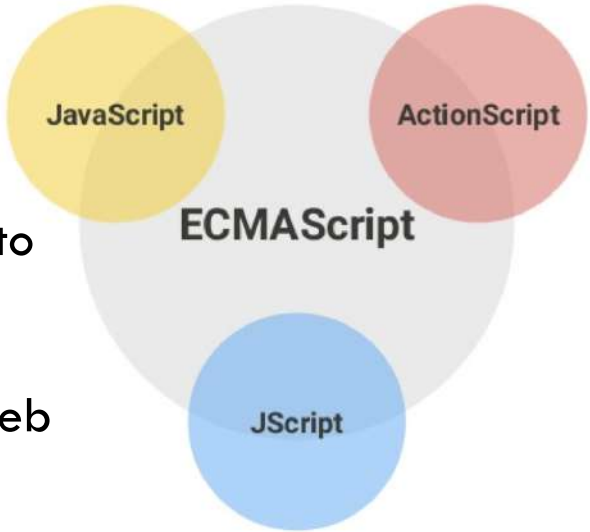
CORSO DI LAUREA IN INFORMATICA

PROGRAMMAZIONE WEB

JAVASCRIPT

a.a 2020-2021

Che cos'è JavaScript



- JavaScript è un linguaggio di scripting sviluppato per dare interattività alle pagine HTML
- Può essere inserito direttamente nelle pagine Web ed è in pratica lo standard client-side
- Il suo nome ufficiale è ECMAScript
  - È diventato standard ECMA (European Computer Manufactures Association) (ECMA-262) nel 1997
  - È anche uno standard ISO (ISO/IEC 16262)
- Sviluppato inizialmente da Netscape (il nome originale era LiveScript) e introdotto in Netscape 2 nel 1995
- In seguito anche Microsoft ha lavorato sul linguaggio producendo una sua variante chiamata JScript

You already know we use HTML, or Hypertext Markup Language, to specify all the **content** of your pages along with their **structure**, like paragraphs, headings and sections.

And you already know that we use CSS, or Cascading Style Sheets, to specify how the HTML is presented...the colors, fonts, borders, margins, and the layout of your page. CSS gives you **style**, and it does it in a way that is separate from the structure of the page.

So let's introduce JavaScript, HTML & CSS's computational cousin. JavaScript lets you create **behavior** in your web pages. Need to react when a user clicks on your "On Sale for the next 30 seconds!" button? Double check your user's form input on the fly? Grab some tweets from Twitter and display them? Or how about play a game? Look to JavaScript. JavaScript gives you a way to add programming to your page so that you can compute, react, draw, communicate, alert, alter, update, change, and we could go on... anything dynamic, that's JavaScript in action.

JAVASCRIPT POPULARITY

Worldwide, Apr 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	30.61 %	+3.9 %
2		Java	18.45 %	-1.9 %
3		Javascript	7.91 %	-0.4 %
4		C#	7.27 %	-0.0 %
5		PHP	6.07 %	-1.1 %
6		C/C++	5.76 %	-0.2 %
7		R	3.8 %	-0.2 %
8		Objective-C	2.4 %	-0.4 %
9		Swift	2.23 %	-0.2 %
10	↑	TypeScript	1.85 %	+0.2 %
11	↓	Matlab	1.77 %	-0.2 %
12	↑↑	Kotlin	1.63 %	+0.4 %
13		VBA	1.33 %	+0.0 %
14	↑↑↑	Go	1.26 %	+0.2 %
15	↓↓↓	Ruby	1.23 %	-0.1 %
16		Scala	0.99 %	-0.1 %
17	↓↓	Visual Basic	0.92 %	-0.2 %

Processo di standardizzazione di JavaScript

- È diventato standard ECMA nel 1997 (ECMA-262)
- Nel dicembre 1999 si è giunti alla versione ECMA-262 Edition 3, anche noto come ECMAScript Edition 3, corrisponde a JavaScript 1.5
- Nel dicembre 2009 si è definita la versione ECMAScript Edition 5 (superset di ECMAScript Edition 3), corrispondente a JavaScript 1.8
- Nel giugno 2011 si è giunti ECMAScript Edition 5.1 (superset di ECMAScript Edition 5), corrispondente a JavaScript 1.8.5
- ECMAScript 2015 – edizione 6
- ECMAScript 2016 – edizione 7
- ECMAScript 2017 – edizione 8
- ECMAScript 2018 – edizione 9 – June 2018
- ECMAScript 2019 – edizione 10 – June 2019

JavaScript e Java

- *Al di là del nome, Java e JavaScript sono due cose completamente diverse*
- L'unica similitudine è legata al fatto di aver entrambi adottato la sintassi del C
- Esistono profonde differenze
  - JavaScript è **interpretato** e non compilato
  - JavaScript è **object-based** ma **non class-based**
    - Esiste il concetto di oggetto
    - Non esiste il concetto di classe
  - JavaScript è **debolmente tipizzato** (weakly typed)
    - Non è necessario definire il tipo di una variabile
    - *Attenzione però: questo non vuol dire che i dati non abbiano un tipo (sono le variabili a non averlo in modo statico)*

