

JavaScript bàsic

IES Son Ferrer

NOTA IMPORTANT:

Tot el que veurem a continuació ho teniu en:

<http://www.w3schools.com/js/>

Aquesta presentació és un resum actualitzat i modificat del que trobareu allà.

JS Tutorial

Què és Javascript? Per què estudiar JavaScript?

HTML + CSS + JavaScript

- JavaScript i Java són llenguatges diferents.
- JavaScript va ser inventat per Brendan Eich el 1995, i es va convertir en un estàndard ECMA en 1997.
- ECMAScript és el nom oficial. ECMAScript 2020 (versió 11, publicada el juny de 2020) és la darrera versió de JavaScript.

JavaScript Introduction

JavaScript per canviar contingut HTML:

```
document.getElementById("demo").innerHTML = "Hello JavaScript";
```

JavaScript per canviar atributs HTML:

```

```

JavaScript per canviar estils (CSS):

```
document.getElementById("demo").style.fontSize = "25px";
```

```
function changeImage() {
    var image =
document.getElementById('myImage');
    if (image.src.match("bulbon")) {
        image.src = "pic_bulboff.gif";
    } else {
        image.src = "pic_bulbon.gif";
    }
}
```

JavaScript Introduction

JavaScript per validar dades (formularis):

```
...  
<button type="button" onclick="myFunction()">Submit</button>  
...
```

```
<script>  
function myFunction() {  
    let x, text;  
    // Get the value of the input field with id="numb"  
    x = document.getElementById("numb").value;  
  
    // If x is Not a Number or less than one or greater than 10  
    if (isNaN(x) || x < 1 || x > 10) {  
        text = "Input not valid";  
    } else {  
        text = "Input OK";  
    }  
    document.getElementById("demo").innerHTML = text;  
}  
</script>
```

JS Where To

<script> ... </script>

Javascript en <head>:

```
<!DOCTYPE html>
<html>

<head>
  <script>
    function myFunction() {
      document.getElementById("demo").innerHTML = "Paragraph changed.";
    }
  </script>
</head>
<body>

  <h1>A Web Page</h1>
  <p id="demo">A Paragraph</p>
  <button type="button" onclick="myFunction()">Try it</button>

</body>
</html>
```

JS Where To

Javascript en **<body>**:

```
<!DOCTYPE html>
<html>
<body>

  <h1>A Web Page</h1>
  <p id="demo">A Paragraph</p>
  <button type="button" onclick="myFunction()">Try it</button>

  <script>
    function myFunction() {
      document.getElementById("demo").innerHTML = "Paragraph changed.";
    }
  </script>

</body>
</html>
```

JS Where To

Javascript en **arxiu independent**. Avantatges:

Separa codi i HTML.

Més fàcil de llegir i mantenir.

Arxius JavaScript en memòria cau poden accelerar càrregues de la pàgina.

```
<!DOCTYPE html>
<html>
<body>

    <script src="myScript.js"></script>

</body>
</html>
```


JS Output

Fent servir **window.alert()**:

```
<script>window.alert(5 + 6);</script>
```

Fent servir **document.write()** (apareix en pantalla el resultat d'executar-lo):

```
<script>document.write(5 + 6);</script>
```

Escrivint en un objecte HTML usant **innerHTML**.

```
<script>document.getElementById("demo").innerHTML = 5 +  
6;</script>
```

Escrivint en la consola del navegador amb **console.log()** (F12 i seleccionar en el menú).

```
<script>console.log(5 + 6);</script>
```

JS Quadres de diàleg

L'objecte **window** del navegador proporciona uns quadres de diàleg per a interactuar amb l'usuari.

- Missatges d'alerta: **alert("missatge")**. Mostra "missatge" en un quadre i l'usuari ha de clicar OK per avançar.
- Missatge de confirmació: **confirm("pregunta")**. Mostra "pregunta" amb 2 botons per acceptar (retorna `true`) o cancel·lar (retorna `false`)
- Quadre entrada de text: **prompt("missatge", valorPerDefecte)**.
Mostra el "missatge", un control de text per escriure i 2 botons. Si l'usuari accepta, el mètode retorna el valor introduït (o el `valorPerDefecte` si l'usuari no l'ha modificat). Si cancel·la, el mètode retorna `null`.

JS Syntax

- Un **programa** és una llista d'"**instruccions**" per ser "**executades**" per l'ordinador.
- JavaScript és un llenguatge de programació.
- En un llenguatge de programació, aquestes instruccions de programa es diuen "**sentències**" ("*statements*").
- Els programes JavaScript són **executats pel navegador web**.
- Les **sentències JavaScript** estan **separades per ;**

JS Syntax

- Es pot ometre un punt i coma en la majoria dels casos quan existeix un salt de línia (inserció automàtica de punt i coma):

```
alert('Hello')  
alert('World')
```

- Però hi ha casos en què una nova línia no significa un punt i coma. Per exemple:

```
alert(3 +  
1  
+ 2);
```

- Però hi ha situacions en què JavaScript "falla" a assumir un punt i coma en què realment és necessari:

```
alert("There will be an error")  
[1, 2].forEach(alert)
```

- **RECOMANACIÓ:** posar punts i coma entre sentències encara que estiguin separades per línies noves.

JS Statements

- Les sentències JavaScript estan compostes per: Valors (literals i variables), operadors (= + - * /), expressions (p. e.: 5 + 6 o 5 *10), paraules clau (var, if, for, etc.) i comentaris (// i /* ... */).
- JavaScript és **Case Sensitive**.
- JavaScript sol seguir **Camel Case**:
 - **Hyphens**: first-name, last-name, master-card, inter-city.
 - **Underscore**: first_name, last_name, master_card, inter_city.
 - **Camel Case**: firstName, lastName, masterCard, interCity.
- La codificació de caràcters de JavaScript és **Unicode**.

JS Statements

Blocs de codi: { ... }

```
function myFunction() {  
    document.getElementById("demo").innerHTML = "Hello Dolly.";  
    document.getElementById("myDIV").innerHTML = "How are you?";  
}
```

Paraules clau (són reservades!): break, continue, debugger, do ... while, function, if, else, return switch, try ... catch, var, etc.

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS HOME -> Temps: 10 min (sense test)

JS Introduction -> 20 min

JS Where To -> 20 min

JS Output -> 30 min

JS Syntax -> 25 min

Fer exercicis 1, 2 i 3 de DWEC_P02.

JS Comentarís

En una l nia:

```
let x = 5;    // Declare x, give it the value of 5
// Change heading:
document.getElementById("myH").innerHTML = "My First
Page";
```

Multi-l nia:

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
```


JS Comentarís

Comentarís per evitar execucions:

```
//document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first  
paragraph.";
```

```
/*  
document.getElementById("myH").innerHTML = "My First Page";  
document.getElementById("myP").innerHTML = "My first  
paragraph.";  
*/
```

JS Variables

Declaració (es poden declarar sense assignar valor i el valor que retornarien és “**undefined**”):

```
let price1 = 5;  
let price2 = 6;  
let total = price1 + price2;  
let person = "John Doe", carName = "Volvo", price = 200;
```

Les variables han de tenir un identificador únic:

- Es poden fer servir lletres, dígitos, _ i \$.
- Han de començar per una lletra (o per \$ i _ encara que això és en desús).
- Són CASE SENSITIVE.
- No es poden emprar paraules reservades.

JS Variables

L'operador d'assignació és = (== és "igual a" en comparacions lògiques).

```
let price1 = 5;  
let price2 = 6;  
let total = price1 + price2;
```

Tipus de dades:

```
let x = true; var y = false; // Booleans  
let length = 16; var pi = 3.14; // Numbers  
let lastName = "Johnson"; // String  
let cars = ["Saab", "Volvo", "BMW"]; // Array  
let x = {firstName:"John", lastName:"Doe"}; // Objecte
```

JS Operators

Operadors aritmètics:

Operador	Descripció
+	Suma
-	Resta
*	Multiplicació
/	Divisió
%	Mòdul
++	Increment
--	Decrement

Operadors d'assignació:

Operador	Exemple	Igual a
=	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

JS Operators

Operadors de cadenes de caràcters:

Operador	Descripció
+	Concatena
+=	Concatena i assigna
Nota:	Si es concatena strings i nombre el resultat és string.

Operadors lògics:

Operador	Descripció
==	igual a
===	igual valor i igual tipus
!=	no igual
!==	no igual valor o no igual tipus
>	major que
<	menor que
>=	major o igual que
<=	menor o igual que

JS Tipus de dades

El concepte de tipus de dades és important per tractar expressions com la següent (fa *autocasting*):

```
let x = 16 + "Volvo";
```

JS avalua les expressions d'esquerra a dreta, per tant:

```
let x = 16 + 4 + "Volvo"; -> "20Volvo"
```

```
let x = "Volvo" + 16 + 4; -> "Volvo164"
```

JS té tipus dinàmics (la mateixa variable pot fer-se servir amb diferents tipus):

```
let x; // Valor undefined
```

```
let x = 5; // Valor Number
```

```
let x = "John"; // Valor String
```

JS Tipus de dades

JS Strings:

```
let carName = "Volvo XC60";    // Using double quotes
let carName = 'Volvo XC60';    // Using single quotes
```

JS Numbers (dos tipus: amb o sense decimals):

```
let x1 = 34.00;                // Amb decimals
let x2 = 34;                   // Sense decimals

let y = 123e5;                 // Exponencial: 12300000
let z = 123e-5;               // Exponencial: 0.00123
```

JS booleans:

```
let x = true; let y = false;
```

JS Tipus de dades

JS Arrays:

```
let cars = ["Saab", "Volvo", "BMW"];  
cars [0] -> "Saab"  
cars [1] -> "Volvo"  
cars [2] -> "BMW"
```

JS Objects:

```
let person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};  
  
person.firstName -> "John"  
person.lastName -> "Doe"  
person.age -> 50  
person.eyeColor -> "blue"
```


JS Tipus de dades

L'operador **typeof**:

```
typeof "John"           // Retorna string
typeof 3.14              // Retorna number
typeof false             // Retorna boolean
typeof [1,2,3,4]         // Retorna object
typeof {name: 'John', age:34} // Retorna object
```

JS Tipus de dades

El valor/tipus (les dues coses) **Undefined**:

```
let person;           //El valor és undefined, el tipus és undefined
person = undefined;   //El valor és undefined, el tipus és undefined
```

Undefined no té res a veure amb els valor "", 0 o 0.0

També té el valor **Null** que es suposa que és una cosa que no existeix (buit, res o "nada"). Es pot fer servir per inicialitzar un objecte (objecte buit):

```
let person = null; // El valor és null, però el tipus és
object
```

Encara que també pot inicialitzar-se un objecte a *undefined*:

```
let person = undefined; // El valor és undefined i el tipus és
undefined
```

JS Tipus de dades

Diferències entre **Undefined** i **Null**:

```
typeof undefined           // undefined
```

```
typeof null                // object
```

```
null === undefined        // false
```

```
null == undefined        // true
```

O sia, `undefined` i `null` són iguals en valor, però diferents en tipus.

JS Tipus de dades

Més endavant veurem **com es converteix explícitament de *number* a *string* i viceversa:**

parseFloat()

parseInt()

toString()

toExponential()

toFixed()

toPrecision()

JS Funcions

Funció: Bloc de codi destinat a complir una tasca particular. Concepte de **encapsulació** i **reutilització de codi (reusabilitat)**.

JS no té *procedures* o *subroutines* (només *functions*).

Sintaxi:

```
function nom(paràmetre1, paràmetre2, paràmetre3) {  
    // Codi a ser executat  
    return valor; // Valor que retorna la funció  
}                // (dels tipus que hem vist)
```

No s'executarà tot el que hagi després de *return valor*;

JS Funcions

```
function toCelsius(fahrenheit) { // Definició de la funció
    return (5/9) * (fahrenheit-32);
}
// Cridada a la funció que es canviarà pel valor retornat
document.getElementById("demo").innerHTML = toCelsius(32);
```

Altre exemple d'ús:

```
let text = "The temperature is " + toCelsius(32) + " Centigrade";
```

Cridar a la funció sense fer servir () retorna la seva definició (el seu codi):

```
document.getElementById("demo").innerHTML = toCelsius;
```

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

- JS Comments -> Temps: 4 min

- JS Variables -> Temps: 15 min (fer exercicis del final)

- JS Operators, JS Arithmetic, JS Assignment -> Temps: 10 min (fer exercicis)


- JS Data Types -> Temps: 20 min

- JS Functions -> Temps: 20 min (fer exercicis del final)

Fer exercici 4 de DWEC_P02.

JS Objectes

Tots els cotxos tenen les mateixes propietats (***properties***) i els mateixos mètodes (***methods***).

Object	Properties	Methods
	car.name = Fiat	car.start()
	car.model = 500	car.drive()
	car.weight = 850kg	car.brake()
	car.color = white	car.stop()

```
let car = {type:"Fiat", model:500, weight:850, color:"white"};
```


JS Objectes

Es poden accedir a les ***properties*** amb:

objectName.propertyName

Exemple: person.lastName = "González";

Exemple: x = person.lastName;

objectName["propertyName"]

Exemple: person["lastName"] = "González";

Exemple: x = person["lastName"];

Es poden cridar els ***methods*** amb:

objectName.methodName(parameters)

Exemple: name = person.fullName();

Exemple: person.setFirstName("Sergi");

JS Objectes

```
<script>
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  },
  setFirstName : function(c) {
    this.firstName = c;
  },
  setLastName : function(c) {
    this.lastName = c;
  }
};
document.getElementById("demo").innerHTML = person.fullName();
person.setFirstName("Sergi");
alert(person.firstName);
</script>
```

Veurem més coses sobre objectes més envant!

JS Scope (àmbit)

En JS l'àmbit (*scope*) és el conjunt de variables, objectes i funcions als que tens accés.

JS té *function scope*: l'àmbit canvia dins les funcions. Les **variables declarades dins d'una funció són locals** a aquesta (poden fer servir el mateixos noms que variables fora de la funció però **no són** la mateixa).

```
// el codi aquí no pot fer servir carName
function myFunction() {
    let carName = "Volvo";
    // el codi aquí sí pot fer servir carName
}
```

JS Scope (àmbit)

Una **variable global** té **àmbit global** (*global scope*): Tot el codi (funcions incloses) poden accedir a elles.

```
let carName = " Volvo";  
// el codi aquí pot fer servir carName  
function myFunction() {  
    // el codi aquí pot fer servir carName  
    // si hi ha una en local que es digui carName, fer servir  
    // per accedir a la global: window.carName / this.carName  
}
```

JS Scope (àmbit)

Si tu assignes un valor a una **variable** que no ha estat declarada es converteix **automàticament** en d'àmbit global:

```
// el codi aquí pot fer servir carName
function myFunction() {
    carName = "Volvo";
    // el codi aquí pot fer servir carName
}
```

JS var vs let

La declaració **var** és similar a **let**. La majoria de vegades podem substituir **let** per **var** o viceversa i tot funciona igual.

Però internament **var** molt diferent. Generalment no s'utilitza en scripts moderns, però encara s'amaga en els antics.

És important entendre les diferències a l'hora de migrar scripts antics de **var** a **let**, per evitar errors estranys.

JS var vs let

- **var** no té àmbit de bloc

```
> {  
  let x=9;  
}  
console.log(x)  
✖ ▶ Uncaught ReferenceError: x is not defined
```

```
> {  
  var x=9;  
}  
console.log(x)  
9
```

let té àmbit de bloc i **var** sobreviu fora del bloc

l'àmbit de **var** és el context d'execució, que pot ser **global** (si se declara fora de tota funció), i abasta tot l'arxiu, o pot ser relatiu a una funció (**local**) si se declara dins ella

JS var vs let

- **var** tolera les redeclaracions

```
> let usuari;  
let usuari;
```

✖ ▶ Uncaught SyntaxError: Identifier 'usuari' has already been declared

```
> var nom;  
var nom;
```

- Les variables «**var**» poden ser declarades després de ser usades

```
> fruita = 'poma';  
var fruita;
```

```
> verdura = 'bleda';  
let verdura;
```

✖ ▶ Uncaught ReferenceError: Cannot access 'verdura' before initialization

JS var vs let

IMPORTANT: Les declaracions s'elevan, però NO les assignacions.

```
> alert(saluda);  
var saluda = 'bon dia';
```

undefined

D'acord

JS Scope (àmbit)

Temps de vida de les funcions (*lifetime*):

El temps de vida de les funcions comença quan aquestes són declarades.

Les **variables locals** són esborrades quan es completa l'execució de la funció.

Les **variables globals** s'esborren quan tanques la pàgina.

En JS, l'**àmbit global** és l'entorn JS per complet. En HTML, l'àmbit global és la finestra (*window object*): totes les variables globals pertanyen al *window object*.

JS Esdeveniments

Els **esdeveniments** (*events*) son “coses” que els hi passen als elements HTML.

Els esdeveniments poden ser:

- Coses que fa el navegador.

- Coses que fa l'usuari.

Exemples d'esdeveniments:

- Una pàgina ha acabat de carregar.

- Ha canviat el camp d'un formulari.

- S'ha clicat un botó.

- Etc.

JS Esdeveniments

JS te deixa **executar codi** quan succeeixen esdeveniments (nom tècnic: ***event handlers***). Exemple:

```
<some-HTML-element some-event="some JavaScript">
```

```
<button  
  onclick='getElementById("demo").innerHTML=Date()'>The  
  time is?</button>
```

```
<button onclick="displayDate()">The time is?</button>
```

JS Esdeveniments

Normalment feim servir els següents esdeveniments:

Esdeveniment	Descripció
onchange	Quan canvia l'element HTML.
onclick	Quan l'usuari clica l'element HTML.
onmouseover	Quan l'usuari mou el punter del ratol·lí sobre l'element HTML.
onmouseout	Quan l'usuari mou el ratol·lí fora de l'element HTML.
onkeydown	Quan l'usuari espitja una tecla.
onload	Quan el navegador acaba de carregar la pàgina.

JS Esdeveniments

Coses que podem fer amb *event handlers*:

- Coses a fer cada vegada que carrega la pàgina.

- Coses a fer quan es tanca la pàgina.

- Accions a realitzar quan un usuari clica un botó.

- Contingut que ha de ser verificat quan un usuari entra dades (típicament validació de formularis).

Més envant veurem més coses sobre això.

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS Objects -> Temps: 15 min (fer exercicis del final)

JS Scope -> Temps: 20 min (intentar entendre perquè és complicat).

JS Events -> Temps: 15 min (fer exercicis del final)

Fer exercici 5 de DWEC_P02.

JS Strings

Tipus de dades per emmagatzemar i manipular text. Exemple:

```
let answer = "He is called 'Johnny'";
```

```
let answer = 'He is called "Johnny"';
```

Els strings tenen la propietat **length** (mida de l'*string*).

```
let txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
let sln = txt.length;
```

Utilitzar \ (backslash) per a caràcters especials:

```
let x = 'It\'s alright';
```

```
let y = "We are the so-called \"Vikings\" from the north."
```


JS Strings

Caràcters especials:

Codi	Sortida
\'	Comilla simple (<i>single quote</i>)
\"	Comilla doble (<i>double quote</i>)
\\	Contrabarra (<i>backslash</i>)
\n	Nova línia (<i>new line</i>)
\r	Retorn de carro (<i>carriage return</i>)
\t	Tabulador (<i>tab</i>)
\b	Retrocedir espai (<i>backspace</i>)
\f	Avanç de pàgina (<i>form feed</i>)

JS Strings

Per fer fàcil la lectura convé no escriure línies de més de 80 caràcters. Diferents formes per fer això:

```
document.getElementById("demo").innerHTML =  
    "Hello Dolly.";
```

```
document.getElementById("demo").innerHTML = "Hello \  
Dolly!";
```

```
document.getElementById("demo").innerHTML = "Hello" +  
    "Dolly!";
```

```
document.getElementById("demo").innerHTML = \  
    "Hello Dolly!";
```

JS Strings

Els *strings* poden ser objectes (`let firstName = new String("John")`) en lloc `let firstName = "John"`).

No convé fer feina amb *strings* com a objectes. Definir-los com una variable normal!

