODUCTION TO OBJECT MODELING

Presentation

- JavaScript is an Object Oriented Programming language that uses Prototypes
- We’ll see more about OOP next year but there are some basic concepts we have to see during this course...

Presentation

- Objects in JavaScript are mutable keyed collections
- *number*, *string*, *boolean*, *null* and *undefined* are primitive types
- *Arrays* are objects

Presentation

- An object contains properties

- A property has a name and a value
  - A property name can be any string
  - A property value can be any JavaScript value
    - Strings
    - Arrays
    - Functions!

Presentation

- There are several ways to declare an object
- We’re going to see just one during this course!
- That’s called : **Object Literals**

Object Literals

- Convenient notation for creating new objects

- A pair of curly braces surrounding zero or more name/value pairs:

```
var barney = {
    "firstName": "Barney",
    "lastName": "Stinson",
    "saySmthg": function() {
        console.log("It’s gonna be...");
    }
}
```

Object Literals

- Quotes around property names are optional if the name is a legal JavaScript name

- Property values can be other object literals

Object Literals

- Examples:

```
var barney = {
  firstName: "Barney",
  lastName: "Stinson",
  saySmthg: function() {
    console.log
   ("It’s gonna be...");
  }
}
```

```
var trip = {
  departure: {
    city: "Paris",
    country: "France"
  },
  arrival: {
    city: "Montreal",
    country: "Canada"
  },
  price: 890
}
```

Object Literals

- To access a property:

```
var firstName = barney.firstName;

var lastName = barney["lastName"];
```

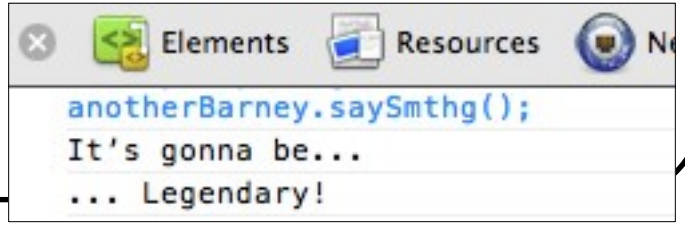
- To call a method:

```
barney.saySmthg();

barney["saySmthg"]();
```

Object Literals

- To create new instances based on an existing object, you can clone it:

```
var anotherBarney = Object.create(barney);

anotherBarney.saySmthg = function() {
    console.log("... Legendary!");
};

barney.saySmthg();
anotherBarney.saySmthg();
```



The *new* operator

- You can also in some specific case use the ***new*** operator to create new instances
  - We'll see more about how to define objects with it in a next course…

```
var anArray = new Array(); // It works!

var newBarney = new barney();  // It doesn't work because
                               // barney is an object literal
```

Standard objects

Main objects provided by the language:

Type	Description
Array	Represent a data array
Math	Provide mathematics functions
Date	Represent a date
RegExp	A useful type to use regular expressions

Standard objects: Math

- Math object
  - Properties: mathematical constants
  - Methods: mathematical functions

```
var nqpi = Math.round(Math.PI); // 3
var logE = Math.log(Math.E); // 1
var al = Math.random(); // Random float between 0 and 1
var al2 = Math.floor(Math.random() * 10);
```

Standard objects: Date

- Date object
  - Represents a date/hour
  - Provide some useful functions

```
var now = new Date(); // today
var unix = new Date(0); // 1970-01-01
var date1 = new Date("Day Mth dd YYYY hh:mm:ss");
var date2 = new Date("YYYY","MM","DD","hh","mm","ss","ms");

var timestamp = Date.now();
```

Standard objects: RegExp

- RegExp object
  - Allow to manipulate regular expressions
  - Can be created by two ways

```
var regex = /PATTERN/<g|i|gi>;
regex = new RegExp("PATTERN",<"g"|"i"|"gi">);
```

