Introduction

JavaScript / HTML5 Canvas

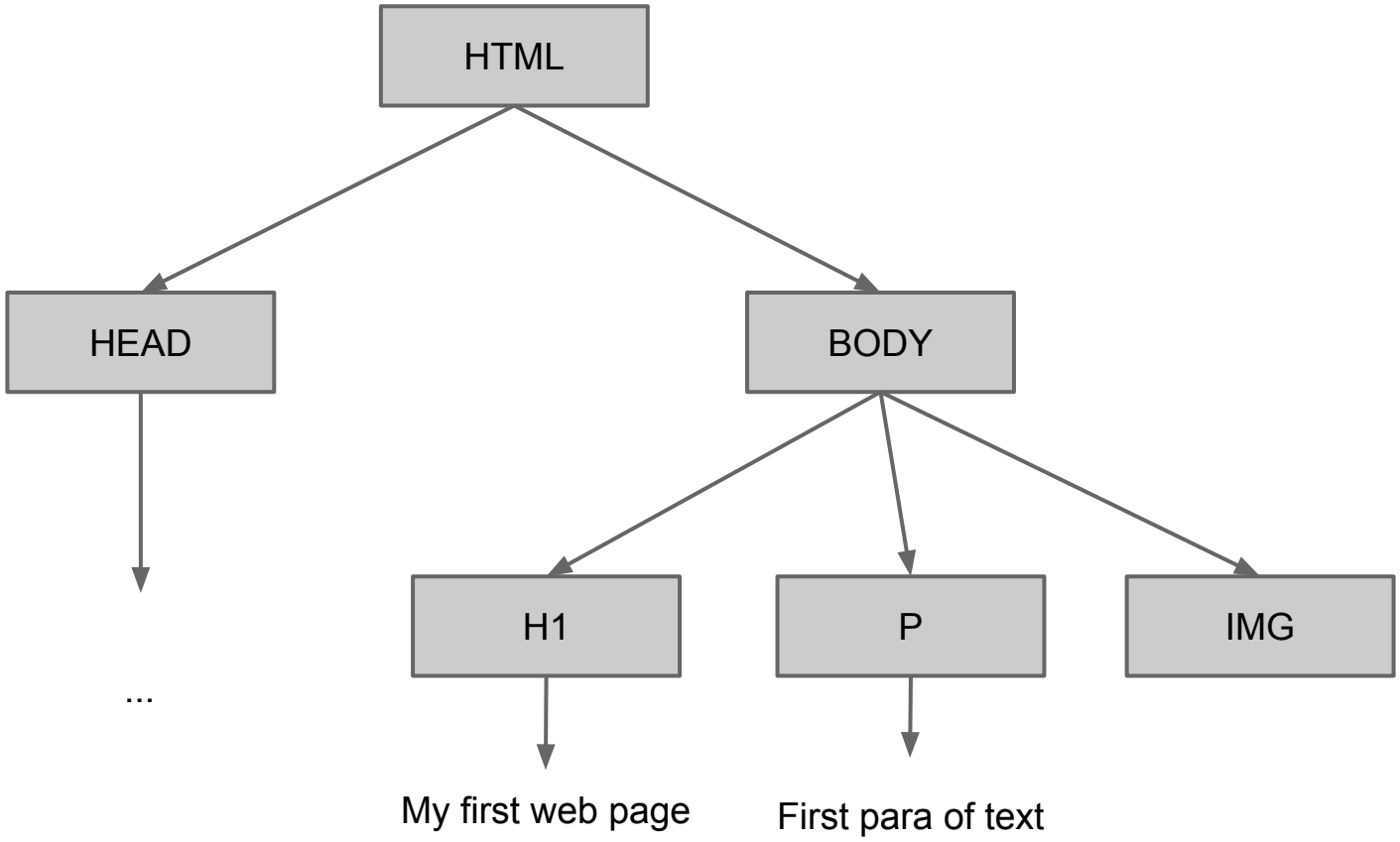
Riccardo Pucella

September 14, 2015

HTML recap

- HTML describes a tree
- That tree “represents” a structured document
- Browsers know how to render the tree into something visually appealing
- Browsers generally agree on how to render that tree...