```
let x = "John";  
let y = new String("John");  
// (x == y) és true perquè x i y tenen valors iguals  
// (x === y) és false perquè x i y tenen diferents tipus  
let x = new String("John");  
let y = new String("John"); // x i y són objectes diferents  
// (x == y) és false perquè els objectes no poden ser  
    comparats
```

JS Strings

Propietats dels *strings*:

Propietat	Descripció
constructor	Retorna la funció que va crear el prototipus d'objecte String.
length	Retorna la longitud de l' <i>string</i> .
prototype	Et permet afegir propietats i mètodes a un objecte.

JS *Strings* Methods

Mètodes dels *strings*:

Mètode	Descripció
charAt()	Retorna el caràcter que es troba en la posició passada com a paràmetre.
charCodeAt()	Retorna el Unicode del caràcter que es troba en la posició passada com a paràmetre.
concat()	Uneix dos o més strings, i retorna una còpia dels strings units.

JS *Strings* Methods

Mètodes dels *strings*:

Mètode	Descripció
fromCharCode()	Converteix valors Unicode en caràcters.
indexOf()	Retorna la posició de la primera ocurrència trobada del valor especificat com a string.
lastIndexOf()	Retorna la posició de la darrera ocurrència trobada del valor especificat com a string.

JS *Strings* Methods

Mètodes dels *strings*:

Mètode	Descripció
match()	Cercar en un string una expressió regular i retorna les coincidències.
replace()	Cerca un valor dins un string i retorna un nou string amb aquest valor reemplaçat per un altre.
search()	Cerca en un string y retorna la posició de la coincidència.

JS *Strings* Methods

Mètodes dels *strings*:

Mètode	Descripció
slice()	Extreu d'un string d'una posició fins a una altra posició.
split()	Xapa un string en un array de substrings.
substr()	Extreu d'un string d'una posició fins un nombre de caràcters.

JS *Strings* Methods

Mètodes dels *strings*:

Mètode	Descripció
substring()	Extreu d'un string d'una posició fins un nombre de caràcters.
toLocaleLowerCase()	Converteix a minúscules (tenint en compte la configuració regional de l'equip client).
toLocaleUpperCase()	Converteix a majúscules (tenint en compte la configuració regional de l'equip client).

JS *Strings* Methods

Mètodes dels *strings*:

Mètode	Descripció
toLowerCase()	Converteix un string a minúscula.
toString()	Retorna el valor d'un objecte String.
toUpperCase()	Converteix un string a majúscula.
trim()	Lleva els espais en blanc del principi i del final de l'string.
valueOf()	Retorna el valor primitiu d'un objecte String.

JS Strings Methods

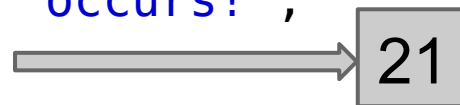
indexOf(): Cercar un string dins un string (primera ocurrència):

```
let str = "Please locate where 'locate' occurs!";  
let pos = str.indexOf("locate");
```



lastIndexOf(): Cercar un string dins un string (darrera ocurrència):

```
let str = "Please locate where 'locate' occurs!";  
let pos = str.lastIndexOf("locate");
```



JS Strings Methods

search(): Cercar un string dins un string
(primera ocurrència):

```
let str = "Please locate where 'locate' occurs!";  
let pos = str.search("locate");
```



7

Si vos fixau `indexOf()` i `search()` pareixen iguals, però aquest darrer accepta expressions regulars (ho veurem més endavant).

JS Strings Methods

slice(): Extreure una part d'un string (d'una posició inicial a una final):

```
let str = "Apple, Banana, Kiwi";  
let res = str.slice(7,13);
```



Banana

slice(): Si els valors de la posició són negatius es compta del final:

```
let str = "Apple, Banana, Kiwi";  
let res = str.slice(-12,-6);
```



Banana

JS Strings Methods

slice(): Si s'omet el segon paràmetre
s'aplicarà fins al final de l'string:

```
let res = str.slice(7);
```



Banana, Kiwi

slice(): O comptant des del final...

```
let res = str.slice(-12);
```



Banana, Kiwi

JS Strings Methods

substring(): és similar a slice(). La diferència és que no accepta índexos negatius.

```
let str = "Apple, Banana, Kiwi";  
let res = str.substring(7,13);
```



Banana

substr(): és similar a slice(). La diferència és que el segon paràmetre especifica la longitud de la part extreta.

```
let str = "Apple, Banana, Kiwi";  
let res = str.substr(7,6);
```



Banana

JS Strings Methods

replace(): Reemplaza un string especificat per un altre.

```
str = "Please visit Microsoft!";  
let n = str.replace("Microsoft", "W3Schools");
```



Please visit W3Schools!

toUpperCase(): Passar a majúscula.

```
let text1 = "Hello World!";  
let text2 = text1.toUpperCase();
```



HELLO WORLD!

JS Strings Methods

toLowerCase(): Passar a minúscula.

```
let text1 = "Hello World!";  
let text2 = text1.toLowerCase();
```



hello world!

A horizontal arrow points from the `toLowerCase()` method call in the code above to a gray rectangular box containing the text "hello world!".

concat(): Uneix dos o més strings.

```
let text1 = "Hello";  
let text2 = "World";  
text3 = text1.concat(" ", text2);
```



Hello World!

A horizontal arrow points from the `concat()` method call in the code above to a gray rectangular box containing the text "Hello World!".

JS Strings Methods

charAt(): Retorna el caràcter especificat per un índex.

```
let str = "HELLO WORLD";  
str.charAt(0);
```



charCodeAt(): Retorna el codi unicode del caràcter especificat per un índex.

```
let str = "HELLO WORLD";  
str.charCodeAt(0);
```



JS Strings Methods

Accedir a un string com un array és insegur per:

No funciona en alguns versions antigues de navegadors
(p. e. IE5, IE6 i IE7).

Fa que els strings paresquin arrays, i no ho són.

~~str[0] = "H"~~ no dóna error (però no funciona)

En tot cas:

```
let str = "HELLO WORLD";
```

```
str[0];
```



JS *Strings Methods*

split(): Per convertir un string a array:

```
let txt = "a,b,c,d,e";    // String
txt.split(",");           // Xapar per comes
txt.split(" ");           // Xapar per espais
txt.split("|");           // Xapar per pipes
txt.split("");            // Xapar per caràcters
```

JS *Strings Methods*

Referència completa pels strings:

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

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS Strings -> Temps: 15 min (fer exercicis del final).

JS String Methods -> Temps: 30 min (fer exercicis del final).

Fer exercicis 6, 7 i 8 de DWEC_P02.

Nombres

JS només té el **tipus de dades “number”**.

Els nombres es poden escriure **amb decimals o sense i en format exponencial**:

```
let x = 123e5;    // 12300000
```

```
let y = 123e-5;   // 0.00123
```

```
let x = 34.00;
```

```
let y = 34;
```

Nombres

En realitat sempre s'emmagatzemen com a nombres de punt flotant de doble precisió (64 bits i definit en IEEE 754).

Valor	Exponent	Signe
52 bits (0 - 51)	11 bits (52 - 62)	1 bit (63)

Nombres

Els nombres sencers tenen una precisió de 15 dígit:

```
let x = 9999999999999999;    // x serà 9999999999999999
let y = 9999999999999999;    // y serà 1000000000000000000
```

Els nombre màxim de decimals és 17, però la precisió aritmètica del punt flotant no és precisa al 100%:

```
let x = 0.2 + 0.1;           // x serà 0.3000000000000000004
Solució: let x = (0.2 * 10 + 0.1 * 10) / 10; // x serà 0.3
```

Nombres

Els hexadecimals es representen començant per 0x:

```
let x = 0xFF;           // x serà 255
```

Els sistemes numerals binaris i octals poques vegades s'utilitzen, però també són compatibles amb els prefixos 0b i 0o:

```
let a = 0b11111111; // 255 en binari
```

```
let b = 0o377; // 255 en octal
```

```
alert( a == b ); // true
```

Nombres

Pots fer servir el mètode **toString()** per convertir els nombres a un text en hexadecimal, octal o binari:

```
var myNumber = 128;  
myNumber.toString(16);    // retorna 80  
myNumber.toString(8);     // retorna 200  
myNumber.toString(2);     // retorna 10000000
```

Nombres

Infinity (∞) o **-Infinity** ($-\infty$) és el valor en JS per representar un nombre superior al nombres més llarg possible:

```
let myNumber = 2;
while (myNumber != Infinity) {    // Executa fins a desbordar
    (overflow) la variable
    myNumber = myNumber * myNumber;
}
```

La divisió per 0 genera **Infinity**:

```
let x = 2 / 0;           // x serà Infinity
let y = -2 / 0;          // y serà -Infinity
```

Nombres

Infinity és de tipus number:

```
typeof Infinity;           // retorna "number"
```

NaN - Not a Number: Paraula reservada per indicar que un valor no és un nombre.

```
let x = 100 / "Apple";    // x serà NaN (Not a Number)  
let x = 100 / "10";       // x serà 10
```

Funció **isNaN()**:

```
let x = 100 / "Apple";  
isNaN(x);               // retorna true perquè x és NaN
```

Nombres

Fent servir **NaN** en operacions matemàtiques:

```
let x = NaN;  
let y = 5;  
let z = x + y;           // z serà NaN
```

NaN és de tipus number:

```
typeof NaN;             // retorna "number"
```

Nombres

Propietats del mètodes (p. e. **let** x = Number.MAX_VALUE;) i no **let** ~~y = x.MAX_VALUE; // y serà undefined~~):

Propietat	Descripció
MAX_VALUE	Retorna el nombre més gran possible en JS.
MIN_VALUE	Retorna el nombre més petit possible en JS.
NEGATIVE_INFINITY	Representa -Infinity (retornado al producirse overflow)
NaN	Representa un valor "Not-a-Number".
POSITIVE_INFINITY	Representa +Infinity (retornado al producirse overflow)

Nombres

Les **funcions globals** (o mètodes globals) de JS es poden fer servir amb qualsevol tipus de dades:

Mètode	Descripció
Number()	Se li passa un argument i el retorna com a nombre.
parseFloat()	Se li passa un argument i el retorna com a nombre de coma flotant.
parseInt()	Se li passa un argument i el retorna com a nombre sencer (Integer).

Nombres

Mètodes de JS per a nombres:

Method	Description
toString()	Retorna un nombre com a string.
toExponential()	Retorna un string amb un nombre arrodonit i escrit fent servir notació exponencial.
toFixed()	Retorna un string amb un nombre arrodonit i escrit fent servir el número indicat de decimals.
toPrecision()	Retorna un string amb un nombre escrit en una longitud determinada.
valueOf()	Retorna un nombre com a nombre.

NOTA: El mètodes de JS per a nombres retornen un valor, però no canvien la variable original.

Nombres

El mètode `toString()`:

```
let x = 123;
```

```
x.toString();           // retorna "123" de la variable x
```

```
(123).toString();       // retorna "123" del literal 123
```

```
(100 + 23).toString();  // retorna "123" de la expressió  
100 + 23
```

Nombres

El mètode **toExponential()**:

```
let x = 9.656;
```

```
x.toExponential(2);    // retorna 9.66e+0
```

```
x.toExponential(4);    // retorna 9.6560e+0
```

```
x.toExponential(6);    // retorna 9.656000e+0
```

Nombres

El mètode **toFixed()**:

```
let x = 9.656;
```

```
x.toFixed(0);           // retorna 10
```

```
x.toFixed(2);           // retorna 9.66
```

```
x.toFixed(4);           // retorna 9.6560
```

```
x.toFixed(6);           // retorna 9.656000
```

Nombres

El mètode **toPrecision()**:

```
let x = 9.656;
```

```
x.toPrecision();           // retorna 9.656
```

```
x.toPrecision(2);         // retorna 9.7
```

```
x.toPrecision(4);         // retorna 9.656
```

```
x.toPrecision(6);         // retorna 9.65600
```

Nombres

Convertint variables a nombres:

Mètode **Number ()**

Mètode **parseInt ()**

Mètode **parseFloat ()**

Els tres mètodes anteriors no són mètodes de Number, sinó que són mètodes de JS.

Nombres

Mètode Number ():

```
x = true;
Number(x);           // retorna 1
x = false;
Number(x);           // retorna 0
x = new Date();
Number(x);           // retorna 1404568027739
x = "10"
Number(x);           // retorna 10
x = "10 20"
Number(x);           // retorna NaN
```

Nombres

Mètode `parseInt()`:

`parseInt("10");` `// retorna 10`

`parseInt("10.93");` `// retorna 10`

`parseInt("10 20 30");` `// retorna 10`

`parseInt("10 years");` `// retorna 10`

`parseInt("years 10");` `// retorna NaN`

Nombres

Mètode `parseFloat()`:

```
parseFloat("10");           // retorna 10
```

```
parseFloat("10.33");        // retorna 10.33
```

```
parseFloat("10 20 30");     // retorna 10
```

```
parseFloat("10 years");     // retorna 10
```

```
parseFloat("years 10");     // retorna NaN
```

Nombres

Mètode `valueOf()`:

```
let x = 123;
```

```
x.valueOf();           // retorna 123 de la variable x
```

```
(123).valueOf();       // retorna 123 del literal 123
```

```
(100 + 23).valueOf();  // retorna 123 de la expressió 100  
+ 23
```

Nombres

Referència completa per Number:

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

L'objecte Math

L'objecte Math te permet realitzar tasques relacionades amb les Matemàtiques.

Crear nombre aleatori:

```
Math.random();           // retorna un nombre aleatori
```

Trobar el valor mínim:

```
Math.min(0, 150, 30, 20, -8, -200);    // retorna -200
```

Trobar el valor màxim:

```
Math.max(0, 150, 30, 20, -8, -200);    // retorna 150
```

L'objecte Math

Arrodonir a sencer:

```
Math.round(4.7);           // retorna 5
```

```
Math.round(4.4);           // retorna 4
```

Arrodonir a sencer cap amunt:

```
Math.ceil(4.4);            // retorna 5
```

Arrodonir a sencer cap abaix:

```
Math.floor(4.7);           // retorna 4
```

Es poden combinar:

```
Math.floor(Math.random() * 11);
```

```
// retorna un nombre aleatori entre 0 i 10
```

L'objecte Math

Constants de l'objecte Math:

Math.E	// retorna el nombre d'Euler
Math.PI	// retorna PI
Math.SQRT2	// retorna l'arrel quadrada de 2
Math.SQRT1_2	// retorna l'arrel quadrada d'1/2
Math.LN2	// retorna el logaritme neperià de 2
Math.LN10	// retorna el logaritme neperià de 10
Math.LOG2E	// retorna el logaritme en base 2 d'E
Math.LOG10E	// retorna el logaritme en base 10 d'E

L'objecte Math

Mètodes de l'objecte Math:

Mètode	Descripció
abs(x)	Retorna el valor absolut d'x.
acos(x)	Retorna l'arccosinus d'x, en radians
asin(x)	Retorna l'arcsinus d'x, en radians
atan(x)	Retorna l'arctangent d'x com a valor numèric entre $-\pi/2$ i $\pi/2$ radians.

L'objecte Math

Mètodes de l'objecte Math:

Mètode	Descripció
<code>atan2(y,x)</code>	Retorna l'arctangent del concient dels seus arguments.
<code>ceil(x)</code>	Retorna x, arrodonit cap amunt al sencer més pròxim.
<code>cos(x)</code>	Retorna el cosinus d'x (x és en radians)

L'objecte Math

Mètodes de l'objecte Math:

Method	Description
<code>exp(x)</code>	Retorna el valor d'E ^x
<code>floor(x)</code>	Retorna x, arrodonit cap avall al sencer més pròxim.
<code>log(x)</code>	Retorna el logarítme neperià (base E) de x

L'objecte Math

Mètodes de l'objecte Math:

Method	Description
<code>max(x,y,z,...,n)</code>	Retorna el nombre amb el valor més alt
<code>min(x,y,z,...,n)</code>	Retorna el nombre amb el valor més baix
<code>pow(x,y)</code>	Retorna x^y
<code>random()</code>	Retorna un nombre aleatori entre 0 i 1

L'objecte Math

Mètodes de l'objecte Math:

Method	Description
round(x)	Arrodoneix x al sencer més pròxim.
sin(x)	Retorna el sinus d'x (x és en radians)
sqrt(x)	Retorna l'arrel quadrada d'x
tan(x)	Retorna la tangent d'un angle

L'objecte Math

Referència completa per l'objecte **Math**:

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

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS Numbers -> Temps: 15 min (fer exercicis del final).

JS Numbers Methods -> Temps: 20 min.

JS Math -> Temps: 20 min (fer exercicis del final).

Fer exercicis 10 i 11 de DWEC_P02.

JS Dates

En JS les dates es poden escriure com a string o com a number:

String: Tue Jun 23 2015 17:15:39 GMT+0200 (CEST)

Number: 1435072539608

Les dates escrites com a number especifiquen el nombre de milisegons desde l'1 de gener de 1970 a les 00:00:00.

JS Dates

Exemple:

```
<p id="demo"></p>
```

```
<script>
```

```
    document.getElementById("demo").innerHTML = Date();
```

```
</script>
```

El codi anterior mostraria una pàgina amb la data actual (en el moment que escric això és 23/6/15 17:20):

Tue Jun 23 2015 17:20:17 GMT+0200 (CEST)

JS Dates

Creant objectes “Date”:

Una data en JS consisteix en any, mes, dia, hores, minuts, segons i milisegons.

Les dates es creen amb el constructor `new Date()`. Quatre formes d’inicialitzar una data:

`new Date()` // agafa data/hora del moment d’execució

`new Date(milliseconds)`

`new Date(dateString)`

`new Date(year, month, day, hours, minutes, seconds, milliseconds)`

Nota important: Com a valor sencer, el mes es representa de 0 a 11, amb 0=gener i 11=desembre.

JS Dates

Exemple de creació d'objectes "Date":

Amb string: `var d = new Date("October 13, 2014 11:13:00");`

En milisegons: `var d = new Date(86400000);`

En l'exemple anterior es mostraria: Fri Jan 02 1970
01:00:00 GMT+0100 (CET)

Zero milisegons seria: 01 January 1970 00:00:00 UTC

Especificant any, mes, etc.: `var d = new Date(99,5,24,11,33,30,0);`

En l'exemple anterior es mostraria: Thu Jun 24 1999
11:33:30 GMT+0200 (CEST)

Especificant any, mes, etc.: `var d = new Date(99,5,24);`

En l'exemple anterior es mostraria: Thu Jun 24 1999
00:00:00 GMT+0200 (CEST)

JS Dates

Amb els objectes **Date** es tenen disponible tota una sèrie de mètodes que es veuran a diapositives posteriors:

`toString()`

`toUTCString()`

`toDateString()`

`...`

JS Date Formats

Hi ha quatre tipus de format de data vàlids en JS:

1. Dates ISO (sintaxi ISO 8601 YYYY-MM-DD):

```
var d = new Date("2015-03-25");
```

```
var d = new Date("2015-03");
```

```
var d = new Date("2015");
```

```
var d = new Date("2015-03-25T12:00:00");
```

JS Date Formats

Hi ha quatre tipus de format de data vàlids en JS:

2. Dates llargues:

```
var d = new Date("Mar 25 2015");  
var d = new Date("25 Mar 2015");  
var d = new Date("2015 Mar 25");  
var d = new Date("January 25 2015");  
var d = new Date("Jan 25 2015");  
var d = new Date("2015, JANUARY, 25");
```

JS Date Formats

Hi ha quatre tipus de format de data vàlids en JS:

3. Dates curtes:

```
var d = new Date("03/25/2015");  
var d = new Date("03-25-2015");  
var d = new Date("2015/03/25");
```

JS Date Formats

Hi ha quatre tipus de format de data vàlids en JS:

4. Dates en format complet (“*full JS format*”):

```
var d = new Date("Wed Mar 25 2015 09:56:24 GMT+0100  
(W. Europe Standard Time)");
```

En la següent data hi ha l'error de què el 25/3/2015 no és divendres i tampoc existeix un Tokyo Time:

```
var d = new Date("Fri Mar 25 2015 09:56:24  
GMT+0100 (Tokyo Time)");
```

El resultat que mostraria el navegador seria:
Wed Mar 25 2015 09:56:24 GMT+0100 (CET)

JS Date Methods

Mètodes per obtenir part d'una data:

Mètode	Descripció
getDate()	Obtenir el dia com un nombre (1-31)
getDay()	Obtenir el dia de la setmana com un nombre (0-6)
getFullYear()	Obtenir l'any amb 4 dígit (yyyy)
getHours()	Obtenir l'hora (0-23)
getMilliseconds()	Obtenir els milisegons (0-999)
getMinutes()	Obtenir els minuts (0-59)
getMonth()	Obtenir el mes (0-11)
getSeconds()	Obtenir els segons (0-59)
getTime()	Obtenir la hora (milisegons de de l'1 de gener de 1970)

JS Date Methods

Els dies de la setmana es retornen amb un dígit de 0 a 6. Si vols que es retornin com un text:

```
<script>
  let d = new Date();
  let days =
["Diumenge", "Dilluns", "Dimarts", "Dimecres", "Dijous", "Divendres", "Dissabte"];
  document.getElementById("demo").innerHTML = days[d.getDay()];
</script>
```


JS Date Methods

Hi ha mètodes per posar a un valor concret una part d'una data:

Mètode	Descripció
setDate()	Posa el dia amb un nombre (1-31)
setFullYear()	Posa l'any (optionalment mes i dia)
setHours()	Posa l'hora (0-23)
setMilliseconds()	Posa els milisegons(0-999)
setMinutes()	Posa els minuts (0-59)
setMonth()	Posa el mes (0-11)
setSeconds()	Posa els segons (0-59)
setTime()	Posa l'hora (en milisegons des de l'1 de gener 1970)

JS Date Methods

Exemples:

Posar data a 14 de gener de 2020:

```
var d = new Date();  
d.setFullYear(2020, 0, 14);
```

Posar 20 com a dia de mes:

```
d.setDate(20);
```

Afegir 50 dies a la data actual:

```
d.setDate(d.getDate() + 50);
```

Afegir 5 mesos a la data actual:

```
d.setMonth(d.getMonth() + 5);
```

Afegir 10 anys a la data actual:

```
d.setFullYear(d.getFullYear() + 10);
```

JS Date Methods

Parsing Dates, es pot convertir un string de data vàlida a milisegons:

```
var msec = Date.parse("March 21, 2012");  
document.getElementById("demo").innerHTML = msec;
```

També es pot convertir un nombre de milisegons a una data:

```
var msec = Date.parse("March 21, 2012");  
var d = new Date(msec);
```

JS Date Methods

Les dates es poden comparar fent servir operadors lògics:

```
let today, someday, text;
today = new Date();
someday = new Date();
someday.setFullYear(2100, 0, 14);

if (someday > today) {
    text = "Avui és anterior al 14 de gener de 2100.";
} else {
    text = "Avui és posterior al 14 de gener de 2100.";
}
```

JS Date Methods

Referència completa per l'objecte **Date**:

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

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS Dates -> Temps: 15 min (fer exercicis del final).

JS Date Formats -> Temps: 15 min.

JS Date Methods -> Temps: 20 min.

Fer exercicis del 12 al 18 DWEC_P02.

JS Arrays

Arrays: Per emmagatzemar múltiples valors en una variable.

Declarar array:

```
let nomArray = [ítem1, ítem2, ...];
```

```
let cars = ["Saab", "Volvo", "BMW"];
```

Nota: es pot crear un array fent servir **new** però no es recomana (let cars = new Array("Saab", "Volvo", "BMW");).

Accedir als elements:

```
nomArray[i]
```

```
cars[1] // accedim al valor "Volvo"
```

Canviar els valors dels elements de l'array:

```
nomArray[i] = valor;
```

```
cars[1] = "Toyota";
```

JS Arrays

Pots tenir elements de l'array que siguin de diferents tipus:

```
myArray[0] = Date.now();  
myArray[1] = myFunction;  
myArray[2] = myCars;
```

Els arrays són objectes (typeof retorna "object"). Però és millor considerar-los com un array convencional. Pot crear confusió amb objectes (als que accedim mitjançant un nom per accedir als seus membres, p.e.: `persona.nom`).

```
let persona = {nom:"John", llinatge:"Doe", edat:46};
```


JS Arrays

Propietats dels arrays:

Propietat **length**:

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.length;    // la longitud de fruits és 4
```

Mètodes dels arrays:

sort():

```
let y = cars.sort(); // El mètode sort()  
// ordena l'array cars en ordre alfabètic
```

push():

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Lemon"); // afegeix un nou element a fruits
```

JS Arrays

Mètodes dels arrays:

push():

```
let fruits = ["Banana", "Orange", "Apple",  
              "Mango"];  
fruits[fruits.length] = "Lemon"; // afegeix nou  
element a fruits
```

Alerta a no introduir valors deixant buits elements que tendran “undefined”:

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits[10] = "Lemon"; // deixa el elements 4, 5, 6, 7, 8 i 9  
a undefined
```

JS Arrays

Recòrrer arrays:

```
let txt = "";  
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
for (let index = 0; index < fruits.length; index++) {  
    txt += fruits[index];  
}
```

JS Arrays

No existeixen arrays associatius en JS
(=que poden fer servir noms com a índex).
Només índexs numèrics!

Els arrays fan servir índexs numèrics.
Els objectes fan servir noms com a índexs.
Si vols índexs que siguin noms fes servir objectes.

JS Arrays

Evita **new Array()**, fes servir **[]**:

```
let points = new Array();           // Malament
```

```
let points = [];                     // Bé
```

```
let points = new Array(40, 100, 1, 5, 25, 10) // Malament
```

```
let points = [40, 100, 1, 5, 25, 10];       // Bé
```

JS Arrays

Com identificar un objecte array?

El problema és que pels arrays **typeof** retorna “**object**”. Solució:

```
let points = [40, 100, 1, 5, 25, 10];  
Array.isArray(points); // true
```

JS Arrays Methods

JS és potent en aquest punt.

Farem un repàs dels mètodes que hi ha.

JS Arrays Methods

`toString()` i `join()`:

Passa l'array a string.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
document.getElementById("demo").innerHTML =  
    fruits.toString();
```

Ficaria en l'element el text: Banana, Orange, Apple, Mango

El mateix amb:

```
document.getElementById("demo").innerHTML = fruits.join();
```

Podem passar per paràmetre un valor diferente de coma:

```
document.getElementById("demo").innerHTML =  
    fruits.join('-');
```


JS Arrays Methods

pop():

Elimina el darrer element del array.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.pop(); // Elimina "Mango"
```

Retorna el valor eliminat ("Mango" en aquest cas).

JS Arrays Methods

push():

Afegeix un element a la darrera posició de l'array.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.push("Kiwi");
```

Retorna la longitud del nou array (en aquest cas seria 5).

També es pot fer amb:

```
fruits[fruits.length] = "Kiwi";
```

JS Arrays Methods

shift():

Elimina el primer element de l'array.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.shift();
```

Retorna el valor eliminat ("Banana" en aquest cas).

JS Arrays Methods

unshift():

Afegeix un element a l'array al principi.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.unshift("Lemon");
```

Retorna la longitud del nou array (en aquest cas seria 5).

JS Arrays Methods

splice():

Per afegir elements a l'array.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2, 0, "Lemon", "Kiwi");
```

El primer paràmetre (2) defineix la posició on s'afegiran els elements.

El segon paràmetre (0) defineix quants elements han de ser eliminats.

La resta de paràmetres són els elements afegits.

Resultat: ["Banana", "Orange", "Lemon", "Kiwi", "Apple", "Mango"]

JS Arrays Methods

splice():

```
let fruits = ["Banana", "Orange", "Apple",  
              "Mango"];
```

```
fruits.splice(0, 1); //Elimina el primer  
                     element de fruits
```

JS Arrays Methods

sort():

Ordena alfabèticament els elements d'un array.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.sort();    // Ordena els elements de fruits
```

JS Arrays Methods

`reverse()`:

Inverteix els elements d'un array. Si ho combines amb `sort` te pot servir per ordenar de forma descendent.

```
let fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
fruits.sort();
```

Resultat: ["Apple", "Banana", "Mango", "Orange"]

```
fruits.reverse();
```

Resultat: ["Orange", "Mango", "Banana", "Apple"]

També les poden encadenar: `fruits.sort().reverse();`

JS Arrays Methods

sort() amb nombres:

Ordena els nombres alfabèticament, per tant no ordena “numèricament” els nombres.

Truc per ordenar nombres:

```
let points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return a-b});  
//ara points[0] conté el valor més baix
```

Truc per ordenar descendentment:

```
var points = [40, 100, 1, 5, 25, 10];  
points.sort(function(a, b){return b-a});  
//ara points[0] conté el valor més alt
```

JS Arrays Methods

concat():

Concatena dos arrays.

```
let arr1 = ["Cecilie", "Lone"];  
let arr2 = ["Emil", "Tobias", "Linus"];  
let arr3 = ["Robin", "Morgan"];  
let myChildren = arr1.concat(arr2, arr3);
```

JS Arrays Methods

slice():

Retalla un tros d'un array en un altre nou.

```
let fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];  
let citrus = fruits.slice(1, 3)
```

Retorna: "Orange", "Lemon"

Comença en 1 i acaba en 2 (3 - 1, el 3 no entra).

JS Matrius

Les **matrius** també es poden dir **arrays multidimensionals**.

Exemple de matriu de 2x2:

```
let numeric = [  
  ['input1', 'input2'],  
  ['input3', 'input4'] ];  
numeric[0][0] == 'input1';  
numeric[0][1] == 'input2';  
numeric[1][0] == 'input3';  
numeric[1][1] == 'input4';
```

JS Matrius

Exemple de matriu 4x4 i de com fer un bucle que la recorri:

```
let matrix = [ [],[],[],[] ]
for (let i=0; i<4; i++) {
  for (let j=0; j<4; j++)
    matrix[i][j] = i*j;
}
```

JS Matrius

JavaScript: The Good Parts 1st Edition (de Douglas Crockford, p. 64). Extèn l'objecte array de JS:

```
Array.matrix = function(numrows, numcols, initial) {  
    let arr = [];  
    for (let i = 0; i < numrows; ++i) {  
        let columns = [];  
        for (var j = 0; j < numcols; ++j) {  
            columns[j] = initial;  
        }  
        arr[i] = columns;  
    }  
    return arr;  
}  
  
//Es crea una matriu 3x4 amb el nom "Marco"  
let names = Array.matrix(3,4,"Marco");  
  
//el num de files serà names.length  
//el núm de columnes serà names[0].length  
names[1][2] = "Joe";  
for (i=0; i < names.length; i++) {  
    for (j=0; j < names[0].length; j++) {  
        alert "["+i+"]"+"["+j+"]"+"="+names[i][j]);  
    }  
}
```

W3SCHOOLS JS: Exercicis.

Referència completa dels arrays:

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

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS Arrays -> Temps: 20 min (fer exercicis del final).

JS Array Methods -> Temps: 25 min (fer exercicis del final).

JS Array Sort

JS Array Iteration

Fer exercicis del 19 al 21 DWEC_P02.

Reparar en ...

- `Array.forEach()`
- `Array.isArray()`
- `Math.max.apply()`, `Math.min.apply()`

JS Booleans

Boolean: true/false.

La funció **Boolean()**:

```
Boolean(10 > 9)           // retorna true
let x = -1; Boolean(x);    // retorna true
    (Els valors reals retornen true)
let x = 0; Boolean(x);     // retorna false
let x = ""; Boolean(x);    // retorna false
let x; Boolean(x);         // retorna false
let x = null; Boolean(x);  // retorna false
let x = false; Boolean(x); // retorna false
let x = 10 / "H"; Boolean(x); // retorna false
```

JS Booleans

Referència completa dels booleans:

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

JS Comparisons

Recordau els operadors lògics que ja hem vist:

Operador	Descripció
==	igual a
===	igual valor i igual tipus
!=	no igual
!==	no igual valor o no igual tipus
>	major que
<	menor que
>=	major o igual que
<=	menor o igual que

JS Comparisons

Les comparacions es fan servir en sentències condicionals per fer una acció o altra depenent del resultat:

```
if (edat < 18) {  
    text = "Molt jove";  
} else {  
    text = "Adult";  
}
```

JS Comparisons

Operadors lògics:

Operador	Descripció	Exemple (x = 6 i y = 3)
&&	and	(x < 10 && y > 1) és true
	or	(x == 5 y == 5) és false
!	not	!(x == y) és true

JS Comparisons

Operador condicional (ternari):

nomvariable = (condició) ? valor1:valor2

```
let text = (edat < 18) ? "Molt jove": "Adult";
```

JS Comparisons

Comparar dades de diferents tipus en JS pot donar resultats estranys. És millor fer la comparació convertint les variables al mateix tipus de dades:

```
age = Number(age);  
if (isNaN(age)) {  
    voteable = "Error in input";  
} else {  
    voteable = (age < 18) ? "Too young" : "Old enough";  
}
```

Nota: La funció **Number()** converteix una cadena buida a 0, i una cadena no numèrica a NaN.

JS Comparisons

Operadors per a bits:

Fan feina amb nombres de 32 bits amb signe. Si qualche dia cursau els estudis universitaris veureu de què va tot això.

Operador	Descripció	Exemple	Igual que*	Resultat	Decimal
&	AND	$x = 5 \& 1$	0101 & 0001	0001	1
	OR	$x = 5 1$	0101 0001	0101	5
~	NOT	$x = \sim 5$	~0101	1010	10
^	XOR	$x = 5 \wedge 1$	0101 ^ 0001	0100	4
<<	Left shift	$x = 5 << 1$	0101 << 1	1010	10
>>	Right shift	$x = 5 >> 1$	0101 >> 1	0010	2

*Penseu que això estaria representat realment en 32 bits amb signe. A causa d'això, en JS, per exemple ~5 no retornaria 10, sinó que retornaria -6 (~00000000000000000000000000000101 retornarà 11111111111111111111111111111010).

JS Conditions

Sentències condicionals en JS:

if per especificar un bloc de codi que s'executarà si la condició és true.

else per especificar un bloc de codi que s'executarà, si la condició del punt anterior és falsa.

else if per especificar una nova condició a comprovar, si la primera condició és falsa.

switch per especificar molts blocs alternatius de codi per a ser executats segons diferents condicions.

JS Conditions

if/else en JS:

```
if (condició) {  
    bloc de codi a ser executat si la condició és true  
}
```

```
if (condition) {  
    bloc de codi a ser executat si la condició és true  
} else {  
    bloc de codi a ser executat si la condició és false  
}
```

```
if (condició1) {  
    bloc de codi a ser executat si la condició és true  
} else if (condició2) {  
    bloc de codi a ser executat si la condició és false i la condició2 és true  
} else {  
    bloc de codi a ser executat si la condició és false i la condició2 és false  
}
```

JS Switch

if/else en JS:

```
switch(expression) {
```

```
  case n:
```

code block

```
    break;
```

```
  case n:
```

code block

```
    break;
```

```
  ...
```

```
  default:
```

default code block

```
}
```


```
switch (new Date().getDay()) {  
  case 1:  
  case 2:  
  case 3:  
  default:  
    text = "Looking forward to the Weekend";  
    break;  
  case 4:  
  case 5:  
    text = "Soon it is Weekend";  
    break;  
  case 0:  
  case 6:  
    text = "It is Weekend";  
}
```

```
switch (new Date().getDay()) {  
  case 6:  
    text = "Today is Saturday";  
    break;  
  case 0:  
    text = "Today is Sunday";  
    break;  
  default:  
    text = "Looking forward to the Weekend";  
}
```

JS Loop For

Les sentències iteratives serveixen per iterar (=repetir).

```
text += cars[0] + "<br>";  
text += cars[1] + "<br>";  
text += cars[2] + "<br>";  
text += cars[3] + "<br>";  
text += cars[4] + "<br>";  
text += cars[5] + "<br>";
```



```
for (i = 0; i < cars.length; i++) {  
  text += cars[i] + "<br>";  
}
```

JS Loop For

Hi ha diferents tipus de sentències iteratives:

for - repeteixen un bloc de codi un nombre de determinat de vegades

for / in - per recòrrer les propietats d'un objecte

while - es repeteix un bloc de codi mentre una certa condició sigui avaluada a true (s'avalua la condició abans d'entrar al bucle)

do / while - es repeteix un bloc de codi mentre una certa condició sigui avaluada a true (s'avalua la condició al final del bucle després d'executar una vegada el bloc de codi)

JS Loop For

El bucle for:

```
for (sentència 1; sentència 2; sentència 3) {  
    code block to be executed  
}
```

```
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}
```

- **Sentència 1:** inicialització de variables abans de què el bucle comenci.
- **Sentència 2:** Defineix la condició per a què es realitzi la iteració. Una mala condició genera un **bucle infinit**.
- **Sentència 3:** S'incrementa/decrementa el valor de la variable del bucle una vegada que s'ha executat.

JS Loop For

El bucle **for**, **sentència 1**:

És opcional.

Es pot iniciar més d'una variable separant-les per comes.

```
for (i = 0, len = cars.length, text = ""; i < len; i++) {  
    text += cars[i] + "<br>";  
}
```



```
var i = 0;  
var len = cars.length;  
var text = "";  
for (; i < len; i++) {  
    text += cars[i] + "<br>";  
}
```

JS Loop For

El bucle **for**, **sentència 2**:

És opcional. Si l'ometes has de ficar una sentència **break** dins del bucle per evitar un bucle infinit.

JS Loop For

El bucle **for**, **sentència 3**:

Allò típic és que sigui `i++`, però hi ha moltes variants:

`i--`

`i = i + 15`

etc.

També és opcional:

```
var i = 0;
var len = cars.length;
for (; i < len; ) {
    text += cars[i] + "<br>";
    i++;
}
```

JS Loop For

El bucle **for/in**:

```
var person = {fname:"John", lname:"Doe", age:25};  
  
var text = "";  
var x;  
for (x in person) {  
    text += person[x];  
}
```

JS Loop While

El bucle **While**:

S'executa mentre una condició sigui true.

```
while (condició) {  
    bloc de codi a ser executat  
}  
  
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}
```

NOTA: Si no fas que la condició sigui false en algun moment generaràs un **bucle infinit**.

JS Loop While

El bucle **Do/While**:

S'executa mentre una condició sigui true.

```
do {  
    bloc de codi a ser executat  
}  
while (condition);
```

```
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);
```

JS Loop While

Tot és el mateix, o dit d'altra forma: Tot es podria fer amb for.

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
for (;cars[i];) {  
    text += cars[i] + "<br>";  
    i++;  
}
```



```
var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
while (cars[i]) {  
    text += cars[i] + "<br>";  
    i++;  
}
```

JS Break

Break romp l'execució d'un bucle (=bota fora del bucle).

Continue passa a la següent iteració del bucle (ignorant el que ve a continuació).

```
for (i = 0; i < 10; i++) {  
  if (i === 3) { continue; }  
  text += "The number is " + i + "<br>";  
}
```



The number is 0
The number is 1
The number is 2
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

JS Break

En JS es poden fer servir **labels**. Amb break/continue pots botar fins a una label determinada.

break label es pot fer servir per botar a qualsevol línia del teu codi.

continue (amb o sense label) només serveix per botar-te una iteració del bucle.

No recoman l'ús de labels ja que van en contra de les bones pràctiques que marca la programació estructurada. Jo no ho demanaré a classe.

Exemples:

[*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/label*](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/label)

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS Booleans -> Temps: 10 min (fer exercicis del final).

JS Comparisons -> Temps: 20 min (fer exercicis del final).

JS Conditions -> Temps: 20 min (fer exercicis del final).

JS Switch -> Temps: 20 min (fer exercicis del final).

JS Loop For -> Temps: 20 min (fer exercicis del final).

JS Loop While -> Temps: 20 min (fer exercicis del final).

JS Type Conversion

En JS hi ha 5 tipus de dades:

string

number

boolean

object

function

En JS hi ha 3 tipus d'objectes:

Object

Date

Array

En JS hi ha 2 tipus de dades que no poden contenir valors:

null

undefined

JS Type Conversion

L'operador **typeof** retorna un string que conté el tipus de dades d'una variable:

```
typeof "John"           // Retorna string
typeof 3.14              // Retorna number
typeof NaN               // Retorna number
typeof false             // Retorna boolean
typeof [1,2,3,4]         // Retorna object
typeof {name:'John', age:34} // Retorna object
typeof new Date()        // Retorna object
typeof function () {}    // Retorna function
typeof myCar              // Retorna undefined (si myCar no ha estat declarada)
typeof null              // Retorna object
```

JS Type Conversion

Es pot consultar la propietat constructor de les variables JS:

```
"John".constructor // Retorna function String() { [native code] }
```

```
(3.14).constructor // Retorna function Number() { [native code] }
```

```
false.constructor // Retorna function Boolean() { [native code] }
```

```
[1,2,3,4].constructor // Retorna function Array() { [native code] }
```

```
{name:'John', age:34}.constructor // Retorna function Object() { [native code] }
```

```
new Date().constructor // Retorna function Date() { [native code] }
```

```
function () {}.constructor // Retorna function Function(){ [native code] }
```

JS Type Conversion

La utilitat pràctica és per determinar si un objecte és un array o una data:

```
function isArray(myArray) {  
    return myArray.constructor.toString().indexOf("Array") > -1;  
}
```

```
function isDate(myDate) {  
    return myDate.constructor.toString().indexOf("Date") > -1;  
}
```

JS Type Conversion

Es pot convertir el contingut de les variables a altre tipus de dades de dues formes en JS:

- Fent servir una funció de JS.
- Automàticament pel JS.

JS Type Conversion

Convertint numbers a strings:

El mètode global String():

```
String(x) // Retorna un string d'una variable numèrica x  
String(123) // Retorna un string d'un número literal 123  
String(100 + 23) // Retorna un string d'una expressió  
numèrica
```

El mètode toString():

```
x.toString()  
(123).toString()  
(100 + 23).toString()
```

JS Type Conversion

Convertint numbers a strings:

toExponential(): Retorna un string amb un nombre, arrodonit i escrit fent servir notació exponencial.

toFixed(): Retorna un string amb un nombre, arrodonit i escrit en un nombre específic de decimals.

toPrecision(): Retorna un string amb un nombre, escrit en una longitud específica.

JS Type Conversion

Convertint booleans a strings:

Amb el mètode global `String()`:

```
String(false)           // Retorna "false"  
String(true)            // Retorna "true"
```

El mètode `toString()`:

```
false.toString()        // Retorna "false"  
true.toString()         // Retorna "true"
```


JS Type Conversion

Convertint dates a strings:

Amb el mètode global String():

```
String(Date()) // Retorna Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)
```

El mètode toString():

```
Date().toString() // Retorna Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)
```

JS Type Conversion

En l'apartat
sobre dates
vàrem
veure més
mètodes
per
convertir
dates a
strings:

<i>getDate()</i>	<i>Retorna el dia com a nombre (1-31)</i>
<i>getDay()</i>	<i>Retorna el dia de la setmana com a nombre (0-6)</i>
<i>getFullYear()</i>	<i>Retorna l'any amb 4 dígit (yyyy)</i>
<i>getHours()</i>	<i>Retorna l'hora (0-23)</i>
<i>getMilliseconds()</i>	<i>Retorna els milisegons (0-999)</i>
<i>getMinutes()</i>	<i>Retorna els minuts (0-59)</i>
<i>getMonth()</i>	<i>Retorna el mes (0-11)</i>
<i>getSeconds()</i>	<i>Retorna els segons (0-59)</i>
<i>getTime()</i>	<i>Retorna el temps (milisegons des de 1/1/1970)</i>

JS Type Conversion

Convertint strings a numbers:

El mètode global `Number()`:

```
Number("3.14")    // Retorna 3.14
Number(" ")        // Retorna 0
Number("")         // Retorna 0
Number("99 88")    // Retorna NaN
```

L'operador unari `+`:

```
let y = "5";       // y és un string
let x = + y;        // x conté el nombre 5
```

JS Type Conversion

Convertint strings a numbers:

parseFloat (): Analitza un string i retorna un nombre de punt flotant.

parseInt (): Analitza un string i retorna un sencer.