JavaScript Testing

Problema:

Javascript: ogni browser vendor crea la propria versione

- Il comportamento dello stesso programma javascript può variare sostanzialmente da un browser all'altro, e anche da una versione all'altra dello stesso browser
- Conseguenza
  - Prima di rilasciare la distribuzione finale, è necessario testare la propria applicazione web su tutti i browser che si prevede di supportare
  - La maggior parte degli sviluppatori
    - Effettua i test iniziali e lo sviluppo su Chrome o Firefox
    - Ma testa anche su Edge e Safari ed altri prima del rilascio finale

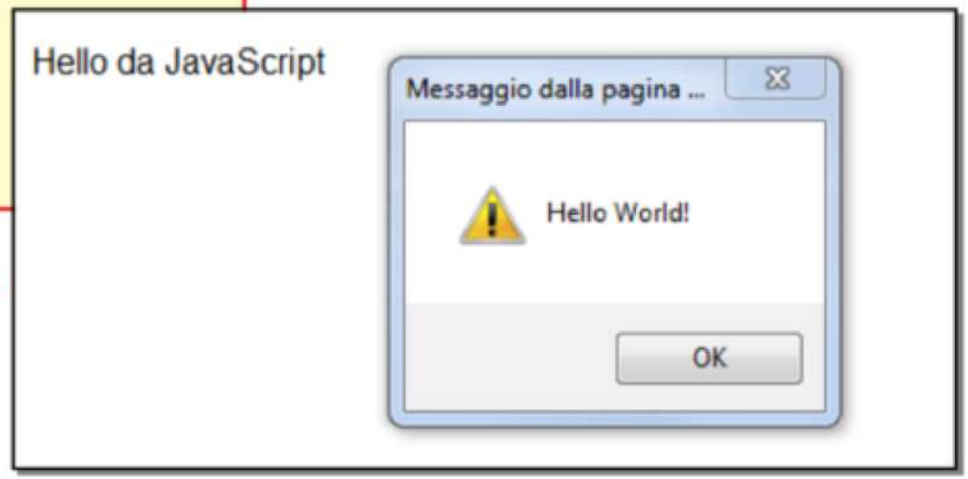
Cosa si può fare con JavaScript

- Il codice JavaScript viene eseguito da un interprete contenuto all'interno del browser
- Nasce per dare **dinamicità** alle pagine Web
- Consente quindi di
  - Accedere e modificare elementi della pagina HTML
  - Reagire ad eventi generati dall'interazione fra utente e pagina
  - Validare i dati inseriti dall'utente
  - Interagire con il browser: determinare il browser utilizzato e la dimensione della finestra in cui viene mostrata la pagina, lavorare con i browser cookie, ecc.

Esempio

- Vediamo la versione JavaScript dell'ormai mitico *HelloWorld!*
- Viene mostrato un popup con la scritta HelloWorld
- Lo script viene inserito nella pagina HTML usando il tag **<script>**:

```
<html>
  <body>
    <p>Hello da JavaScript</p>
    <script type="text/javascript">
       alert("Hello World!");
    </script>
  </body>
</html>
```



Script element

The type attribute tells the browser you're writing JavaScript. The thing is, browsers assume you're using JavaScript if you leave it off. So, we recommend you leave it off, and so do the people who write the standards.

The <script> opening tag.

```
<script type="text/javascript">
alert("Hello world!");
</script>
```

Don't forget the right bracket on the opening tag.

Everything between the script tags must be valid JavaScript.

You must end the script with a closing </script> tag, always!

Esempio 2: test.html

Here's our standard HTML5 doctype, and <html> and <head> elements.

And we've got a pretty generic <body> for this page as well.

Ah, but we've added a script element to the <head> of the page.

And we've written some JavaScript code inside it.

Again, don't worry too much about what this code does. Then again, we bet you'll want to take a look at the code and see if you can think through what each part might do.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Just a Generic Page</title>
    <script>
      setTimeout(wakeUpUser, 5000);
      function wakeUpUser() {
         alert("Are you going to stare at this boring page forever?");
      }
    </script>
  </head>
  <body>
      <h1>Just a generic heading</h1>
      <p>Not a lot to read about here. I'm just an obligatory paragraph living in
an example in a JavaScript book. I'm looking for something to make my life more
exciting.</p>
  </body>
</html>
```

Sintassi del linguaggio

- La sintassi di JavaScript è modellata su quella del C con alcune varianti significative
- In particolare
  - È un linguaggio **case-sensitive**
  - Le istruzioni sono terminate da **';'** ma il terminatore può essere omesso se si va a capo
  - Sono ammessi sia commenti multilinea (delimitati da /* e */) che monolinea (iniziano con //)
- Gli identificatori possono contenere lettere, cifre e i caratteri **'_'** e **'$'** <u>ma non possono iniziare con una cifra</u>

Variabili

- Le variabili vengono dichiarate usando la parola chiave **var**

  var nomevariabile;

- **Non hanno un tipo**
  - possono contenere valori di qualunque tipo
- È prevista la possibilità di inizializzare una variabile contestualmente alla dichiarazione

  var f = 15.8

- Possono essere dichiarate in linea

  for (var i = 1, i<10, i++) { … }

- Esiste lo **scope globale** e quello **locale** (ovvero dentro una funzione) ma, a differenza di Java, non esiste lo scope di blocco

Example

