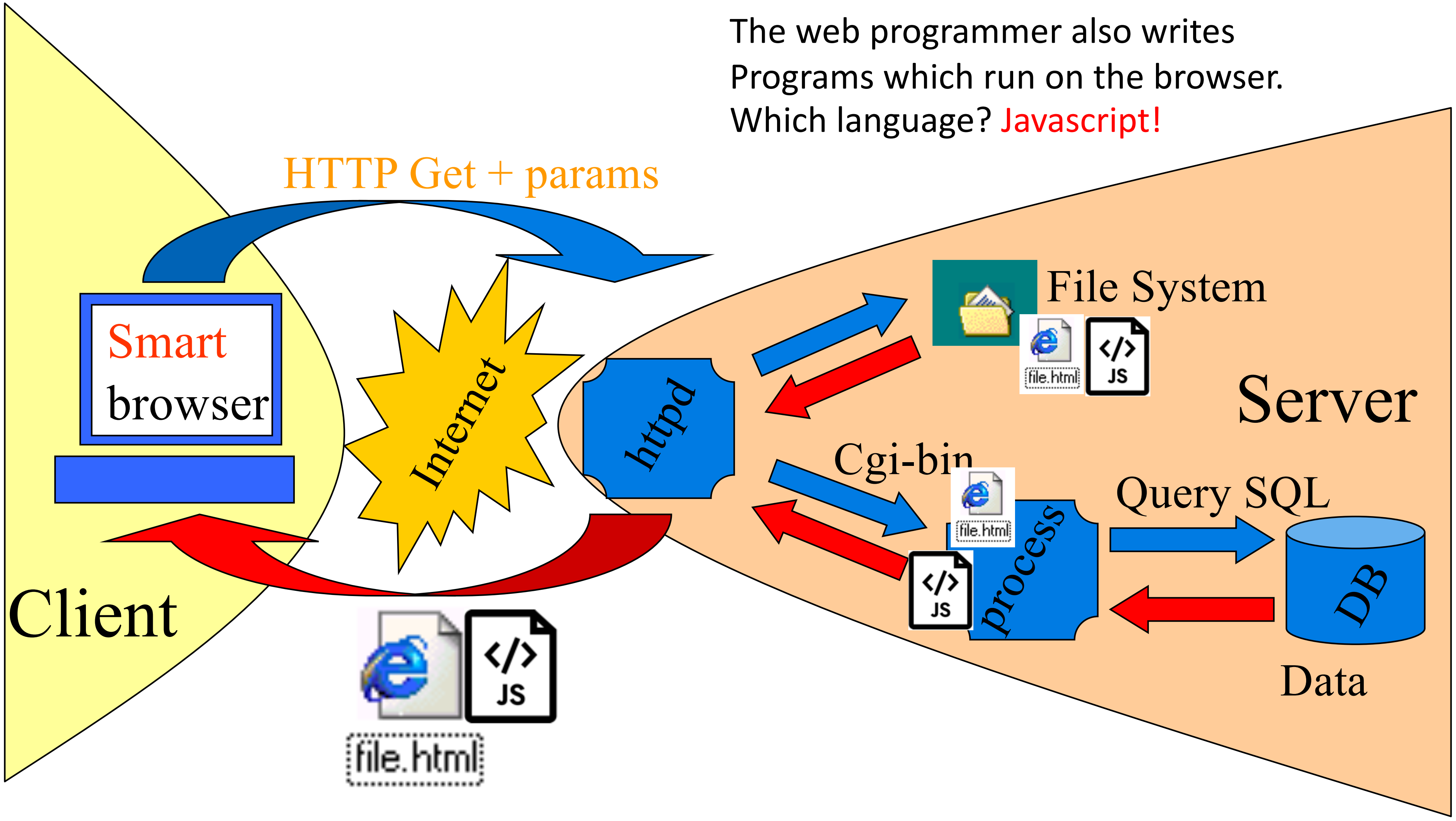
Javascript :
the basis of the language

The web architecture with smart browser

The web programmer also writes Programs which run on the browser. Which language? Javascript!



Evolution 3: execute code also on client! (How ?)

Example 1: onmouseover, onmouseout

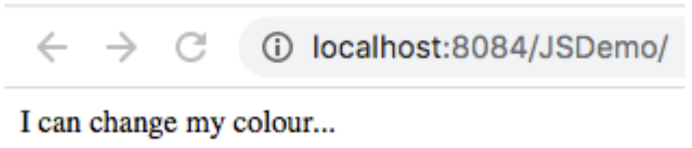
```
<!DOCTYPE html>
<html>
    <head>
        <title>Dynamic behaviour</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
  initial-scale=1.0">
    </head>
    <body>
        <div onmouseover="this.style.color = 'red'"
  onmouseout="this.style.color = 'green'">
  I can change my colour!</div>
    </body>
</html>
```

JAVASCRIPT

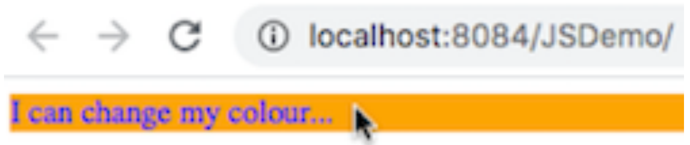
The dynamic behaviour is
on the client side!
(The file can be loaded locally)

Example 2: onmouseover, onmouseout

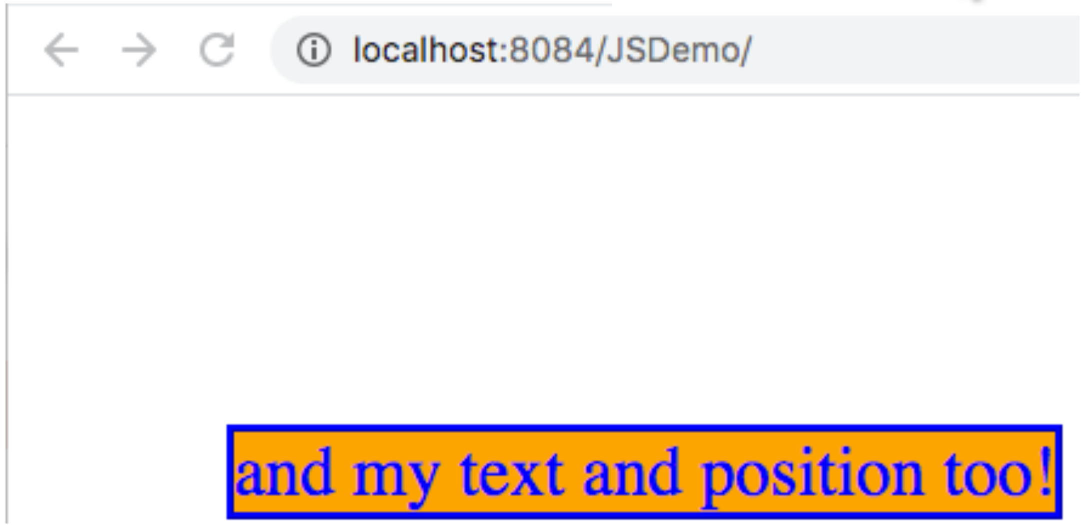
```
<body>
  <div
    onmouseover="this.style.background='orange';
      this.style.color = 'blue';"
    onmouseout="
      this.innerText='and my text and position too!';
      this.style.position='absolute';
      this.style.left='100px’;
      this.style.top='150px';
      this.style.borderStyle='ridge';
      this.style.borderColor='blue';
      this.style.fontSize='24pt';">
   I can change my colour...
  </div>
</body >
```


localhost:8084/JSDemo/

I can change my colour...


localhost:8084/JSDemo/

I can change my colour...


localhost:8084/JSDemo/

and my text and position too!

