

Rémy LENTZNER



**Getting started**  
*with*  
**JAVASCRIPT**

PROFESSIONAL TRAINING  
ÉDITIONS REMYLENT

**REMY LENTZNER**

**GETTING STARTED WITH JAVASCRIPT**

French original title : Bien débiter avec JavaScript

EDITIONS REMYLENT, Paris, 1ère édition, 2021

R.C.S. 399 397 892 Paris

25 rue de la Tour d'Auvergne - 75009 Paris

REMYLENT@GMAIL.COM

www.REMYLENT.FR



ISBN EPUB : 9782490275427

The Intellectual Property Code prohibits copies or reproductions intended for collective use. Any representation or reproduction in whole or in part by any means whatsoever, without the consent of the author or his successors in title or cause, is unlawful and constitutes an infringement, pursuant to articles L.335-2 and following of Intellectual Property Code.

This book is dedicated to Anna and Hélène.

I could not have written it without their support, advice, encouragements and proofreading.

Graphic illlustration : Anna Lentzner

In the same collection

[Improve your PivotTables with Excel](#)

[Improve your skills with Google Sheets](#)

[Programming macros with Google Sheets](#)

[Getting started with HTML](#)

Welcome to studying JavaScript language. This book is intended for all those who wish to learn it in order to organize and control the Internet page elements better. JavaScript fits into the organization of HTML tags, and you should know them before programming any functions.

You will discover how the language is structured through its organization : keywords, variables and control structures such as loops or conditional expressions. Because an Internet user can act on any part of an Internet page, the JavaScript language is event-driven, that is to say it "listens" to what happens, thanks to very specific commands you program in the code. Do not hesitate to reproduce the book examples to take up the syntax of the language.

This book is structured in 4 chapters.

Chapter 1 recalls some HTML code examples and shows that JavaScript is closely linked to it. You will learn how to write code inside simple functions that trigger at a specific time. You will study the DOM objects (Document Object Model) and its specificities.

Chapter 2 shows how to control the flow of the code using control structures. You will learn how to manipulate conditional expressions and loops.

Chapter 3 details the events management, such as mouse click or input area entry (or exit). You will appreciate the *AddEventListener* handler which listens to what happens within a web page. The events bubbling will show you how to avoid redundancies in the code.

Chapter 4 shows the most common methods that manipulate texts, numbers and dates.

I hope this book will give you a better understanding of the JavaScript language structure.

If you have any comments about this book or any questions, please do not hesitate to contact me at the following address : REMYLENT@GMAIL.COM. I will be sure to answer you.

Enjoy your reading.

The author

# TABLE OF CONTENTS

## **Chapter 1**

### **Fundamentals**

#### 1.1 The efficient JavaScript/HTML duo

##### 1.1.1 The Brackets editor helps to code

##### 1.1.2 A web page without JavaScript code

##### 1.1.3 An example with JavaScript coding

#### 1.2 The DOM or the tags tree

##### 1.2.1 The `querySelector()` and `getElementById()` methods

##### 1.2.2 Creating the web page directly in JavaScript

##### 1.2.3 Comments

#### 1.3 Where can I find information about JavaScript?

#### 1.4 The variables

##### 1.4.1 Declaration and assignment

##### 1.4.2 Scope of variables

##### 1.4.3 Scripts without function

#### 1.5 The functions

##### 1.5.1 Creating a function

##### 1.5.2 Calling a function

##### 1.5.3 Passing values into a function

##### 1.5.4 Passing variables into a function

##### 1.5.5 Anonymous functions

## **Chapter 2**

### **Control structures**

#### 2.1 The conditions

##### 2.1.1 if

##### 2.1.2 if ... else

##### 2.1.3 if ... elseif ... else

##### 2.1.4 switch

#### 2.2 Comparison operators

#### 2.3 Loops

##### 2.3.1 while

##### 2.3.2 do...while

##### 2.3.3 for

##### 2.3.4 for ... of

## **Chapter 3**

### **Events**

#### 3.1 Events caused by the Internet user

### [3.1.1 Keyboard events](#)

[Capturing a key with keypress](#)

[Coloring when entering text](#)

[Ctrl, Alt, Shift and A with a keydown event](#)

[Pressing several keys at the same time](#)

### [3.1.2 Mouse events](#)

[Displaying two images according to the left click](#)

[Mouse position on a map of Europe](#)

[An image cut into several parts](#)

## [3.2 Events in a form](#)

### [3.2.1 About the form element structure](#)

### [3.2.2 Tags that create elements](#)

### [3.2.3 The <input> tag](#)

### [3.2.4 Workshop : a form to request information](#)

## [3.3 AddEventListener and bubbling](#)

### [3.3.1 Listening to a click on 3 images](#)

### [3.3.2 Checking empty fields in a form](#)

### [3.3.3 The boiling of events](#)

[How can this bubbling help us ?](#)

[An example without bubbling](#)

[Bubbling avoids redundancy](#)

## **[Chapter 4](#)**

### **[Text, number and date](#)**

## [4.1 Manipulating strings of characters](#)

### [4.1.1 Concatenating string](#)

### [4.1.2 Concatenating special characters](#)

### [4.1.3 The length property](#)

### [4.1.4 The confirm dialog box](#)

### [4.1.5 Text manipulation methods](#)

## [4.2 Number manipulation](#)

### [4.2.1 Accuracy and rounding](#)

### [4.2.2 The very important parseFloat method](#)

### [4.2.3 The Math objects](#)

## [4.3 Handling dates](#)

### [4.3.1 Calculating your age](#)

### [4.3.2 Methods that affect dates](#)

# Chapter 1

## Fundamentals

This chapter recalls some HTML code examples and shows that JavaScript is closely linked to it, within the `<html>` and `</html>` tags. You will study how to write code inside simple functions that are triggered at a given moment. An HTML page is structured according to a document object pattern called *DOM*. The JavaScript language is object-oriented and is able to manage all the objects inside the Internet page, thanks to several specific expressions, such as methods and properties. JavaScript can also control the flow of the code using variables.

### 1.1 The efficient JavaScript/HTML duo

JavaScript is a programming language created in the 1990s to efficiently manipulate the information stored in a web page. When you want to create an Internet page, you use HTML (Hypertext Markup Language), that is a markup language designed to display data with any browsers. Under any circumstances, HTML cannot perform calculations or other complex operations. JavaScript will take care of this since it will be able to execute commands. When developing applications for the Internet, you work with HTML that helps you to create the structure of the web pages. The tags enable you to put the objects in place. For instance, you can create titles, text, images, links, input fields, headers, footers, logos, menus and many others.

At the same time, CSS (Cascading Styles Sheets) is used to organize the data presentation. For example, you can set a page logo, right-aligned and wrapped with a border. Likewise, a specific font and a text size can be specified.



The job of CSS language is essentially to manage the presentation.

Finally, you will use JavaScript to create functions that perform operations above the objects. For example, you can make a calculation from different dates, find a result from numerical values, or manage web user actions when using the mouse or the keyboard.

There is also the PHP language that is used to manage data in a client-server context. This language is preferred by programmers when information is sent to a server and when an answer is needed. It is also possible to use some derivative JavaScript language, that handles these client-server operations, but this kind of programming is complex and beyond the scope of this book.

The HTML-CSS-JAVASCRIPT trio is often referred to as *standard web language*.

If you don't know HTML-CSS at all, you should start by learning them. If you want to practise, you can purchase the book *Getting started with HTML* at [www.remylent.fr](http://www.remylent.fr).

Creating Web pages is always an integration between HTML tags, CSS presentation commands and JavaScript functions.

### **1.1.1 The Brackets editor helps to code**

With any text editor, you can write a web page that contains the HTML/JavaScript code. You type the code and you save the file, first with an *.html* extension, then with a *.txt* extension. Afterwards, you open your *.html* file with your browser. You will see the result immediately. To modify the code, reopen the *.txt* file. This solution is not practical at all because it forces you to open the *.txt* file to make a change, then you have to save the file again with *.html* and *.txt* extension.

On other way (among many others) is to use the free *Brackets Editor* that I use for my program tests. This tool allows you to write the code, then you will see the result immediately, via the *Live Preview* option. If you change the code, simply refresh the web page in the

browser to see the result. You can download Brackets for Mac at <https://brackets.fr.uptodown.com/mac>, but you can also download it for Windows or another operating system.

The software Brackets always shows the result in the default browser. If you want to use another browser, simply change the default one.

Not all browsers display the HTML/CSS/JAVASCRIPT code in the same way. And that is why we have difficulties to create web pages that work correctly on many software platforms as computers, tablets, mobiles, connected objects, etc). If you want to develop strong applications, your code will have to take these differences into account.

### 1.1.2 A web page without JavaScript code

Figure 1.1 shows an example of an HTML page that displays a list of choices. A menu offers to click on four links that call up another specific page.



*Figure 1.1 : A menu in a HTML page*

The code below shows how to make this page. HTML instructions are tags, i.e. expressions wrapped with < and > signs.

```
<html>
<!-- beginning of the header -->
<head>
<!-- beginning of style definition -->
<style>
/* characteristics of the objects present in the body*/
body {
```

```
/* the properties below apply to the body of the page
/*I define the left margin*/
margin-left: 10px;
/* I define the top margin*/
margin-top: 50px;
/* I define the size and the font
and the family in case the font does not exist*/
font: 18px Verdana, sans-serif;
}
/* characteristics of menu items*/
.menu {
/*I say there are no chips in the list*/
list-style-type:none;
}
/* characteristics of the items in the list*/
.li menu {
/* I declare a floating menu which is positioned on the left and whose options are
vertical*/
float:none;
}
/* characteristics of the anchor elements (anchor)*/
.menu a {
/* width of the menu option*/
width: 200px;
/* height of the menu item*/
height: 30px;
/* I display per block*/
display:block;
/* I centre the text*/
text-align: center;
/*I place a border*/
border: 1px solid gray;
/*simple text. no underscores, strikethroughs, etc*/
text-decoration:none;
/* red colour for text*/
color:red;
```

```

/* sky blue color for the background*/
background-color:azure;
}
/* characteristics of element a when I fly over it*/
.menu a:hover {
/*if I hover over the option, its background turns yellow*/
background: yellow;
/*if I hover over the option, its border turns black*/
border: 1px solid black;
}
/* characteristics of element a when I click on it*/
.menu a:active {
/*when I click on the option, its background turns blue*/
background: blue;
/*when I click on the option, its background turns blue*/
}
</style>
</head>
<!beginning of the body of the page>
<body>
Choose a year from these options
<!start of menu items>
<ul class="menu">
<li><a href="prog1.html"> Year 2018</a></li>
<li><a href="prog2.html"> Year 2019</a></li>
<li><a href="prog3.html"> Year 2020</a></li>
<li><a href="prog1.html"> Year 2021</a></li>
</ul>
</body>
</html>

```

In this example, the HTML and CSS languages are fully integrated without JavaScript coding. The purpose of this page is to display a new page when the user clicks above a menu option.

The following list details the main aspects of this code :

The program starts with the tag `<html>` which gives information about the type of language used. We could have had `<XML>`.

Each tag is wrapped by the `<` and `>` signs.

The `<head>` tag ends with the `</head>` tag meaning header parameters.

The `<style>` tag ends with the `</style>` tag. It defines the CSS styles that will be used for the objects presentation created in the body of the web page.

The `<body>` tag ends with the `</body>` tag. It shows the body of the web page, where different objects are defined.

In the styles, the characteristics of each element are defined by a word followed by braces `{` and `}`.

In the body of the page, any style that must format specific elements uses the instruction `class="keyword"`. When a style applies to a body element, there is no need to use the instruction `class`.

### **1.1.3 An example with JavaScript coding**

JavaScript is a programming language that enhances the HTML language, allowing you to execute commands written between the two tags `<script>` and `</script>`.

As the code will act on the elements present in the web page, JavaScript will be able to recognize all the objects defined by the HTML code. In programming terms, the web page is called a *document*.

Let's take a simple example : Figure 1.2 shows a web page that asks the user to enter two dates, an invoice date and a payment date. The JavaScript code embedded in the HTML code will calculate the number of days between these two dates.

Figure 1.2 shows a form that proposes to enter two dates and will perform the result as soon as the user presses the *Calculate* button.