```
var x;

x = 6;
```

```
var x = 5 + 6;
var y = x * 10;
```

```
var x = 5;
var y = 6;
var z = x + y;
```

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

```
var person = "John Doe", carName = "Volvo", price = 200;
```

```
var x = 5;   // I will be executed

// var x = 6;   I will NOT be executed
```

Javascript Keyword

Keyword	Description
break	Terminates a switch or a loop
continue	Jumps out of a loop and starts at the top
debugger	Stops the execution of JavaScript, and calls (if available) the debugging function
do ... while	Executes a block of statements, and repeats the block, while a condition is true
for	Marks a block of statements to be executed, as long as a condition is true
function	Declares a function
if ... else	Marks a block of statements to be executed, depending on a condition
return	Exits a function
switch	Marks a block of statements to be executed, depending on different cases
try ... catch	Implements error handling to a block of statements
var	Declares a variable

If: One or Two Options

- Single option

```
if (condition) {
   statement_1;
   ...
   statement_N;
}
```

- Two options

```
if (condition) {
   ...
} else {
   ...
}
```

JavaScript has a liberal definition of what condition is "false" (fails the test):

- "false": false, null, undefined, "" (empty string), 0, NaN
- "true": anything else (including the string "false")

How to make a statement

A set of statements.

Each statement does a little bit of work, like declaring some variables to contain values for us.

```
var age = 25;
var name = "Owen";

if (age > 14) {
    alert("Sorry this page is for kids only!");
} else {
    alert("Welcome " + name + "!");
}
```

Here we create a variable to contain an age of 25, and we also need a variable to contain the value "Owen".

Or making decisions, such as: Is the age of the user greater than 14?

And if so alerting the user they are too old for this page.

Otherwise, we welcome the user by name, like this: "Welcome Owen!" (but since Owen is 25, we don't do that in this case.)

Decisions

```
if (scoops >= 5) {
   alert("Eat faster, the ice cream is going to melt!");
} else if (scoops < 3) {
   alert("Ice cream is running low!");
}
```

another test with if/else if

Add as many tests with "else if" as you need, each with its own associated code block that will be executed when the condition is true.

Alert and decisions

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Icecream!</title>
  </head>
  <body>
  </body>
  <script>
    scoops = 5;
    while (scoops > 0) {
        document.write("Another scoop!<br>");
        if (scoops < 3) {
            alert("Ice cream is running low!");
        } else if (scoops >= 5) {
            alert("Eat faster, the ice cream is going to melt!");
        }
        scoops = scoops - 1;
    }
    document.write("Life without ice cream isn't the same");
  </script>
</body>
</html>
```

Debugging

Firefox File Modifica Visualizza Cronologia Segnalibri Strumenti Finestra Aiuto

Download ⌘J
Componenti aggiuntivi ⇧⌘A
Accedi a Sync...
Sviluppo web
Informazioni sulla pagina ⌘I

✓ Attiva strumenti ⌥⌘I
Analisi pagina ⌥⌘C
Console web ⌥⌘K
Debugger
Editor stili ⇧F7
Prestazioni ⇧F5
Rete ⌥⌘E
Analisi archiviazione ⇧F9
Accessibilità ⇧F12
WebIDE ⇧F8
Console del browser ⇧⌘J
Modalità visualizzazione flessibile ⌥⌘M
Prelievo colore
Blocco per gli appunti ⇧F4
Service worker
Sorgente pagina ⌘U
Altri strumenti di sviluppo

JavaScript Loops

BMW
Volvo
Saab
Ford
Fiat
Audi

Analisi pagina | Console | Debugger | Editor stili | Pres | Archiviazione | Accessibilità

Sorgenti | Struttura

file://
Users/marello/Desktop
moz-extension://a926f39d-cbcc-3345-943

io.html

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <h2>JavaScript Loops</h2>
6
7  <p id="demo"></p>
8
9  <script>
10 var cars = ["BMW", "Volvo", "Saab", "Ford", "Fiat", "Audi"];
11 var text = "";
12 var i;
13 for (i = 0; i < cars.length; i++) {
14   text += cars[i] + "<br>";
15 }
16 document.getElementById("demo").innerHTML = text;
17 </script>
18
19 </body>
20 </html>
```

Espressioni di controllo
i: 6
text: "BMW
Volvo
Saab
Ford·
cars: (6) [...]

Punti di interruzione
Metti in pausa per le eccezioni
io.html
var cars = ["BMW", "Volvo", "Saab… 10
document.getElementById("demo").i… 16

Punti di interruzione XHR

If: Example 2