HTML as a tree (DOM)



HTML as text

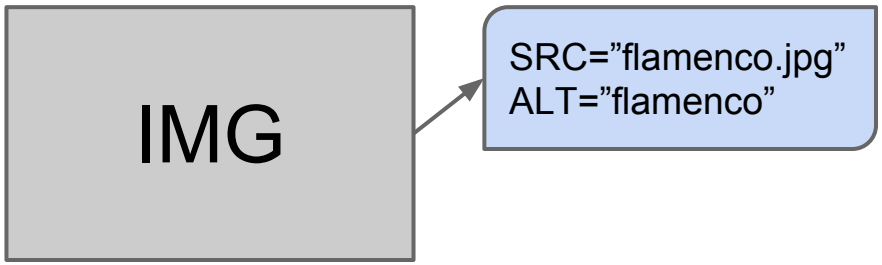
```
<html>
  <head> ... </head>

 <body>
    <h1>My first web page</h1>
    <p>First para of text</p>
    <img src=”flamenco.jpg” alt=”flamenco”>
      </img>
  </body>

</html>
```

Attributes

Attributes modify the meaning of elements



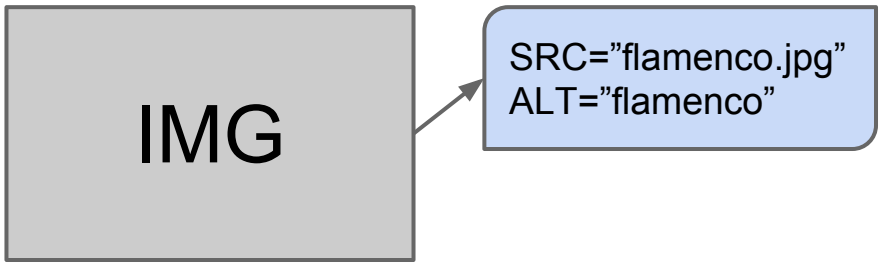
(Non-HTML parameters)

Some global

Some specific

Attributes

Attributes modify the meaning of elements



(Non-HTML parameters)

Some global

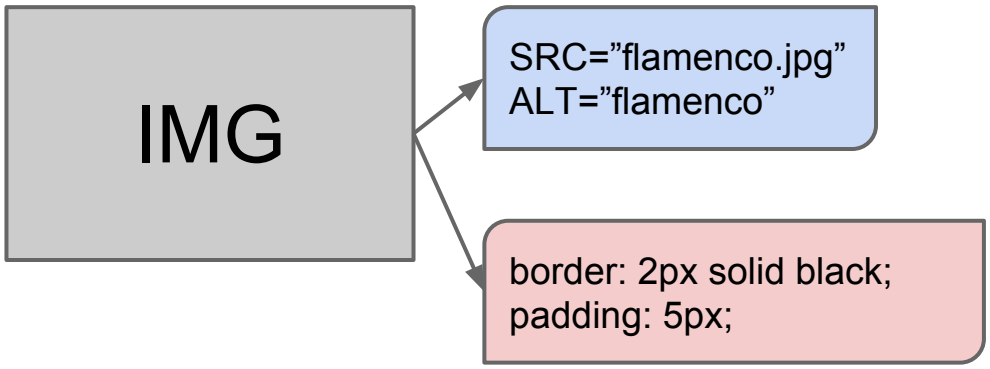
Some specific

ID = “…”

CLASS = “…”

Styles

Styles modify the appearance of elements

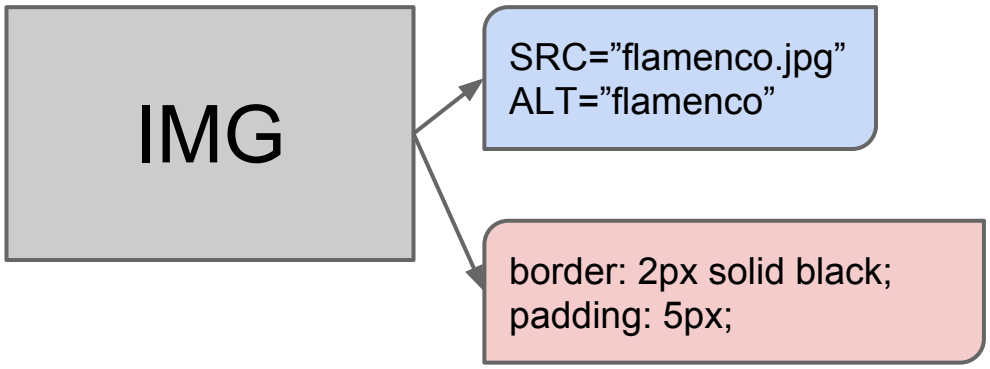


Can be attached to any element via STYLE attribute

Can be applied through a style sheet

Styles

Styles modify the appearance of elements



Can be attached to any element via STYLE attribute

Can be applied through a style sheet

Defined inline
<STYLE> element

Defined in a file
<LINK> element

CSS

Apply style to all elements with a given tag:

```
img { border: 5px solid black; }
```

Apply style to an element with given ID:

```
#someId { border: 5px solid black; }
```

Apply style to all elements with given CLASS:

```
.someClass { border: 5px solid black; }
```

CSS

Apply style to all elements with a given tag:

```
img { border: 5px solid black; }
```

Apply style to an element with given ID:

```
#someId { border: 5px solid black; }
```

Apply style to all elements with given CLASS:

```
.someClass { border: 5px solid black; }
```

Somewhat rich language of selectors

JavaScript

- A programming language in the browser
- Why is that even useful?
- Modify the content of a page in response to events
  - Modify the content of the page = Manipulating the DOM
  - Events = interesting stuff that happens on a page (button clicks, mouseovers, selectors, etc)

JavaScript

- A p
- Wh
- Mo
  res
  - Events = interesting stuff that happens on a page (button clicks, mouseovers, selectors, etc)

Why JavaScript programming is painful:

It is not procedural
(where the program is in control)

It is reactive
(where the browser is in control)

Inverted program structure

Ye Olde Programming Language

Nothing particularly special going on

- Standard imperative language
- C-like syntax
- Dynamically typed
- First-class functions - `function (...) { … }`
- Types include: numbers, strings, objects
- Objects to a first approximation like dictionaries in Python

```
{ x: 10, y: 20, z: 30}
```

- OO: Not class-based, but prototype-based

JavaScript in HTML

Embedded in a <SCRIPT> element

```
<SCRIPT>
    javascript code
</SCRIPT>
```

Or loaded via a <SCRIPT> element

```
<SCRIPT SRC="file.js"></SCRIPT>
```

How do you run the code?

Running JavaScript code

The browser executes JavaScript code when it gets to the <SCRIPT> element

- that can lead to race conditions

Best practice:

- the JavaScript code should only define functions and set up event handlers
- Create a handler to execute code upon load if needed

```
window.addEventListener("load",run,false);
```

JavaScript and DOM

Way too much to describe -- look it up

But basically, can lookup an element by ID.

```
var e = document.getElementById("foo");
```

The result is an object representing the element

```
e.setAttribute(...)
e.getAttribute(...)
e.style.border="5px solid black";
```

JavaScript and DOM

Navigate the tree:

```
e.parentNode        e.firstChild
e.nextSibling       e.childNodes
```

Create new elements:

```
var img = document.createElement("img")
img.setAttribute("src","flamenco.jpg")
e.appendChild(img)
```

etc…

The <canvas> element

Provides a simple canvas in which to draw

- Bitmap - no structure to the drawing
- The canvas remembers the result of drawing so that it can refresh if needed
- Use JavaScript to draw on the canvas

```
<canvas id="demo" width="500" height="500">
</canvas>
```

Drawing in a canvas (1)

```
var cnvs = document.getElementById("demo");
var ctxt = cnvs.getContext("2d");
```

All drawing operations are methods of the context

(0,0) is the upper-left corner

(width,height) is the lower-right corner

- where width/height come from <canvas>

Drawing in a canvas (2)

Most drawing via paths:

- draw lines and arcs
- (optional: close the path)
- stroke (or fill) to actually draw

```
ctxt.beginPath();
ctxt.moveTo(100,100);
ctxt.lineTo(200,200);
ctxt.lineTo(300,100);
ctxt.stroke();  // draw the path
```

Drawing in a canvas (2)

Most drawing via paths:

- draw lines and arcs
- (optional: close the path)
- stroke (or fill) to

```
ctxt.beginPath();
ctxt.moveTo(100,10
ctxt.lineTo(200,2
ctxt.lineTo(300
ctxt.stroke();  //
```

Alternatively, close the path, and fill in the figure:

```
ctxt.closePath();
ctxt.fill();
```

Drawing in a canvas (3)

Can change the stroke color and the fill color:

```
ctxt.fillStyle=”red”;
ctxt.fillStyle=”#5c5c5c”;
ctxt.strokeStyle=”blue”;
```

Stroke/fill styles persist until changed

- **`beginPath()`** resets stroke/fill styles

Modifying a canvas

Drawing over something literally draws over.

A canvas doesn’t remember “how” something was drawn.

Corollary: you can’t delete from a canvas

You can erase part or all the canvas by drawing a filled rectangle of the appropriate background color

Modifying a canvas

Drawing over something literally draws over.

A canvas doesn't remember "how" something was drawn.

Corollary: you can't delete from a canvas

You can erase part or all the canvas by drawing a filled rectangle of the appropriate background color *(Yech!)*

Going further

There’s a lot more you can do with canvases:

- apply transformations to distort
- write text
- use gradients for color
- display images (PNG, JPG,...)

You can also do animations, of course.

See Readings on the course web page.