Number of days between two dates

Enter invoice date: 10/12/2020

Enter payment date: 05/01/2021

Number of days between the two dates: 26

Calculate

*Figure 1.2 : A difference between two dates*

The following code shows how to carry out this HTML page with the embedded JavaScript code.

```
<html>
<head>
<style>
/* presentation of the input fields*/
input {
/*absolute fixed position*/
position: absolute;
left: 240px;
/*family of fonts*/
font-family: sans-serif;
/*character size*/
font-size:12px;
width:160px;
}
/* presentation of label*/
label {
/* left position*/
left: 4px;
/*family of fonts*/
font-family: sans-serif;
/*character size*/
font-size:12px;
}
/* button presentation*/
input [type=button]
{
```

```
position: absolute;
/* left position*/
top:130px;
left:10px;
font-size:12px;
}
/* end of styles*/
</style>
<!-- start of script-->
<script language="JavaScript">
function calculation(MF)
{
/* I retrieve the value of the invoice date in a variable*/
var d1=document.MF.dateI.value;
/*I transform the string d1 into date*/
var Td1=new Date(d1);
/*I calculate the date in milliseconds*/
md1=Td1.getTime();
/* I retrieve the value of the settlement date*/
var d2=document.MF.dateS.value;
/*I transform the string d2 into date*/
var Td2=new Date(d2);
/*I calculate the date in milliseconds*/
md2=Td2.getTime();
/* I make the difference in milliseconds*/
var diff=md2-md1;
/*I retransform milliseconds into number of days*/
var result=diff/(1000 * 3600 * 24);
/* I place the result in the 3rd zone*/
document.MF.nbr.value=result;
}
/* end of script*/
</script>
<!-- end of header-->
</head>
```

```

<! beginning of the body of the page>
<body>
<h5>Number of days between two dates</h5>
<!--start of form definition-->
<form name="MF">
<label> Enter invoice date:</label>
<input type="date" name="date1"><br><br>
<!--the expression br means : you go to the line-->
<label>Enter payment date:</label>
<input type="date" name="dateS"><br><br>
<label> Number of days between the two dates:</label>
<input type="text" name="nbr"><br><br>
<!-- if I click on the button, then I trigger the calculation function-->
<input type="button" value="Calculate" OnClick="calculation(MF)">
</form>
</body>
</html>

```

1. The code is structured in several parts :

- The header is defined by the *<head>* tag.
- The element presentation is defined by the *<style>* tag.
- The JavaScript function is defined by the *<script>* tag.
- The body of the Web page is defined by the *<body>* tag.
- The elements are embedded in a form defined by the *<form>* and *</form>* tags.

2. The document elements

An HTML page is structured in a set called DOM as *Document Object Model*. It represents a hierarchy of objects or a tag tree, with these objects fitting into each other.

In the body of the page, elements are named by words written in inverted commas.

```

<input type="date" name="date1">

```



```
<input type="date" name="dateS">
```

```
<input type="text" name="nbr">
```

These words enable us to point to the object.

The expression `var d1=document.MF.date1.value` indicates that a variable `d1` (defines with the word `var`) is declared. In this variable, we will store the value entered on the keyboard by the user. The value of the invoice date `value` is defined by the word `date1` which is part of the form called `MF`, itself integrated in the document.

This way of writing comes from the object-oriented nature of the JavaScript language. It is said to be a *dot language*.

```
document.form_name.name_of_the_input_zone.value=some_value
```

3.The function is triggered by a click event.

When you click on a button in a form, you trigger a *click* event. It is programmed using the expression `OnClick="function_to_execute()"`.

```
<Input type="button" value="Calculate" OnClick="calculation(MF)">
```

If you click on the *Calculate* button, you trigger the calculation function integrating all the characteristics from the form named `MF` defined in the page body.

```
<form name="MF">
```

When you place the name of the form as a parameter in the brackets of the function, you send all the form characteristics to the function. You can then retrieve the fields and assign results to them.

This is what happens here, since the calculation of the number of days is sent to the 3rd field.

```
document.MF.nbr.value=result;
```

4. The necessary transformation of the dates into milliseconds to calculate the number of days is done in the following way :

Managing dates is not easy in JavaScript because the language must be able to juggle different countries and time zones. The manipulation of days, months, years and the calculations are carried out with specific commands. In order to calculate dates without

problems, you have to transform dates into milliseconds. After the calculations, the milliseconds will be transformed back into days, months or years.

To facilitate the dates input into a *date-type* input field, you can use an integrated template (figure 1.3).

The figure shows a web form titled "Number of days between two dates". It contains two input fields: "Enter invoice date:" and "Enter payment date:", both with a "dd/mm/yyyy" placeholder and a calendar icon. Below these is a label "Number of days between the two dates:" followed by a calendar widget. The calendar is for "January 2021" and shows a grid of days from 1 to 31. The 4th of January is highlighted with a blue square. There are up and down arrows at the top right of the calendar, and a "Today" link at the bottom right.

Figure 1.3 : Facilitated date entry

Keep in mind that the result of any input is a string of characters, with which you cannot perform calculations. You will have to transform the string into a date.

```
var Td1=new Date(d1);
```

In that expression, a new variable is defined that will store a new date type object taking the value entered on the keyboard into account. This expression is called *Date constructor*. It has many properties and action verbs called *methods*.

JavaScript stores dates internally in the form of a *unix timestamp* that means the number of milliseconds elapsed since January 1, 1970 at midnight UTC. It facilitates the comparison between the dates. After computing, you will have to convert back into days.

The *getTime()* method transforms the date into milliseconds.

```
/*I calculate the date in milliseconds*/
```

```
md2=Td2.getTime();
```

The expression below carries out the difference between the two blocks of milliseconds.

```
/* The difference of milliseconds*/
```

```
var diff=md2-md1;
```

Then the milliseconds found are converted back into days. The value is divided by 3600 seconds \* 24 hours \* 1000 milliseconds.

```
/*I retransform the milliseconds into days*/  
var resultat=diff/(1000 * 3600 * 24);  
/* I place the result in the 3rd zone*/  
document.MF.nbr.value=result;
```

The code in this example may be a little bit complicated to start with, but it shows how to retrieve data from a form and how to manage dates. Feel free to come back to it later, after studying the other examples in this book.

## 1.2 The DOM or the tags tree

The DOM (*Document Object Model*) is a programming interface, standardized by the regulatory organization called W3C. It is the group that defines the various HTML tags and CSS properties all along. As browsers have evolved, some tags have been deprecated and the W3C asks developers to stop using them.

For example, you should no longer use the `<frame>` tag but `<iframe>` with styles.

The DOM is a structured representation of the document as a form of a tree, automatically created by the browser. Each branch has a node that contains objects. JavaScript has access to it through properties and methods.

Therefore, the internal structure of an HTML program looks like a tag tree. It always has a root which is an entry point for the organisation of the tags. This is followed by one or more paths, nodes, leaves, other branches, and finally objects such as fields, tags, images, texts, menus, etc.

Figure 1.4 shows the tree and its root that is always the `<html>` tag. The root has at least two leaves : the `<body>` and `<head>` tags. The `<body>` tag is the body of the web page, the `<head>` tag is the header. Other components will be added to the tree later.

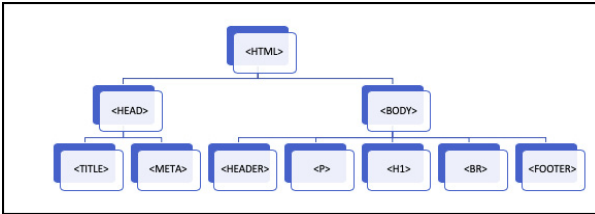


Figure 1.4 : An example of a tag tree

### 1.2.1 The `querySelector()` and `getElementById()` methods

The DOM interface allows you to have access to the objects by special orders. For example, you can use `querySelector()`, `querySelectorAll()`, `getElementById()` or other methods. They allow you to point any nodes in the DOM tag tree.

These expressions allow you to modify the properties of the objects or simply retrieve the values.

Note. Programming is always defined in the `<script>` part.

The following code shows how to modify elements in a web page, using the methods `querySelector()` and `getElementById()`.

```

<!doctype html>
<html>
<style>
/* I declare the styles for the texts defined inside the <p> tag */
p{
/* I define the character font*/
/*font family*/
font-family : sans-serif;
/*size*/
font-size:12px;
}
body{
background-color:azure;
}
input[type=button]{
background-color:lightgrey;
font-size:12px;
}
  
```

```

}
</style>
<body>
<!-- I declare a form with its name
<form name="myForm">
<H5> List of books to publish</H5>
<br>
<!-- first text with an ID P1>
<p id="p1"> Getting started with JavaScript</p>
<!-- second text with an ID P2>
<p id="p2"> Improve your skills with Google Sheets</p>
<!-- third text with an ID P3>
<p id="p3"> Programming macros with Google Sheets and JavaScript</p>
<br>
<!-- the button>
<input type="button" value="Test with QuerySelector and getElementById"
onclick="modify(myForm)">
</form>
</body>
<script language="JavaScript">
function modify(myForm)
{
/* I modify the backgroundcolor of the body */
document.querySelector("body").backgroundColor="green";
/* I modify the text of the H5 tag*/
document.querySelector("H5").textContent="List of books already published";
/* I modify the text color of the H5 tag*/
document.querySelector("H5").style.color="red";
/* I modify the text of the p1 tag*/
document.getElementById("p1").textContent="Getting started with HTML"
document.getElementById("p1").style.color="blue";
/* I modify the text of the p2 tag*/
document.getElementById("p2").textContent="The macros with Google Sheets";
/* I modify the text color of the p2 tag */
document.getElementById("p2").style.color="green";
document.getElementById("p2").style.fontSize="14px";

```

```

/* I modify the text color of the p3 tag */
document.getElementById("p3").textContent="Google Sheets on line";
document.getElementById("p3").style.color="pink";
document.getElementById("p3").style.fontSize="16px";
}
</script>
</html>

```

In the `<body>` part, you characterize a `<p>` tag with the keyword *id*, then you use the `getElementById` method that points again to the `<p>` tag. Sometimes, the keyword *name* is also used to characterize a tag and the `getElementsByName` method will be used afterwards.

There are several ways of doing this pointing because browsers are very different. If one technique doesn't work, another will probably have better luck. Figure 1.5 shows the result of step 1 when displaying the web page.

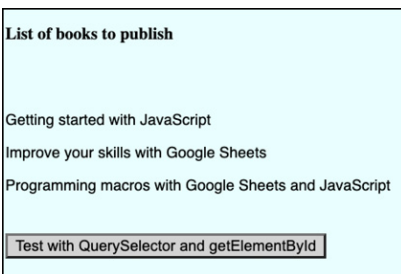


Figure 1.5 : Displaying text in the page

Figure 1.6 shows the modification after clicking on the *Test* button.

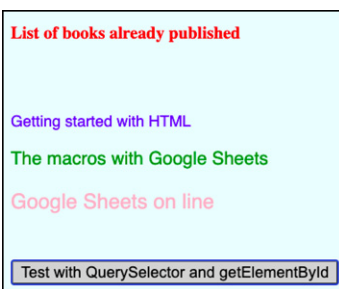


Figure 1.6: Changing texts and properties

## 1.2.2 Creating the web page directly in JavaScript

JavaScript allows you to create web pages without writing many lines

of HTML code. To do this, you will use the properties of the DOM, i.e. the hierarchy of tags, branches and nodes in the tree.

A program coded with the HTML language starts with the tag `<html>`. In the header `<head>`, you write the styles in the `<style>` part, the functions in the `<script>` part, then finally the body of the program in the `<body>` part.

The DOM is built by the browser when reading the program instructions.

Sometimes, you can insert the `<body>` part before the scripts, so that the objects are placed in memory before the rest of the code. It depends on the application you wish to develop.

Most of the time, the `<body>` part is placed before the JavaScript code. In the script, it is possible to integrate event management to control areas, mouse moving or other events (see chapter 3).

When you define a `<p>` tag, you create what is called a *node*.

The following code creates three nodes :

```
<!-- place the 1st text characterizing it with a P1> identifier.
<p id="p1"> Getting started with JavaScript</p>
<!-- place the 2nd text characterizing it with a P2> identifier.
<p id="p2"> Improve your skills with Google Sheets</p>
<!-- place the 3rd text characterizing it with a P3> identifier.
<p id="p3"> Programming macros with Google Sheets and JavaScript</p>
```

When you define styles for a group of tags `<p>`, you create properties for the group of nodes.

```
p{
/* I set the font for the texts*/
/*police family*/
font-family: sans-serif;
/*character size*/
font-size:12px;
}
```

The previous code uses HTML with tags.

Figure 1.7 shows the representation of the DOM.

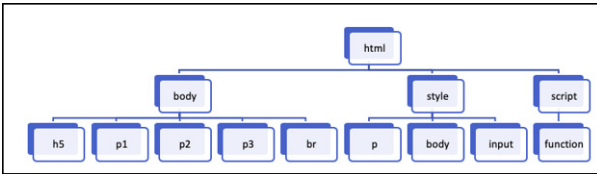


Figure 1.7: The DOM is built in the browser.

Note that you can write the code to build a web page without writing all the tags. Several methods are important for creating the nodes (the tags), attaching them and defining their properties.

- *document.createElement*. This expression is used to create a tag in memory.
- *appendChild*. This method allows you to add an element in the tree.
- *document.querySelector*. This expression is used to point to a tag (an element, a node) which is already created in the DOM tree.