```
function flipcoin() {
  if (Math.random() < 0.5) {
    return("heads");
  } else {
    return("tails");
  }
}
alert(flipcoin());
```

Math.random() returns a number between 0.0 (inclusive) and 1.0 (exclusive). If the random number generator is good, the values should be evenly distributed and unpredictable

Random

- we need is an integer between 0 and 4

Our variable randomLoc. We want to assign a number from 0 to 4 to this variable.

Math.random is part of standard JavaScript and returns a random number.

```
var randomLoc = Math.random();
```

First, if we multiply the random number by 5, then we get a number between 0 and 5, but not including 5. Like 0.13983, 4.231, 2.3451, or say 4.999.

We can use Math.floor to round down all these numbers to their nearest integer value.

```
var randomLoc = Math.floor(Math.random() * 5);
```

So, for instance, 0.13983 becomes 0, 2.34 becomes 2 and 4.999 becomes 4.

If: Example 3

```
function max(n1, n2) {
  if (n1 >= n2) {
    return(n1);
  } else {
    return(n2);
  }
}
```

Switch Statement

```
function dayname(daynumber) {
  var dayname;
  switch(daynumber) {
    case 0: dayname = "sunday"; break;
    case 1: dayname = "monday"; break;
    case 2: dayname = "tuesday"; break;
    case 3: dayname = "wednesday"; break;
    case 4: dayname = "thursday"; break;
    case 5: dayname = "friday"; break;
    case 6: dayname = "saturday"; break;
    default: dayname = "invalid day";
             break;
  }
  return(dayname);
}
```

Boolean Operators

- **==, !=**
  - Equality, inequality
- **===**
  - In addition to comparing primitive types, === tests if two objects are identical (the same object), not just if they appear equal (have the same fields). More details when we introduce objects. Not used frequently
- **<, <=, >, >=**
  - Numeric less than, less than or equal to, greater than, greater than or equal to
- **&&, ||**
  - Logical and, or. Both use short-circuit evaluation to more efficiently compute the results of complicated expressions
- **!**
  - Logical negation

JavaScript Comparison and Logical Operator

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

```
x = (1 < 2) ? true : false;
```

Javascript Operator

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

```
x = 5 + 5;
y = "5" + 5;
z = "Hello" + 5;
```

```
var x = 5;
var y = 2;
var z = x % y;
```

**	Exponentiation	10 ** 2

Javascript Assignment Operator

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

```
txt1 = "What a very ";
txt1 += "nice day";
```

While

A while statement starts with the keyword while.

While uses a boolean expression that we call a conditional test, or conditional for short.

If the conditional is true, everything in the code block is executed.

```
while (scoops > 0) {
    document.write("Another scoop!");
    scoops = scoops - 1;
}
```

What's a code block? Everything between the curly braces; that is, between {}.

And, if our conditional is true, then, after we execute the code block, we loop back around and do it all again. If the conditional is false, we're done.

Like we said, lather, rinse, repeat!

HappyBirthday

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Happy Birthday</title>
  </head>
  <body>
  <script>
    var name = "Joe";
    var i = 0;
    while (i < 2) {
        document.write("Happy Birthday to you.<br>");
        i = i + 1;
    }
    document.write("Happy Birthday dear " + name + ",<br>");
    document.write("Happy Birthday to you.<br>");
  </script>
</body>
</html>
```

Console output (bottle.html)