JavaScript is event-based

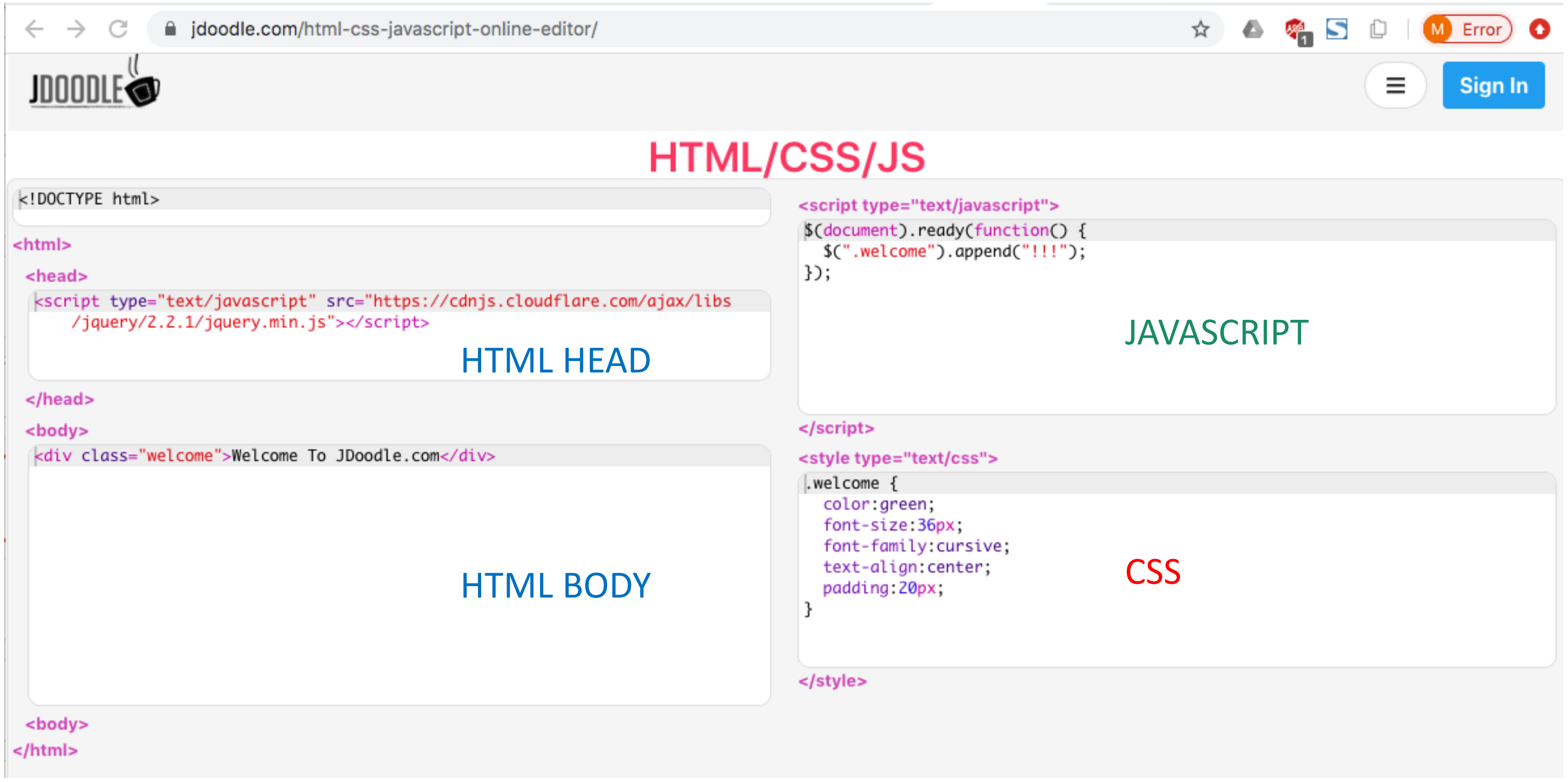
UiEvents:

These event objects inherits the properties of the UiEvent:

- The FocusEvent
- The InputEvent
- The KeyboardEvent
- The MouseEvent
- The TouchEvent
- The WheelEvent

See https://www.w3schools.com/jsref/obj_uievent.asp

Test and Gym



```
<!DOCTYPE html>

<html>
  <head>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs
        /jquery/2.2.1/jquery.min.js"></script>
  </head>
  <body>
    <div class="welcome">Welcome To JDoodle.com</div>
  <body>
</html>
```

```
<script type="text/javascript">
$(document).ready(function() {
   $(".welcome").append("!!!");
});
</script>
```

```
<style type="text/css">
.welcome {
   color:green;
   font-size:36px;
   font-family:cursive;
   text-align:center;
   padding:20px;
}
</style>
```

https://www.jdoodle.com/html-css-javascript-online-editor/

JavaScript History

- JavaScript was born as Mocha, then “**LiveScript**” at the beginning of the 94’s.
- Name changed into JavaScript (name owned by Netscape)
- Microsoft responds with **Vbscript**
- Microsoft introduces **JScript** (dialect of Javascript)
- A standard is defined: ECMAScript (ECMA-262, ISO-16262)
- Jscript survives (as ECMAScript incarnation till 2009, then as Chakra till 2015)
- Another incarnation of ECMAScript is **ActionScript** (Adobe, for Flash)

JavaScript: History

Name	Edition	Date published			
Mocha		May-95			
LiveScript		Sep-95			
JavaScript		Dec-95			
			Jscript	Aug-96	Microsoft
ECMAScript	1	Jun-97			
ECMAScript	2	Jun-98	ActionScript	1998	Macromedia/Adobe
ECMAScript	3	Dec-99			
ECMAScript	4	Abandoned			
ECMAScript	5	Dec-09			
ECMAScript	5,1	Jun-11			
ECMAScript 2015 (ES2015)	6	Jun-15			
ECMAScript 2016 (ES2016)	7	Jun-16			
ECMAScript 2017 (ES2017)	8	Jun-17			
ECMAScript 2018 (ES2018)	9	Jun-18			
ECMAScript 2019 (ES2019)	10	Jun-19			

ECMAScript Engine

An ECMAScript engine is a program that executes source code written in a version of the ECMAScript language standard

Examples:

- **V8 (Chrome, NodeJS, Opera)**
- **SpiderMonkey (Mozilla)**
- **Chakra (Microsoft)**
- **JavaScriptCore(Apple)**
- **Nashorn (Oracle – JDK)**

See https://en.wikipedia.org/wiki/List_of_ECMAScript_engines

JavaScript and HTML

- Between `<script>` and `</script>` tags
- In a `<script src="url"></script>` tag
- Between `<server>` and `</server>` tags
- In an event handler:

```
<input type="button" value="Click me"
             onClick="js code">
<div onmouseover="this.style.color =
   'red'" onmouseout="this.style.color =
   'green'”>
```

Base

- Syntax is C-like (C++-like, Java-like)
  case-sensitive,
  statements end with (optional) semicolon ;
  //comment /*comment*/
  operators (=,*,+,++,+=,!=,==,&&,…)

```
if (expression) {statements} else {statements}
switch (expression) {
        case value: statements; break;
        …
        default: statements; break;
 }
while (expression) {statements}
do (expression) while {statements}
for (initialize ; test ; increment) {statements}
  for (a in s) {statements}
```

Operators

- Mathematical operators: standard, plus ** for exponentiation (ES6)
- Assignment operators: standard, plus **= for exponentiation (ES6)
- String operators: + (concatenation), see later
- Comparison operators: standard, plus type and value

===	equal value and equal type
!==	not equal value or not equal type

- Logical and bitwise operators: standard
- Type operators

typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

see https://www.w3schools.com/js/js_operators.asp

Data types

- Primitive data types
  number, string, boolean, undefined

- Complex data types
  object, function (more later!)

- Loosely, dynamic typed variables (Basic-like)

```
 t0==typeof(x);
var x=3;
var t1=typeof(x);
x="pippo";
var t2=typeof(x);
```

t0: undefined

t1: number

t2: string

Data types: Objects and DOM

Javascript also has Objects, somehow similar to Java Objects (even though their implementation is quite different, and their definition not as clean and straightforward as in Java).

Objects have variables and methods.

Similarly to Java Objects, they can be printed: in such case they use their customized toString() method, or give a generic indication such as **[object HTMLDivElement]**

Some Javascript Objects represent fragments of an HTML document. The collection of these Objects represent the whole page. Such representation is called Document Object Model.

More here:

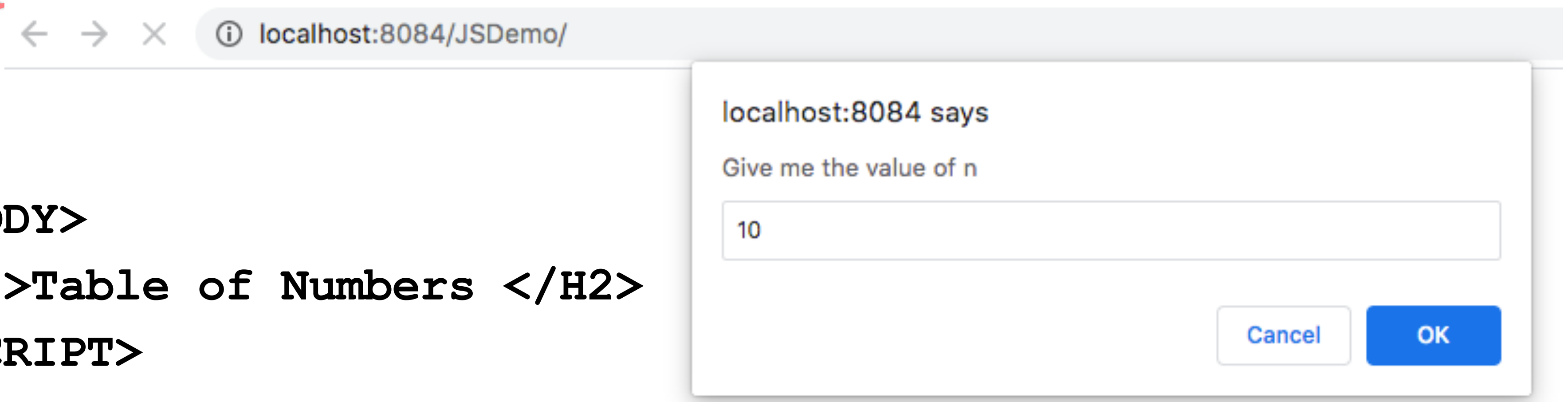
https://www.w3schools.com/js/default.asp

Q

How can I do basic I/O in JavaScript?

Core

User I/O


localhost:8084/JSDemo/

localhost:8084 says

Give me the value of n

10

```
<BODY>
<H2>Table of Numbers </H2>
<SCRIPT>
n=window.prompt("Give me the value of n",3);
for (i=1; i<n; i++) {
   document.write(i);
   document.write("<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

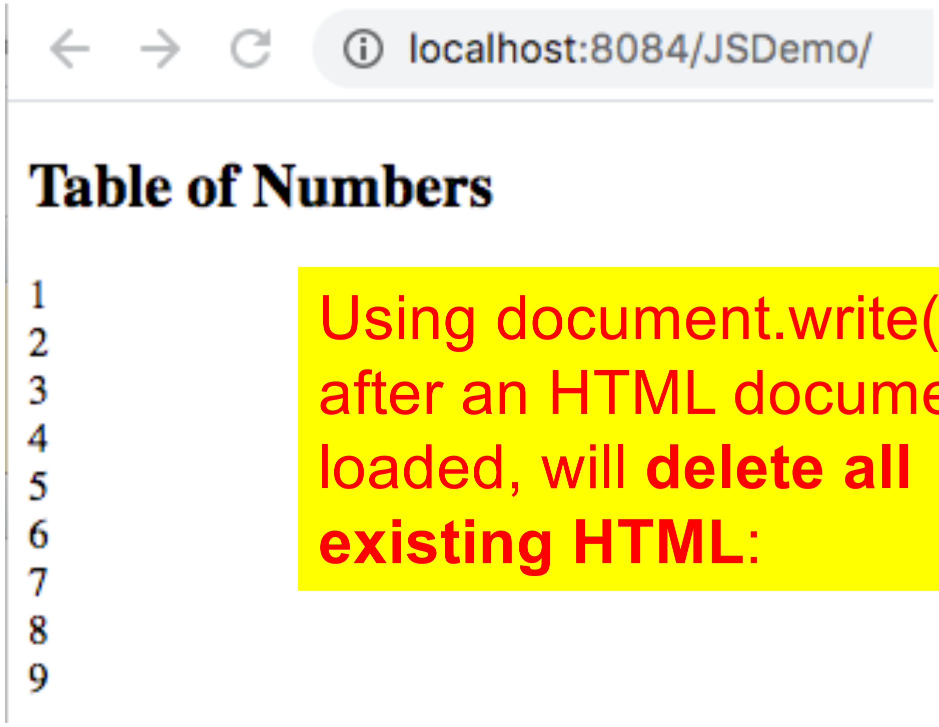

localhost:8084/JSDemo/

Table of Numbers

