Practical work Javascript CSS

**Evaluated skills :**

*→ Javascript CSS lesson*

**Mini Practical Work :**

Let's face it: the exercises we did previously were still not very useful without real interaction with the user. The alert (), confirm () and prompt () is nice for a moment, but we quickly went around! It is therefore time to move on to something more interesting: a system of drag & drop! Finally ... a very simple version!

This is a mini-TP, which means it is not very long to achieve, but it still asks a little thought. This lab will make you use CSS events and manipulations.

**Presentation of the exercise**

First, what is drag & drop? It is a system allowing the displacement of elements by a simple displacement of mice. To put it simply, it's like when you have a file in one folder and move it to another folder by dragging it with your mouse.

And am I really able to do that?

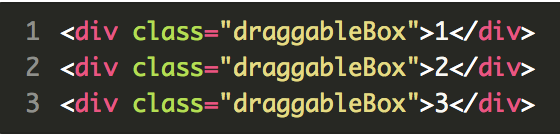
Of course ! Well, you must have followed the course carefully and struggling a little, but it is perfectly possible, you are able!

Before embarking on the code, let's list the operating steps of a drag & drop system:

* The user presses (and does not release) the left mouse button on an item. The drag & drop is initialized then knowing that it will have to manage the displacement of this element. For information, the event to use here is mousedown.
* The user, while leaving the button of his mouse depressed, begins to move his cursor, the targeted element then follows his movements to the trace. The event to use is mousemove and we advise you to apply it to the document element, we will explain why in the correction.
* The user releases the button of his mouse. The drag & drop then ends and the element no longer follows the mouse cursor. The event used is mouseup.

So ? It does not look so crooked, is it?

Now that you know pretty much what to do, we'll provide you with the basic HTML and CSS so you do not have to worry about doing it yourself:





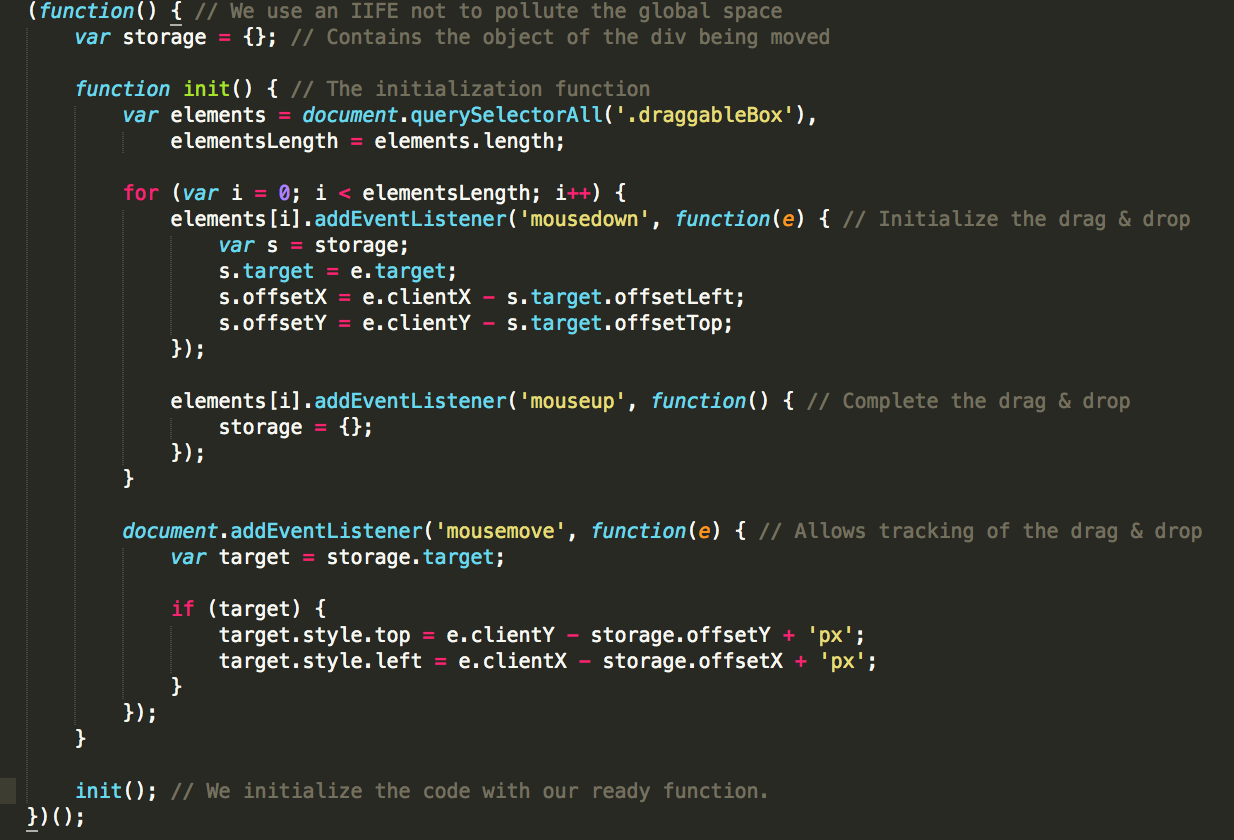
Just two last little things. It would be good :

* Whether you use an IIFE in which you will place all the functions and variables necessary for the proper functioning of your code, it will be much cleaner. Thus, your script will not pollute the global space with its own variables and functions;
* That your code does not apply to all existing <div> but only to those who own the .draggableBox class.

With that, good luck!