JS Type Conversion

Convertint booleans a numbers:

Mètode global **Number ()**:

```
Number(false)    // retorna 0  
Number(true)     // retorna 1
```

JS Type Conversion

Convertint dates a numbers:

Mètode global **Number()**:

```
d = new Date();
```

```
Number(d)          // retorna 1404568027739
```

Mètode **getTime()**:

```
d = new Date();
```

```
d.getTime()        // retorna 1404568027739
```

JS Type Conversion

Conversió automàtica de tipus:

Quan feim operacions entre diferents tipus de dades JS fa l'autoconversió de tipus:

`5 + null` // retorna 5 perquè null és convertit a 0

`"5" + null` // retorna "5null" perquè null és convertit a "null"

`"5" + 1` // retorna "51" perquè 1 és convertit a "1"

`"5" - 1` // retorna 4 perquè "5" és convertit a 5

JS Type Conversion

Conversió automàtica a string:

JS crida automàticament a `toString()` quan “treus” a la pàgina un objecte o variable:

```
document.getElementById("demo").innerHTML = myVar;  
// if myVar = {name:"Fjohn"} // toString converts to "[object Object]"  
// if myVar = [1,2,3,4]      // toString converts to "1,2,3,4"  
// if myVar = new Date()    // toString converts to "Fri Jul 18  
2014 //09:08:55 GMT+0200"
```

Els numbers i booleans també s'autoconverteixen:

```
// if myVar = 123           // toString converts to "123"  
// if myVar = true          // toString converts to "true"  
// if myVar = false         // toString converts to "false"
```


JS Type Conversion

<i>Valor Original</i>	<i>Convertit a Number</i>	<i>Convertit a String</i>	<i>Convertit a Boolean</i>
<i>false</i>	<i>0</i>	<i>"false"</i>	<i>false</i>
<i>true</i>	<i>1</i>	<i>"true"</i>	<i>true</i>
<i>0</i>	<i>0</i>	<i>"0"</i>	<i>false</i>
<i>1</i>	<i>1</i>	<i>"1"</i>	<i>true</i>
<i>"0"</i>	<i>0</i>	<i>"0"</i>	<i>true</i>

JS Type Conversion

<i>Valor Original</i>	<i>Convertit a Number</i>	<i>Convertit a String</i>	<i>Convertit a Boolean</i>
<i>"1"</i>	<i>1</i>	<i>"1"</i>	<i>true</i>
<i>NaN</i>	<i>NaN</i>	<i>"NaN"</i>	<i>false</i>
<i>Infinity</i>	<i>Infinity</i>	<i>"Infinity"</i>	<i>true</i>
<i>-Infinity</i>	<i>-Infinity</i>	<i>"-Infinity"</i>	<i>true</i>
<i>""</i>	<i>0</i>	<i>""</i>	<i>false</i>

JS Type Conversion

<i>Valor Original</i>	<i>Convertit a Number</i>	<i>Convertit a String</i>	<i>Convertit a Boolean</i>
"20"	20	"20"	true
"twenty"	NaN	"twenty"	true
[]	0	""	true
[20]	20	"20"	true
[10,20]	NaN	"10,20"	true

JS Type Conversion

<i>Valor Original</i>	<i>Convertit a Number</i>	<i>Convertit a String</i>	<i>Convertit a Boolean</i>
<i>["twenty"]</i>	<i>NaN</i>	<i>"twenty"</i>	<i>true</i>
<i>["ten","twenty"]</i>	<i>NaN</i>	<i>"ten,twenty"</i>	<i>true</i>
<i>function(){}</i>	<i>NaN</i>	<i>"function(){}"</i>	<i>true</i>
<i>{ }</i>	<i>NaN</i>	<i>"[object Object]"</i>	<i>true</i>
<i>null</i>	<i>0</i>	<i>"null"</i>	<i>false</i>

JS Type Conversion

<i>Valor Original</i>	<i>Convertit a Number</i>	<i>Convertit a String</i>	<i>Convertit a Boolean</i>
<i>undefined</i>	<i>NaN</i>	<i>"undefined"</i>	<i>false</i>

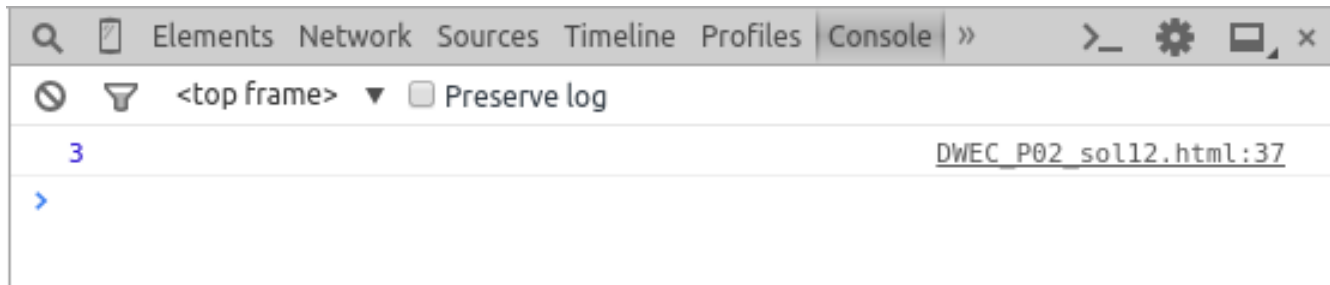
JS Debugging

És complicat escriure codi JS sense *debugger*. La depuració (=debugging) és el procés de proves i recerca per reduir errors (*bugs*) en el codi d'un programa. El primer error informàtic conegut va ser un insecte real (=bug), atrapat en la circuiteria.

En el navegador, fent clic a F12 apareix el ***debugger***.

També fent servir el mètode **`console.log()`** al teu codi pots mostrar valors en una consola.

```
let a = 3;  
console.log(a);
```



JS Debugging

En el debugger
es poden
definir
breakpoints.

Objecte Paused in debugger 12:15

Sat Mar 21 2009 12:15:00 GMT+0100 (CET)

La mateixa data amb el format exacte de l'enunciat:

Sources Content scripts Snippets DWEC_P02_sol12.html x

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <p>Objecte date amb la data 21/03/2009 12:
6
7 <p id="demo"></p>
8
9 <p>La mateixa data amb el format exacte de
10
11 <p id="demo2"></p>
12
13 <script>
14 //Funció per formatar un sencer a dos dígi
15 //number: nombre a formatar
16 //zeros: string amb el nombre de xifres de
17 //numZeros: sencer amb el nombre de xifres
18 //Exemple d'ús formatNumer(2,"000",3) i re
19 function formatNumer(number, zeros, numZe
20     var format = zeros + number;
21     format = format.substring(format.lengt
22     return format;
23 }
24
25 //Funció que dona a la data el format que
26 function formataData (d1, separadorDies, s
27
28     return formatNumer(d1.getDay(), "00",
29         formatNumer((parseInt(d1.
30             formatNumer(d1.getHours()
31             formatNumer(d1.getMinutes
32
33 }
34
35 var d = new Date(2009, 2, 21, 12, 15, 00,
36
37 document.getElementById("demo").innerHTML
38 document.getElementById("demo2").innerHTML
39
40
41 { } Line 38, Column 1
```

Call Stack (anonymous function) DWEC_P02_sol12.html:38

Paused on a JavaScript breakpoint.

Breakpoints

- ✓ DWEC_P02_sol12.html:38 document.getElementById("demo2").inne...

DOM Breakpoints

XHR Breakpoints

Event Listener Breakpoints

Scope Watch Global Window

JS Debugging

A més tenim la paraula clau ***debugger*** que ens permet incloure-la en el codi per afegir un breakpoint en el debugger.

```
let x = 15 * 5;  
debugger;  
document.getElementById("demo").innerHTML = x;
```

Chrome, Firefox, Internet Explorer, Opera i Safari tenen debugger. No funcionen 100% igual. Vosaltres sou els que ho heu de mirar i decidir quins voleu fer servir.

JS Hoisting

JS mou totes les declaracions de variables al principi, això es coneix com a **hoisting**. Però no mou les inicialitzacions!
Passa d'això i declara les variables al principi!!

```
var x = 5;  
  
var elem = document.getElementById("demo");  
elem.innerHTML = "x = " + x + ", y = " + y;  
  
var y = 7;
```



x = 5, y = undefined

```
let x = 5;  
  
let elem = document.getElementById("demo");  
elem.innerHTML = "x = " + x + ", y = " + y;  
  
let y = 7;
```



Uncaught ReferenceError: Cannot access 'y' before initialization

JS Strict Mode

La directiva “**use strict**” es nova en JS 1.8.5 (ECMAScript version 5). Les altres versions l'ignoren.

Si es declara al principi s'aplica a tot el codi.
Dins d'una funció s'aplica només dins ella.