```
1
2
3
4
5
6
7
8
9
```

Using document.write() after an HTML document is loaded, will **delete all existing HTML**:

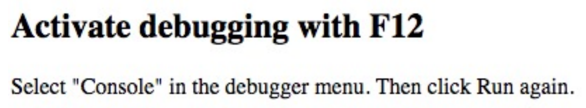
The dynamic behaviour is on the client side!
(The file can be loaded locally)

JS output

Writing into the HTML output using **`document.write()`**.

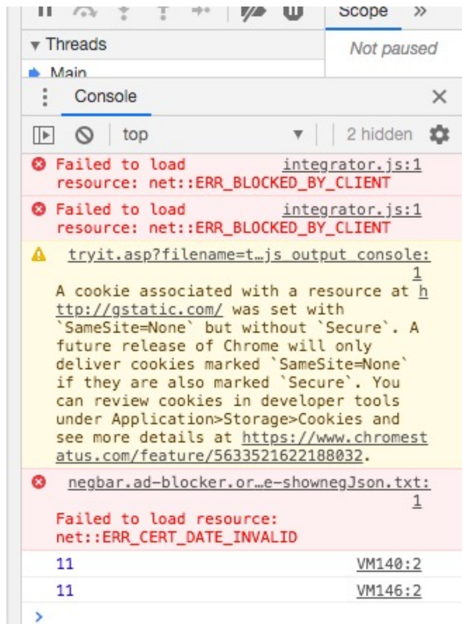
Writing into an alert box, using **`window.alert()`**.

Writing into the browser console, using **`console.log()`**.



Activate debugging with F12

Select "Console" in the debugger menu. Then click Run again.



```
Failed to load                    integrator.js:1
resource: net::ERR_BLOCKED_BY_CLIENT
Failed to load                    integrator.js:1
resource: net::ERR_BLOCKED_BY_CLIENT
 tryit.asp?filename=t…js_output_console:
                                        1
A cookie associated with a resource at h
ttp://gstatic.com/ was set with
`SameSite=None` but without `Secure`. A
future release of Chrome will only
deliver cookies marked `SameSite=None`
if they are also marked `Secure`. You
can review cookies in developer tools
under Application>Storage>Cookies and
see more details at https://www.chromest
atus.com/feature/5633521622188032.
 negbar.ad-blocker.or…e-shownegJson.txt:
                                        1
Failed to load resource:
net::ERR_CERT_DATE_INVALID
11                                VM140:2
11                                VM146:2
>
```

JS output

Writing into an HTML element, using:

- innerHTML

```
<div onmouseover="this.innerHTML='How are you?';">
   Hello</div>
```

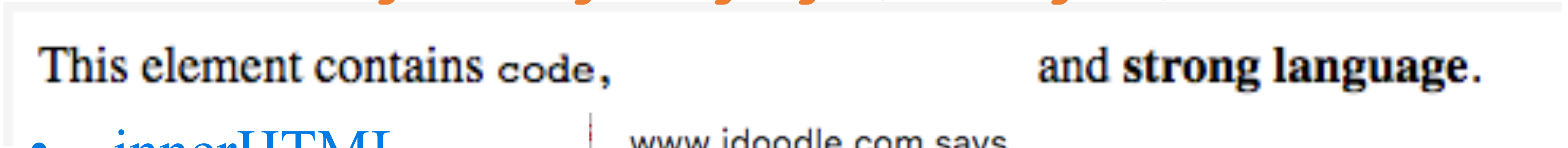
- innerText

```
<div onmouseover="this.innerText='How are you?';">
   Hello</div>
```

- textContent

```
<div onmouseover="this.textContent='How are you?';">
   Hello</div>
```

```
<div onmouseover="window.alert(this.property)";>
This element contains <code>code</code>, <span
   style="visibility:hidden">hidden information, </span>
   and <strong>strong language</strong>.</div>
```



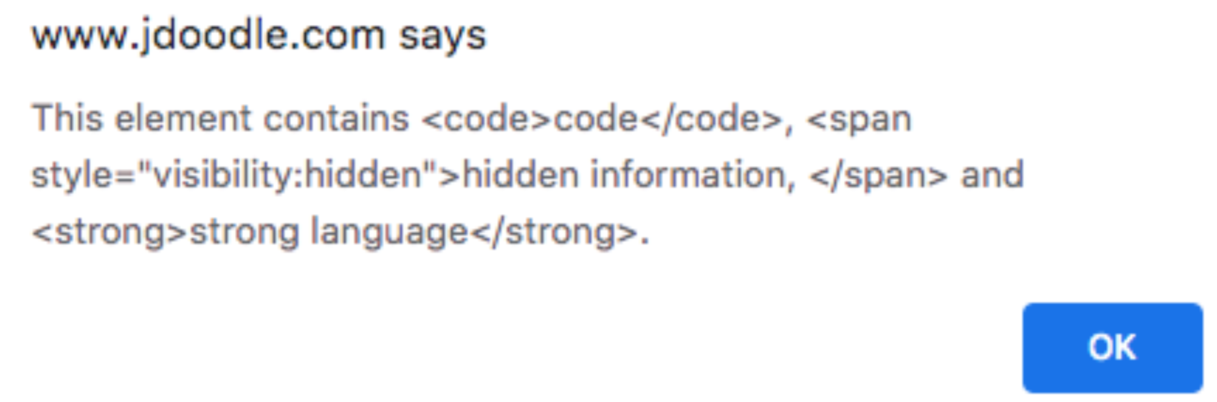
This element contains `code`, and **strong language.**

- innerHTML
  - Full HTML content

property

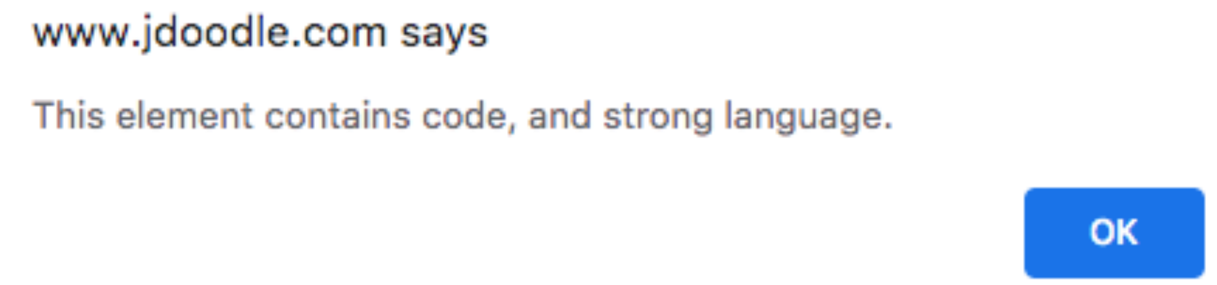
- innerText
  - Text only, CSS aware
- textContent
  - Text only, CSS unaware

JS output



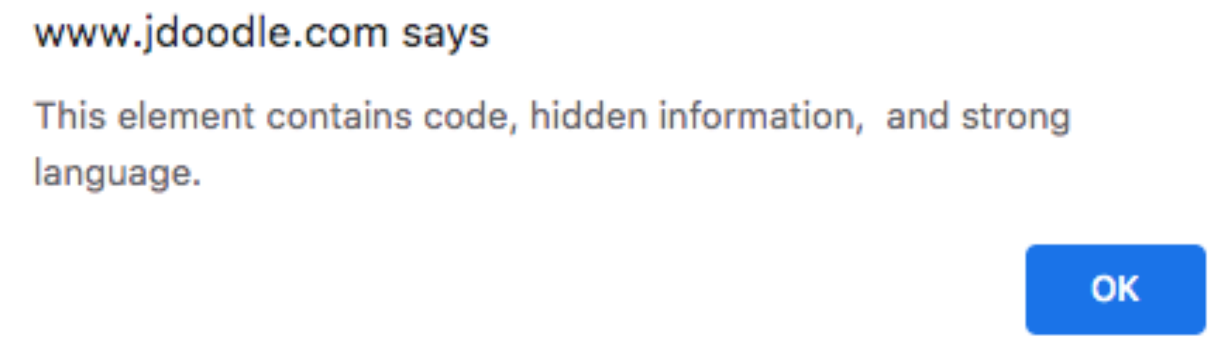
```
www.jdoodle.com says