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>My First JavaScript</title>
</head>
<body>
<script>
var word = "bottles";
var count = 99;
while (count > 0) {
    console.log(count + " " + word + " of beer on the wall");
    console.log(count + " " + word + " of beer,");
    console.log("Take one down, pass it around,");
    count = count - 1;
    if (count > 0) {
        console.log(count + " " + word + " of beer on the wall.");
    } else {
        console.log("No more " + word + " of beer on the wall.");
    }
}
</script>
</body>
</html>
```

Valori speciali

- Ad ogni variabile può essere assegnato il valore **null** che rappresenta l'assenza di un valore
- Come in SQL, **null** è un concetto diverso da *zero* (0) o *stringa vuota* (“”)
- Una variabile non inizializzata ha invece un valore indefinito **undefined**
- I due concetti si assomigliano ma non sono uguali:
  - **undefined** significa una variabile è stata dichiarata, ma non è ancora stato assegnato un valore
  - **null** è un valore di assegnazione. Esso può essere assegnato ad una variabile come una rappresentazione di valore

Tipi primitivi: numeri e booleani

- Javascript prevede pochi tipi primitivi: **numeri**, **booleani** e **stringhe**
- Numeri (**number**)
  - Sono rappresentati in formato floating point a 8 byte
  - Non c'è distinzione fra interi e reali
  - Esiste il valore speciale ***NaN*** (not a number) per le operazioni non ammesse (ad esempio, radice di un numero negativo)
  - Esiste il valore ***infinite*** (ad esempio, per la divisione per zero)
- Booleani (**boolean**)
  - ammettono i valori ***true*** e ***false***

Il concetto di tipo in JavaScript

- Come abbiamo detto, alle variabili non viene attribuito un tipo
  - lo assumono **dinamicamente** in base al dato a cui vengono agganciate
- I dati hanno un tipo e per ogni tipo esiste una sintassi per esprimere le costanti (**literal**)
- Per i numeri, ad esempio, le costanti hanno la forma usuale: **1.0**, **3.5** o in altre basi (i.e., 015, 0x123A, 0b110)
- Per i booleani sono gli usuali valori **true** e **false**

```
var v; // senza tipo

v = 15.7; // diventa di tipo number

v = true; // diventa di tipo boolean
```

Communicate with your user

- **Create an alert**
  - the browser gives you a quick way to alert your users through the **alert** function. Just call alert with a string containing your alert message, and the browser will give your user the message in a nice dialog box: **alert("Hello world!");**
  - Alert really should be used only when you truly want to stop everything and let the user know something
- **Write directly into your document**
  - Think of your web page as a document (that's what the browser calls it). You can use a function **document.write** to write arbitrary HTML and content into your page at any point. In general, this is considered **bad form**, although you'll see it used here and there

Communicate with your user (2)

- **Use the console**
  - Every JavaScript environment also has a console that can log messages from your code. To write a message to the console's log you use the function **console.log** and hand it a string that you'd like printed to the log. You can view console.log as a great tool for troubleshooting your code, but typically your users will never see your console log, so it's not a very effective way to communicate with them
- **Directly manipulate your document**
  - This is the way you want to be interacting with your page and users - using JavaScript you can access your actual web page, read & change its content, and even alter its structure and style! This all happens by making use of your browser's ***document object model***. This is the best way to communicate with your user
  - Using the document object model requires knowledge of **how your page is structured** and of the programming interface that is used to read and write to the page

How do I add code to my page?

- **You can place your code inline, in the <head> element**
- **You can add your code inline in the body of the document**
  - When your browser loads a page, it loads everything in your page's <head> before it loads the <body>
    - if your code is in the <head>, users might have to wait a while to see the page
    - If the code is loaded after the HTML in the <body>, users will get to see the page content while they wait for the code to load

How do I add code to my page? (2)

- **Put your code in its own file and link to it from the <head>**
  - This is just like linking to a CSS file. The only difference is that you use the **src** attribute of the **<script>** tag to specify the URL to your JavaScript file
  - When your code is in an external file, it's easier to maintain (separately from the HTML) and can be used across multiple pages. But this method still has the drawback that all the code needs to be loaded before the body of the page
- **You can link to an external file in the body of your page**
  - The best of both worlds. We have a maintainable JavaScript file that can be included in any page, and it's referenced from the bottom of the body of the page, so it's only loaded after the body of the page

How do I add code to my page? (3)

- Open bottle.html and select all the code; that is, everything between the <script> tags
- Now create a new file named “code.js” in your editor, and place the code into it. Then save “code.js”

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script src="code.js">

    </script>
  </body>
</html>
```

Use the src attribute of the <script> element to link to your JavaScript file.

Where your code was.

Believe it or not we still need the ending <script> tag, even if there is no code between the two tags.

Loading Scripts: Usual Approach

- Script tag with src (in head section of HTML page)

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

- Purpose
  - To define functions, objects, and variables
  - Functions will <u>later</u> be triggered by buttons, other user events, inline script tags with body content, etc.
- Html5 note
  - The type attribute is optional and will be omitted in future examples
- Best practice
  - Use subfolder for the javascript files (just as with images and CSS files)

    <script src="scripts/my-script.js"></script>
  - Some prefer the equivalent version prefaced with "./"

    <script src="./scripts/my-script.Js"></script>

Loading Scripts: Alternative Approach

- Script tag with body content (in body section of HTML page)

```
<body>
<script>javascript code that uses document.write(…)
</script>
</body>
```

- Purpose
  - To directly invoke code that will run as page loads
    - e.g., to output HTML content built by Javascript using **document.write**
  - Don't use this approach for defining functions
    - Slower (no browser caching) and less reusable

Example 1: code.js

```
function getmessage() {
var amount = math.Round(math.Random() * 100000);
  var message =
    "you won $" + amount + "!\n" +
    "to collect your winnings, send your credit card\n" +
    "and bank details to oil-minister@phisher.Com.";
  return(message);
}

function showwinnings1() {
  alert(getmessage());
}

function showwinnings2() {
  document.write("<h1>" + getmessage() + "</h1>");
}
```

