Université de Bordeaux
Licence Parcours International

Web design and data management

Event programming using Javascript : basics

Marie Beurton–Aimar, Florent Grélard

florent.grelard@u-bordeaux.fr

https://fgrelard.github.io/#teaching

Version : 26 avril 2023 (diapositives de présentation)

Aims

- Know the basics of the Javascript language for event handling (mouse or keyboard input) on a webpage
- Create dynamic webpages whose content can be modified in live.

Part 1: Web – Plan

Plan: Event programming

Event programming

Definition (Event)

An event corresponds to an action from a user (mouse click, keyboard input) onto an element (button, video, link...). It can also result from an internal task, e.g. file loading. An event changes the state of a webpage.

Definition (Event programming)

Event programming is a coding model that reacts when an event is triggered.

Examples

- Image carousel : images that automatically scroll
- Mask a section from a webpage on click
- Form interaction

Function

The code is structured into functions.

A function is set of coding instructions, such as variable definitions, selection of HTML elements...

Function

The code is structured into functions.

A function is set of coding instructions, such as variable definitions, selection of HTML elements...

Event programming relies on function execution when an event is raised

We say a function is is registered to an element for an event.

Function

The code is structured into functions.

A function is set of coding instructions, such as variable definitions, selection of HTML elements...

Event programming relies on function execution when an event is raised

We say a function is is registered to an element for an event.

.. which is equivalent to :
An element is listened to by an event listener that triggers a function

Languages

Overview of languages for Web programming :

- For content : **HTML**.
- For styling : **CSS**.
- For event programming : **Javascript**, **PHP**.
- For communication with databases and secured processing : **PHP**.

Plan: Javascript

Overview

Javascript is an interpreted language : lines of codes are executed one at a time.

- Javascript code can be inserted directly in the HTML file or in a separate file (`.js` extension), then included in the HTML
- browsers all support Javascript

Including a Javascript file in a HTML file (at the bottom of the `body`) :

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

Plan: Javascript

Code debugging

- Syntax errors can prevent code execution.
  $\Rightarrow$ need for error inspection and debugging.
  - Browser console (shortcut in Firefox $\rightarrow$ F12)
  - To display a message or a variable in the console (`print` equivalent in Python) :

```
console.log("Message" + variable);
```

- Permissive language $\Rightarrow$ it is strongly advised to add `"use strict";` at the beginning of the code to facilitate debugging.

Syntax

- Generalities
- Variables, data types
- Functions
- Arrays

Generalities

- Lines of codes end with a semi-column “;”
- Functions and conditional statements are surrounded with curly brackets.

Variables

Variables are used to **store information**. They can be used throughout the program.

Variable declarations (only once) are done by prefixing the variable name with `var`.

Example : `var one = 1;`

```
1       var number = 18;
2       number = 2;
```

Conditions

Condition checking : if–else.

Modify how code is executed.

Conditions

Condition checking : if–else.

Modify how code is executed.

```
1    if (condition) {
2     // if condition true
3    }
4    else {
5     // if condition false
6    }
```

Conditions

Condition checking : if–else.

Modify how code is executed.

```
1    if (condition) {
2     // if condition true
3    }
4    else {
5     // if condition false
6    }
```

```
1    var raining = True;
2    if (raining) {
3     console.log("We␣stay␣indoors.");
4    }
5    else {
6     console.log("We␣go␣outside.");
7    }
```

Equality and inequality operators

Two equality operators :

- == : checks for equality by forcing type conversion.
  Ex : '1' == 1 yields true

- === : checks for strict equality (variables with same type and same value).
  Ex : '1' === 1 yields false, because it is a character versus a number.

Good practices

Prefer the === operator.

Inequality operators (less than or greater than) : <, <=, >, >=.

Conditions : combination

Combination of conditions :

- && (and) : both conditions need to be true
- || (or) : one condition needs to be true
- ! (not) : invert the conditions (a true condition becomes false)

Conditions : combination

Combination of conditions :

- && (and) : both conditions need to be true
- || (or) : one condition needs to be true
- ! (not) : invert the conditions (a true condition becomes false)

```
1   var weather = "Sunny";
2   var work = False;
3   if (weather === "Sunny" && work) {
4    console.log("It's␣sunny␣and␣I'm␣working.");
5   }
6   else if (weather !== "Sunny" && work) {
7    console.log("Raining␣and␣working.");
8   }
9   else if (weather === "Sunny" && !work) {
10   console.log("It's␣sunny␣and␣I'm␣free.");
11  }
```

Conditions : nesting

Possible to nest multiple if–else statements.

```
1    var weather = "Sunny";
2    var work = False;
3    if (weather === "Sunny") {
4     if (work) {
5      console.log("It's␣sunny␣and␣I'm␣working.");
6     }
7     else {
8      console.log("It's␣sunny␣and␣I'm␣free.");
9     }
10   }
11   else {
12    if (work) {
13     console.log("Raining␣and␣working.");
14    }
15    else {
16     console.log("It's␣raining,␣and␣I'm␣free.");
17    }
18   }
```

Loops

Loops : to repeat code instructions.

