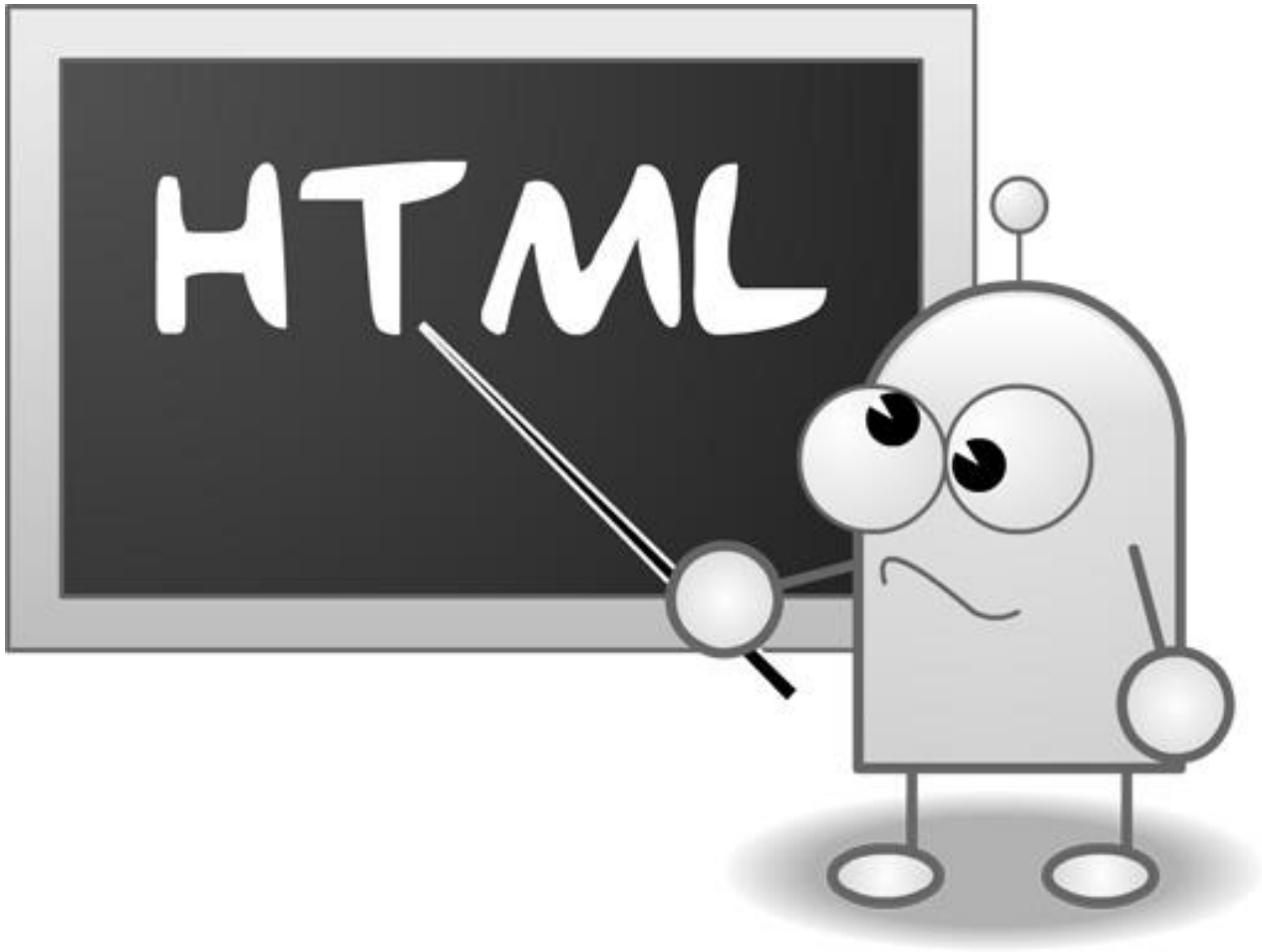
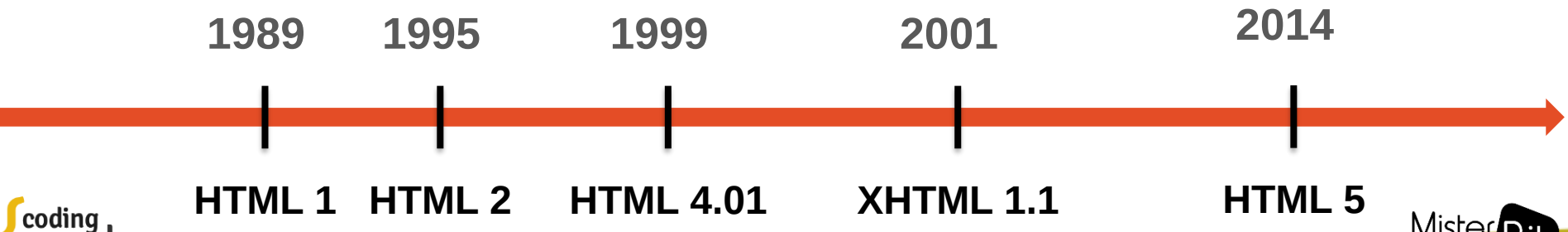