The following code creates a web page with only the tags `<html>` and `<body>`.

```
<!doctype html>
<html>
<script>
function creation()
{
var N_body=document.querySelector("body");
N_body.style.background="azure";
var new_H5=document.createElement("H5");
N_body.appendChild(new_H5);
new_H5.textContent="List of the books to publish";
var li1=document.createElement("br");
N_body.appendChild(li1);
var new_p1=document.createElement("p");
new_p1.id="p1";
```



```

new_p1.textContent="Getting started with JavaScript";
N_body.appendChild(new_p1);
var new_p2=document.createElement("p");
new_p2.id="p2";
new_p2.textContent="Improve your skills with Google Sheets";
N_body.appendChild(new_p2);
var new_p3=document.createElement("p");
new_p3.id="p3";
new_p3.textContent="Programming macros with Google Sheets and JavaScript";
N_body.appendChild(new_p3);
var new_button=document.createElement("input");
new_button.type="button";
new_button.value="Test de modification";
new_button.style.backgroundColor="pink";
new_button.fontSize="12px";
new_button.addEventListener("click",change);
N_body.appendChild(new_button);
}
window.onload=function(){
creation();
}
function change()
{
document.querySelector("H5").textContent="List of books already published";
document.querySelector("H5").style.color="red";
document.getElementById("p1").textContent="Getting started with HTML";
document.getElementById("p1").style.color="blue";
document.getElementById("p2").textContent="The macros with Google Sheets";
document.getElementById("p2").style.color="green";
document.getElementById("p2").style.fontSize="14px";
document.getElementById("p3").textContent="Google Sheets on line";
document.getElementById("p3").style.color="pink";
document.getElementById("p3").style.fontSize="16px";
}
</script>

```

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

### 1.2.3 Comments

You can comment on the rows of code with several ways :

- In the `<script>` part, enter `//` at the beginning of the line.
- In the `<script>` part, for a group of comments, enter `/*` at the beginning of the line then `*/` at the end.
- In the `<html>` part and outside the script part, enter `<!--` at the beginning of the line and `-->` at the end of the line.

## 1.3 Where can I find information about JavaScript?

Several sites can help you to study the very numerous JavaScript programming expressions.

Here are three sites that are very well-known in the developer community.

1. <https://www.w3schools.com/js/default.asp>

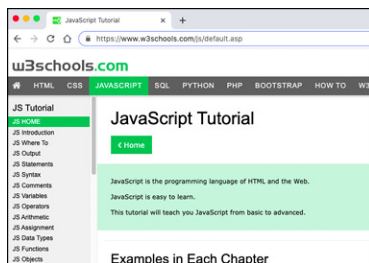


Figure 1.8 : The W3C schools

2. <https://developer.mozilla.org/fr/docs/Web/Tutoriels>

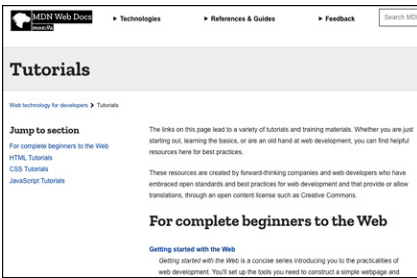


Figure 1.9: The Mozilla community

3. <https://github.com/search?q=JAVASCRIPT>

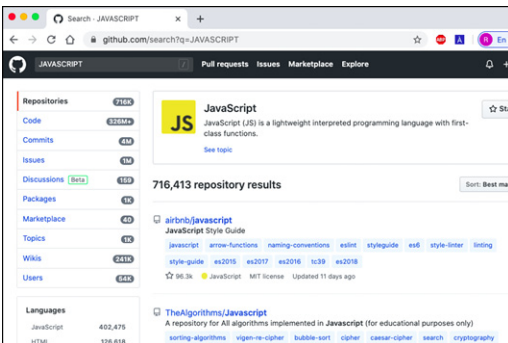


Figure 1.10: The Github community

## 1.4 The variables

A variable is a container, a kind of "shoebox" in which only one shoe is stored. A variable always has a name that must begin with a letter, never a number. Do not put any special characters or spaces in the name of a variable.

Caution : variable names are case sensitive like all JavaScript instructions.

### 1.4.1 Declaration and assignment

- You declare a variable using either the keyword *var* or the keyword *let*.
- To manipulate a variable, you have to declare it. Then you assign a value with the sign equal (=).
- You assign a value to a variable writing it from right to left.

```
let firstname="joe";
```

```
var name="bow";
```

or

```
var namecomplete;
```

```
nomcomplete = "joe bow";
```

The word *let* has a block span defined between braces, as *if{}*, *while{}*, etc. The word *var* has a function range. The declaration of a variable with the word *let* must be done before its assignation. A variable can be declared with *var* after it has been assigned. It is always preferable to declare the variables before using them.

The following code shows an example of variable assignment in an existing tag.

```
<!doctype html>
<html>
<script>
function test_var()
{
// I define a variable
var vName="joe bow";
// I put the variable content into the tag p1
document.getElementById("p1").innerHTML=vName;
}
</script>
<body>
<!-- create a tag p1 -->
<p id="p1">Value of the variable</p>
<!-- I create a push button that calls the function -->
<input type="button" value="test" onclick="test_var()">
</body>
</html>
```

The variable *vName* is assigned in the *<p>* tag identified by *p1* thanks to the following expression :

```
document.getElementById("p1").innerHTML=vName;
```

The *innerHTML* property (*inner*) designates the content between the incoming tag and the closing tag.

### 1.4.2 Scope of variables

The scope of a variable is the location where the variable can be used in a function. A distinction is made between local scope and global scope.

A variable defined in a function is considered to have a local scope, and will not be recognized in another function.

A variable defined outside a function has a global scope and can be accessible by several functions.

The following code shows local and global variables defined in some functions.

```
<html>
<script>
function part1()
{
// local variables
var v1;
var v2;
v1="paris";
v2="nice";
alert(v1);
alert(v2);
}
function part2()
{
// the variables are not accessible
alert(v1);
alert(v2);
}
//I perform both functions
part1();
part2();
```

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

### 1.4.3 Scripts without function

The code below shows another way to write the script. Indeed, this one is placed inside the body but without using any function. The program performs the code and finally displays the cities.

```
<!doctype html>
<html>
<!-- declare the body -->
<body>
<p id="p1">Paris</p>
<p id="p2">Lyon</p>
<p id="p3">Nice</p>
<p id="p4">Marseille</p>
<p id="p5">Toulouse</p>
<script>
let line;
for(line=1;line<=5;line++)
{
alert(document.getElementById("p"+line).textContent);
}
</script>
</body>
</html>
```

A *for* loop is used to display the contents of the different tags. The translation is : for the variable *line* equal from 1 to 5, repeat the code placed between the braces, then at each item, increment the variable by one.

The expression *line++* is equivalent to *line = line + 1*.

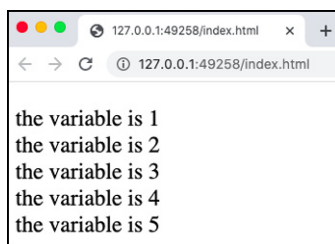
The expression *"p"+line* means to concatenate the string *"p"* to the value of the variable *line*. The result will be *p1* then *p2*, then *p3* then *p4*

then *p5*.

The following code fills the body with the result of successive concatenation.

```
<!doctype html>
<html>
<!-- declare the body -->
<body>
<p id="test"></p>
<script>
// I declare two variables i and text
var i
var text="" // I initialise the variable
// I loop
for(i=1;i<=5;i++)
{
// I affect the text variable
text += "the variable is " + i + "<br>";
}
//I display the concatenation result after the 5 items.
document.getElementById("test").innerHTML=text;
</script>
</body>
</html>
```

Figure 1.11 shows the result.



*Figure 1.11: Successive concatenation*

The expression `text += "the variable is " + i + "<br>;` is equivalent to `text = text + "the variable is " + i + "<br>;`

Note the `<br>` tag that causes the line break. The code uses the

keyword *var* to declare variables.

## 1.5 The functions

A function is a group of lines of codes whose purpose is to perform a specific task. A function is named and reusable. Its name begins with a letter, contains no space or special character, and must not be a JavaScript keyword.

### 1.5.1 Creating a function

The functions are declared in the `<script> </script>` part.

The syntax is as follows :

```
function function_name()
{
expressions
}
```

For example :

```
function info()
{alert("The message has been sent successfully");
}
```

Variables can also be included in the definition of the function.

```
function function_name(var1,var2,var3,etc)
{content of the function}
```

### 1.5.2 Calling a function

When you want to call a function, you invoke it by typing its name and parameters if any. The code below performs three functions, one after the other.

```
part1();
part2();
part3();
```

Figure 1.12 shows a web page in which you enter three parameters : height, width and length. A first function places the pointer in the first input field as soon as the page is loading. A second function



calculates the volume by retrieving the values entered in the input fields.

**Volume calculation**

Enter height	<input type="text" value="10"/>
Enter width	<input type="text" value="12"/>
Enter length	<input type="text" value="3"/>

Click to calculate the volume

The volume is: 360.00

*Figure 1.12: A function that calculates a volume*

The code below shows how to invoke the two functions :

```
<html>
<style>
/*styles for headings*/
label{
position:absolute;
left:30px;
}
/*styles for the input areas*/
input[type=text]
{
position:absolute;
left:150px;
width:40px;
}
/*styles for tag p*/
p{
position:absolute;
left:30px;
}
input[type=button]
{
position:absolute;
top:120px;
height:30px;
```

```
left:30px;
width:200px;
}
</style>
<!--when the page loads, I invoke a function-->
<body onload="parameter()">
<H3>Volume calculation</H3>
<label> Enter height </label>
<input type="text" value="10" id="v1"><br>
<label> Enter width </label>
<input type="text" value="12" id="v2"><br>
<label> Enter length </label>
<input type="text" value="3" id="v3"><br>
<input type="button" value="Click to calculate the volume" id="v4" onclick="calculate()">
<br><br>
<p name="v5">The volume is : </p>
</body>
<script>
function parameter()
{
//I place the focus (the pointer) in the height area
document.getElementById("v1").focus();
}
function calculate()
{
// I calculate the volume
var height =document.querySelectorAll("input")[0].value;
var width=document.querySelectorAll("input")[1].value;
var length=document.querySelectorAll("input")[2].value;
var volume=height*width*length;
//I concatenate the volume to the text of the p-tag
document.querySelectorAll("p")[0].textContent="The volume is: " + volume.toFixed(2);
}
</script>
</html>
```

### 1.5.3 Passing values into a function

You can pass values into a function when you invoke it. Be sure to follow the order of the variables. The example below shows a function called *watch* that is defined with 3 parameters.

```
function watch(v1,v2,v3)
{ alert(v1+" "+v2+" "+v3);}
```

The code below calls the function passing values.

```
watch("joe", "pass", "the jazzman")
```

### 1.5.4 Passing variables into a function

The following example shows a function in which variables are transferred.

```
<html>
<style>
/*styles for headings>*/
label{
position:absolute;
left:10px;
}
/*styles for the input areas>*/
input[type=text]{
position:absolute;
left:200px;
width:40px;
}
/*styles for tag p*/
p{
position:absolute;
left:30px;
}
</style>
<!--when the page loads, I call the function-->
<body onload="parameter()">
<H3>calculation of the volume</H3>.
```

```
<label> Enter height </label>
<input type="text" value="10" id="v1"><br>
<label> Enter width </label>
<input type="text" value="12" id="v2"><br>
<label> Enter length </label>
<input type="text" value="3" id="v3"><br>
<input type="button" value="volume calculation" id="v4" onclick="process()"><br>
<p name="v5">volume is -> </p>
</body>
<script>
//-----
function parameter()
{
//I place the focus (the pointer) on the height
document.getElementById("v1").focus();
}
//-----
function calculate_volume(v_height, v_width, v_length)
{
// I calculate the volume
return v_height * v_width * v_length;
}
//-----
function process()
{
//I retrieve the values typed in the fields
var f_height =document.querySelectorAll("input")[0].value;
var f_width=document.querySelectorAll("input")[1].value;
var f_length=document.querySelectorAll("input")[2].value;
//I invoke the function for volume calculation
volume=calculate_volume(f_height,f_width,f_length);
//I concatenate the volume to the text of the p tag
document.querySelector("p")[0].textContent="the volume is: " + volume.toFixed(2);
}
</script>
```

</html>

### 1.5.5 Anonymous functions

There are functions that do not have a name. They are used when the function does not need to be reused. However, reading the code is less simple. Anonymous functions are often used in events management.

Here is the syntax of such a function :

```
function()  
{  
  code;  
}
```

Example 1 : without anonymous function

The following code transforms an image of a rabbit into a flower when you click on it (figure 1.13).



*Figure 1.13: A rabbit to flower*

```
<!DOCTYPE html>  
<html>  
<head>  
<base src="/Applications/Brackets.app/Contents/samples/en/First steps">  
</head>  
<style>  
/*I set the left position of the image*/  
img{  
  left:20px;  
}  
</style>  
<body>  
<img width="10%" height="10%" id="image1">  
</body>
```

```

<script>
//execute the function to display the rabbit
rabbit();
//I listen if I click on the picture
document.getElementById("image1").addEventListener("click", flower, false);
function rabbit() {
//I display the rabbit
document.getElementById("image1").setAttribute("src", "rabbit.jpg");
}
function flower() {
//I display the flower
document.getElementById("image1").setAttribute("src", "flower.jpg");
}
</script>
</html>

```

Example 2 : with an anonymous function

```

<!DOCTYPE html>
<html>
<head>
<base src="/Applications/Brackets.app/Contents/samples/en/First steps">
</head>
<style>
/*I set the left position of the image*/
img{
left:20px;
}
</style>
<body>
<script>
/* I start listening when the page loads*/
/* an anonymous function is used here*/
window.onload = function() {
document.getElementById('image1').addEventListener('click', flower, false);
function flower() {
document.getElementById("image1").setAttribute("src", "flower.jpg");
}
}

```

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

Here, the anonymous function runs when the page is loading, listening to the *onload* event.

```
window.onload = function() {  
  document.getElementById('image1').addEventListener('click', flower, false);
```

The events are discussed in chapter 3.