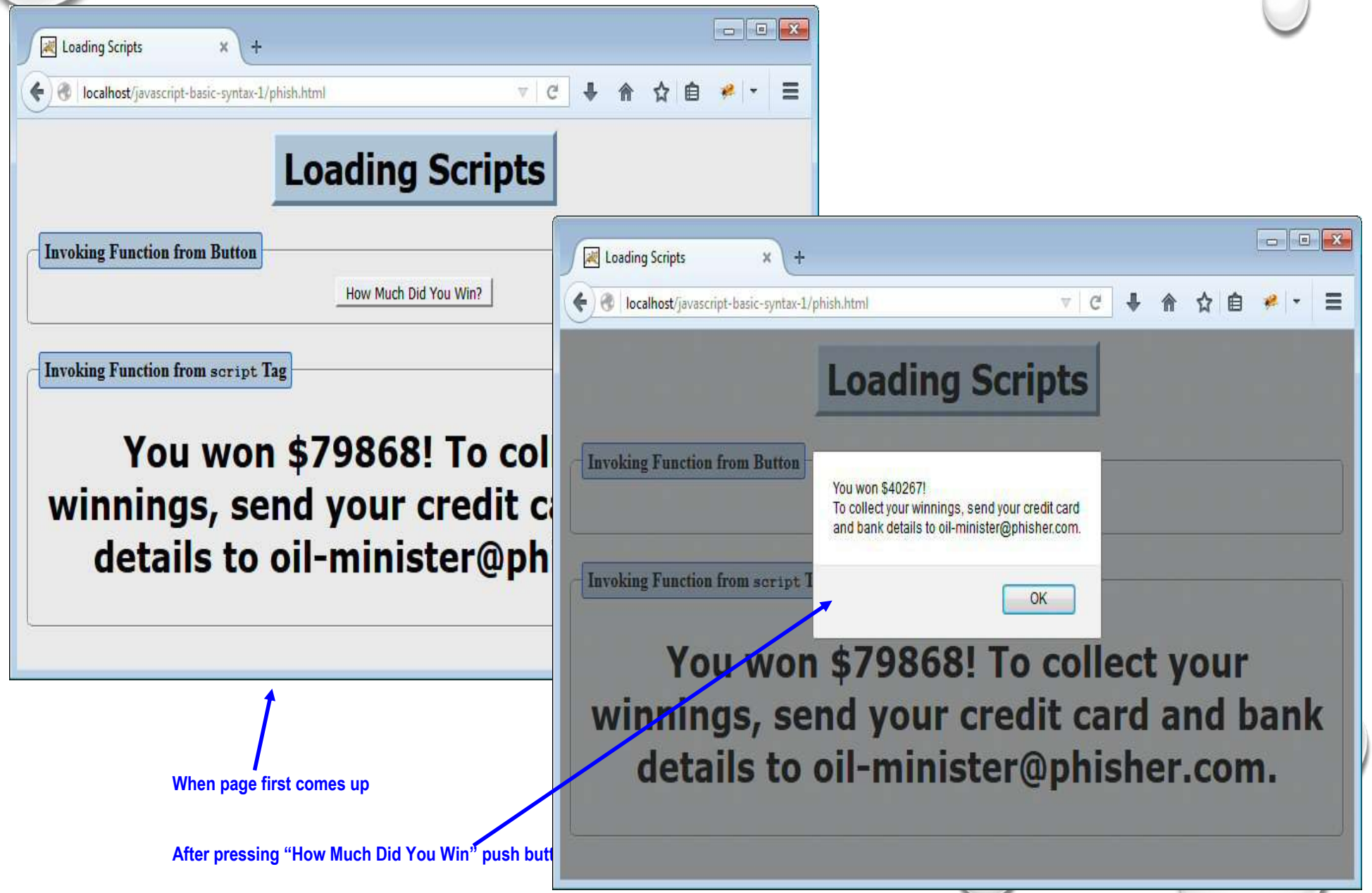
"document.write" inserts text into page at location that it is called

Example 2

```
<!doctype html><html>
<head><title>loading scripts</title>
...
<script src="scripts/code.js"></script>
</head>
<body>
...
   <input type="button" value="how much did you
 win?"
            onclick='showwinnings1()'/>
...
  <script>showwinnings2()</script>
...
</body></html>
```

Loads script shown on previous page

Example (Results)