HTML

What is it?

HTML5 is a markup language used for structuring and presenting content on the World Wide Web.



What is HTML

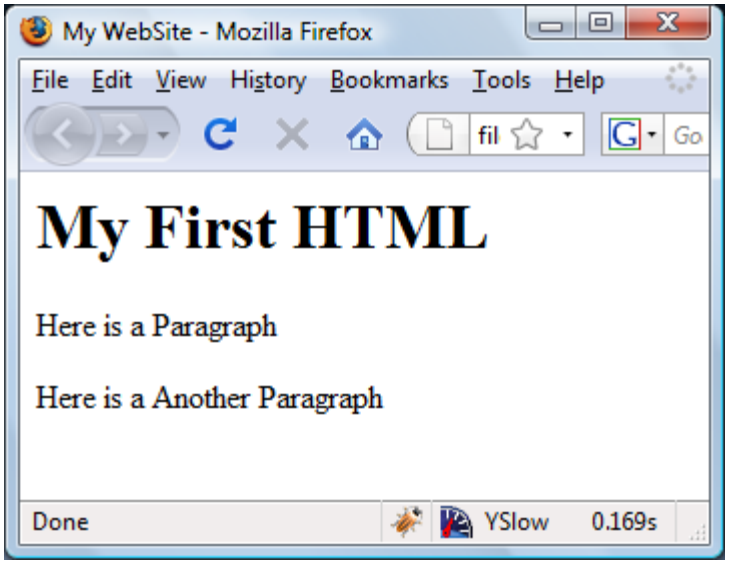
HTML is a language for describing web pages.

HTML stands for **H**yper **T**ext **M**arkup **L**anguage.

```
<html>
<head>
   <title>My WebSite</title>
</head>
<body>
   <h1> My First HTML </h1>
</body>
</html>
```