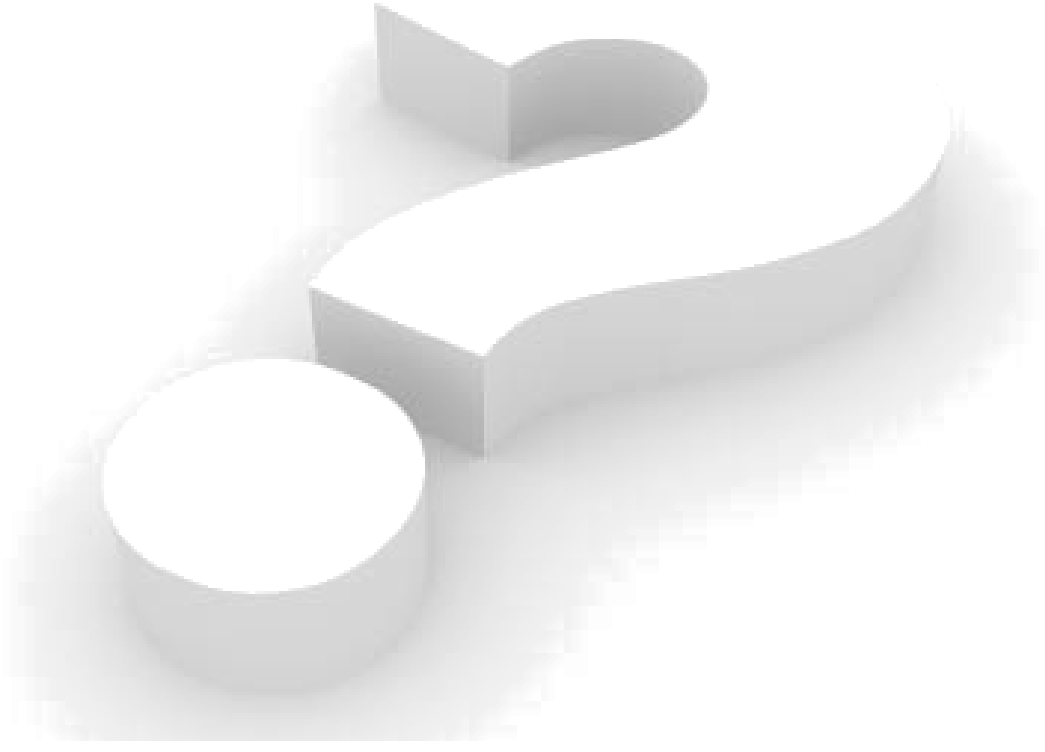
  - Use modifiers:
    - "g" for "global"
    - "i" for "insensitive"

Standard objects: RegExp

- RegExp object:
  - Patterns also have functions:
    - *test(str)* to check if there is a match
    - *exec(str)* to return matched string parts

```
var str = "cdbBdbsbz";
var re = /d(b+)(d)/ig;
var match = re.test(str);
console.log(match); // Display: « true »

var result = re.exec(str);
console.log(result.toString()); // Display: « dbBd,bB,d »
```

Questions ?



Exercise

- Let’s create a Calculator!
  - Create an HTML file with a sample structure
  - Add to it a tag with class « calculator »
  - Create a « style.css » file and link it to the HTML
  - Create a « script.js » file and link it to the HTML

Exercise

- In the JavaScript file:
  - Create a variable Calculator with in it:
    - A property currentSign
    - A property number1
    - A property number2

```
var Calculator = {
  currentSign: false,
  ...
};
```

Exercise

- Add in the App variable a ***init(selector)*** function:
  - Will query the selector passed in parameter
  - For each element targetted, will:
    - Draw in JavaScript a <table> tag
    - The first row will display the result (colspan 3) & +
    - The second row will display 7, 8, 9 and -
    - The third row will display 4, 5, 6, *
    - The fourth row will display 1, 2, 3, /
    - The fifth row cill display 0 (colspan 3) and =
  - Use the generateCell function described after