When page first comes up

After pressing "How Much Did You Win" push butt

Exercise: battleship game

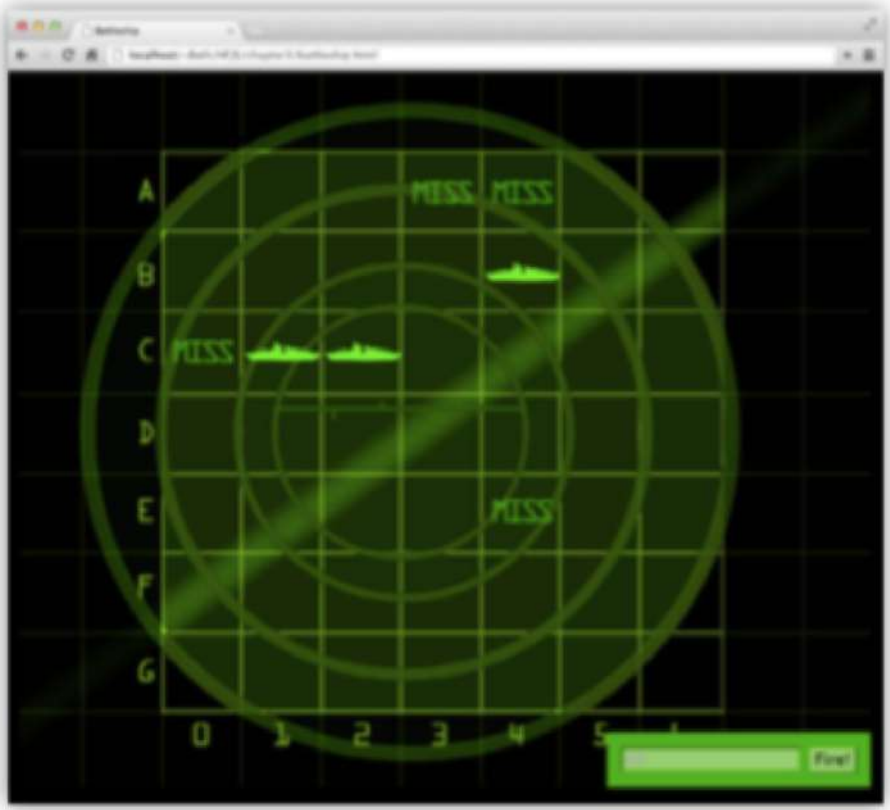
- **Goal:** Sink the browser's ships in the fewest number of guesses. You're given a rating, based on how well you perform
- **Setup**: When the game program is launched, the computer places ships on a virtual grid. When that's done, the game asks for your first guess
- **How you play:** *The browser will prompt you to enter a guess and you'll type in a grid location. In response to your guess, you'll see a result of "Hit", "Miss", or "You sank my battleship!" When you sink all the ships, the game ends by displaying your rating*

Simplified battleship

- *Objective: 7x7 graphical version with three ships, but…*
- We're going to start with a nice 1-D grid with seven locations and one ship to find

Instead of a 7x7 grid, like the one above, we're going to start with just a 1x7 grid. And, we'll worry about just one ship for now.

Notice that each ship takes up three grid locations (similar to the real board game).



High-level design

1 User starts the game

- **A** Game places a battleship at a random location on the grid.

2 Game play begins

Repeat the following until the battleship is sunk:

- **A** Prompt user for a guess ("2", "0", etc.)
- **B** Check the user's guess against the battleship to look for a hit, miss or sink.

3 Game finishes

Give the user a rating based on the number of guesses.

Details

- **Representing the ships**
  - Keep in mind the virtual grid
  - The user knows that the battleship is hidden in three consecutive cells out of a possible seven (starting at zero), the row itself doesn't have to be represented in code
- **Getting user input**
  - Use the **prompt** function. Whenever we need to get a new location from the user, we'll use prompt to display a message and get the input, which is just a number between 0 and 6, from the user
- **Displaying the results**
  - Use alert to show the output of the game