This element contains <code>code</code>, <span
style="visibility:hidden">hidden information, </span> and
<strong>strong language</strong>.
```



```
www.jdoodle.com says

This element contains code, and strong language.
```

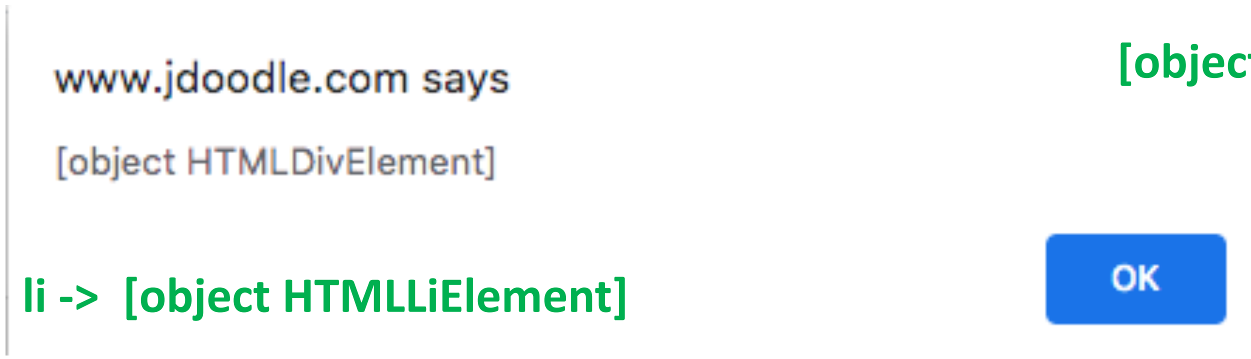


```
www.jdoodle.com says

This element contains code, hidden information,  and strong
language.
```

What is "this"?

```
<div onmouseover="window.alert(this)";"> Hello</div>
```


www.jdoodle.com says

[object HTMLDivElement]

[object HTMLDivElement]

li -> [object HTMLLiElement]

h1,…h5 -> [object HTMLHeadingElement]

b, i -> [object HTMLElement]

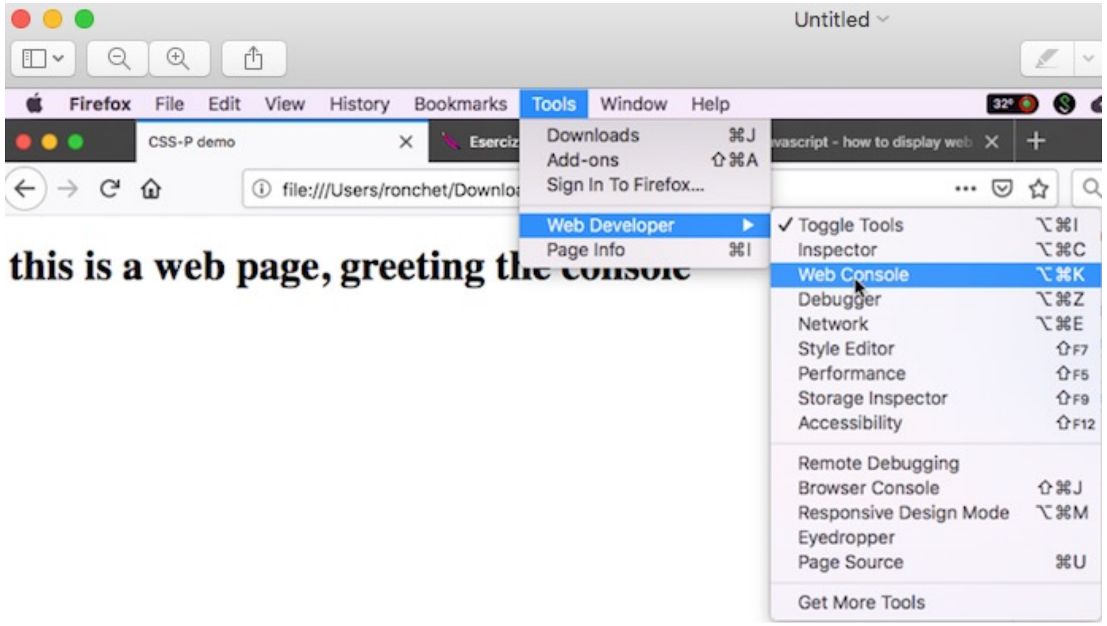
```
<a onmouseover="window.alert(this)";"
   href="http://localhost"> a link</a>
```

a -> http://localhost

```
<b onmouseover="window.alert(this.nodeName)";"> Hello</b>
```

b -> b

a -> a

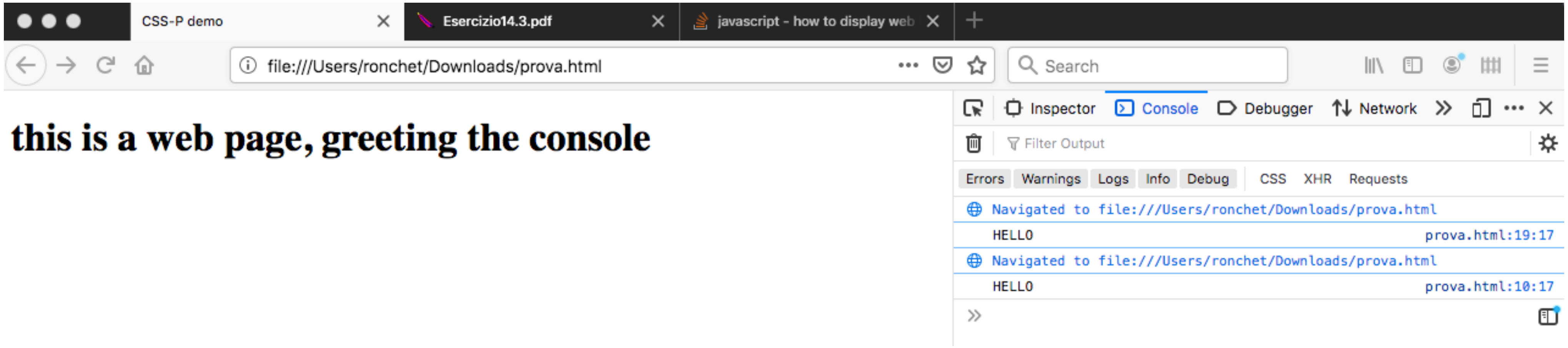


this is a web page, greeting the console

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>CSS-P
   demo</title>
           </head>
    <body>
        <h1>this is a web page,
   greeting the console
        <script>
        console.log("HELLO")
        </script>
    </body>
</html>
```