### **In brief**

The HTML and JavaScript languages are intimately linked within the *<html>* and *</html>* tags. They describe exactly what happens in the web page. The JavaScript code can be written with functions or directly in the *<script>* tag. Methods and properties enable direct access to the objects defined in the tag tree (or DOM) created when the browser reads the code.

Chapter 2 will deal with control structures, such as loops and conditional expressions.

# Chapter 2

## Control structures

As any programming language, JavaScript allows you to control the flow of the code. In this chapter you will study how to control the process with conditional expressions and loops.

### 2.1 The conditions

You use conditions with the following expressions :

- if
- if...else
- if...elseif...else
- switch

#### 2.1.1 if

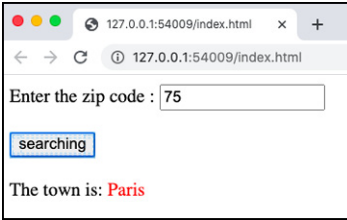
This instruction uses comparison operators. The syntax is as follows :

```
if ( condition)  
{actions;}
```

Figure 2.1 shows an example about the first two characters of a postcode.

- If the value is "75" then "Paris" is displayed.
- If the value is "69" then "Lyon" is displayed.
- If the value is "29" then "Brest" is displayed.





*Figure 2.1 : A test with a substring*

The code below shows the different tests with three *if* conditions.

```
<!doctype html>
<html>
<body>
<form name="mf">
Enter the zip code : <input type="text" id="b1">
<br> <br>
<input type="button" style="background-color: red" value="searching" onclick="test(mf)">
<br> <br>
The town is: <label id="label2" style="color:red"></label>
<br> <br>
</form>
</body>
<script>
function test(mf)
{
//I reset the result
document.getElementById("label2").innerHTML = "";
//I store the contents of the input field
vpostal=document.mf.b1.value;
//I extract the first 2 characters of the postal code
code=vpostal.substr(0,2);
// I test for 75
if(code=="75")
{
document.getElementById("label2").innerHTML = "Paris";
} // end of if
// I test for 69
```

```

if(code=="69")
{
document.getElementById("label2").innerHTML = "Lyon";
} // end of if
// I test for 29
if(code=="29")
{
document.getElementById("label2").innerHTML = "Brest";
} // end of if
} // end of function
</script>
</html>

```

The code identifies an area with the `<label>` tag.

```
<label id="label2" style="color:red"></label>
```

Note the comparison operator `==` used in the brackets of each if instruction.

```
if(code=="29")
```

### 2.1.2 if ... else

If a condition is true, you perform an action otherwise an other action.

The syntax is as follows :

```

if ( condition)
{action1;}
else
{action2;}

```

The following code shows the difference.

```

<!doctype html>
<html>
<body>
<form name="mf">
Enter the zip code : <input type="text" id="b1">
<br> <br>
<input type="button" style="background-color: red" value="searching" onclick="test(mf)">
<br> <br>

```

```
The town is: <label id="label2" style="color:red"></label>
<br> <br>
</form>
</body>
<script>
function test(mf)
{
//I reset the result
document.getElementById("label2").innerHTML = "
//I store the contents of the input field
vpostal=document.mf.b1.value;
//I extract the first 2 characters of the postal code
code=vpostal.substr(0,2);
// I test for 75
if(code=="75")
{
document.getElementById("label2").innerHTML = "Paris";
} // end of if
// otherwise
else
{
// I test for 69
if(code=="69")
{
document.getElementById("label2").innerHTML = "Lyon";
} // end of if
else // otherwise
{
// I test for 29
if(code=="29")
{
document.getElementById("label2").innerHTML = "Brest";
} // end of if
else
{
```

```

document.getElementById("label2").innerHTML = "unknown";
}
} // end of 2nd else
} // end of 1st else
} // end of function
</script>
</html>

```

### 2.1.3 if ... elseif ... else

In this case, the *if* follows the *else* for the new condition.

```

If ( condition)
{action1;}
else if (condition2)
{action2;}
else if (condition3)
{action3;}
else
{action4;}

```

The programming is slightly different.

```

<!doctype html>
<html>
<body>
<form name="mf">
Enter the zip code : <input type="text" id="b1">
<br><br>
<input type="button" style="background-color: red" value="searching" onclick="test(mf)">
<br> <br>
The town is: <label id="label2" style="color:red"></label>
<br><br>
</form>
</body>
<script>
function test(mf)
{
//I reset the result

```

```

document.getElementById("label2").innerHTML = "";
//I store the contents of the input field
vpostal=document.mf.b1.value;
//I extract the first 2 characters of the postal code
code=vpostal.substr(0,2);
// I test for 75
if(code=="75")
{
document.getElementById("label2").innerHTML = "Paris";
} // end of if
// otherwise if
else if(code=="69")
{
document.getElementById("label2").innerHTML = "Lyon";
} // end of elseif
else if(code=="29")
{
document.getElementById("label2").innerHTML = "Brest";
} // end of elseif
else
{
document.getElementById("label2").innerHTML = "unknown";
}
} // end of function
</script>
</html>

```

#### **2.1.4 switch**

This instruction allows you to control the progress of the code according to a variable value. It is a case-by-case management.

The syntax is as follows :

```

switch(variable)
{
case value1 :
action1;

```

```

break;
case value2 :
action2 :
break;
case value3 :
action3;
break;
default :
action4;
}

```

Each *case* instruction must end with a *break* command that tells JavaScript you exit the loop. The last *default* command is used to test all other cases, and the code is only performed if no case matches the value.

```

<!doctype html>
<html>
<body>
<form name="mf">
Enter the zip code : <input type="text" id="b1">
<br><br>
<input type="button" style="background-color: red" value="searching" onclick="test(mf)">
<br> <br>
The town is: <label id="label2" style="color:red"></label>
<br><br>
</form>
</body>
<script>
function test(mf)
{
//I reset the result
document.getElementById("label2").innerHTML = "";
//I store the contents of the input field
vpostal=document.mf.b1.value;
//I extract the first 2 characters of the postal code
code=vpostal.substr(0,2);

```

```

// I test
switch(code)
{
case "75" :
{
document.getElementById("label2").innerHTML = "Paris";
break;
}
case "69" :
{
document.getElementById("label2").innerHTML = "Lyon";
break;
}
case "29" :
{
document.getElementById("label2").innerHTML = "Brest";
break;
}
default :
{
document.getElementById("label2").innerHTML = "unknown";
}
} // end of switch
} // end of function
</script>
</html>

```

The switch instruction can be useful when you have many values to compare.

## 2.2 Comparison operators

JavaScript provides operators that allow you to compare values. The table below reminds them.

Operator	Description
==	To test an equality between two values.
===	To test an equality between two values, also taking the type of values (string, number, date) into account.

!= or <>	To test the difference between two values.
!==	To test the difference between different values or types.
<	To test if a value is strictly lower than another.
<=	To test if a value is less than or equal to another.
>	To test if a value is strictly higher than another.
>=	To test if a value is greater than or equal to another.
+=	To concatenate. The expression a+=6 is equivalent to a=a+6.

## 2.3 Loops

They allow some blocks of code to be executed several times, depending on a variable value. See below different forms of loops available in JavaScript :

- *while loop* (as long as a condition is true)
- *loop do ... while* (do as long as a condition is true)
- *for* (for a pointer from a value to another)
- *for ... of* (to browse according to an enumeration)

Loop instructions use a start value which is considered as a counter. This value is incremented and compared to an end value in a condition. If the condition is true, the loop continues. As soon as the condition is false, the loop stops.

Here are some examples of loops used to test the elements of a web page.

### 2.3.1 while

It means : as long as a condition is verified, the code is performed. The syntax is as follows :

```
while(condition)
{actions;}
```

Figure 2.2 shows a web page where information has to be entered in fields. When the *Test* button is pressed, the program performs a loop to test whether any field remains empty. In that case, a message is displayed and the empty area is shown in red.



127.0.0.1:54009/index.html

Confirm your personal information

First name

Name

Address

zip

City

Figure 2.2 : A test with several input fields

The following code shows how to test the fields using a *while* loop.

```
<html>
<style>
/*I set the styles for the labels*/
label{
position: absolute;
left:25px;
}
/*I define the styles for the input areas*/
input{
position: absolute;
left:90px;
width:200px;
}
/*I set the styles for the button*/
input[type=button]{
position:absolute;
left:10px;
font-style:italic;
width:60px;
}
/*I define the styles for the body*/
body{
font-family:sans-serif;
font-size:13px;
}
```

```
</style>
<body>
<h4>Confirm your personal information</h4>
<label>First name</label>
<input type="text"><br><br>
<label>Name</label>
<input type="text"><br><br>
<label>Address</label>
<input type="text"><br><br>
<label>zip</label>
<input type="text"><br><br>
<label>City</label>
<input type="text"><br><br>
<input type="button" value="Test" onclick="test()">
</body>
<script>
function test()
{
// I test with a loop that each field is well filled in.
// I initialise a variable
var p=0;
// I repeat 5 times in a while loop (from 0 to 4)
// as long as the variable p is less than 4
while(p<=4)
{
// I retrieve the contents of the 1st input zone. The index is 0.
var zone=document.querySelectorAll("input")[p].value;
// I test if the zone is empty.
if(zone=="")
{
//I put the concerned area in red
document.querySelectorAll("input")[p].style.backgroundColor="red";
alert("the area is empty");
}
//I increment the variable
```

```
p++;  
//end of loop  
}  
}  
</script>  
</html>
```

The program starts defining the styles for each group of elements. There are three groups : *label* for the field names, *input* for the text/button input and the *body*. CSS styles apply each group.

The instruction below shows how to perform the function as soon as the button is clicked.

```
<input type="button" value="Test" onclick="test()">
```

The instruction below retrieves the content entered in each input field, according to a pointer *p*, that takes successively 0, 1, 2, 3 and finally 4 as values.

```
var zone=document.querySelectorAll("input")[p].value;
```

The *querySelectorAll* method provides an indexed table in which the elements of the web page are located.

The expression *document.querySelectorAll("input")[0]* indicates the first input field.

The expression *document.querySelectorAll("input")[1]* indicates the second input field.

The expression *document.querySelectorAll("input")[2]* indicates the third input field.

Thanks to the parameter value, you retrieve the content of the field.

The *while* loop performs a test with a pointer from 0 to 4 that is the number of the input fields. The inside of the loop tests if the field content is empty. If so, the field is colored in red.

### 2.3.2 do...while

It is very similar to the *while* loop. The change is the order in which the action and the conditional test will take place. With *do...while*, the code

is performed first, then the output condition is tested. The loop is performed at least once.

The syntax is as follow :

```
do{  
actions;  
}  
while (condition);
```

The following lines show the previous script transformed with the *do...while* loop.

```
<script>  
function test()  
{  
var p=0;  
//do  
do  
{  
// I retrieve the contents of the 1st input zone. The index is 0.  
var zone=document.querySelectorAll("input")[p].value;  
// I test if the zone is empty.  
if(zone=="")  
{  
//I put the concerned area in red  
document.querySelectorAll("input")[p].style.backgroundColor="red";  
alert("the area is empty");  
}  
//I increment the variable  
p++;  
// as long as the condition is true  
}  
while(p<=4);  
}  
</script>
```

### 2.3.3 for

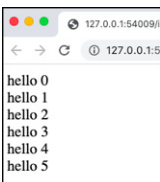
A *for* loop indicates the repetition of an action a certain number of times depending on the value of a pointer.

The *for* structure has three parameters : a start, a test and an increment that is performed at the end of each loop.

Example 1: The following code displays 5 times the word *hello* with the *for* loop pointer value.

```
<html>
<body>
<a id="p1"></a>
<script>
for(let p=0;p<=5;p++)
{
document.getElementById("p1").innerHTML+="hello "+p+"<br>";
}
</script>
</body>
</html>
```

Figure 2.3 shows the result.



*Figure 2.3 : A for loop in action*

The code line below changes a *p* pointer from 0 to 5. The third parameter increments *p* by 1.

```
for(let p=0;p<=5;p++)
```

The code line below adds the word *hello* followed by the *p* pointer value to the element already present in the body. The expression `<br>` indicates to go to the line.

The `+=` operator means the result is added to what was there before.