Sample game interaction

Battelship

The HTML for the Battleship game is super simple; we just need a page that links to the JavaScript code, and that's where all the action happens.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Battleship</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Play battleship!</h1>
    <script src="battleship.js"></script>
  </body>
</html>
```

We're linking to the JavaScript at the bottom of the <body> of the page, so the page is loaded by the time the browser starts executing the code in "battleship.js".

Battleship (2)

DECLARE three *variables* to hold the location of each cell of the ship. Let's call them location1, location2 and location3
DECLARE a *variable* to hold the user's current guess. Let's call it guess
DECLARE a *variable* to hold the number of hits.We'll call it hits and *set* it to 0
DECLARE a *variable* to hold the number of guesses.We'll call it guesses and *set* it to 0
DECLARE a *variable* to keep track of whether the ship is sunk or not. Let's call it isSunk and *set* it to false

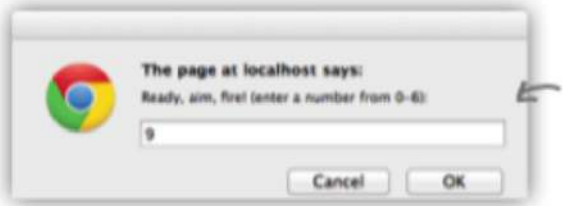
```
var randomLoc = Math.floor(Math.random() * 5);
var location1 = randomLoc;
var location2 = location1 + 1;
var location3 = location1 + 2;
var guess;
var hits = 0;
var guesses = 0;
var isSunk = false;
```

Battleship (3)

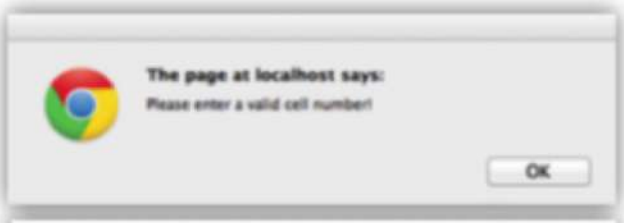
```
while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
        alert("Please enter a valid cell number!");
    } else {
        guesses = guesses + 1;
        if (guess == location1 || guess == location2 || guess == location3) {
            alert("HIT!");
            hits = hits + 1;
            if (hits == 3) {
                isSunk = true;
                alert("You sank my battleship!");
            }
        } else {
            alert("MISS");
        }
    }
}
var stats = "You took " + guesses + " guesses to sink the battleship, " +
            "which means your shooting accuracy was " + (3/guesses);
alert(stats);
```

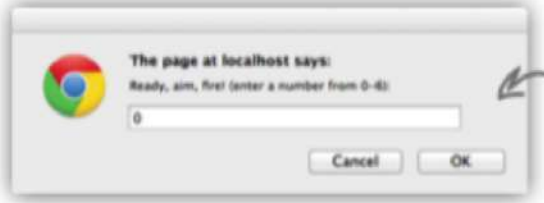
Test it!

Here's what our game interaction looked like.

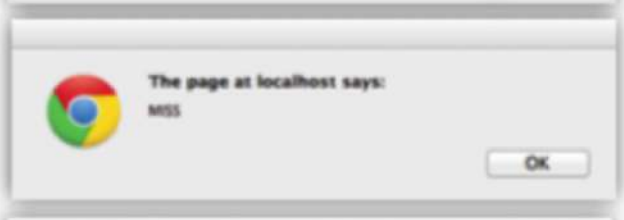


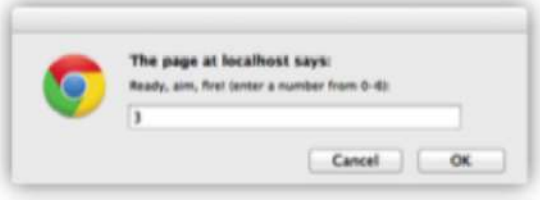
First we entered an invalid number, 9.



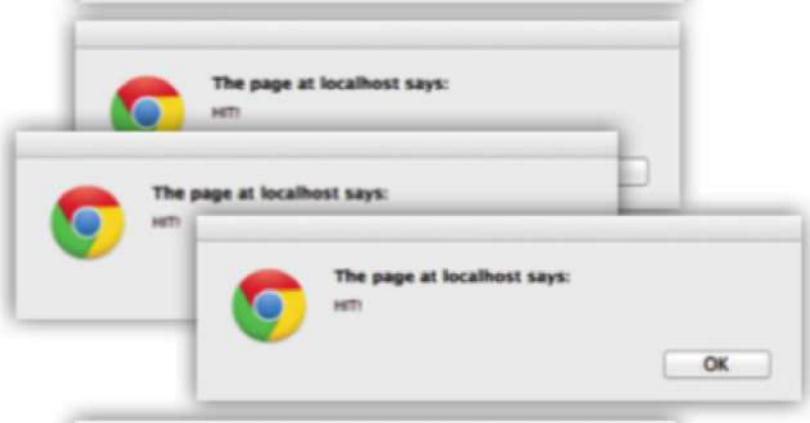


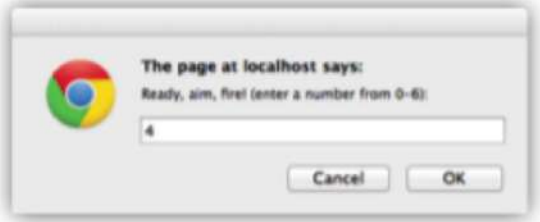
Then we entered 0, to get a miss.

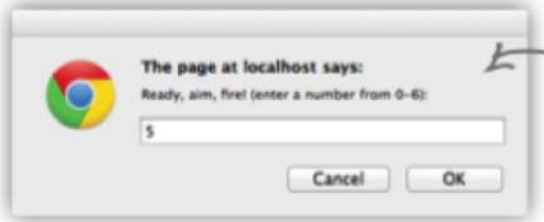




But then we get three hits in a row!







On the third and final hit, we sink the battleship.

And see that it took 4 guesses to sink the ship with an accuracy of 0.75.