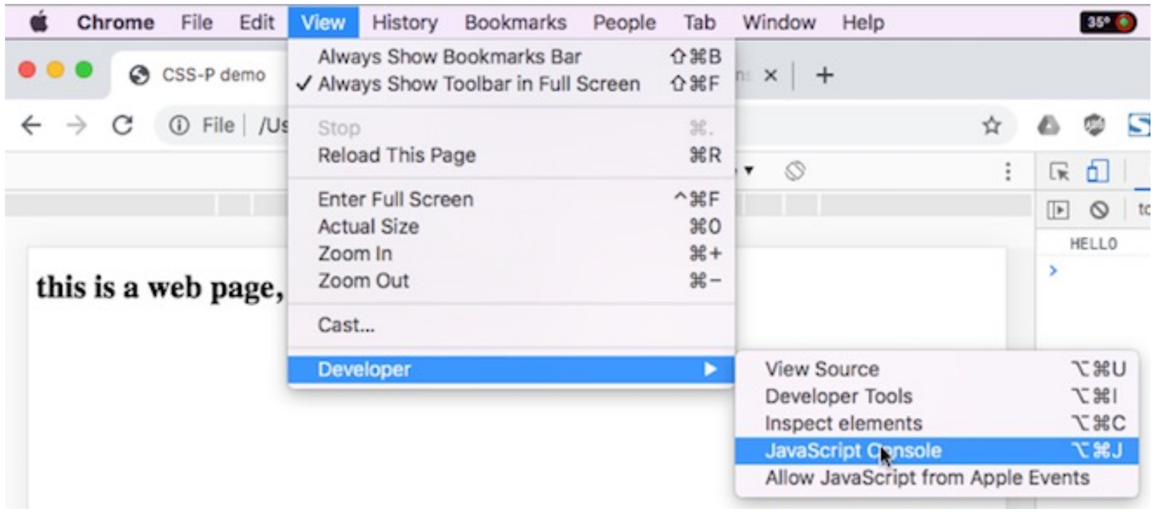
The Javascript console

Firefox


CSS-P demo
Esercizio14.3.pdf
javascript - how to display web
file:///Users/ronchet/Downloads/prova.html
Search
Inspector
Console
Debugger
Network
Filter Output
Errors Warnings Logs Info Debug CSS XHR Requests

this is a web page, greeting the console

```
Navigated to file:///Users/ronchet/Downloads/prova.html
HELLO                                                    prova.html:19:17
Navigated to file:///Users/ronchet/Downloads/prova.html
HELLO                                                    prova.html:10:17
```

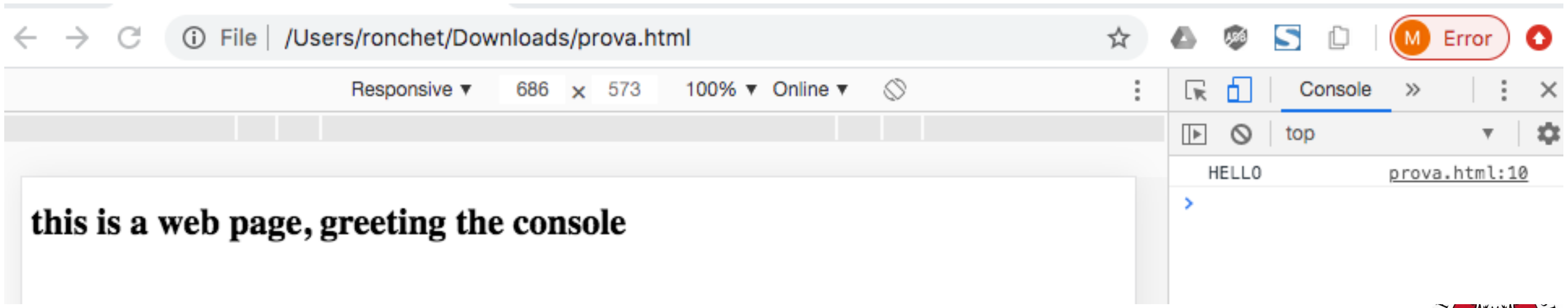


this is a web page,

```
<!DOCTYPE html>
<html>
    <head>
         <meta charset="UTF-8">
         <title>CSS-P
   demo</title>
            </head>
    <body>
        <h1>this is a web page,
   greeting the console
        <script>
        console.log("HELLO")
        </script>
    </body>
</html>
```

The Javascript console

Chrome


File /Users/ronchet/Downloads/prova.html
Error
Responsive 686 × 573 100% Online
Console
top

this is a web page, greeting the console

```
HELLO                    prova.html:10
```

Q

What's so peculiar in JavaScript string (and String)?

Strings

JavaScript Strings are a strange, double-headed beast…

They are both

a primitive data type and

an object

View also:
https://www.w3schools.com/js/js_strings.asp

Strings as Objects

Usually, JavaScript strings are primitive values, created from literals:

```
var firstName_primitive = "John";
```

But strings can also be defined as objects with the keyword new:

```
var firstName_object = new String("John");
```

When using `===` , firstName_primitive and firstName_object are NOT EQUAL, because the === operator expects equality in both type and value. With == they are EQUAL.

String operators

```
a="foot";
b="ball";
a+b => football

a>b => true
```

String methods

There are a lot of java-like methods

Some examples:

charAt(0)
indexOf(substring), lastIndexOf(substring)
charCodeAt(n),fromCharCode(value,…)
concat(value,…),slice(start,end)
toLowerCase(), toUpperCase()
replace(regexp,string), search(regexp)

```
a="foot";
b="ball";
a>b => true
```

- List: https://www.w3schools.com/jsref/jsref_obj_string.asp
  (Ignore constructor and prototype for now)

- Detailed examples: https://www.w3schools.com/js/js_string_methods.asp

Q

How is the + operator behaving in JavaScript?

operand1 + operand2 = result

+ Operator : rules

Phase 1: conversion

1) Any Object operand is converted to a primitive value (String, Number, Boolean);
2) If an operand is a String and the other is not, the non-String operand is converted to String
3) Any remaining Boolean operand is converted to Number (true->1, false->0)
4) Any remaining null operand is converted to Number (0)
5) Any remaining Undefined is converted to Number (NaN)

Phase 2: execution

6) If both operands are String, concatenation is performed.
7) If both operands are Number, sum is performed.