For example,  $D+=6$  is equivalent to  $D=D+6$ .

```
document.getElementById("p1").innerHTML+="hello "+p+"<br>";
```

Example 2 : The following code changes in red the color of all labels. We start by retrieving all the names of the objects, then we read each label in a table setting the new color.

```
<script>
function test()
{
// I retrieve in a variable all of the label objects
var elements = document.querySelectorAll("label");
// I loop through this set
for (var i = 0; i < elements.length; i++)
{
// I change the color
elements[i].style.color="red";
}
}
</script>
```

### 2.3.4 for ... of

It is a control structure designed to browse through an enumeration of objects, a series of objects without defining a particular order. The following code displays a word in the browser, then reads each letter, displaying them one under the other.

```
<html>
<body>
<script>
//I define a variable
var txt = "the beautiful car";
document.write(txt);
// I skip two lines
document.write("<br><br>");
var x;
// I repeat
for (x of txt)
```

```
{  
  // I take each letter and skip a line  
  document.write(x + "<br >");  
}  
</script>  
</body>  
</html>
```

Figure 2.4 shows the result.



*Figure 2.4: The loop for ... of in action*

Note : Inside a loop, you can use a *break* instruction to force an exit.

## **In brief**

Control structures allow you to evaluate the flow of the code according to conditional expressions. Variables are used to read the elements to be enumerated.

Chapter 3 deals with events that can be controlled in web pages.

# Chapter 3

## Events

This chapter focuses on events you can capture when you manipulate JavaScript code, in relation to HTML tags. An event is an action performed by the user, such as moving the cursor from an input field to another. Moving the mouse or loading a web page are also events.

### 3.1 Events caused by the Internet user

The table below shows the main events you can capture in the web pages.

#### 3.1.1 Keyboard events

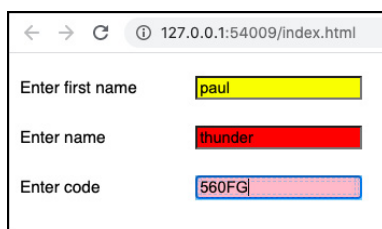
They are used when you want to control the keystrokes of the keyboard such as *Ctrl*, *Esc*, *Alt* or others.

Event	Description
onkeydown	When a key is pressed.
onkeypress	When a key is pressed then released.
onkeyup	When a key is released.

Let's see an example where the *onkeypress* event is triggered when pressing a key inside three input zones.

The triggered action colored the 1st input field in yellow, the 2nd field in red and the 3rd in pink.

Figure 3.1 shows the result.



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:54009/index.html". Below the address bar, there are three input fields with labels: "Enter first name", "Enter name", and "Enter code". The first input field contains the text "paul" and has a yellow background. The second input field contains the text "thunder" and has a red background. The third input field contains the text "560FG" and has a pink background.



*Figure 3.1: Triggering a keyboard event*

The code below shows the three functions. The call depends on the cursor position in the input fields. The *onkeypress* event is invoked. CSS styles are used to format fields and labels. Note the *<div>* tag that contains *common* properties to be applied by inheritance.

```
<!DOCTYPE html>
<html>
<style>
.common{
/*I define here the common properties which will be reflected by inheritance*/
font-size:14px;
/*family of fonts*/
font-family:sans-serif;
/* line spacing 3 */
line-height:3;
}
label{
position:absolute;
left:10px;
}
input[type=text]{
position:relative;
left:150px;
}
</style>
<script>
function color1()
{
document.getElementById("zone1").style.backgroundColor="yellow";
}
function color2()
{
document.getElementById("zone2").style.backgroundColor="red";
}
```

```

function color3()
{
document.getElementById("zone3").style.backgroundColor="pink";
}
</script>
<body>
<div class="common">
<label> Enter first name</label>
<input type="text" id="zone1" onkeypress="color1()"><br>
<label> Enter name</label>
<input type="text" id="zone2" onkeypress="color2()"><br>
<label> Enter code</label>
<input type="text" id="zone3" onkeypress="color3()"><br>
</div>
</body>
</html>

```

### Capturing a key with keypress

The example below shows how to capture any key on the keyboard to find out the unicode value of the key pressed. The *addEventListener* method is used and "listens" to the event.

```

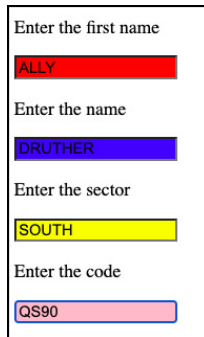
<html>
<script>
//I set up a passive earphone that listens to what happens on the keyboard.
window.addEventListener("keypress", capture);
function capture(e)
{
//I display the unicode of the pressed key
alert(e.charCode);
}
</script>
</html>

```

### Coloring when entering text

The example below "listens" if you type any text into each input field.

The code colors the field immediately. Figure 3.2 shows the result.



*Figure 3.2: The AddEventListener method listens to the events*

```
<html>
<body>
<p>Enter the first name</p> <input type="text" id="p1"><br>
<p>Enter the name <p> <input type="text" id="p2"><br>
<p>Enter the sector </p> <input type="text" id="p3"><br>
<p>Enter the code</p> <input type="text" id="p4"><br>
<script>
// I listen to the keyboard keystrokes for each input field
//for the first name
document.getElementById("p1").addEventListener("keypress",function color1()
{document.getElementById("p1").style.backgroundColor="red";})
// for the name
document.getElementById("p2").addEventListener("keypress",function color2()
{document.getElementById("p2").style.backgroundColor="blue";})
// for the sector
document.getElementById("p3").addEventListener("keypress",function color3()
{document.getElementById("p3").style.backgroundColor="yellow";})
// for the code
document.getElementById("p4").addEventListener("keypress",function color4()
{document.getElementById("p4").style.backgroundColor="pink";})
</script>
</body>
</html>
```

Ctrl, Alt, Shift and A with a keydown event

You can control whether you press *Ctrl*, *Alt* or *Shift* keys. We do not use the *KeyPress* event that is reserved for the alphabetical keyboard keys.

The following code shows an example of listening to these keys.

```
<html>
<body>
<script>
document.addEventListener("keydown",mykey,false);
//listen to the Ctrl key
function mykey(e)
{
// we test if the values and types are equal
// I test the Ctrl key
if (e.keyCode === 17)
{
alert ("the Ctrl key has been pressed");
};
// testing the Shift key
if (e.keyCode === 16)
{
alert ("the Shift key has been pressed");
};
// testing the A key
if (e.keyCode === 65)
{
alert ("the A key has been pressed");
};
}
</script>
</body>
</html>
```

Figure 3.3 shows the result.

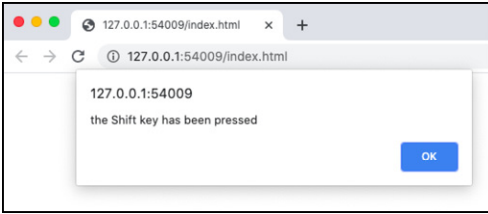


Figure 3.3 : Listening to keyboard keys

### Pressing several keys at the same time

You can control if several keys have been pressed, using properties of the event *e*.

- *e.ctrlkey* means the *Ctrl* key.
- *e.shiftkey* means the *Shift* key.
- *e.altkey* indicates the *Alt* key.
- *e.ctrlkey* && *e.shiftkey* indicates the *Ctrl* and *Shift* keys

The following code shows an example of capturing both the *Ctrl* and *Shift* keys.

```
<html>
<body>
<script>
// keys listening
document.addEventListener("keydown", mykeys, false);
function mkeys(e)
{
// I test the Ctrl and Shift keys
if (e.ctrlKey && e.shiftKey)
{
alert ("the Ctrl and Shift keys have been pressed");
}
}
</script>
</body>
</html>
```

### 3.1.2 Mouse events

The table below details the events you can capture manipulating the mouse.

Attribute	Description
onclick	If you click on the element.
ondblclick	If you double-click on the element.
ondrag	When an element is moved.
ondragend	At the end of a move/drag operation.
ondrag	When an element has been moved to a valid target.
ondragleave	When an item leaves a valid target.
ondragover	When an item is dragged to a valid target.
ondragstart	At the beginning of a drag/drag operation.
ondrop	When the dragged item is released.
onmousedown	When a mouse button is pressed.
onmousemove	When the mouse cursor moves.
onmouseout	When the mouse cursor moves out of an item.
onmouseover	When the mouse cursor hovers over an element.
onmouseup	When a mouse button is released.
onmousewheel	When the mouse wheel is turned.
onscroll	When using the scroll bars.

#### Displaying two images according to the left click

The following code displays a flower in the `<body>`, as soon as the web page loads. If you click on the flower with the left mouse button, the flower turns into a car. As soon as you release the button, the car turns back into a flower.

```
<!DOCTYPE html>
<html>
<head>
<!-- set the base directory for the images -->
<base src="/Applications/Brackets.app/Contents/samples/en/first steps">
<style>
/*I set the left position of the image*/
img{
left:50px;
```

```

}
</style>
<script>
function flower()
/*I assign a flower image to the img object in the page*/
{document.getElementById("image1").setAttribute("src", "flower.jpg");
}
function mycar()
/*I assign a car image to the img object in the page*/.
{document.getElementById("image1").setAttribute("src", "citroen.jpg");
}
</script>
</head>
<body>

</body>
</html>

```

### Mouse position on a map of Europe

The following example shows you how to retrieve the horizontal and vertical position of the mouse, on a map inserted in a web page. The *AddEventListener* event is performed.

Figure 3.4 shows this web page.

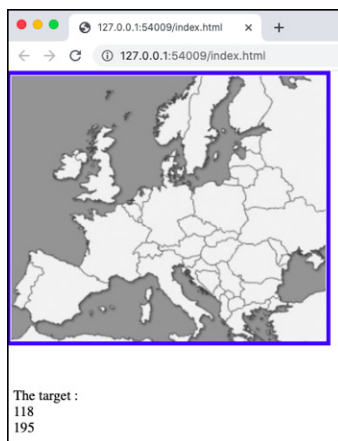


Figure 3.4: The AddEventListener Event Listener

The following code shows how JavaScript retrieves the mouse coordinates.

```
<!DOCTYPE html>
<html>
<head>
<base src="/Applications/MAMP/htdocs/europe.jpg">
</head>
<style>
/*image position*/
img{
position: absolute;
top:0px;
left:0px;
border: thick solid blue;
}
/*position of texts*/
a{
position: relative;
top:350px;
left:0px;
}
</style>
<body>
<a id="the target">The target : </a><br>
<a id="posx">The x position of the mouse : </a><br>
<a id="posy">The y position of the mouse : </a><br>
<!creating a container but you don't have to>
<div>

</div>
</body>
<script>
//on the div container, I capture the event of the left mouse pressing and perform the test1
function
// The 3rd parameter false indicates to let the event go up in the object hierarchy. It is said
```



that the event is bubbling

```
let c1=document.querySelector("div").addEventListener("mousedown", test1, false);
function test1(event) {
// I display the name of the target
document.getElementById("the target").innerHTML += event.target.tagName;
// I display the name of the target's container
document.getElementById("current_target").innerHTML += event.currentTarget.tagName;
// I display the horizontal position of the mouse pointer
document.getElementById("posx").innerHTML += event.clientX;
// I display the vertical position of the mouse pointer
document.getElementById("posy").innerHTML += event.clientY;
}
//On the image, I capture the mouse movement event
let i1=document.querySelector("img").addEventListener("mousemove", test2, false);
function test2(event) {
//I reset the values to zero
document.getElementById("posx").innerHTML="";
document.getElementById("posy").innerHTML="";
// I display the mouse coordinates
document.getElementById("posx").innerHTML += event.clientX;
// I display the vertical position of the mouse pointer
document.getElementById("posy").innerHTML += event.clientY;
}
</script>
</html>
```

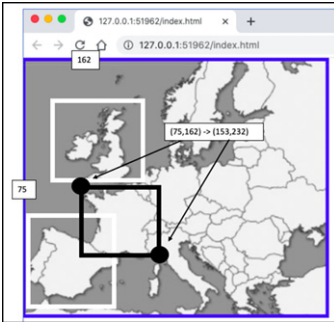
The *addEventListener* event allows you to listen to any event that occurs on the web page. It returns event object properties that can be reused. For instance, *event.clientX* returns the horizontal coordinate of the mouse pointer, *event.clientY* returns the vertical coordinate of the mouse pointer. The listener has been attached to the *div* container.

### An image cut into several parts

You can cut an image into several parts to display information when the user clicks on it. This is what happens with Google Maps or the Surveying Office.

The command that allows you to select parts of an image is *area*. The coordinates (X1 position, Y1 position, X2 position, Y2 position) define a rectangle identified by a character string in the *id* attribute.

Figure 3.5 shows the map cut into three parts with the coordinates of France.



*Figure 3.5 : An image cut into several parts*

The following code shows how to select multiple parts of an image.

```
<!DOCTYPE html>
<html>
<head>
<base src="/Applications/MAMP/htdocs/">
</head>
<style>
/*image position*/
img{
position: absolute;
top:0px;
left:0px;
border: thick solid blue;
}
</style>
<body>

<map name="card">
<area id="France" shape=RECT COORDS=75,162,153,232>
<area id="Spain" shape=RECT COORDS=8,204,101,280>
```

```

<area id="England" shape=RECT COORDS=47,60,139,145>
</map>
</body>
<script>
let en=document.getElementById("France").addEventListener("mousedown", displayFR, false);
let es=document.getElementById("Spain");
addEventListener("mousedown", displayES, false);
let uk=document.getElementById("England");
addEventListener("mousedown", displayUK, false);
// I trigger the opening of a new window with the following information
function displayFR() {
ref=window.open("http://www.remylent.fr", "books", "winSize");
// I resize the window
ref.resizeBy(500,500);
//I position the window
ref.moveTo(500,300);
}
function displayES(event) {alert("hola, que tal?");
}
function displaysUK(event) { alert("hi, how are you?");
}
</script>
</html>

```

## 3.2 Events in a form

These events occur on the HTML elements that design the form. The table below lists the main events you can control with JavaScript.

Attribute	Description
onblur	The code starts when the item loses the focus, i.e. when the pointer leaves the input field.
onchange	The code starts when an element is changed.
oncontextmenu	The code starts when a menu option is triggered.
onfocus	The code starts when the focus is placed in the field.
onformchange	The code starts when a form is modified.
onforminput	The code starts when you enter in the form.

oninput	The code starts when an element is entered in the form.
oninvalid	The code starts when an element is invalid.
onselect	The code starts when an element is selected.
onsubmit	The code starts when a form is submitted.

You can capture that kind of event via the *AddEventListener* method.

A form is often used to perform calculations, but most of the time, it is used to send information to a web server that will process your data.

### 3.2.1 About the form element structure

A form is defined by the *<form>* tag that contains items such as input fields, text input buttons, submitting button, headings, radio buttons, lists, etc. All the elements should be formatted with CSS styles.

A form object has properties :

- *name*. It is the name of the form.
- *action*. It is the URL (*Uniform Resource Locator*) address of the page the validated information will be sent to.
- *method*. It is the communication protocol used to send data when submitting the form. The values are either *post* or *get*. With *post*, the data sent is not visible, there is no URL sharing. The server determines the size of the data sent and no automatic back-navigation is allowed. With *get*, the data is visible in the URL destination, which can be copied and pasted. The size of the data is limited and you can go backwards.
- *target*. It is the page destination. With *\_blank* parameter, a new window is opened.
- *enctype*. It is the type of content encoding.

### 3.2.2 Tags that create elements

The list below shows the different elements you can manipulate with HTML tags.

- `<form>`. Defines the beginning of the form.
- `<label>`. Creates a label for an area and helps you to validate some controls, such as *checkboxes* or *radio buttons*.
- `<input>`. It is used for text input but also to define a radio button, a checkbox and other elements.
- `<select>`. Create a drop-down list.
- `<option>`. Select several options from a list using the *Ctrl/Cmd* key.
- `<textarea>`. Define a memo, i.e. a large multi-line text area.
- `<button>`. Create a push button.
- `<fieldset>`. Group elements for a form.
- `<datalist>`. Create a predefined list of options in an input field.
- `<output>`. Displays the result of a calculation.

### 3.2.3 The `<input>` tag

It is the most important tag for creating forms. It allows you to enter text, validate buttons, enter dates, drop-down data lists and more. It can be manipulated with styles just like any other tag. Styles can be embedded or grouped.

- `<input type="submit">` Displays a push button that validates the form. This validation performs a program defined in the `<form>` tag, in the *action* attribute (*action="program name"*). The program can be written in JavaScript, PHP or another language. It retrieves the data filled in the form.
- `<input type="button">`. This type defines a push button that can trigger a program defined in the *formaction* attribute.
- `<input type="text">`. This type allows you to enter text in an area whose characteristics are defined with styles. For instance, you can specify that an input field is mandatory with a specific length.

- `<Input type="checkbox">`. Displays a checkbox.
- `<Input type="radio">`. Displays a radio button.
- `<Input type="color">`. This type displays a color container to help you to choose.
- `<Input type="date">`. This type allows you to set a date in a calendar.
- `<Input type="email">`. This type checks that you have rightly entered an email address with the @ sign.
- `<Input type="file">`. This type allows you to select a file on your computer.
- `<Input type="password">`. This type allows you to enter data that will be displayed as asterisks.
- `<Input type="month">`. This type allows you to choose a month from a drop-down list.
- `<Input type="number">`. This type allows you to choose a number between a minimum and a maximum value, using a selector.
- `<Input type="range">`. This type displays a slider to choose a number (a circle you move on a horizontal bar).
- `<Input type="reset">`. This type allows you to reset the default value defined in the *value* attribute.

### 3.2.4 Workshop : a form to request information

Figure 3.5 shows a form in which you enter business information. When it is completed, you click on a button that displays (in a new pop-up window) all the options you have chosen.

In that workshop, you will :

- Place correctly the objects in the form using CSS styles and the `<form>` tag.
- Capture certain events in the form.

- Retrieve the objects of the form.
- Create a new window and place the desired information in it.
- Print and close the window.

*Figure 3.6: A request form*

The code below details this program.

```
<!DOCTYPE html>
<html>
<style>
/*the labels*/
form,select,ul,textarea,input,qt,label{
/*I defined for the form*/
font-family: sans-serif;
font-size: 11px;
}
/*the list of choices*/
select{
position:absolute;
top:90px;
width:240px;
}
```

```
/*the label for the quantity area*/
.qt{
position:absolute;
top:70px;
left:300px;
}
/*entry of the quantity*/
input[type=number]{
position:absolute;
top:90px;
left:300px;
width:40px;
}
/*the lists for the radio buttons*/
ul{
position:absolute;
top:160px;
width:150px;
padding:20px;
list-style-type:none;
border-style:solid;
outline-width:3px;
}
/*within the list*/
li{
color:red;
}
/*the checkbox*/
input[type=checkbox]{
position:absolute;
top:270px;
left:220px;
}
/*the text in the checkbox*/
.thetext_ofthecheck{
```



```
position:absolute;
top:272px;
left:240px;
}
/*the link*/
a{
position:absolute;
top:240px;
left:222px;
color:black;
}
/* the text for the GSC General Sales Conditions*/
.gsc{
position:absolute;
top:290px;
left:240px;
}
/*the text above the blue zone*/
.info{
position:absolute;
top:300px;
left:14px;
}
/* the comment area*/
textarea{
position:absolute;
top:330px;
left:10px;
height:100px;
width:200px;
background-color:azure
}
/*the form submission button*/
input[type=submit]{
position:absolute;
```

```

font-weight:600;
position:absolute;
top:450px;
left:10px;
}
/*the print button*/
.bt_print{
position:absolute;
top:450px;
left:100px;
}
/*the close button*/
.bt_close{
position:absolute;
top:450px;
left:170px;
}
</style>
<body>
<H5>Form to fill in</H5>
<label>Select the book to order</label>
<form name="myform" onsubmit="process(myform)"><br>
<select name="list" size=5 multiple="true" id="choice" >
<option>Google Gmail €20</option>
<option>Macros with Google Sheets 20€</option>
<option> Getting started with HTML 20€</option>
<option> Getting started with Javascript 20€</option>
</select>
<label class="qt">Enter the desired quantity</label>
<input type="number" value="1" name="vqt">
<ul>Delivery period
<li><input type="radio" name="period" value="1" id="p1" checked>
<label for="p1">9h-13h </label></li>
<li><input type="radio" name="period" value="2" id="p2">
<label for="p2">14h-18h </label></li>

```

```

</li><input type="radio" name="period" value="3" id="p3">
<label for="p3">19h-22h </label></li>
</ul>
<!-- the checkbox -->
<input type="checkbox" name="payment" value="1" checked id="p4">
<!-- the link -->
<label class="thetext_ofthecheck">I have read the Sales General Conditions</label>
<a class="SGC" href="http://www.remylent.fr" target="_blank"> See general terms and
sales conditions</a>
<p class="info">Other information</p>
<!-- the text area -->
<textarea name="notice" style=></textarea>
<!-- the validation button -->
<input type="submit" value="Validate">
<!-- a button to print the page -->
<input type="button" class="bt_print" value="Print" onclick="toprint()">
<!-- a button to close the page -->
<input type="button" class="bt_close" value="Close" onclick="close()">
</form>
</body>
<script>
function process(myform)
{
// I retrieve the number of the selected book
let selection = document.myform.list.selectedIndex;
// I retrieve the name of the selected book
let book = document.myform.list.options[selection].value;
// I retrieve the quantity
let qt = document.myform.vqt.value;
// I retrieve the delivery date
// I store the number of the radio button
let rb=document.myform.period.value;
// I test the button number
switch(rb)
{
case "1":

```

```

{
delivery="delivery 9h-13h" ;
break;
}
case "2":
{
delivery="delivery 14h-18h";
break;
}
case "3":
{
delivery="delivery 19h-22h";
break;
}
} // end of switch
// I retrieve the information from the text zone
let info=document.myform.notice.value;
//open a new window
var ref=window.open("",_blank", "width=600, height=400");
//I locate the window
ref.moveTo(500,200);
// I create the body of the document
ref.document.write("<body></body>");
// I write a title in the window
ref.document.write("<H1 align='center'>Order summary</H1>");
// I skip a line
ref.document.write("<br>");
ref.document.write("<h3>you have ordered: "+ book+"</h3>");
ref.document.write("<h3>Quantity: "+qt+"</h3>");
ref.document.write("<h3>Delivery period: "+delivery+"</h3>");
ref.document.write("<h3>Your information: "+info+"</h3>"); }
// The code to print
function print(){window.print();}
// The code to close
function close(){window.close();}

```

</script></html>

When the form is validated, a window is created displaying all the information previously entered (figure 3.7).

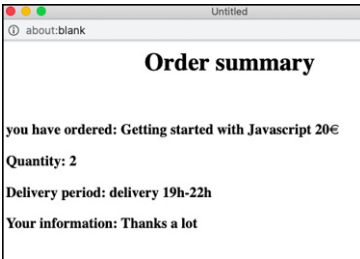


Figure 3.7 : Opening a new window

### 3.3 AddEventListener and bubbling

The *AddEventListener* handler allows you to listen and capture events that occur in the web page. You can manage input fields, images, titles, links, etc.

The syntax is quite simple :

`AddEventListener("event", function to be executed, false or true)`

- The first setting is the event you want to capture. It should be placed in inverted commas or with apostrophes. For instance, if you want to listen to a mouse click, you write the string "click". You could have "mousedown" "mousemove", "keydown" or others.
- The second parameter is the function that will process the captured event. The name is placed without quotes and without parenthesis.
- The last parameter *false* or *true* indicates the behaviour of the event. By default, an event starts from the top of the window, that is the top of the DOM tree. Then, the event propagates through the different elements to the target. If an *AddEventListener* is involved, i.e. if you try to capture a *click*, a *keystroke* or something else, the event will be captured as it travels down to the target. When it reaches the target, it will go

up to the different elements, back to the `<html>` tag. The event *bubbles*, like in a bottle.

- If the third parameter is set to *false*, the function will only run during the bubbling phase, from bottom to top. If the third parameter is set to *true*, the function will be run only during the capture phase (from top to bottom). The event handler is only active once in all cases.

You can prevent the event propagation with several methods :

- *stopPropagation()*. This method will stop the event propagation after the handler has been reached. If several *AddEventListeners* are attached to the same target, they will run in succession.
- *stopImmediatePropagation()*. The first handler will be executed.
- *preventDefault()*. It prevents the event behaviour. For instance, you can neutralise the validation of a form (*submit button*) if some fields are not correctly filled in.

### 3.3.1 Listening to a click on 3 images

Figure 3.8 shows a web page containing three different images. If you click on one of them, an informative text appears at the bottom of the page. The purpose of this program is to show how an event manager can be used to act on the different clicks.

The code below realises this kind of feature.

```
<html>
<head>
<base src="/Applications/MAMP/htdocs/">
</head>
<style>
.im1{
position:absolute;
width:8%;
```

```
height:170px;
left:30px;
}
.im2{
position:absolute;
width:8%;
left:180px;
height:170px;
}
.im3{
position:absolute;
width:8%;
left:325px;
height:170px;
}
.info{
font-family:sans-serif;
font-size:15px;
position:absolute;
top:200px;
width:300px;
left:20px;
height:100px;
background-color:aliceblue;
border-radius:80px;
padding:30px;
text-align:center;
}
</style>
<body>
<!--the 3 images-->



<!--the information part-->
```

```
<p class="info" id="inf"> click on the images</p>
</body>
<script>
//I start a click event manager on the images
// I select the first image and listen to the click
document.getElementById("image1").addEventListener("click", display1, false);
//I select the second image and listen to the click
document.getElementById("image2").addEventListener("click", display2, false);
//I select the third image and listen to the click
document.getElementById("image3").addEventListener("click", display3, false);
function display1(){
// I display information in the text box
var the_text1="Collection : Everyday computing<br>";
the_text1+="2020 - 130 pages - 148 x 210 mm<br>";
the_text1+="EAN: 978-2-490275-21-2<br>";
the_text1+="Price: 12.99 €<br>";
the_text1+="Public: Excel User";
// I update the information zone
document.getElementById("inf").innerHTML=the_text1;
}
function display2(){
// I update the information zone
document.getElementById("inf").innerHTML="the_text2";
// code here
}
function display3(){
// I update the information zone
document.getElementById("inf").innerHTML="the_text3";
// code here
}
</script>
</html>
```



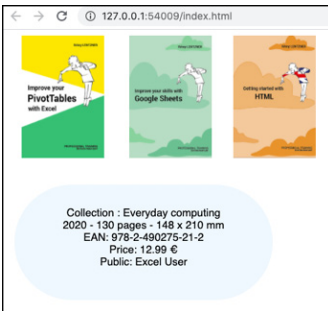


Figure 3.8 : Listening a click

### 3.3.2 Checking empty fields in a form

Any standard form gives the possibility to enter information and offers you to validate the content with a specific button. If all the data is correct, it is sent to a server for processing. If some information is missing, some JavaScript code can warn the Internet user. To create a form, you use a `<form>` tag with submission parameters that project the data to a remote server and more specifically to a program defined in the *action* attribute. You prevent a form from being validated by writing the expression *onsubmit="return false"* as long as the form validation button is a *submit* one.

