**Reference tables**

### Quick summary of event management in JavaScript

#### HTML5 EVENTS

There is no input or output in JavaScript. We treat events caused by user actions as inputs, and we manipulate the DOM structure as output. Usually in a JavaScript application, we will get info such as the key strokes, the mouse button clicks and the mouse position, and we will refer to these variables when determining what action to perform.

In any case, the events are called DOM events, and we use the DOM APIs to create event handlers.

#### HOW TO LISTEN TO EVENTS

There are three ways to manage events in the DOM structure. You could attach an event inline in your HTML code like this:

##### **Method 1: declare an event handler in the HTML code**

1. <div id="someDiv" **onclick**="alert('clicked!')"> content of the div </div>

This method is very easy to use, but it is not the recommended way to handle events. Indeed, although it currently works, it is deprecated (will probably be abandoned in the future). Mixing 'visual layer' (HTML) and 'logic layer' (JavaScript) in one place is really bad practice and causes a host of problems during development.

##### **Method 2: attach an event handler to an HTML element in JavaScript**

1. document.getElementById('someDiv').**onclick**= function() {
2. alert('clicked!');
3. }

This method is fine, but  you will not be able to attach multiple listener functions. If you need to do this, use the version shown below.

##### **Method 3: register a callback to the event listener with the addEventListener method (preferred  method)**

1. document.getElementById('someDiv').**addEventListener**('click', function() {
2. alert('clicked!');
3. }, false);

Note that the third parameter describes whether the callback has to be called during the captured phase. This is not important for now, just set it to false or ignore it (you can even pass only two parameters to the addEventListener function call and do not set this boolean parameter at all).

#### Details of the DOM event are passed to the event listener function

When you create an event listener and attach it to an element, the listener will create an event object to describe what happened. This object is provided as a parameter of the callback function:

1. element.addEventListener('click', function(**event**) {
2. **// now you can use event object inside the callback**
3. }, false);

Depending on the type of event you are listening to, you will consult different properties from the event object in order to obtain useful information such as: "which keys are pressed down?", "what is the location of the mouse cursor?", "which mouse button has been clicked?", etc.

In the following lessons, we will remind you how to deal with the keyboard and the mouse.

### Further reading

In Method 1 (above), we mentioned that "mixing 'visual layer' (HTML) and 'logic layer' (JavaScript) ... is bad practice", and this is similarly reflected in many style features being deprecated in HTML5 and moved into CSS3. The management philosophy at play here is called "the separation of concerns" and applies in several ways to software development - at the code level, through to the management of staff. It's not part of the course, but professionals may find the following references useful:

* [Separation of concerns - Wikipedia, the free encyclopedia](https://en.wikipedia.org/wiki/Separation_of_concerns)
* [Chapter 5. Separation of Concerns](http://chimera.labs.oreilly.com/books/1234000000262/ch05.html) from Programming JavaScript Applications, by Eric Elliott, O'Reilly, 2013.
* [The Art of Separation of Concerns](http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/) by derekgreer, January 3, 2008

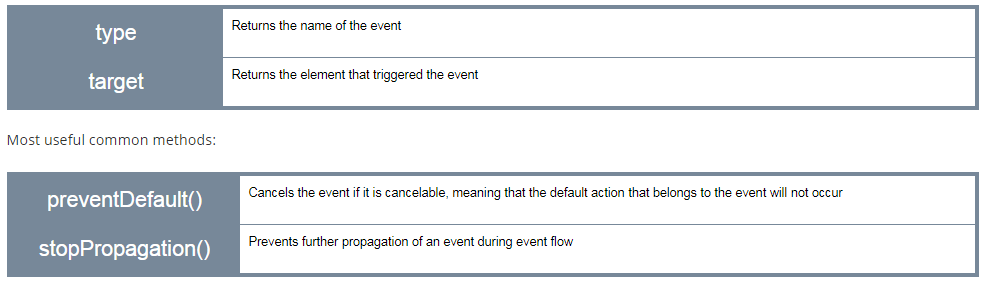
### Reference tables for events and properties/methods

These tables are provided as a reference. They are a compilation of the most common event types sorted by domain (key, mouse, forms, etc.). For each domain you will see the most useful event types and their properties.

In the following sections, we will show  examples that use most of the events displayed in these tables.