+ Operator : examples without instantiated Objects

x	y	Is there a String operand?	x+y
1	2	NO	3
"1"	2	YES -> Rule 2	12
1	null	NO -> Rule 4	1
"1"	null	YES -> Rule 2	1null
1	undefined	NO-> Rule 5	NaN
"1"	undefined	YES -> Rule 2	1undefined
1	true	NO ->Rule 3	2
"1"	true	YES -> Rule 2	1true
false	true	NO ->Rule 3	1
true	null	NO ->Rule 3, 4	1
true	undefined	NO->Rule 3, 5	NaN
null	null	NO ->Rule 4	0
null	undefined	NO->Rule 4, 5	NaN

Operations with integers

```
<script>
var x = "100";
var y = "10";
document.write(x * y);document.write("<br>");
document.write(x + y);document.write("<br>");
document.write(x * 1 + y);document.write("<br>");
document.write(x * 1 + y * 1);document.write("<br>");
</script>
```

```
OUTPUT:
1000
10010
10010
110
```

+ as unary operator

Unary + can be used to convert string to number

```
var y = "5";          // y is a string
var x = + y;          // x is a number (5)

var y = "Pippo";      // y is a string
var x = + y;          // x is a number (NaN)
```

Q

How can functions be defined in JavaScript?

```
function f(x) {return x*x}
```

Functions

____

```
<script>
function add(x,y) {return x+y;};
function multiply(x,y) {return x*y;};
function operate(op,x,y) {
                   return op(x,y);};
document.write(operate(add,3,2));
</script>
```

Output: 5

View also https://www.w3schools.com/js/js_functions.asp

Functions

JavaScript functions:

- do not specify data types for parameters.
- do not perform type checking on the passed arguments.
- do not check the number of arguments received.

If a function is called with **missing arguments** (less than declared), the missing values are set to undefined.

EcmaScript 2015 allows default parameter values in the function declaration:

```
function (x, y = 2) {
    // …
  }
```

Recursive f.

```
<HTML>
<HEAD>
<SCRIPT>
function fact(n) {
   if (n==1) return n;
   return n*fact(n-1);
}
</SCRIPT>
</HEAD>
<BODY>
<H2>Table of  Factorial Numbers </H2>
<SCRIPT>
for (i=1; i<10; i++) {
   document.write(i+"!="+fact(i));
   document.write("<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

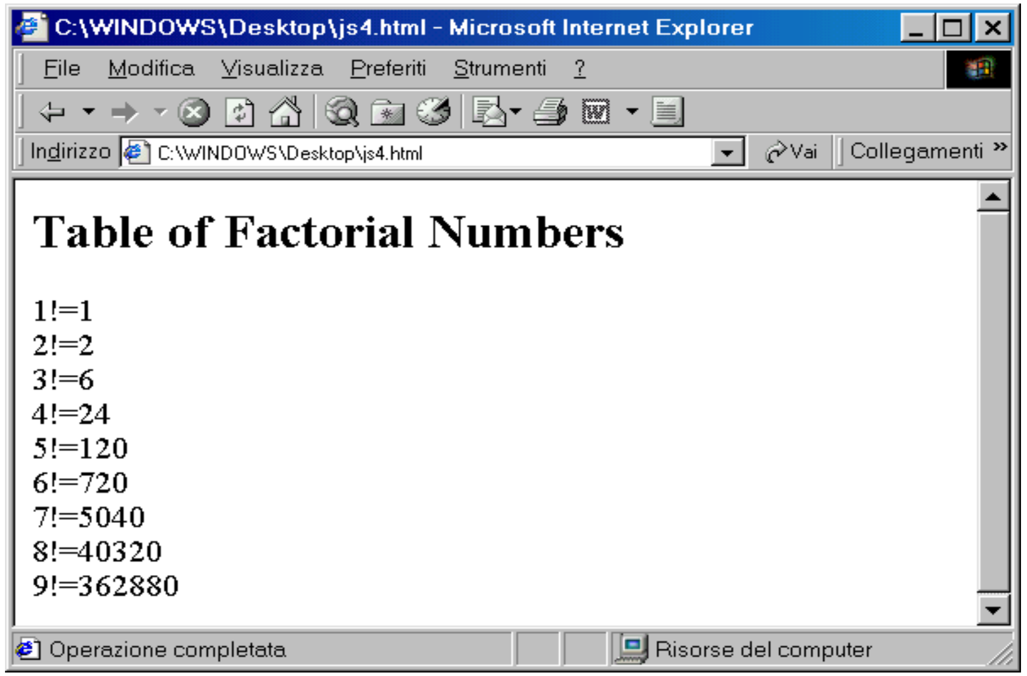
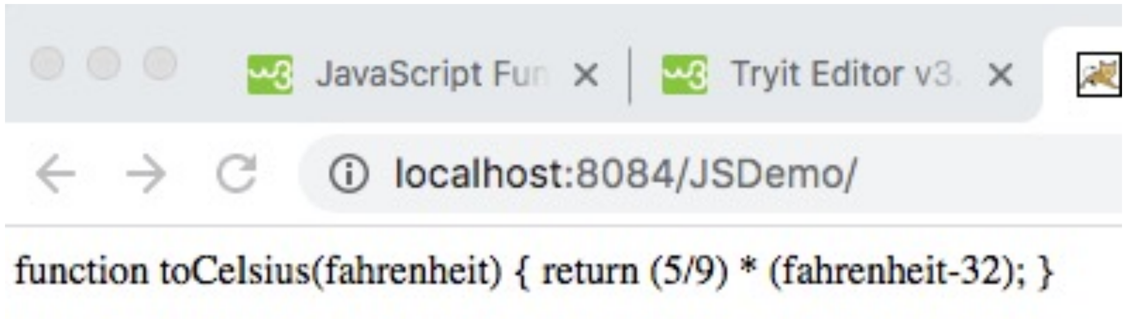

C:\WINDOWS\Desktop\js4.html - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
Indirizzo C:\WINDOWS\Desktop\js4.html
Vai
Collegamenti
Operazione completata
Risorse del computer

Table of Factorial Numbers

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

Functions

```
<!DOCTYPE html>
<html>
<body>
<script>
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}
document.write(toCelsius);
</script>
</body>
</html>
```



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

Function hoisting

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

The hoisting mechanism only moves the declaration. The assignments are left in place.

Function statements

The function statement declares a function.

```
hoisted();
```

OUTPUT:

```
This function has
been hoisted
```

```
function hoisted() {
  console.log('This function has been hoisted.');
};
```

Function declaration are hoisted

Function expressions

A JavaScript function can also be defined using an **expression**.
A function expression can be stored in a variable:
`var x = function (a, b) {return a * b};`

After a function expression has been stored in a variable, the variable can be used as a function: `product=x(2,3);`

Functions stored in variables do not need function names, as they are always invoked (called) using the variable name.