Exercise

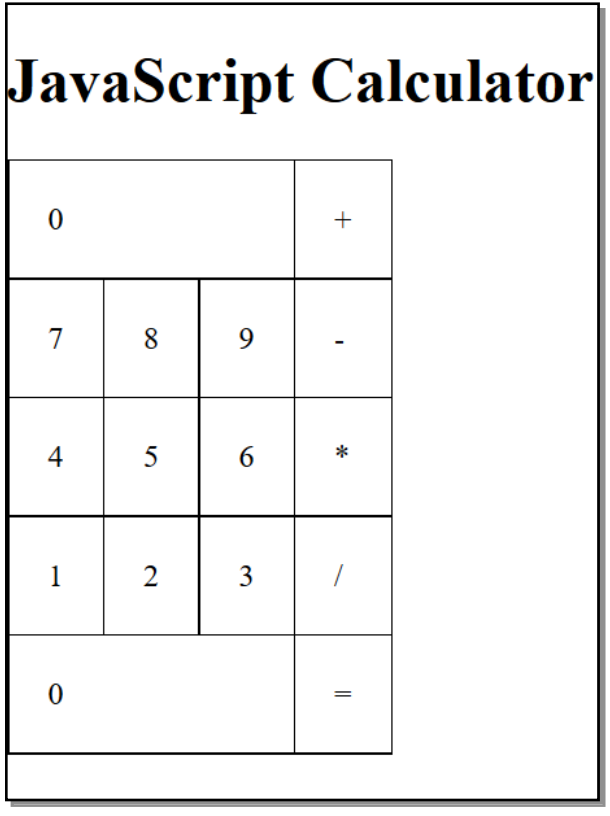
- About this table construction...
  - Copy/pasting a lot of document.createElement() isn’t a good idea
- Create a function ***generateCell(text, colspan)*** in your App variable which should:
  - Create a td element
  - Set its textContent
  - If colspan is defined, specify the cell colspan
  - Return the created element

Exercise

- Use your ***generateCell(text, colspan)*** function to populate your <table>

```
init: function() {
  ...
  var row1 = document.createElement("tr");
  row1.appendChild(App.generateCell(0, 3));
  row1.appendChild(App.generateCell("+"));
  table.appendChild(row1);
  ...
}
```

Exercise – Example rendering




```
/*
  This is an example
  Feel free to change it
*/
.calculator table,
.calculator td {
    border: 1px black solid;
    border-collapse: collapse;
}
.calculator td {
    padding: 20px;
}
```

Exercise

- Update your ***generateCell(text, colspan)*** function
  - Create a switch structure based on t
    - Add an event on operators (+, -, *, /)
      - » Should set the currentSign property (+, -, *, /) on click
    - Add an event on numbers
      - » Should set number1 and number2
      - » See more information in comments
    - Add an event on the equal sign
      - » Should call a ***App.compute()*** function (we'll define it next)

```
switch(sign)
{
  case "+":
  case "-":
  case "*":
  case "/":
    break;
  case "0":
  ...
    break;
}
```

Exercise

- Still inside App variable, add a function compute()
  - Should compute the result between number1 & number2
  - Do another switch case here on App.currentSign
    - To know which operator you're doing
- Add a function updateResult(result)
  - Should target the result cell of your calculator
  - And set its textContent with the provided result

Exercise

- In your ***compute()*** function:
  - Call ***updateResult(result)***
  - Set back number2 to 0
  - Set back currentSign to false

- Update your event on operators
  - If a number2 is set, call the ***compute()*** function
  - Else, keep it as is, should still update the sign

Exercise

- Last thing to do in the JavaScript:
  - Handle division by zero
    - Do an alert
    - Set back number1, number2 and currentSign to initial

- Update your CSS & make your calc beautiful

Exercise

- Example rendering:

Exercise - Bonuses

- Explanation in comments:
  - Allow your App to create several calculators on the same web page
  - Add a cell « . » and handle float numbers
  - Add event on body to handle keyboard input