Figure 3.9 shows a personal information entry form with a control for each field. The program tests whether the fields are correctly filled in.

Figure 3.9 : Controlling text input in a form

JavaScript sends the *Submit* event to the form, just before sending the data to the server. Therefore, any script (or function) placed in the

*onSubmit* attribute parameter is performed before the data is actually submitted.

The code below shows how to create this form.

```
<!doctype html>
<html>
<style>
body{
font-family: sans-serif;
font-size: 15px;
}
p{
position: relative;
top:40px;
left:10px;
}
/*the styles for the text input*/
input[type=text] {
position: absolute;
width : 300px;
left : 100px;
border-bottom-style:solid;
background-color:white;
color:black;
padding:10px;
}
input[type=submit] {
position: absolute;
left:10px;
top:400px;
}
input[type=button] {
position: absolute;
left:100px;
top:400px;
```

```
}
</style>
<script>
function test()
{
// I retrieve the inputs
var v0= document.querySelectorAll("input")[0].value;
var v1= document.querySelectorAll("input")[1].value;
var v2= document.querySelectorAll("input")[2].value;
var v3= document.querySelectorAll("input")[3].value;
var v4= document.querySelectorAll("input")[4].value;
// I test if all the fields are filled in
if (v0=="")
{alert("Please, Enter the first name");
document.querySelectorAll("input")[0].focus;
return false;
}
if (v1=="")
{alert("Please, Enter the last name");
document.querySelectorAll("input")[1].focus
return false;
}
if (v2=="")
{alert("Please, Enter the address");
document.querySelectorAll("input")[2].focus
return false;
}
if (v3=="")
{alert("Please, Enter the zip code");
document.querySelectorAll("input")[3].focus
return false;
}
if (v4=="")
{alert("Please, Enter the town");
document.querySelectorAll("input")[4].focus
```

```

return false;
}
}
</script>
<body>
<form name="myform" action="" method="post" onsubmit="return test()">
<h3>Indicate the following fields</h3>
<p>First name :</p> <input type="text" placeholder="type your first name here"
maxlength="30" autofocus value="Rémy"><br>
<p>Name :</p><input type="text" placeholder="type your last name here" maxlength=30
id="c2" value="Lentzner"><br>
<p>Address :</p><input type="text" placeholder="type your address here" maxlength="30"
id="c3" value=""><br>
<p>Code :</p><input type="text" placeholder="type your zip code here" maxlength="30"
id="c4" value=""><br>
<p>City :</p><input type="text" placeholder="type the town here" maxlength="30" id="c5"
value=""><br>
<input type="submit" value="Validate">
<input type="button" value="Close the windows" onclick="javascript:window.close()">
</form>
</body>
</html>

```

The expression `document.querySelectorAll("input")[0].value` allows you to access to the values entered in the input fields.

### 3.3.3 The boiling of events

In order to be able to control all events that occur in a web page, you need a mechanism to capture them and trigger actions. This event mechanism is called *bubble event*.

As soon as an event occurs, JavaScript locates the *target* immediately. From the programmer's point of view, the event starts from the top of the tree and spreads to the target through the various elements of the page.

If you listen to any event with an *AddEventListener* handler, you will be able to capture the event while it is descending towards the target.

When it reaches the target, the event will go up to the different elements towards the source of the DOM, the `<html>` tag. We say : the event bubbles.

### How can this bubbling help us ?

Imagine a web page showing some fruit names in a list. If you click on a fruit name, an image must appear immediately.

To program this appearance, you can use two ways :

- You use an *AddEventListener* handler for each fruit text and any click will trigger a function that displays the image. If you have 100 texts, you will have 100 functions and a lot of code redundancies.
- You use an *AddEventListener* handler only for a container (`<div>`) that contains the texts. When you click on a text, the event starts from the `<html>` tag to the *target* (the text) then bubbles to the container (the `<div>` tag) where only one function will display the image.

Let's see now these two different programming ways.

### An example without bubbling

Figure 3.10 shows the fruit texts in a list. The cherry link is selected then the image is displayed.

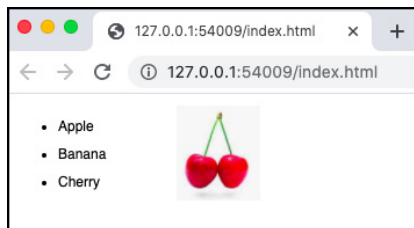


Figure 3.10: Displaying an image according to a click

The following code shows this programming.

```
<!doctype html>  
<html>
```

```
<head>
<base src="/Applications/MAMP/htdocs/">
</head>
<style>
li{
font-family:sans-serif;font-size: 12px;
line-height:2;
}
img{
position: absolute;
/*image top position*/
top:10px;
/*image left position*/
left:150px;
width:5%
}
</style>
<body>
<ul id="fruits">
<li>Apple</li>
<li>Banana</li>
<li>Cherry</li>
</ul>
<img id="im">
</body>
<script>
//I select the 1st text
var vApple = document.querySelectorAll("ul li")[0];
// I call the event manager
vPom.addEventListener("click",function(e) {
// I test the target. Caution to lowercase and uppercase node
if(e.target.nodeName == "LI") {
// I display the image
document.getElementById("im").src="apple.jpg"}
},false);
```

```

//I select the 2nd text
var vBan = document.querySelectorAll("ul li")[1];
// I call the event manager
vBan.addEventListener("click",function(e) {
// I test the target. Caution to lowercase and uppercase node
if(e.target.nodeName == "LI") {
// I display the image
document.getElementById("im").src="banana.jpg"}
},false);
//I select the 3rd text
var vCer = document.querySelectorAll("ul li")[2];
// I call the event manager
vCer.addEventListener("click",function(e) {
// I test the target. Caution to lowercase and uppercase node
if(e.target.nodeName == "LI") {
// I display the image
document.getElementById("im").src="cherry.jpg"}
},false);
</script>
</html>

```

Here the same group of codes is copied three times. That is redundancy. To avoid it, use bubbling technique.

### Bubbling avoids redundancy

If we group all the text elements in a *div* container, we will be able to control the different possible *click* events while bubbling.

```

<!doctype html>
<html>
<head>
<base src="/Applications/MAMP/htdocs/">
</head>
<style>
li{
font-family:sans-serif;font-size: 12px;
line-height:2;

```

```

}
/* I define the container properties*/
div{
width: 200px;
/* menu height*/
height: 100px;
/*I define a border*/
border: 1px solid gray;
/* blue color for the background*/
background-color:azure;
}
img{
position: absolute;
/*image top position*/
top:10px;
/*ipage left position*/
left:220px;
width:5%
}
</style>
<body>
<div id="container">
<ul id="fruits">
<li>Apple</li>
<li>Banana</li>
<li>Cherry</li>
</ul>
<img id="im">
</div>
</body>
<script>
//I select the container
var vcont = document.querySelectorAll("div")[0];
// I call an event manager
vcont.addEventListener("click",function(e){

```

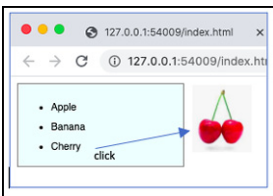


```

//I test if I click on one of the fruit name
if(e.target.nodeName == "LI")
{
// I test the apple
if(e.target.textContent=="Apple")
// I display the apple
{document.getElementById("im").src="Apple.jpg";}
// I test the banana
if(e.target.textContent=="Banana")
// I display the banana
{document.getElementById("im").src="Banana.jpg";}
// I test the cherry
if(e.target.textContent=="Cherry")
// I display the cherry
{document.getElementById("im").src="cherry.jpg";}
}
}
,false);
</script>
</html>

```

Figure 3.11 shows the result.



*Figure 3.11: The event stops at the div container*

## In brief

An event is an action performed by the user, such as moving the cursor from a field to another, moving the mouse or clicking on a list item. The *AddEventListener* handler is a passive event listener that allows you to control all parts of the page defined between the *<html>* and *</html>* tags.

Chapter 4 will show you the methods and the properties you may use

to properly manipulate text, numbers and dates.

# Chapter 4

## Text, number and date

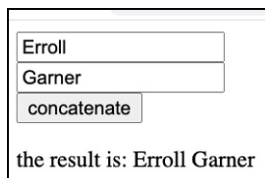
In this chapter, you will study the expressions that allow you to manipulate strings, numbers and dates.

### 4.1 Manipulating strings of characters

A string is managed with the *string* object that has numerous methods and properties. A string is wrapped by inverted commas or apostrophes that can be mixed if necessary, while respecting parity. There are special characters used to simulate non-visual characters. These characters are preceded by a backslash (\).

#### 4.1.1 Concatenating string

To concatenate strings, the *plus* sign (+) is used.



Erroll  
Garner  
concatenate  
the result is: Erroll Garner

Figure 4.1: Concatenation of strings of characters

```
<html>
<body>
<input type="text" id="c1" placeholder="Enter the 1st word"><br>
<input type="text" id="c2" placeholder="Enter the 2nd word"><br>
<input type="button" onclick="test1()" value="concatenate">
<p id="result">the result is: </p>
</body>
<script>
function test1(){
// I retrieve the value of the 1st input field
var r1=document.getElementById("c1").value;
```

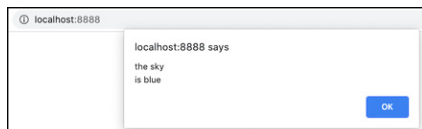
```
// I retrieve the value of the 2nd input field
var r2=document.getElementById("c2").value;
// I concatenate
var r3=r1+" "+r2;
// I select the text and add the following with the operator +=
document.querySelector("p").textContent+=r3;
}
</script>
</html>
```

### 4.1.2 Concatenating special characters

The code below concatenates strings of characters with the special `\n` that can be used to force a line break.

```
<html>
<script>
var a1="the sky";
var a2="is blue";
var a3=a1+"\n"+a2;
alert(a3);
</script>
</html>
```

Figure 4.2 shows the result.

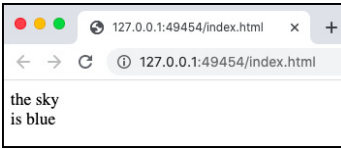


*Figure 4.2 : Special character*

Another way is possible.

```
<html>
<script>
var a1="the sky";
var a2="is blue";
var a3=a1+"<br>" +a2;
document.write(a3);
</script>
```

</html>



*Figure 4.3 : Line feed in the browser*

The table below shows special characters.

Special character	Description
\n	Line feed
\r	Press the Enter key
\t	Tabulation
\"	Double inverted comma
\'	Single quotes
\\	Anti-slash character

The following code shows an example of inserting an inverted comma in a string.

```
<html>
<style>
p{
position: absolute;
left:50px;
}
</style>
<script>
var a1="the sky";
var a2="is blue";
var a3=a1+"<br>" +a2;
document.write("<body>");
document.write("<p id=\"result\">");
document.write("</body>");
document.getElementById("result").innerHTML=a3;
</script>
</html>
```

Figure 4.4 shows the result.



Figure 4.4 : Inserting an inverted comma in a string

### 4.1.3 The length property

This property refers to the length of the chain. The syntax uses a *point* to attach the property to the variable that contains the string.

The following code asks a question to know a name, then the length of the string entered is displayed. The code uses the *prompt* command calling a dialog box that automatically proposes an *OK* button and a *Cancel* button.

```
<html>
<script>
var person = prompt("Enter your name", "Question");
if (person == null || person == "")
{alert("operation cancelled");}
else
{alert("the name has " + person.length + " characters")}
</script>
</html>
```

Figure 4.5 shows the result.

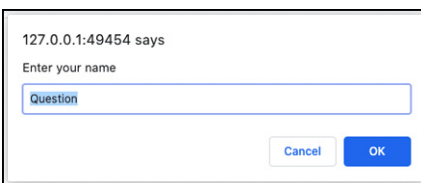


Figure 4.5 : A dialog box that attends an entry

The *prompt* method allows you to ask a question while waiting for a string. The *alert* method only displays a result.