- `for` loop :

```
1     for (var i = 0; i < 10; i++) {
2      //Code
3     }
```

- `while` loop :

```
1     var i = 0;
2     while (i < 10) {
3      //Code
4      i++;
5     }
```

Functions

Functions are small units of code that can be **called and reused several times**. This allows to decompose a problem into smaller chunks, each of which performs a **particular task**.

Functions

Functions are small units of code that can be **called and reused several times**. This allows to decompose a problem into smaller chunks, each of which performs a **particular task**.

A function needs to be defined. The function is not executed until explicitly called.

Functions

Functions are small units of code that can be **called and reused several times**. This allows to decompose a problem into smaller chunks, each of which performs a **particular task**.

A function needs to be defined. The function is not executed until explicitly called.

A function usually requires some inputs, or parameters.

Functions

Functions are small units of code that can be **called and reused several times**. This allows to decompose a problem into smaller chunks, each of which performs a **particular task**.

A function needs to be defined. The function is not executed until explicitly called.

A function usually requires some inputs, or parameters.

Functions can return values which can be captured as variables and used in the code.

Function definition and call

Function definition, usually done in the top of the `.js` file :

```
1    function function_name(parameter1, parameter2) {
2     var sum = parameter1 + parameter2;
3     return sum;
4    }
```

Call to this function, further down the `.js` file :

```
1    var added_values = function_name(2,3);
2    console.log(added_values); //Displays 5 in console.
```

Arrays

Arrays are data structures consisting in a collection of elements (values or variables), each identified by an index. The indices range from 0 to n-1.

1. Declaration and initialization :

```
var tab = [];
```

2. Setting a value in the first cell :

```
1  tab[0] = value;
2  /* OR instead of steps 1 and 2: */
3  var tab = [value, value2, value3];
```

3. Iterating :

```
1  for (var i = 0; i < tab.length; i++) {
2   //Code
3  }
4  /* OR */
5  for (var value in tab) {}
```

Plan: Javascript

Javascript for event programming

Javascript can :

- act on HTML elements (tag, id, class) and their properties
- modify the HTML tree

$\Rightarrow$ Dynamic web pages whose structure changes upon receiving events.

window and document

Two variables are defined by default in Javascript :

- `window` : browser window in which the HTML document is loaded
- `document` : encloses the HTML tree

Function example :

- `window.alert()` : opens a dialog (popup) in the webpage
- `document.write()` : writes text inside the HTML document

Plan: Javascript

Element selection

Before interacting with elements, it is necessary to get them as Javascript variables.
HTML element selection can be done by :

- `tag`
- `id`
- `class`
- any combination of CSS selectors.

Selection by identifier

The `getElementById("myid")` function from the `document` variable selects the unique element whose id is given as a parameter (here : "myid").

Example : updating an image with Javascript :

html file

```
...
<img src="image.
    jpg" id="
    myImage" />
...
```

Javascript file

```
1   var imageJS = document.
    getElementById("myImage");
2   imageJS.src="newimage.jpg"
```

Selection of several elements

Different methods :

- From a tag : `getElementsByTagName()`
- From a class : `getElementsByClassName()`
- From a CSS selector : `querySelectorAll()`.

$\Rightarrow$ return an element array

Example :

Fichier html

```
...
<h1 class="left">
<div class="left"
    >
...
```

Fichier javascript

```
1   var elements = document.
    getElementsByClassName("left")
    ;
2   console.log(elements[0]);
```

Element manipulation

- Javascript variables that are obtained from HTML have attributes : they are the same as the HTML attributes.
- These attributes can be read and modified.