Basic Example

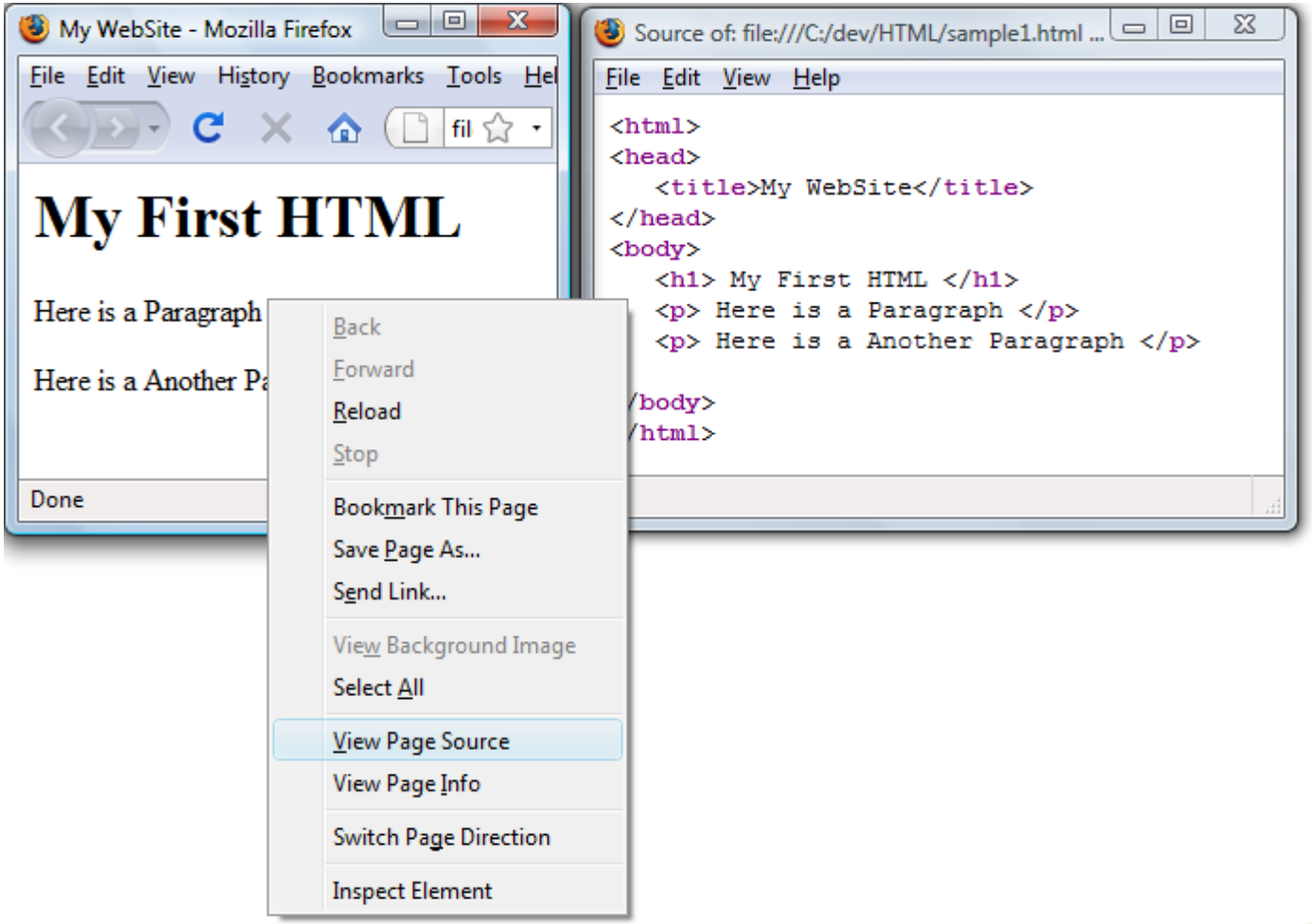
```
<html>
<head>
   <title>My WebSite</title>
</head>
<body>
   <h1> My First HTML </h1>
   <p> Here is a Paragraph </p>
   <p> Here is a Another Paragraph </p>

</body>
</html>
```



- <html> describes the web page.
- <body> is the visible page content.
- <h1> is displayed as a heading.
- <p> is displayed as a paragraph.

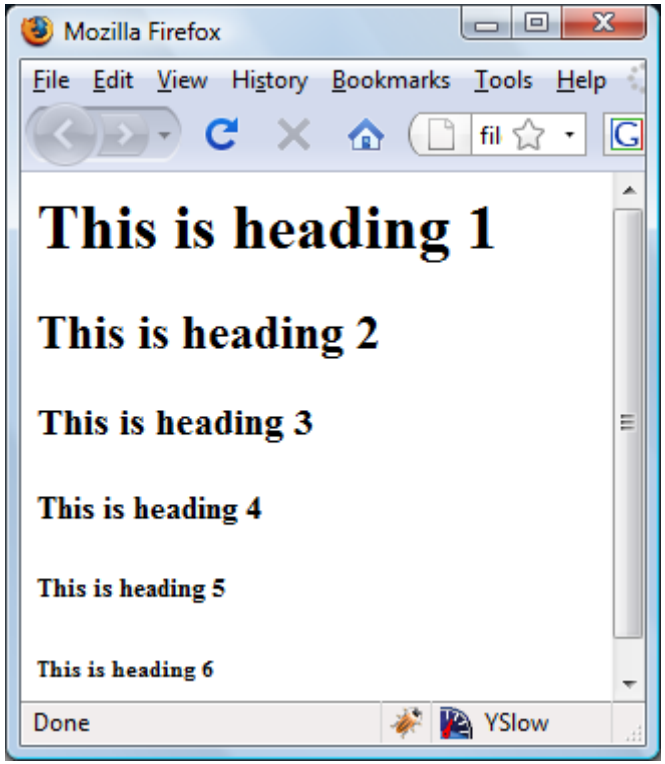
- <head> gives information about the page.
- <title> is the browser's window title.

HTML is the Browser's "Mother-Tongue"



Headings

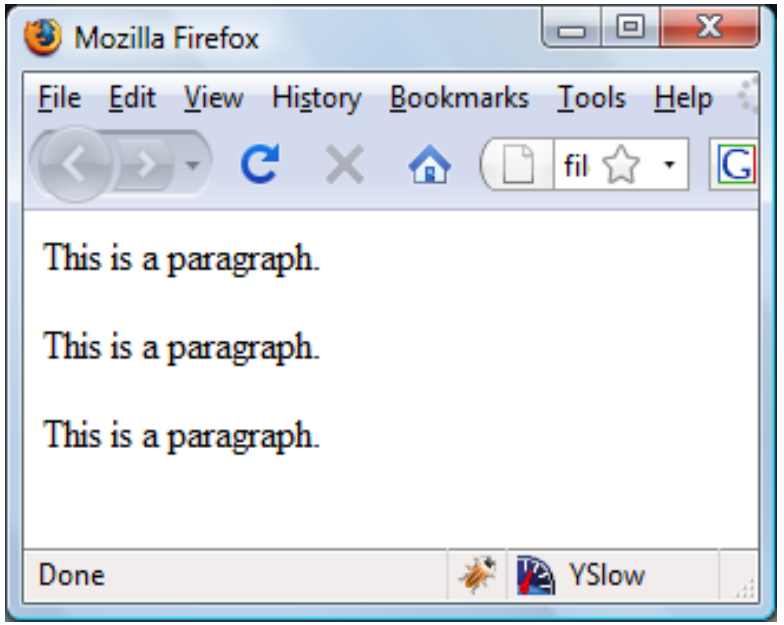
```
<html>
<body>
    <h1>This is heading 1</h1>
    <h2>This is heading 2</h2>
    <h3>This is heading 3</h3>
    <h4>This is heading 4</h4>
    <h5>This is heading 5</h5>
    <h6>This is heading 6</h6>
</body>
</html>
```



- These are the headers provided in HTML.
- All those <h*> elements are block elements
  - So a line-break is added
- Browsers also add some margin at the top and bottom

Paragraphs

```
<html>
<body>

<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>

</body>
</html>
```



- Paragraphs are also block elements
  - So a line-break is added
- Browsers also add some margin at the top and bottom

Images

- Note that *<img>* is an **empty element** – no closing tag is needed.
  - AKA – Self enclosing Tag
- The source for the image is provided through an **attribute**

```
<html>
<body>

    <img src="lala.jpg"   />

</body>
</html>
```

Links

```
<html>
<body>

Go here:
<a href="http://baba.com" > Baba Buba </a>

</body>
</html>
```

- Links allow us to navigate in the web.
- The link address is provided through the *href* **attribute**

More Useful Elements

- **
** - (break-row) used to break lines.
  - Note that normal line breaking (through keyboard) are translated to spaces.
- **<hr />** - (horizontal rule) used to draw a line.

```
<hr   />
<br   />
```