**Correction :**

You have completed the exercise? We hope you did it, but if it does not, it does not matter! Look carefully at the correction, and everything should be clearer :

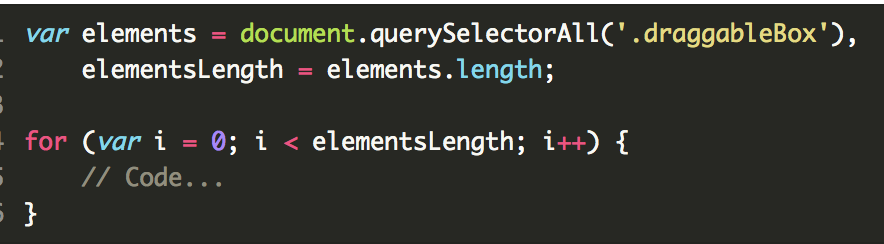


Regarding the storage variable, this is only a storage space which we will explain how it works during the study of the init () function.

Let's start!

**Exploring the HTML code**

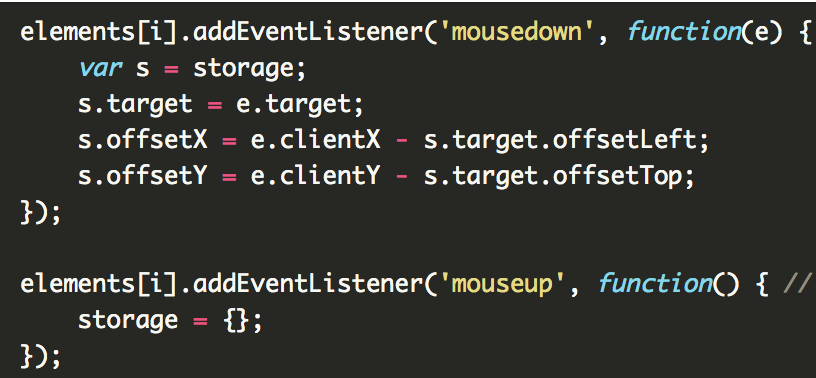
Our init () function starts with the following code:



In this code, we have deliberately hidden the event addition codes, because what interests us is this loop. This loop coupled to the querySelectorAll () method - you've already seen it in the HTML manipulation chapter - allows you to browse all the HTML elements filtered by a CSS selector. In our case, we browse all the elements with the class .draggableBox.

**The addition of mousedown and mouseup events**

In our loop that goes through HTML, we have two additions of events:



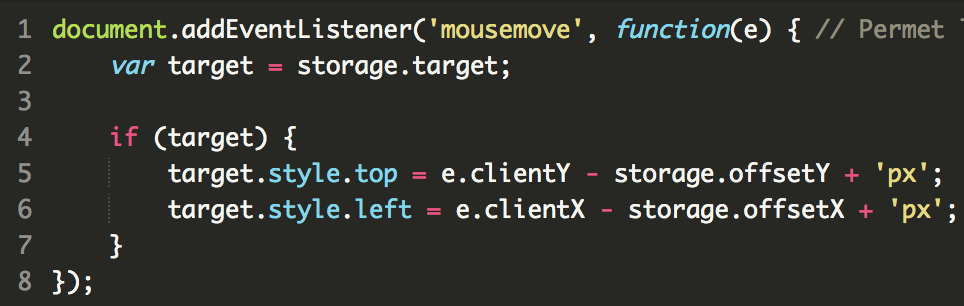
As you can see, these two events just access the storage variable. What is this variable for? It is simply an object that serves us as storage space, it allows to memorize the element currently being moved as well as the position of the cursor relative to our element (we will return to this last point more later).

In short, in our mousedown event (which initializes drag & drop), we add the targeted event in the storage.target property and then the cursor positions relative to our item in storage.offsetX and storage.offsetY.

Regarding our mouseup event (which ends the drag & drop), we just assign an empty object to our storage variable, so that everything is emptied!

**The management of the displacement of our element**

Until now, our code only stores the target element for our drag & drop in the storage variable. However, our goal is to move this element. That's why our mousemove event is coming!



Why is our event applied to the document element?

Let's think! If we apply this event to the targeted item, what will happen? As soon as we move the mouse, the event will trigger and everything will happen as we wish, but if I start moving the mouse too quickly, the cursor will then go out of our element before it has had time to move, so the event will not fire until we move our cursor back to the item. The probability that this will happen is higher than we think, as much as necessary.

Another problem can arise: in our current code, we do not manage the CSS style z-index, so when we move the first element and we place our cursor on one of the other two elements, the first element is then below them. How is this a problem? Well if we applied the mousemove on our element instead of the document then this event will not be triggered as we move our cursor on one of the other two elements and not on our element being moved.

The solution is to put the event mousemove on our document. Since this event is spreading to children, we are sure it will fire whenever the cursor is moved on the page.

**The rest of the code is not complicated:**

We use a condition that verifies that there is a target index in our storage space. If there is none, there is no drag & drop running.

We assign to our target element its new coordinates relative to the cursor.

So let's go back to an important point of the previous code: we had to record the position of the cursor in relation to the upper left corner of our element as soon as the drag & drop is initialized.

Why ? Because if you do not, each time you move your item, it will place its upper left edge under your cursor and this is clearly not what you want.

So try for yourself to check!

**Prevent selection of the contents of movable elements**

As you may have noticed, it is possible that the user selects the text contained in your moveable elements, this is a little random. Fortunately, it is possible to simply solve this problem with some CSS properties applied to movable elements:

