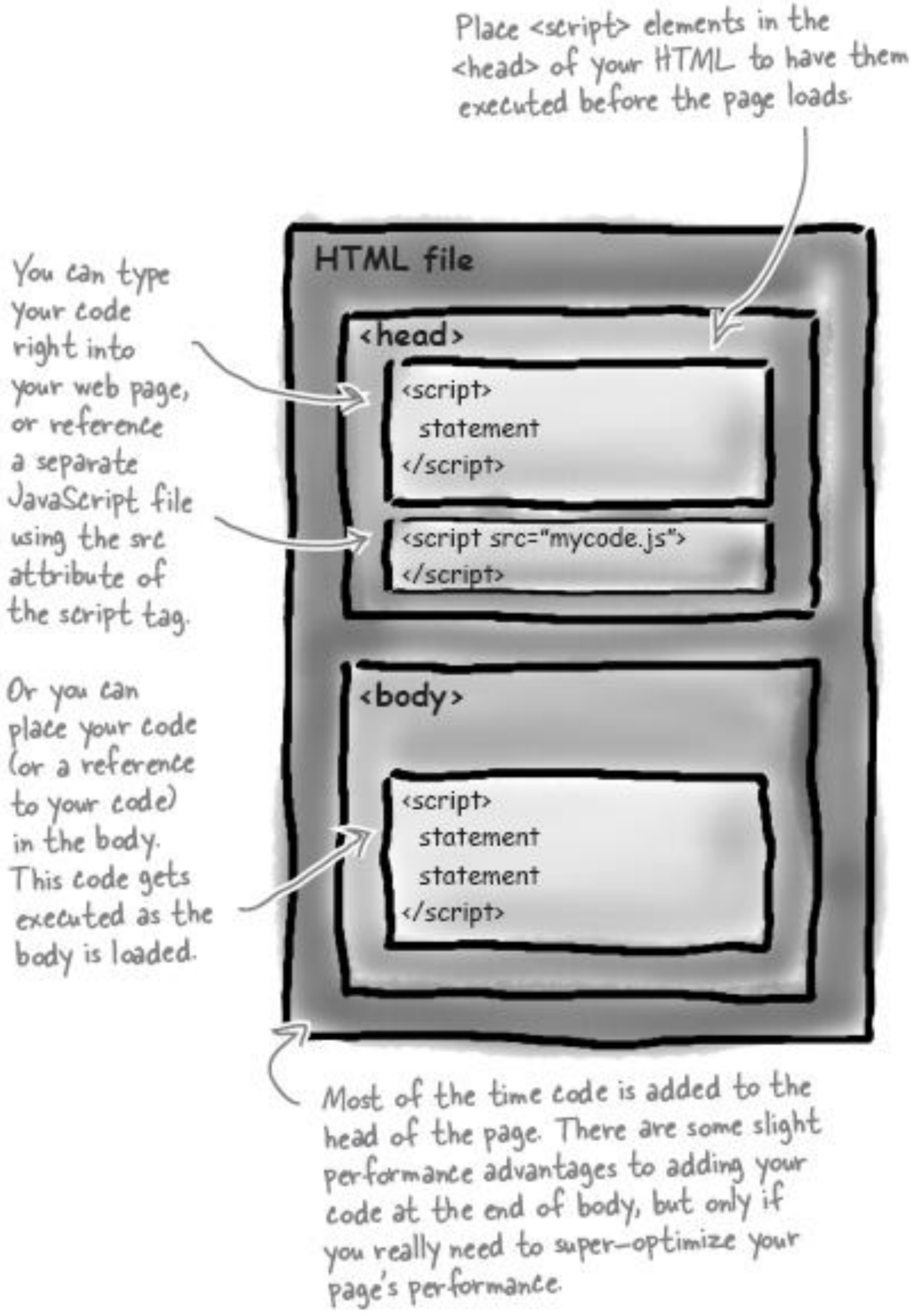
EVENTS

JavaScript: when

Where to place your JS code


Place <script> elements in the <head> of your HTML to have them executed before the page loads.
You can type your code right into your web page, or reference a separate JavaScript file using the src attribute of the script tag.
Or you can place your code (or a reference to your code) in the body. This code gets executed as the body is loaded.
HTML file
<head>
<script>
statement
</script>
<script src="mycode.js">
</script>
<body>
<script>
statement
statement
</script>
Most of the time code is added to the head of the page. There are some slight performance advantages to adding your code at the end of body, but only if you really need to super-optimize your page's performance.

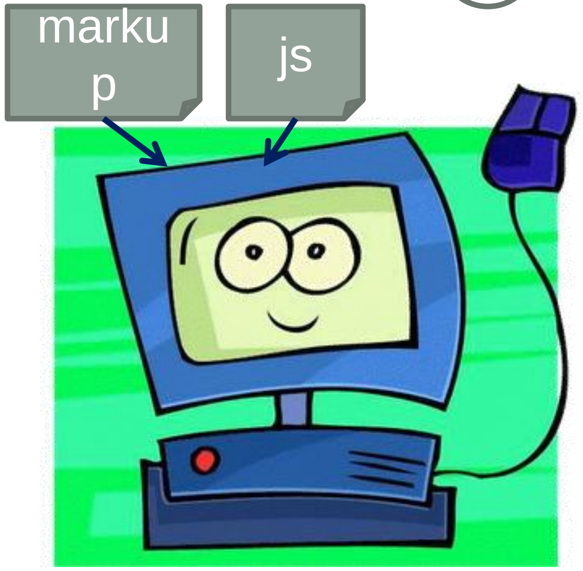
The way (client-side) JavaScript works

1

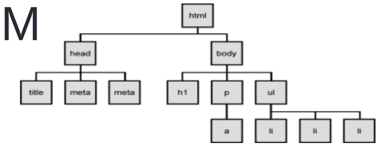
```
<!DOCTYPE HTML>

<html>
<head>
   <title> JS tutorial 17 </title>
   <meta charset="utf-8">
</head>
<body>
   <pre>
      <script>
         document.writeln("<H1> Immediate function invocation
</H1>");
         var squareRoot = function (x){return Math.sqrt(x)} (100);
                           //defined and called inline
         document.writeln ("squareRoot=" + squareRoot + "<br>");
      </script>
   </pre>
</body>
</html>
```

Write HTML file (index.html), JavaScript file (index.js)

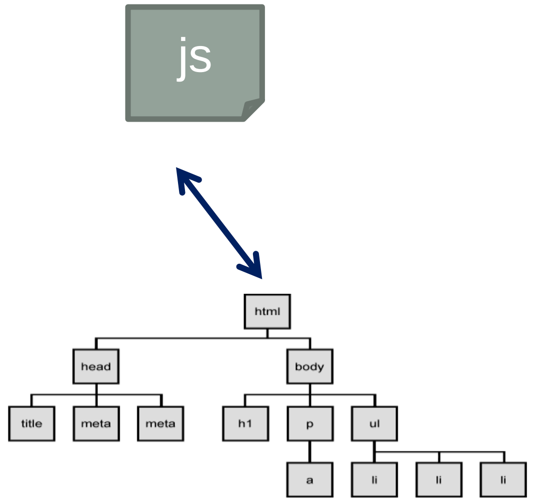
2



Browser parses HTML. When JS is encountered, it is executed. In addition, browser builds an internal model of HTML page: DOM



3



After page has been loaded, JS continues to interact with elements of DOM, but only when some **event** occurs

Interacting with DOM

- JS and HTML are two different things
- When the page is loaded, the browser creates an internal model of the document: DOM
- JS interacts with DOM to get access to the elements of the page: can add, remove, change appearance, content
- When JS modifies DOM, the browser re-renders the page

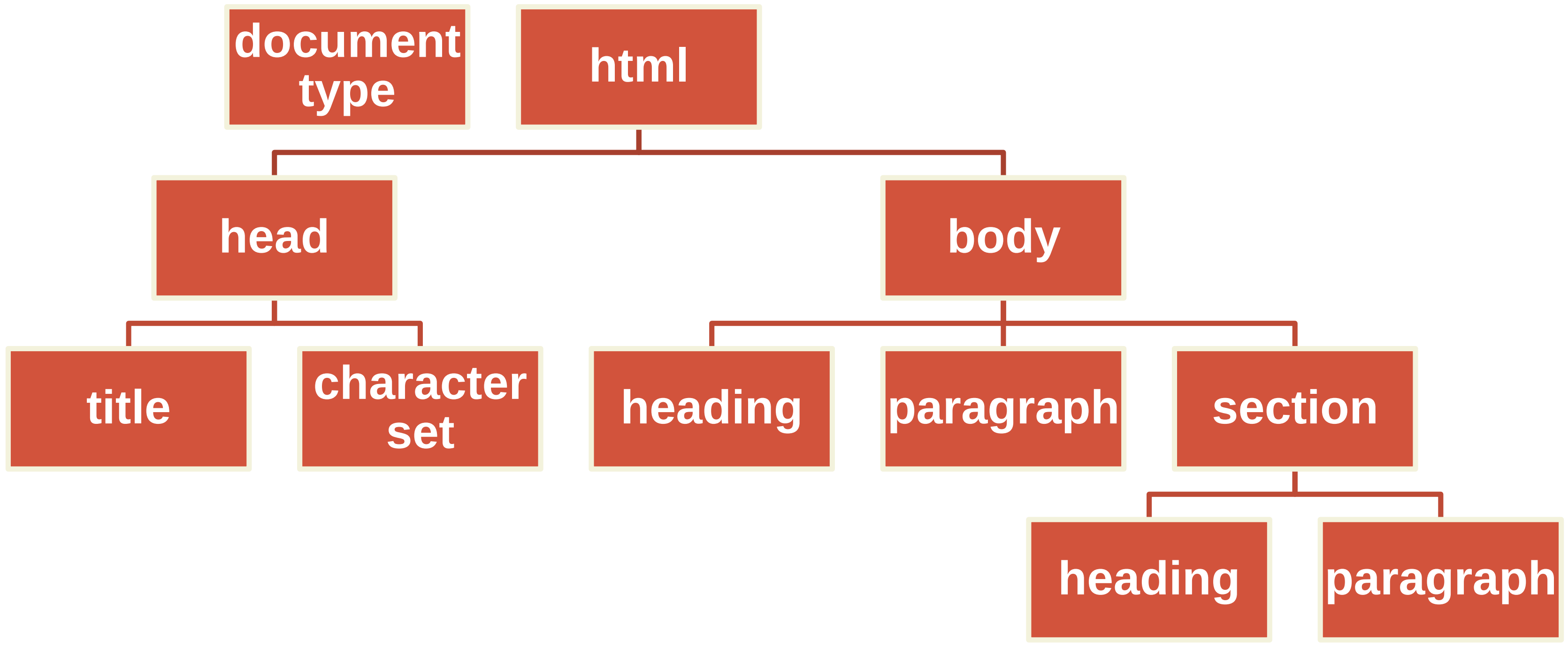
Nested elements

- All HTML documents consist of at least two levels of nesting
- At the top level is an <html> element, defining the bounds of the document

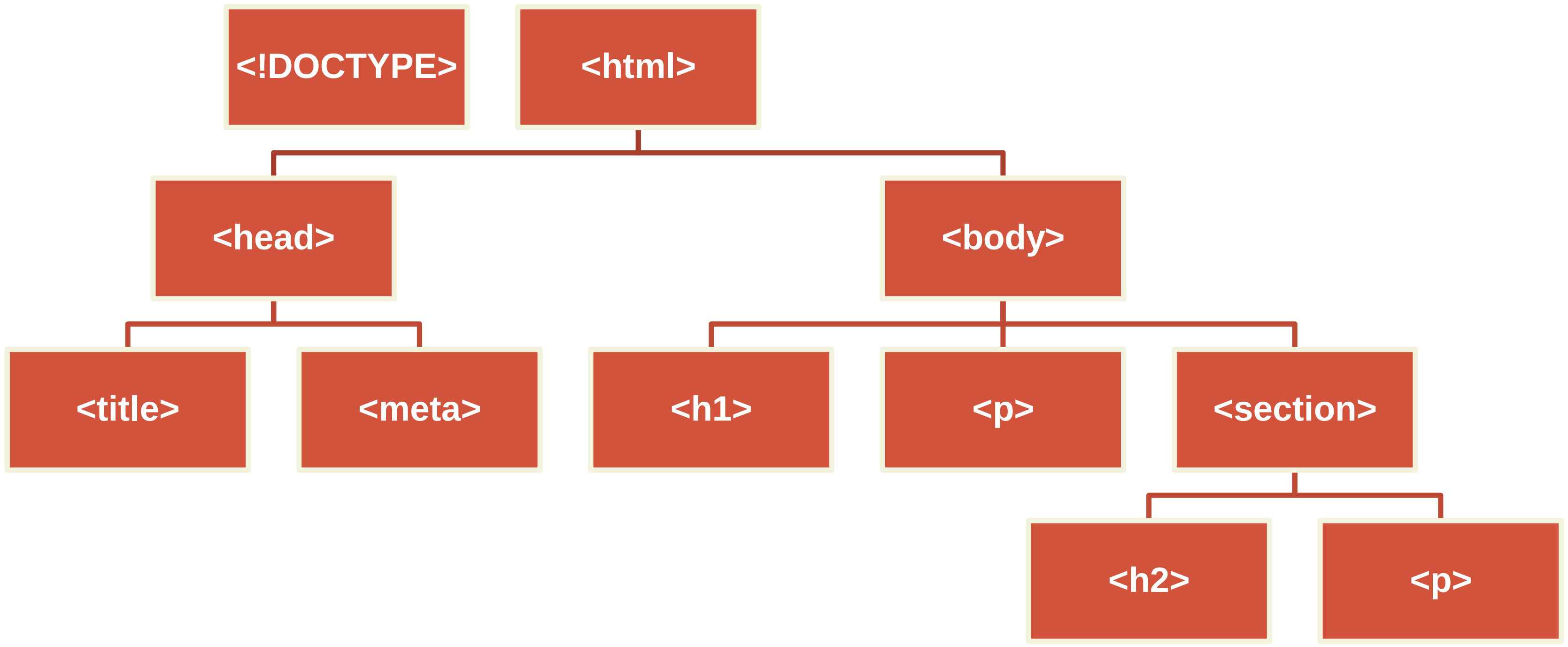
- The <head> and <body> elements are nested within <html>
- <head> contains meta data – information about a document as a whole, as opposed to content
- <body> holds all document content and all content elements, both block and inline

- both the <html> and <body> elements are always present in an HTML document – if they were not included by the author the browser will add them

The nested structure of HTML can be visualized as a tree

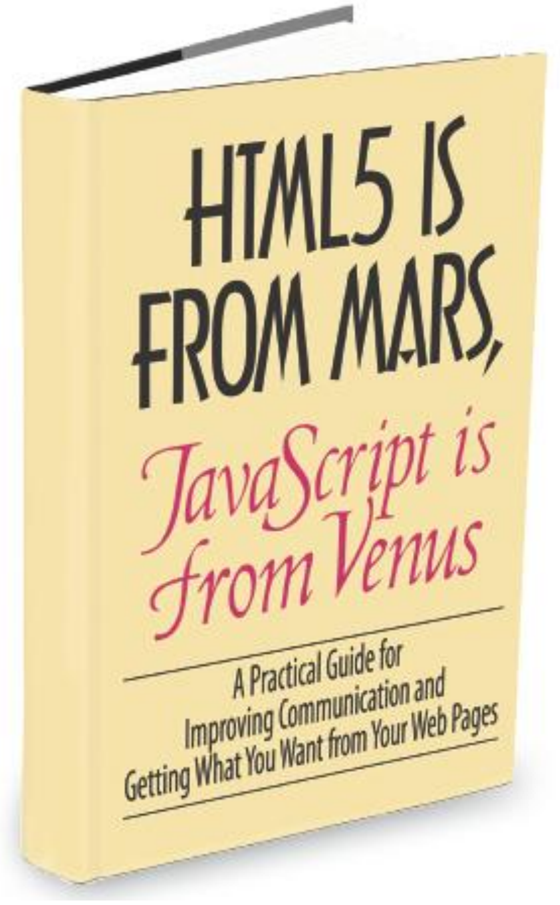


The nested structure of HTML can be visualized as a tree



HTML structure visualizer

http://www.aharef.info/static/htmlgraph/


HTML5 IS FROM MARS,
JavaScript is from Venus
A Practical Guide for Improving Communication and Getting What You Want from Your Web Pages

HTML and JS communicate through DOM:

```
var elem =
document.getElementById(“elem_id”);

elem.innerHTML = “new content”;
```

You can’t mess with DOM until the page is fully loaded

```
<script>

   function init() {
       var planet = document.getElementById("greenplanet");
       planet.innerHTML = "Red Alert: hit by phaser fire!";
   }
   window.onload = init;
</script>
```

This tells the browser: when the page is fully loaded – *onload* event fires – execute code in *init*

Inner HTML – everything enclosed in the referenced tag

```
<script>
  function init() {
      var planet = document.getElementById("greenplanet");
      planet.innerHTML = "Red Alert: hit by phaser fire!";
   }
   window.onload = init;
</script>
```

```
<p id="greenplanet">
        This is the
        <b>
                content
        </b>
        of HTML element p
</p>
```

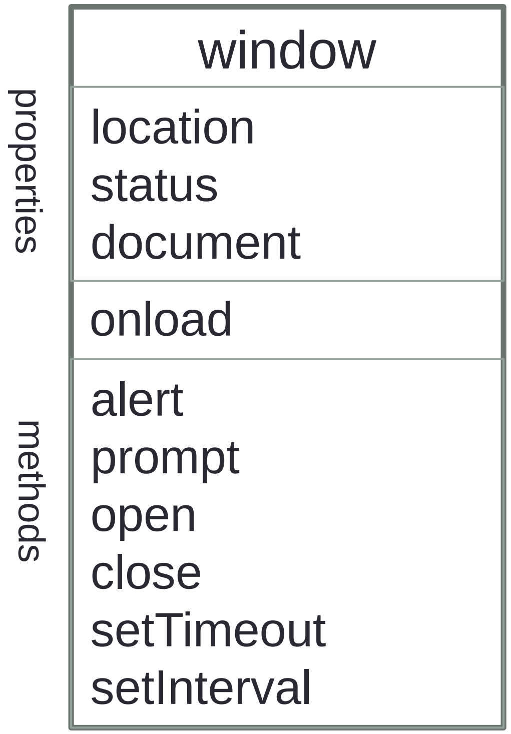
What JS can do with DOM

- **Get** elements from DOM: retrieve one element or a set of element
- **Create** element, **add** elements
- **Remove** elements
- Get and set **attributes** of elements

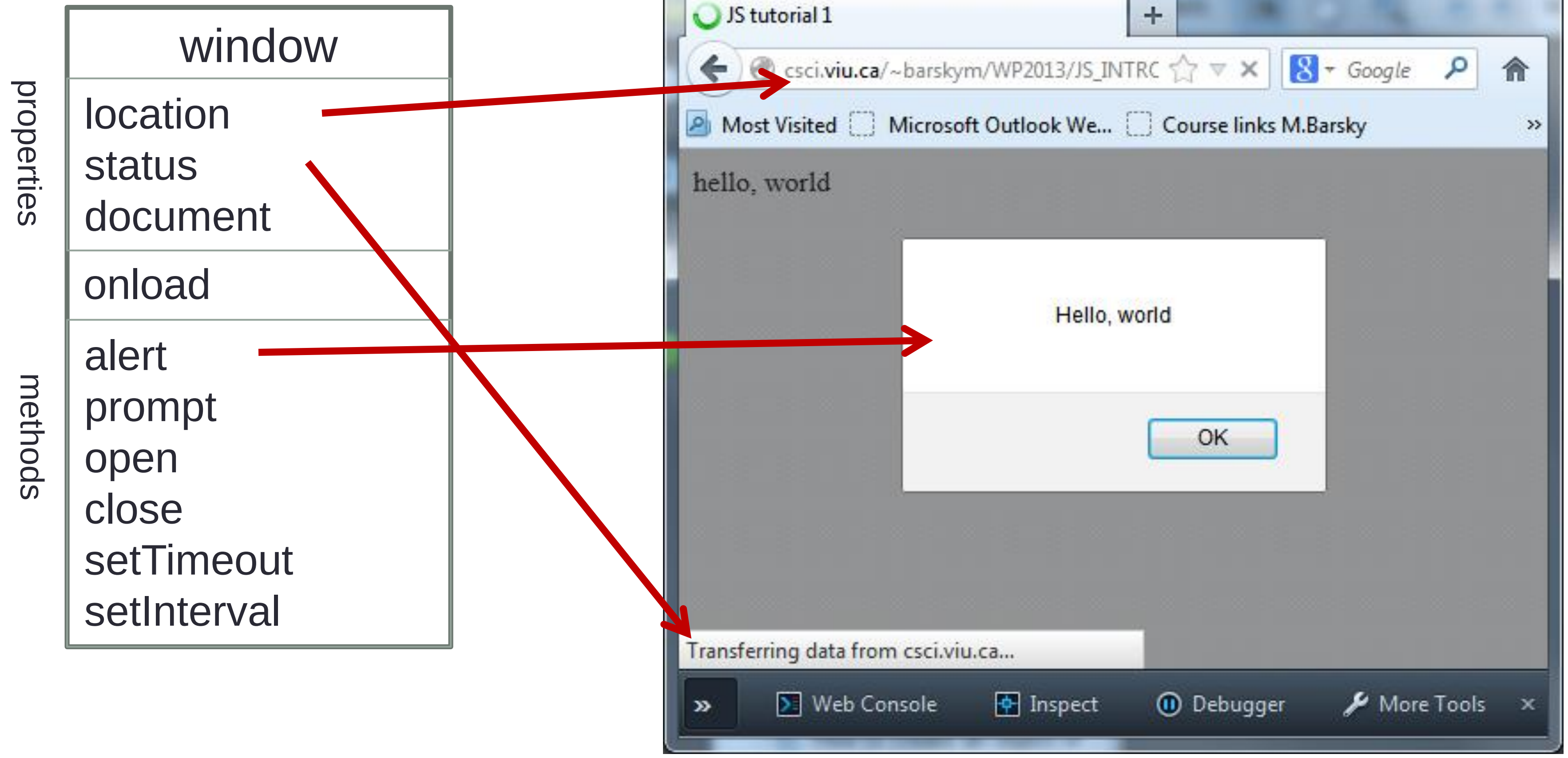
Built-in browser objects

- Window
- Document
- Each element

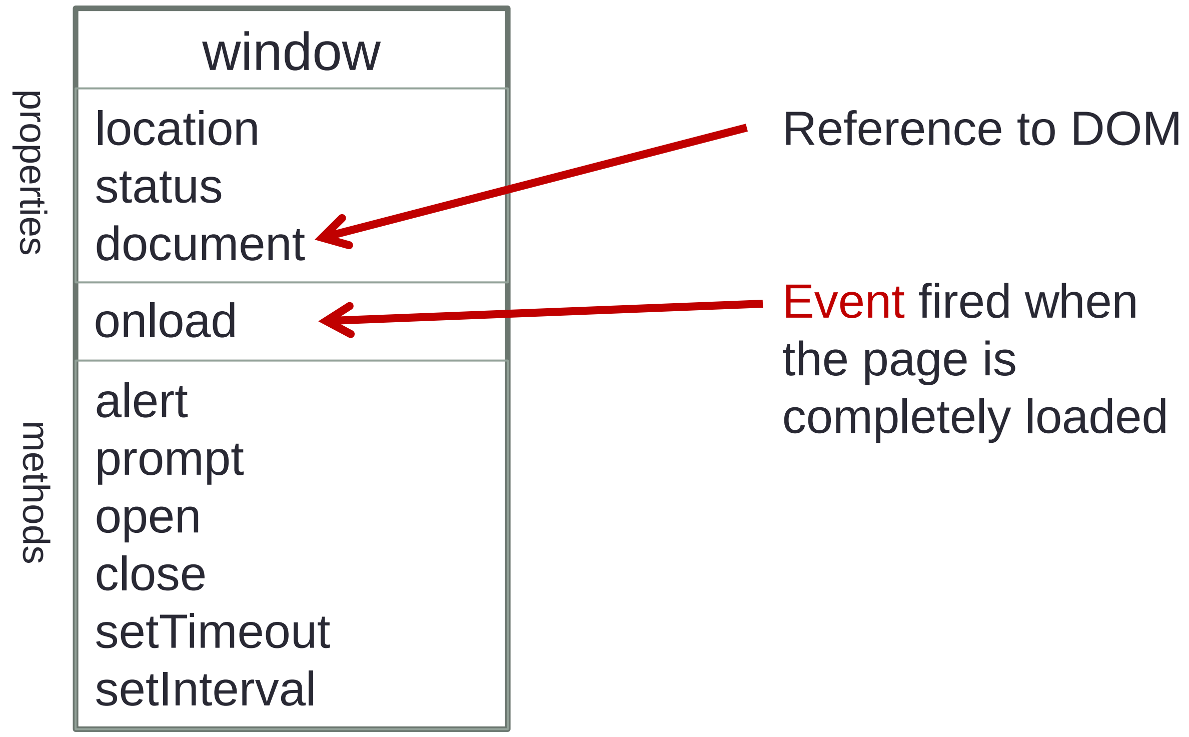
Window object:
core properties and methods



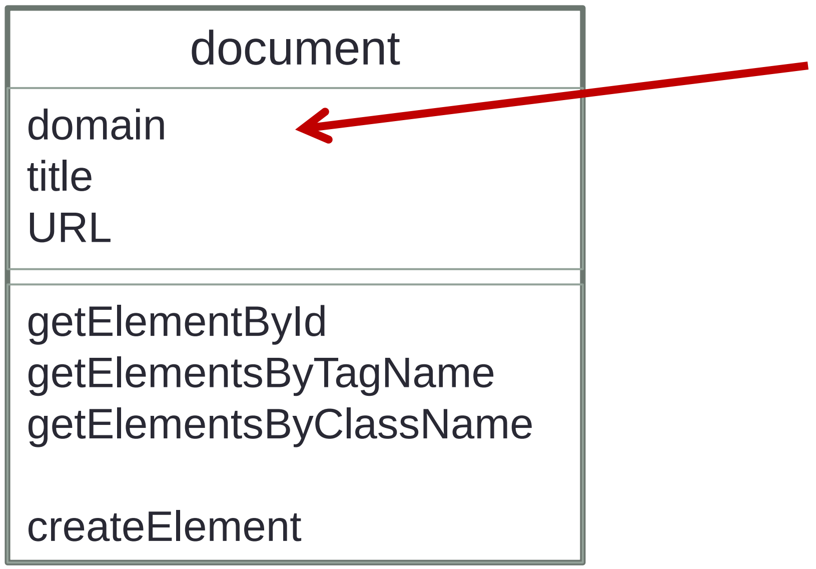
Window object: core properties and methods



Window object: core properties and methods

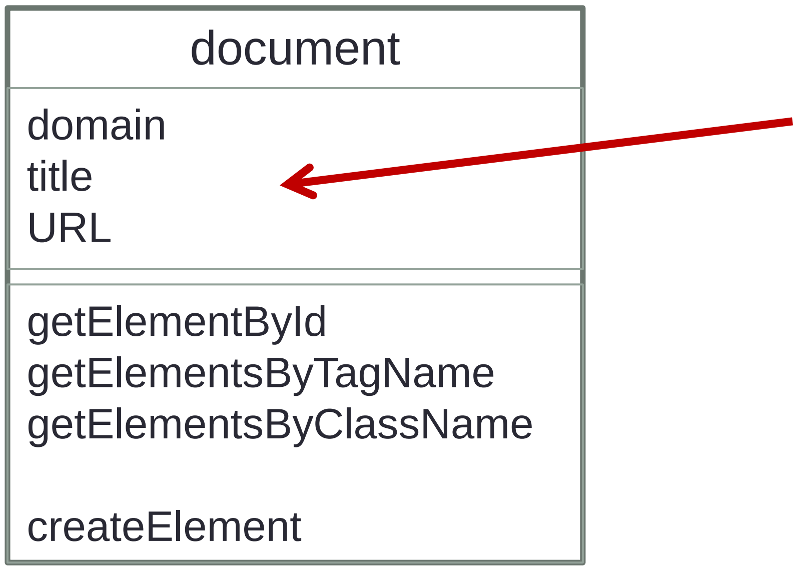


Document object



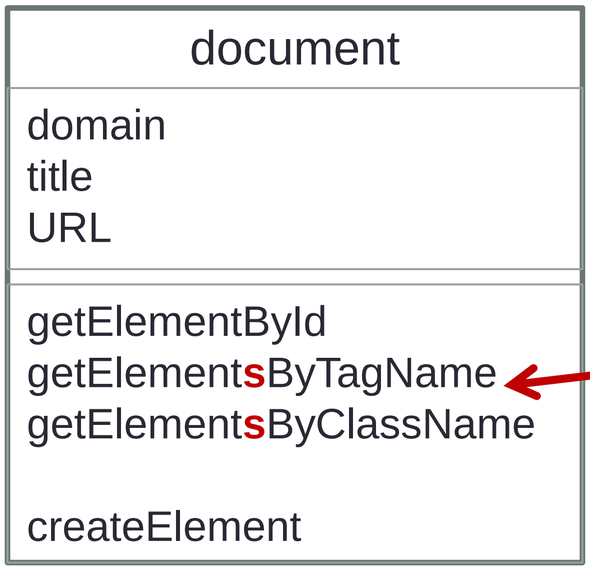
The domain is the name of a server the document was served from: csci.viu.ca

Document object



The document title

Document object



Similar to getElementById, but returns an array of all elements with the specified Tag or Class

Element objects

p
innerHTML childElementCount firstChild
appendChild insertBefore setAttribute getAttribute

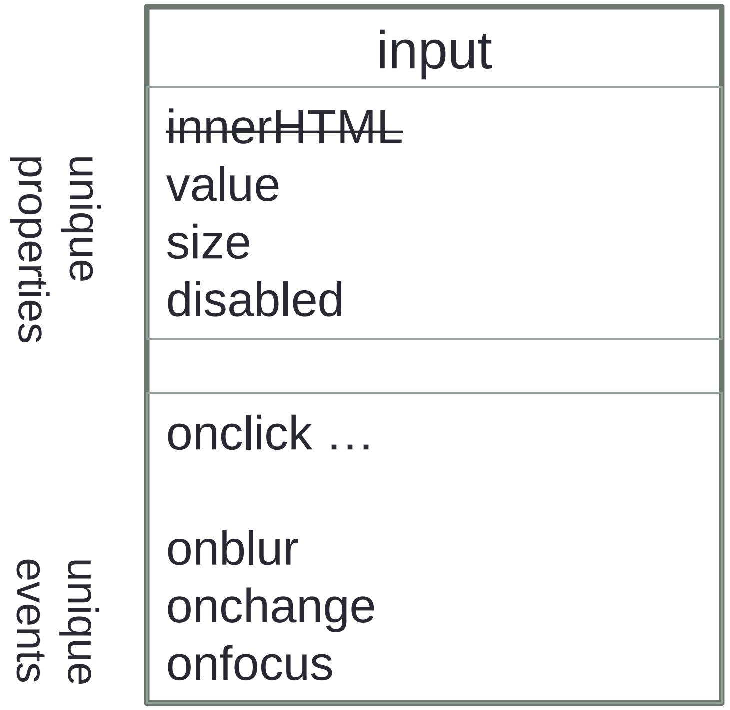
HTML Forms

- HTML forms are used to pass data to a server.
- An HTML form can contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more.
- For now we learn how to access these elements with client-side JavaScript

HTML input element

- The most important form element is the <input> element.
- The <input> element is used to select user information.
- An <input> element can vary in many ways, depending on the type attribute. An <input> element can be of type text field, checkbox, password, radio button, submit button, and more.

Input element



```
function test()
{
   var elem =
document.getElementById("test");
   elem.value = "asdf";
   elem.size = 125;
   elem.disabled=true;
}
```

Event listeners

- For each event we may attach a function, which will be executed when event fires:

```
window.onload = init;
```

Pre-defined events

- Mouse events
- Keyboard events
- Page-level events

Mouse events

- onclick
- ondblclick

- onmousedown
- onmouseup (when a user releases a mouse button over an element)

- onmousemove (when the pointer is moving while it is over an element)
- onmouseover (when the pointer is moved onto an element)
- onmouseout (when a user moves the mouse pointer out of an element)

Keyboard events

- onkeypress (when the user presses a key)

- onkeydown (when the user is pressing a key)
- onkeyup (when the user releases a key)

Attaching a single event listener

```
window.onload = init;

var button = document.getElementById (“testButton”);
button.onclick = handleButtonClick();
```

or as a tag attribute:

```
<body onload=“init”>

<input type=“button” onclick=“handleButtonClick()”>
```

Registering multiple event listeners

```
element.addEventListener('click‘,
                                    startDragDrop,false);
element.addEventListener('click',spyOnUser,false);
```

Syntax:

addEventListener()	Allows the registration of event listeners on the event target (IE8 = attachEvent())
removeEventListener()	Allows the removal of event listeners on the event target (IE8 = detachEvent())

Checking for browser support

```
if (window.addEventListener) {
        window.addEventListener('load', videoPlayer, false);
}
else if (window.attachEvent) {
        window.attachEvent('onload', videoPlayer);
}
```

Event handling in nested elements

- If an element and one of its ancestors have an event handler for the same event, which one should fire first?

```
----------------------------------------
| elementP                            |
|   -------------------------         |
|   |elementC                         |
|   -------------------------         |
|                                     |
----------------------------------------
```

Event handling in nested elements

- Netscape said that the event on Parent takes place first. This is called event capturing.
- Microsoft maintained that the event on Child takes precedence. This is called event bubbling.

```
---------------------------------------
| elementP                            |
|   -------------------------         |
|   |elementC                         |
|   -------------------------         |
|                                     |
---------------------------------------
```

W3C reconciliation

- Any event taking place in the W3C event model is first captured until it reaches the target element and then bubbles up again.
- You, the web developer, can choose whether to register an event handler in the capturing or in the bubbling phase. This is done through the addEventListener()'s parameter 3:

If its last argument is true the event handler is set for the capturing phase, if it is false the event handler is set for the bubbling phase.