```
var fundef = function() {
  document.write('Hello');
};
```

```
fundef();
```

OUTPUT:

```
Hello
```

Function expressions and hoisting

Function expressions load only when the interpreter reaches that line of code.

```
fundef();
```

OUTPUT:

```
"TypeError: expression is not
a function
```

```
var fundef = function() {
  console.log('This will not work.');
};
```

Function expressions are not hoisted

Arrow functions

- `function multiplyByTwo(num){ return num * 2;}`

Possible redefinitions:

- `const multiplyByTwo=function (num){ return num * 2;}`
- `const multiplyByTwo= (num) => { return num * 2;}`
- `const multiplyByTwo= num =>{ return num * 2;}`
- `const multiplyByTwo= num => num * 2;`

`Usage (in all cases): multiplyByTwo(4);`

Mapping and filtering functions

```
<script>
twodArray = [1,2,3,4];
document.write( twodArray);
document.write( "<BR>");
document.write( twodArray.map( num => num * 2) );
</script>
```

```
OUTPUT:
1,2,3,4
2,4,6,8
```

```
<script>
twodArray = [1,2,3,4];
document.write( twodArray);
document.write( "<BR>");
document.write( twodArray.filter( num => num % 2 == 0));
</script>
```

```
OUTPUT:
1,2,3,4
2,4
```

More examples

```
<script>
multiplyByTwo=function(num){ return num * 2;}
document.write( multiplyByTwo(6));
document.write("<BR>");
myArray=[1,2,3];
document.write(myArray.map(multiplyByTwo));
</script>
```

```
OUTPUT:
12
2,4,6,
```

Q

How is scoping defined in JavaScript, and what is the behavior caused by hoisting?

Undeclared variables

In JavaScript, an undeclared variable is assigned the value "undefined" at execution and is also of type "undefined".

a ReferenceError is thrown when trying to access a previously undeclared variable.

```
<script>
  var t0=typeof(x);
  var x=3;
  t1=typeof(x);
  x="pippo";
  var t2=typeof(x);
  document.write(t0+", "+t1+", "+t2+"<br>");
  document.write(z);
</script>;
```

OUTPUT:
undefined, number, string

NO OUTPUT:
reference error

Variable scope 1

Type of declaration	Scope	Note
`x=10;`	always global	
`var x=10;`	function scope	(global if external to any function)
`let x=10;`	block scope	ES 6

Variable scope 2

- Variables declared with var live in their function scope (which, if outside any function, is global)

- Variables declared with let have block scope instead of function scope (ES 6)

- Variables declared without var or let are always global.

Variable hoisting

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

The hoisting mechanism only moves the declaration. The assignments are left in place.

```
…

…

var x=10;
```

```
var x;

…

…

x=10;
```

Variable declarations are processed before any code is executed.

Variable scope

```
{ var x = 2; }
// x CAN be used here

{ let y = 2; }
// y can NOT be used here
```

Variable scope

```
try {
p2 = (n,s) => document.write(n+":"+s+"<br>");
p1 = (n) => document.write(n+"<br>");
// code here can NOT use carName
      function f() {
          var carName="volvo"
          p1(carName); // code here CAN use carName
      }
      f();
      p1(carName); // code here can NOT use carName
} catch (err) {
    p2("ERROR",err.message);
}
```

```
OUTPUT:
volvo
ERROR:carName is not defined
```

Variable redefinition

```
var x = 10;
// Here x is 10


{
  x = 2;
  // Here x is 2
}


// Here x is 2
```

```
var x = 10;
// Here x is 10


{
  var x = 2;
  // Here x is 2
}


// Here x is 2
```

```
var x = 10;
// Here x is 10


{
  let x = 2;
  // Here x is 2
}


// Here x is 10
```

Variable scope - example 1

```
try {
p2 = (n,s) => document.write(n+":"+s+"<br>");
p1 = (n) => document.write(n+"<br>");
function f() {
   a = 20;
   var b = 100;
}
f();
p1(a);
p1("<hr>");
p1(b);
} catch (err) {
     p2("ERROR",err.message);
}
```

```
OUTPUT:
20
----------------------------
ERROR : b is not defined
```

Variable scope - example 2

```
try {
p2 = (n,s) => document.write(n+":"+s+"<br>");
p1 = (n) => document.write(n+"<br>");
function f() {
   p1(a);
   a = 20;
   var b = 100;
}
f();
} catch (err) {
     p2("ERROR",err.message);
}
```

NO OUTPUT:

ERROR: a is not defined

Variable scope - example 3

```
try {
p2 = (n,s) => document.write(n+":"+s+"<br>");
p1 = (n) => document.write(n+"<br>");
function f() {
   p1(b);
   a = 20;
   var b = 100;
}
f();
} catch (err) {
     p2("ERROR",err.message);
}
```

OUTPUT:

```
undefined
```

because of hoisting!

Variable scope - example 4

```
try {
p2 = (n,s) => document.write(n+":"+s+"<br>");
p1 = (n) => document.write(n+"<br>");
 p1(b);
function f() {
   a = 20;
   var b = 100;
}
f();
} catch (err) {
     p2("ERROR",err.message);
}
```

NO OUTPUT:

```
ERROR: b is not defined
```

Variable scope - example 5

```
<script>
p2 = (n,s) => document.write(n+": "+s+"<br>");
x=null; // DEF 1
function f() {
   var x = "A"; // DEF 2
   p2(2,"x in f "+x);
   {    var x=1; // DEF 3
        p2(3,"x in inner block in f "+x);
   }
   p2(4,"x in f "+x);
}
p2(1,x);
f();
p2(5,x);
</script>
```

OUTPUT:

```
1: null
2: x in f A
3: x in inner block in f 1
4: x in f 1
5: null
```

Variable scope - example 5 B

DEF 1	DEF 2	DEF 3	P1	P2	P3	P4	P5	
x=null let x=null var x=null	var x="A"	var x=1	null	A	1	1	null	
		x=1	null	A	1	1	null	
		let x=1	null	A	1	A	null	
	x="A"	var x=1	null	A	1	1	null	WHY?
		x=1	null	A	1	1	1	
		let x=1	null	A	1	A	A	
	let x="A"	var x=1			ERROR			(*)
		x=1	null	A	1	1	null	
		let x=1	null	A	1	A	null	

(*) A let variable cannot be redefined with a larger scope in an inner block.

Why then I can put let in def 1 without problems?

Variable scope - example 6

```
<script>
p2 = (n,s) => document.write(n+":"+s+"<br>");
function f() {
  x = "A"; // DEF 1
  p2(1,"x in f "+x);
  {   p2(2,"x in inner block in f "+x);
      let x=1; // DEF 2
  }
  p2(3,"x in f "+x);
}
try {
  f();
} catch(err) {
    p2("ERROR",err.message);
} </script>
```

OUTPUT:

```
1:x in f A
ERROR:Cannot access 'x' before initialization
```