#### Event object

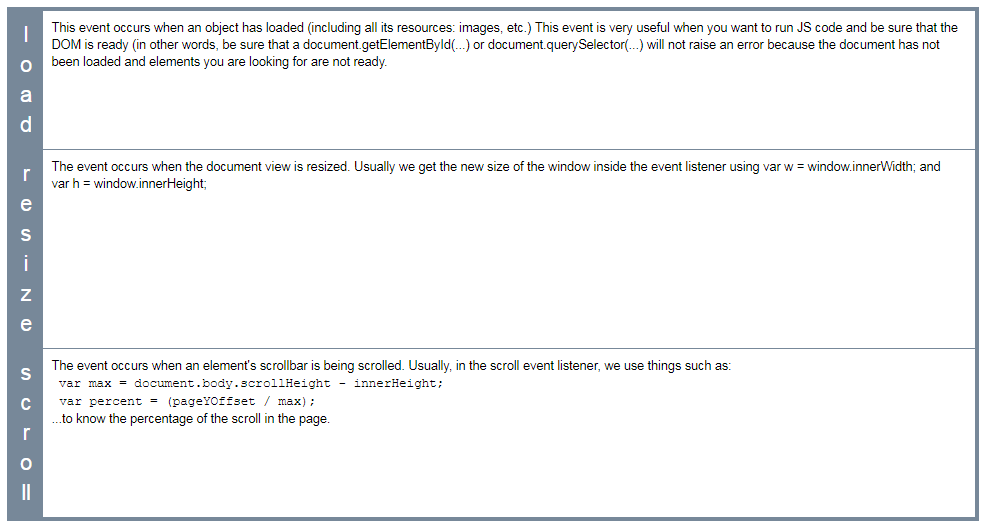
Most useful common properties:



#### Page

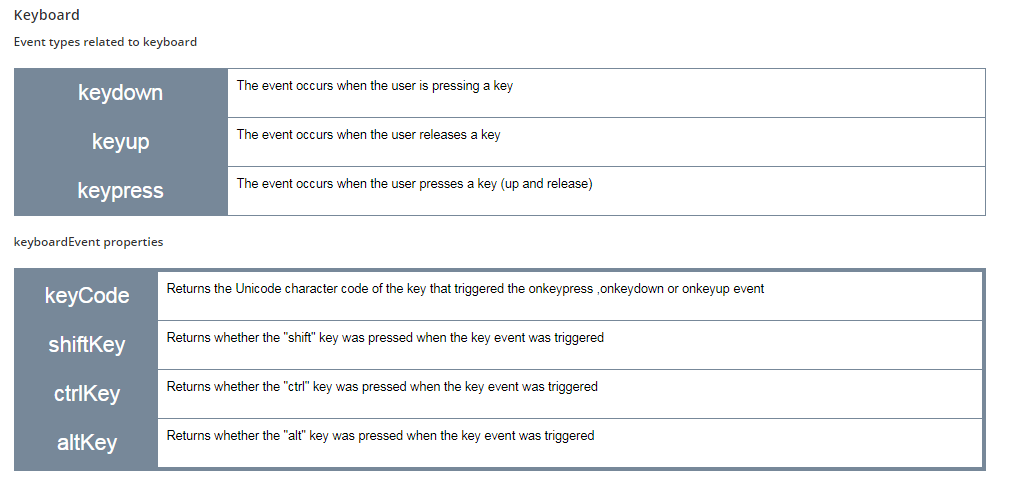
##### Events related to the page lifecycle

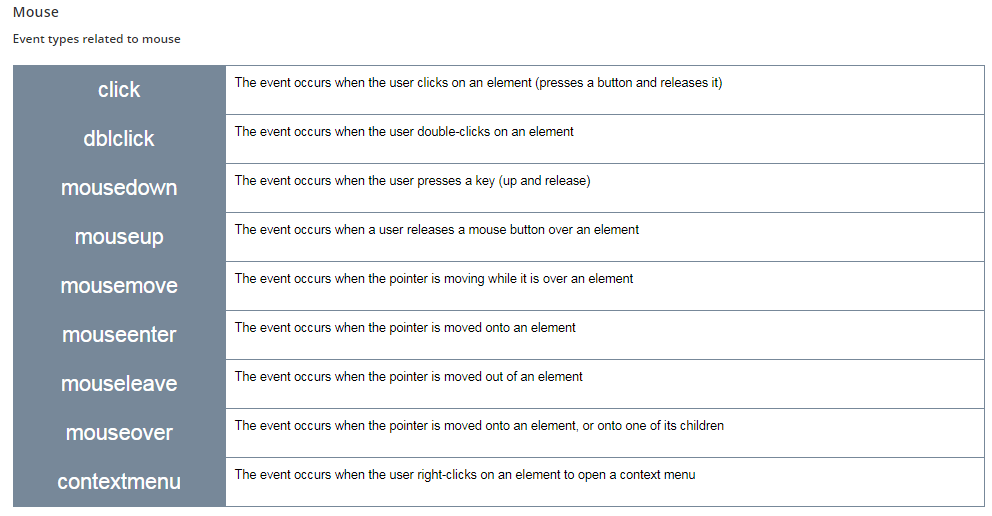
There are many other events related to the page life cycle. Below are the most useful ones for an intro course:

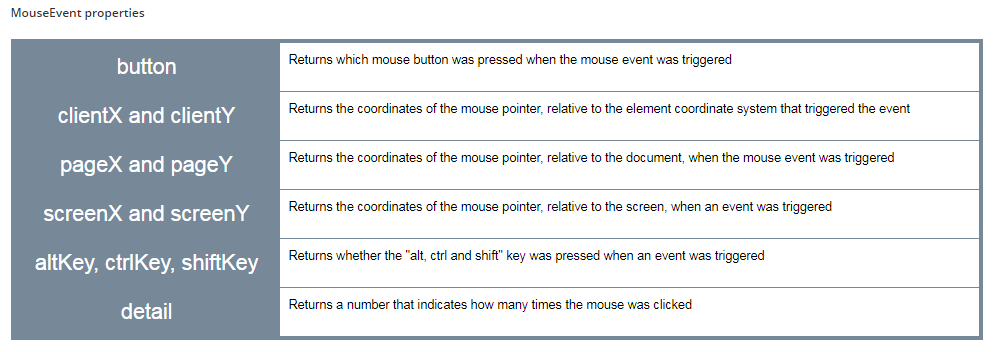


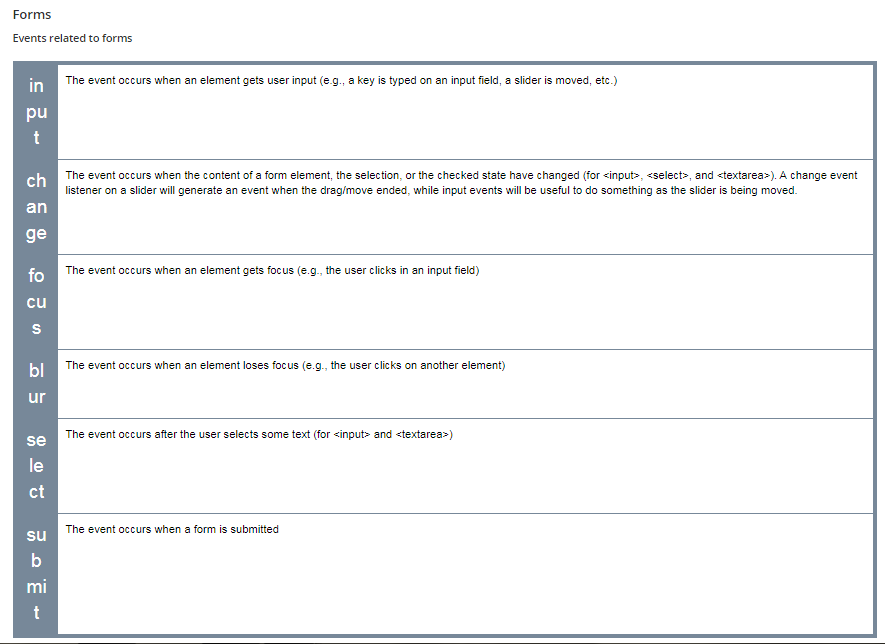
##### Page event properties

There are no particular properties that need to be mentioned here. Usually, the load event listener corresponds to a JavaScript function that can be seen as "the main" function of your Web Application. It is good practice to start everything after the page has been completely loaded. In the resize listener, you get the new size of the window, or the new size of some HTML elements in the page (as they might have been resized too when the window was resized) and then you do something (redraw a graphic in an HTML canvas that takes into account the new canvas size, for example).









##### FormEvent properties

There are no particular properties that need to be mentioned here. Usually, on a form event listener, we check the content of the different input fields, using their value property.