Example :

```
1    var imageJS = document.getElementById("myImage");
2    imageJS.alt = "Description␣text";
3    console.log(imageJS.alt);
4    imageJS.src = "newImage.png";
5    imageJS.className = imageJS.className + "suffixClass";
```

Changing the content of a tag

- The `innerHTML` attribute corresponds to the HTML content of an element :
  - access : contains tags
  - modification : tags are considered as HTML
- The `textContent` attribute corresponds to the text inside a tag :
  - access : does not contain tags
  - modification : tags are considered as text

Differences between innerHTML and textContent

```
<div id="example">
<p> This is <span> my content </span>
</p>
</div>
```

innerHTML :

```
var element = document.
getElementById("example");
var htmlText = element.
innerHTML;
console.log(htmlText); //
text contains <p> and <span
> tags
element.innerHTML = "<p>
My brand new content </p>";
 //tags are interpreted
```

textContent :

```
var element = document.
getElementById("example");
var htmlText = element.
textContent;
console.log(htmlText); //
text does not contain <p>
and <span> tags
element.textContent = "<p>
 My brand new content </p>"
; //tags are considered as
text
```

Changing the HTML tree structure

Possible to create new HTML elements with `document.createElement()`

To modify tree structure :

- add : `appendChild()`, `insertBefore()`
- remove : `removeChild()`
- replace : `replaceChild()`

Event capture

Aim : link a function when an event is triggered on an element.

The link between an event and an element is done by an event listener.

Examples :

Element (HTML) $\rightarrow$	**Event (JS)** $\rightarrow$	**Event listener (JS)** $\rightarrow$
Button	Click	Send form data
Image	Click	Image update

Events

There are various types of events :

- mouse and keyboard actions : `click`, `keyup`, `mouseover` ...
- state update : `change`, `focus`...
- begin or end after a large element is loaded on the page : `load`.

Event listener (1/2)

The `addEventListener()` performs the registration of a function to an event for an element (possible to have multiple listeners on an element).

Function usage :

```
object.addEventListener(eventType, triggeredFunction);
```

- `object` : targetted element (ex : `document` or object obtained from `getElementById()`).
- `eventType` : description of the event (click, keyup...).
- `triggeredFunction` : triggered function (e.g. : update of an image or the content of a tag)

Event listener (2/2)

Alternative to `addEventListener()` to define function registration (only one per element) :

```
object.onevent = triggeredFunction;
```

Examples :

```
1  document.onkeyup = triggeredFunction;
2  //Similar to
3  document.addEventListener("keyup", triggeredFunction);
```

Event listening : example (1/2)

HTML :

```
...
<p id="myParagraph"> If you click on this button, you
    will never see me again :(
</p>

<button id="myButton" type="submit"> Send </button>
...
```

Javascript :

```
1  function changeMyParagraph() {
2   var textJS = document.getElementById("myParagraph");
3   textJS.textContent = "My brand new text";
4  }
5
6  var buttonJS = document.getElementById("myButton");
7  button.addEventListener("click", changeMyParagraph);
8  //Equivalent to : button.onclick = changeMyParagraph;
```

Event listening : example (2/2)

At first :

If you click on this button, you will never see me again :(

Send

After a click on the button :

My brand new text

Send

event object

- An `event` object is created at each interaction
- This object has various attributes :
  - `type` : event type (click, ...)
  - `key` : information about the pressed key
  - `target` : target HTML element
- The event object is the first parameter of the registered function.

Object event : example

```
1   function changeColor(event) {
2    var text = document.getElementById("myParagraph");
3    if (event.key === "r") {
4     texte.style.color = "red";
5    }
6    else if (event.key === "g") {
7     texte.style.color = "green";
8    }
9    else if (event.key === "b") {
10    texte.style.color = "blue";
11   }
12  }
13
14  function setupListeners() {
15   document.addEventListener("keyup", changeColor);
16  }
17
18  window.addEventListener("load", setupListeners);
```

Object event : example (2/2)

Result :

Try "R", "G", and "B" keys!

Object event : example (2/2)

Result :

Try "R", "G", and "B" keys!

Object event : example (2/2)

Result :

Try "R", "G", and "B" keys!

Object event : example (2/2)

Result :

Try "R", "G", and "B" keys!

Registration of several functions

For one element, it is possible to have :

- several listeners for the various events
- several listeners for the same event

Good practices :

1. Define a `setupListeners` function in charge of setting up all the listeners
   ⇒ facilitates debugging and code maintenance
2. Call the `setupListeners` function when the HTML page is fully loaded
   ⇒ `window.addEventListener("load", setupListeners)`

Sources et références

Sources et références

Références :

- Cours de Jean-Christophe Routier (Université Lille 1) : `http://www.fil.univ-lille1.fr/~routier/enseignement/licence/tw1/spoc/#chap8`