### 4.1.4 The confirm dialog box

The *confirm* dialog box asks for confirmation with two buttons *Ok* or

*Cancel*. You can manage the answer. The following code asks the user to close his browser.

```
<html>
<script>
var r= confirm ("Do you want to close the browser?")
if(r==true)
{alert("I close");
window.close();
}
else
{alert("I continue");}
</script>
</html>
```

When you press the *OK* button, JavaScript sets the return of the *confirm* method to *true*. If you press the *Cancel* button, the method returns *false*.

#### **4.1.5 Text manipulation methods**

They look like other programming languages. The following list details them.

- *charAt(string, position)*. Returns the character at a given position.

```
<html>
<script>
var a = "JavaScript Studying";
alert(a.charAt(0)); // returns J
alert(a.charAt(11)); // returns S
</script>
</html>
```

- *indexOf(substring, position)*. Returns the position of a substring in a character string. The searching is from left to right.

```
<html>
```

```
<script>
var a = "I love JavaScript";
alert(a.indexOf("JavaScript",0)); // returns 7
</script>
</html>
```

- *lastIndexOf(substring, position)*. The method is similar to *indexOf()* but the searching apply from right to left.

```
<html>
<script>
var a = "At breakfast, JavaScript, I love";
alert(a.lastIndexOf("JavaScript", a.length)); // returns 15
</script>
</html>
```

- *substring(position1, position2)*. The method returns the substring between two positions.

```
<html>
<script>
var a = "JavaScript studying";
alert(a.substring(4,10)); // returns "Script"
</script>
</html>
```

- *substr(position1, length)*. Returns a substring starting at a position with a defined length.

```
<html>
<script>
var a = "I write with JavaScript";
alert(a.substr(8,4)); // returns "with"
</script>
</html>
```

- *toLowerCase()*. Converts all characters to lowercase.



```
<html>
<script>
var a = "JavaScript";
alert(a.toLowerCase()); // transforms to lowercase letters
</script>
</html>
```

- *toUpperCase()*. Converts all characters to uppercase.

```
<html>
<script>
var a = "JavaScript";
alert(a.toUpperCase()); // transforms to lowercase letters
</script>
</html>
```

- *concat(string1,string2, etc)*. Concatenates strings.

```
<html>
<script>
var a = "the sky";
var b = " is blue";
alert(a.concat(b)); // returns "the sky is blue"
</script>
</html>
```

- *startsWith()* and *endsWith()*. Determines if a string matches another.

```
<html>
<script>
var a = "the sky is blue";
alert(a.startsWith("the")); // returns true
alert(a.endsWith("blue")); // returns true
</script>
</html>
```

- *slice()*. Extracts a string of characters. The starting string is not modified. It is mandatory to provide the start position and optionally, as a second argument, the position where the extraction should stop. If negative values are given, the search starts from the end.

```
<html>
<script>
var a = "I like JavaScript for breakfast only";
alert(a.slice(7,17)); // returns "JavaScript".
</script>
</html>
```

- *replace()*. Search an expression in a character string then replaces it with another one.

```
<html>
<script>
var a = "I like JavaScript for breakfast only";
alert(a.replace("breakfast", "dinner")); // returns "I like JavaScript for dinner only";
</script>
</html>
```

- *trim()*. Deletes spaces in the string.

```
<html>
<script>
var a = " I like JavaScript ";
alert(a.trim()); // remove the first and the last spaces
</script>
</html>
```

## 4.2 Number manipulation

You have to pay attention to the numbers in JavaScript because nothing is logical in this area. Writing a number in a field does not mean that it will be considered as such. JavaScript is not very

rigorous because historically, it had to adapt itself to different browsers. Web regulators (W3C and ECMA) do their best to avoid errors.

#### 4.2.1 Accuracy and rounding

When manipulating numbers, you will have to handle decimals, rounding and other features. There are some methods that can help you.

- *toFixed()*. Sets precisely the number of decimals. The result is a string.

```
<html>
<script>
var num = 3.14159;
alert(num.toFixed(2)); // returns 3.14
</script>
</html>
```

- *toPrecision()*. Returns the number with a decimal precision, a rounding. The result is a string.

```
<html>
<script>
var num = 3.14159;
alert(num.toPrecision(4)); // returns 3.142
</script>
</html>
```

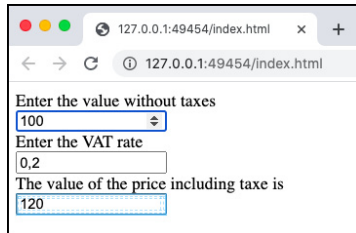
#### 4.2.2 The very important parseFloat method

*parseFloat* converts a string containing a number into a *number* type. You should use it because entering data in the input fields produces a string.

- *parseFloat* is used with numbers with commas.
- *parseInt* is used for integers (without commas).

Let's see how to manipulate numbers in a form.

Figure 4.6 shows three input fields. One field for a price excluding VAT, another for the value of the VAT rate (20%) and a last one for the price including all taxes.



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:49454/index.html'. The page content includes three input fields. The first field is labeled 'Enter the value without taxes' and has a value of '100'. The second field is labeled 'Enter the VAT rate' and has a value of '0,2'. The third field is labeled 'The value of the price including tax is' and has a value of '120'.

*Figure 4.6 : A calculation in a form*

The code below shows the programming of the events and the calculation.

```
<!DOCTYPE html>
<html>
<body onload="start()">
Enter the value without taxes<br>
<input type="number" id="M1" value="100"><br>
Enter the VAT rate<br>
<input type="number" id="Rate"><br>
The value of the price including tax is <br>
<input type="number" id="M2" onfocus="calc()"><br>
</body>
<script>
function start()
{
/* I fill in the VAT rate in the zone*/
document.getElementById("Rate").value=0.2;
}
function calc()
{
/* I put the cursor back in the M1 field*/
document.getElementById("M1").focus();
/*I retrieve the value of the M1 field that I transform into a numerical */
```

```

var vM1=parseFloat(document.getElementById("M1").value) ;
/* I retrieve the content of the rate that I transform into numerical*/
vRate=parseFloat(document.getElementById("Rate").value);
/* I calculate the VAT*/
var vM2=vM1*(vRate+1);
/* I place the value in the zone*/
document.getElementById("M2").value=vM2;
}
</script>
</html>

```

As soon as the page is opened, the VAT rate field is filled in with the value of the current rate (20% in France, at the time of writing this book). The *onload* event is used. When the pointer arrives in the VAT rate field, a calculation is performed. The *onfocus* is used and the formula is equal to  $value \times (1 + rate)$ .

Caution. Because the value from an input field is always a *string* type, you have to *parseFloat* it. Don't forget the letter F in capital letters. JavaScript is extremely case sensitive.

### 4.2.3 The Math objects

JavaScript offers other methods that fit for mathematical calculations.

- *Math.round()*. Produces the closest integer.

```

<html>
<script>
alert(Math.round(38.1)); // returns 38
alert(Math.round(38.6)); // returns 39
</script>
</html>

```

- *Math.ceil()*. Produces the upper integer.

```

<html>
<script>

```

```

alert(Math.ceil(38.1)); // returns 39
alert(Math.ceil(38.6)); // returns 39
</script>
</html>

```

- *math.floor()*. Indicates the lower integer.
- *Math.random()*. Generates a random number between 0 and 1.

The following code generates 10 random numbers without decimals.

```

<html>
<script>
for(i=1;i<=10;i++)
{document.writeln(Math.round(Math.random()*100))}
</script>
</html>

```

Figure 4.7 shows the result.

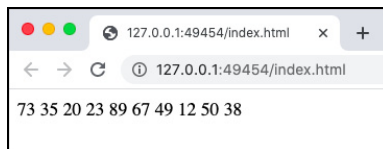


Figure 4.7: Generating random numbers

- *toString()*. Converts a number into a string.

```

<html>
<script>
a=3.14;
alert(typeof(a)); // returns type number
a=a.toString();
alert(typeof(a)); // returns the type function considered as string
</script>
</html>

```

## 4.3 Handling dates

Admittedly, the date management is a real headache in JavaScript.

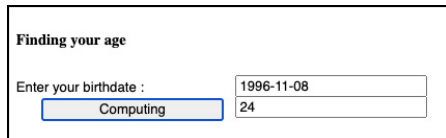
Because the language is likely to be used in different countries, with different time zones, manipulating days, months, years and performing calculations between them is not very simple.

In order to calculate dates from different countries and time zones without problems, you have to transform the dates into milliseconds. After the calculations, you change the milliseconds into days, months or years.

Several methods are available for this very special date management.

### 4.3.1 Calculating your age

Here is a simple example of date manipulation. You calculate the age of a person after entering its birthdate. Figure 4.8 shows this form.



<b>Finding your age</b>	
Enter your birthdate :	<input type="text" value="1996-11-08"/>
<input type="button" value="Computing"/>	<input type="text" value="24"/>

*Figure 4.8: Computing your age*

The following code shows how to calculate a difference between two dates, with a result in years. In this example, we use methods that avoid transforming dates into milliseconds.

```
<!doctype html>
<html>
<head>
<style>
input {
/*absolute position*/
position: absolute;
left: 200px;
/*font family*/
font-family: sans-serif;
/*character size*/
font-size:12px;
width:160px;
}
```

```

label {
/* left position*/
left: 4px;
font-family: sans-serif;
font-size:12px;
}
input[type=button] {
position: absolute;
top:80px;
left: 30px;
font-size:12px;
}
</style>
<script>
function age(MF)
{
/* I extract the year from today*/
var year_ofday=new Date().getFullYear()
/*I extract the input data from the form*/
var D1= document.MF.mydate.value;
/*I extract the 4th first chararters from my date*/
year_birth=D1.substr(0,4)
/* I compute the difference between the 2 values*/
age=year_ofday - year_birth-1
/* I place the age*/
document.MF.res.value=age;
}
</script>
</head>
<body>
<h5>Finding your age</h5>
<form name="MF">
<label>Enter your birthdate :</label>
<input type="text" name="mydate" value="1993-07-04"><br>
<input type="text" name="res"><br>

```



```
<input type="button" value="Computing" OnClick="age(MF)">
</form>
</body>
</html>
```

### 4.3.2 Methods that affect dates

The following list shows different commands that allow you to manipulate dates. You have to create a date variable with a *date()* constructor that allow you to work with methods and properties.

- *getDate*. This method retrieves the date from the current date. Respect the case, JavaScript is being obsessive about upper and lower case in commands.

```
<html>
<script>
var d = new Date();
alert(d.getDate()); // returns the day's number
</script>
</html>
```

- *getMonth()*. Returns the digit of the month (0 for January, 1 for February, 11 for December).

```
<html>
<script>
var d = new Date();
alert(d.getMonth()+1); //returns the number of the month with an offset of 1
</script>
</html>
```

- *getFullYear*. Returns the year in 4 digits.

```
<html>
<script>
var d = new Date();
alert(d.getFullYear());
```

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

- *getDay()*. Returns the day of the week as a number with 0 for Sunday, 1 for Monday and 6 for Saturday.

```
<html>
<script>
var d = new Date();
alert(d.getDay());
</script>
</html>
```

- *getHours()*. Returns the time in digits for the specified date according to the local time.

```
<html>
<script>
var d = new Date();
alert(d.getHours());
</script>
</html>
```

- *getMinutes()*. Returns the minutes for the specified date according to the local time.

```
<html>
<script>
var d = new Date();
alert(d.getMinutes());
</script>
</html>
```

- *Seconds()*. Returns the seconds in digits for the specified date according to the local time.

```
<html>
```

```
<script>
var d = new Date();
alert(d.getSeconds());
</script>
</html>
```

- *getMilliseconds()*. Returns the milliseconds in digits for the specified date based on local time.

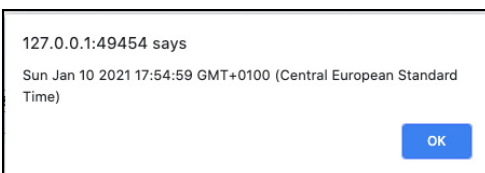
```
<html>
<script>
var d = new Date();
alert(d.getMilliseconds());
</script>
</html>
```

Note. The expressions that start with the word *get* are called *getters*. Those that start with the word *set* are called *setters*.

The code below - displays the system date.

```
<html>
<script>
var d = new Date();
alert(d);
</script>
</html>
```

Figure 4.9 shows the result of the current date.



*Figure 4.9 : The default current date.*

To retrieve a date with the format *dd/mm/yyyy*, you can use the following code :

```
<html>
<script>
```

```
var d = new Date();  
alert(d.getDay()+"/"+d.getMonth()+"/"+d.getFullYear());  
</script>  
</html>
```

If you want to see other calculations with dates, study again the code of the figure 1.2, at the beginning of this book.

### **In brief**

JavaScript offers many methods to manipulate strings, numbers and dates. The syntax is object-oriented using objects, methods and properties.

I hope this book has helped you to control your web pages better with JavaScript.

Do not hesitate to contact me at REMYLENT@GMAIL.COM if you have any comments or questions.

The next book will focus on PHP programming.

To be published

**Getting started with PHP**

Rémy LENTZNER



**Getting started**  
*with*  
**JAVASCRIPT**

PROFESSIONAL TRAINING  
ÉDITIONS REMYLENT