```
"use strict";  
myFunction();  
  
function myFunction() {  
    y = 3.14;    // This will also cause an error  
}
```

```
x = 3.14;        // This will not cause an error.  
myFunction();  
  
function myFunction() {  
    "use strict";  
    y = 3.14;    // This will cause an error  
}
```

JS Strict Mode

Característiques:

- Totes les variables han de ser declarades.
- No es permet esborrar una variable (`delete x;`).
- No es pot definir una propietat més d'una vegada (`let x = {p1:10, p1:20};`).
- No es permet duplicar un paràmetre d'una funció (`function x(p1, p1) {};`).
- No es permeten literals en octal (`let x = 010;`).
- No es permeten literals amb escape (`let y = \010;`).

JS Strict Mode

Característiques:

- No es permeten propietats read-only
(obj.defineProperty(obj, "x", {value:0, writable:false})).
- No es poden fer servir com a variables paraules com eval o arguments.
- No es pot fer servir la sentència with.
- No es pot fer servir eval() per crear variables en l'àmbit (scope) on va ser cridat (eval ("let x = 2");).
- Etc.

JS Style Guide

La idea de seguir convencions quan escrius el teu codi és que es pugui llegir fàcilment (i això du a què sigui més fàcil de mantenir). Per exemple, els comentaris són molt importants quan agafes un codi fet per una altra persona.

JS Style Guide

Noms de variables i funcions fent servir
camelCase:

```
firstName = "John";  
lastName = "Doe";  
price = 19.90;  
tax = 0.20;  
fullPrice = price + (price * tax);
```

Variables globals i constants escrits en
MAJÚSCULA.

JS Style Guide

Fer servir espais entre operadors (= + - * /) i després de comes:

```
let x = y + z;  
let values = ["Volvo", "Saab", "Fiat"];
```

Indentar blocs de codi amb 4 espais:

```
function toCelsius(fahrenheit) {  
    return (5 / 9) * (fahrenheit - 32);  
}
```

NOTA: És recomanable NO fer servir tabuladors ja que s'interpreten de forma diferent segons l'editor.

JS Style Guide

Sempre acabar una sentència amb ;

```
let values = ["Volvo", "Saab", "Fiat"];  
let person = {  
  firstName: "John",  
  lastName: "Doe",  
  age: 50,  
  eyeColor: "blue"  
};
```

Fixa't que en l'exemple { es posa en la primera línia (amb un espai en blanc davant) i } es posa en una nova línia al final. A més tens una propietat de l'objecte per línia .Mirau també a la diapositiva anterior l'exemple amb la funció.

JS Style Guide

Per incloure una llibreria:

```
<script src="myscript.js">
```

Extensions de fitxers:

.html (no .htm)

.css

.js

Penseu també que és convenient usar les mateixes convencions en els noms HTML. Per exemple, les dues sentències següents donarien resultats diferents:

```
let obj = getElementById("Demo")
```

```
let obj = getElementById("demo")
```

JS Style Guide

Fer servir minúscules en els noms de fitxers:

foto.jpg no és el mateix que Foto.jpg (en IIS sí és el mateix, servidors Windows no són case sensitive, els Linux/Unix sí).

JS Best Practices

- Minimitzar l'ús de variables globals.
- Sempre declarar les variables (globals i locals).
- Posar les declaracions de variables al principi i sempre inicialitzar-les.
- No tractar numbers, strings i booleans com a objectes (~~let~~ y = ~~new~~ String(~~"John"~~)).

JS Best Practices

No usar new Object():

- Fer servir {} en lloc de new Object().
- Fer servir "" en lloc de new String().
- Fer servir 0 en lloc de new Number().
- Fer servir false en lloc de new Boolean().
- Fer servir [] en lloc de new Array().
- Fer servir /(())/ en lloc de new RegExp().
- Fer servir function (){} en lloc de new function().

```
let x1 = {};           // new object
let x2 = "";           // new primitive string
let x3 = 0;            // new primitive number
let x4 = false;        // new primitive boolean
let x5 = [];           // new array object
let x6 = /(())/;       // new regexp object
let x7 = function(){}; // new function object
```

JS Best Practices

Evitar conversions de tipus automàtiques (el tipat de dades no és fort).

Fer servir ===

```
0 == "";           // true
1 == "1";          // true
1 == true;         // true
0 === "";          // false
1 === "1";         // false
1 === true;        // false
```

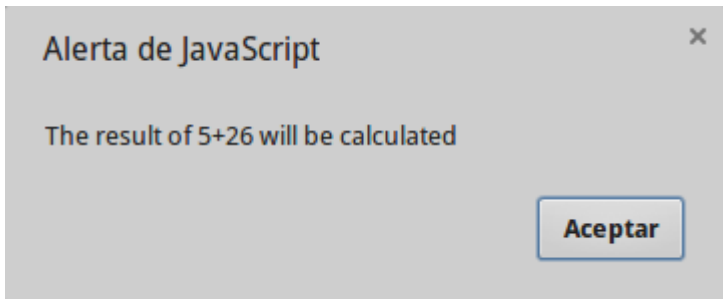
JS Best Practices

Acabar les sentències **switch** amb **default**.

Evitar l'ús d'**eval()**. Aquesta funció permet executar text com a codi (en la gran majoria de casos això no és necessari i a més és un problema de seguretat).

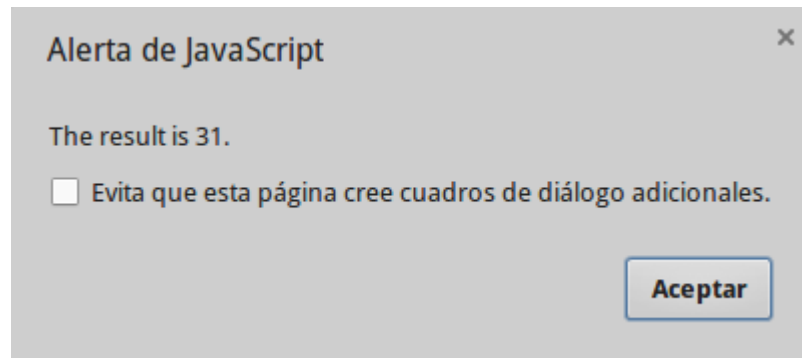
```
alert("The result is " + eval("alert('The result of 5+26 will be calculated');5+26;") + ".")
```

1



```
let x = 10;  
let y = 20;  
let a = eval("x * y") + "<br>";
```

2



JS Mistakes

Ús de = en lloc de == (la condició `x = 10` és true sempre).

```
let x = 0;  
if (x = 10)
```

Comparació entre tipus diferents:

```
let x = 10;  
let y = "10";  
if (x == y) -> true  
if (x === y) -> false
```

JS Mistakes

La sentència switch fa servir una comparació estricta.

```
var x = 10;  
switch(x) {  
    case 10: alert("Hello");  
}
```



Mostra l'alert.

```
var x = 10;  
switch(x) {  
    case "10": alert("Hello");  
}
```



No mostra l'alert.

JS Mistakes

+ és fa servir per sumar numbers i per concatenar strings.

```
let x = 10;  
let y = 5;  
let z = x + y; // el resultat dins z és 15
```

```
let x = 10;  
let y = "5";  
let z = x + y; // el resultat dins z és "105"
```

JS Mistakes

JS fa servir nombres de punt flotant de 64 bits.

```
let x = 0.1;
let y = 0.2;
let z = x + y // z conté realment
0.30000000000000004
if (z == 0.3) // la condició és avaluada com a
false
```

Una possible solució és dividir i multiplicar:

```
var z = (x * 10 + y * 10) / 10; // z és ara 0.3
```

JS Mistakes

Rompre strings:

```
let x =  
"Hello World!";
```

```
let x = "Hello  
World!";
```

```
let x = "Hello \  
World!";
```

JS Mistakes

Ús de **return**:

```
function myFunction(a) {  
    let  
    power = 10;  
    return  
    a * power;  
}
```

// Aquesta funció retorna sempre undefined

// El problema és que return no necessita acabar en ;

JS Mistakes

Com hem vist, els arrays en JS no tenen índexs que siguin noms (són sempre nombres!).

```
let person = [];
```

```
person[0] = "John";
```

```
person[1] = "Doe";
```

```
person[2] = 46;
```

```
let x = person.length; // person.length tornarà 3
```

```
let y = person[0]; // person[0] tornarà "John"
```

JS Mistakes

Els objectes fan servir índexs que són noms.

```
let persona = {};
```

```
persona["firstName"] = "John";
```

```
persona["lastName"] = "Doe";
```

```
persona["age"] = 46;
```

```
let x = persona.length; //persona.length undefined
```

```
let y = persona[0]; //persona[0] tornarà undefined
```

JS Mistakes

JS ens permet acabar la definició d'arrays i objectes amb una coma a fi de facilitar la tasca d'afegir i/o eliminar elements o propietats, per tant, NO ho considerem un error:

```
punts = [  
  40,  
  100,  
  1,  
  5,  
  25,  
  10,  
]
```

```
persona = {  
  nom: "Joan",  
  cognoms: "Doe",  
  edat: 46,  
}
```

undefined (per variables, propietats i mètodes) no és el mateix que **null** (per objectes). És a dir, un objecte inicialitzat a null significa que es buit si és undefined és que no ha estat definit.

JS Performance

El rendiment és pot millorar un poc depenent de la forma en què programes en JS.

Reduir l'activitat en els bucles:

És pitjor (el codi de la propietat s'accedeix en cada iteració del bucle):

```
for (i = 0; i < arr.length; i++) {
```

És millor:

```
l = arr.length;
```

```
for (i = 0; i < l; i++) {
```


JS Performance

Reduir l'accés al DOM (més envant ens ficarem en aquest tema):

Accedir al DOM és molt lent. Si has d'accedir a ell moltes vegades, guarda-ho en una variable local:

```
obj = document.getElementById( "demo" );  
obj.innerHTML = "Hello";
```

Reduir la mida del DOM.

Millor fer feina amb un nombre petit d'elements del DOM.

Això millorarà la velocitat de càrrega de la pàgina. A més així es redueix el temps d'accés al DOM.

JS Performance

Fer servir les variables que siguin necessàries
i no més!!!

Per 3 variables:

```
let fullName = firstName + " " + lastName;  
document.getElementById("demo").innerHTML = fullName
```

El mateix amb 2 variables:

```
document.getElementById("demo").innerHTML = firstName + " " + lastName
```

JS Performance

Si posem els script al final de la pàgina, deixem que el navegador la carregui primer...

Mentre es carrega l'script el navegador no inicia altres descàrregues ni renderitza la pàgina...

Hi ha fòrmules per millorar això, com afegir l'script amb codi una vegada que s'ha carregat la pàgina:

```
<script>
```

```
window.onload = downScripts;
```

```
function downScripts() {  
  var element = document.createElement("script");  
  element.src = "myScript.js";  
  document.body.appendChild(element);  
}
```

```
</script>
```

JS Performance

Evitar l'ús de `with`. Baixa el rendiment i no és permès en strict mode.

W3SCHOOLS JS: Ho heu de fer!

Fer els apartats:

JS Hoisting -> Temps: 10 min.

JS Strict Mode -> Temps: 15 min.

JS Style Guide -> Temps: 20 min.

JS Best Practices -> Temps: 20 min.

JS Mistakes -> Temps: 20 min (fer exercicis del final).

JS Performance -> Temps: 15 min.

Fer exercicis del 24 al 28 DWEC_P02.

JS Reserved words

Les **paraules reservades** no es poden fer servir per noms de variables i funcions.

Si vols veure una llista de les paraules reservades en JS visita:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords

Bibliografia

<http://www.w3schools.com/js/>

<https://javascript.info/>

<http://aprende-web.net/javascript/>

<http://www.desarrolloweb.com/manuales/20/>

<http://stackoverflow.com/questions/7545641/javascript-multidimensional-array>