Comments

- We can use comments (comments are ignored by the browser):
  - To improve readability and order
  - Remove part of the markup for debugging

```
<!-- Sample Comment! -->
```

Lists

- Unordered List:

```
<h5>Great Carrot Drink</h5>
<ul>
   <li>Carrots - 5 pieces</li>
   <li>Honey - 1 spoon</li>
   <li>Ice - one glass</li>
</ul>
```

Great Carrot Drink

- Carrots - 5 pieces
- Honey - 1 spoon
- Ice - one glass

- Ordered List:

```
<h5>Follow the following steps:</h5>
<ol>
   <li>Go straight to the End</li>
   <li>Turn left</li>
   <li>Leave the case by the tree</li>
   <li>Start running</li>
</ol>
```

Follow the following steps:

1. Go straight to the End
2. Turn left
3. Leave the case by the tree
4. Start running

Nested Lists

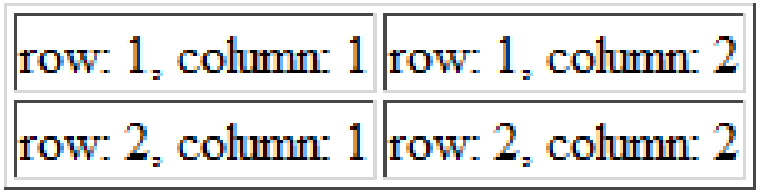
Here is an example of nested lists:

```
<ul>
        <li>Japan</li>
        <li>Israel:
            <ul>
                <li>Tel Aviv:
                    <ul>
                        <li>Florentin</li>
                        <li>HaTikva</li>
                    </ul>
                </li>
                <li>Jerusalem</li>
            </ul>
        </li>
        <li>China</li>
    </ul>
```

Tables

Here is an example:

```
<table border="1">
<tr>
    <td>row: 1, column: 1</td>
    <td>row: 1, column: 2</td>
</tr>
<tr>
    <td>row: 2, column: 1</td>
    <td>row: 2, column: 2</td>
</tr>
</table>
```



- <*tr*> represents a table-row
- <*td*> represents a cell: table-data

Table Headers

Tables may use <th> define headers:

```
<table border="1">
<tr>
    <th>header1</th>
    <th>header2</th>
</tr>
<tr>
    <td>row: 1, column: 1</td>
    <td>row: 1, column: 2</td>
</tr>
<tr>
    <td>row: 2, column: 1</td>
    <td>row: 2, column: 2</td>
</tr>
</table>
```

header1	header2
row: 1, column: 1	row: 1, column: 2
row: 2, column: 1	row: 2, column: 2

Tables

Cells may span more that one column, here is how:

```
<table border="1">
<tr>
   <th>Name</th>
   <th colspan="2">Phones</th>
</tr>
<tr>
   <td>Muki D.</td>
   <td>763-8796-980</td>
   <td>763-3746-731</td>
</tr>
</table>
```

Name	Phones	
Muki D.	763-8796-980	763-3746-731

Tables

Cells may span more that one row:

```
<table border="1">
<tr>
  <th>Name:</th>
  <td>Puki G.</td>
</tr>
<tr>
  <th rowspan="2">Phones:</th>
  <td>763-8796-980</td>
</tr>
<tr>
  <td>763-3746-731</td>
</tr>
</table>
```

Name:	Puki G.
Phones:	763-8796-980
	763-3746-731

Tables

Good tables include a <tbody> and optionally <thead>

```
<table>
 <thead>
    <tr>
      <th> column: 1 </th>
      <th> column: 2 </th>
    </tr>
  </thead>
<tbody>
   <tr>
      <td>row: 1, column: 1</td>
      <td>row: 1, column: 2</td>
   </tr>
   <tr>
      <td>row: 2, column: 1</td>
      <td>row: 2, column: 2</td>
   </tr>
</tbody>
</table>
```

HTML Entities

- Some characters (like <) are better not placed inside the text (the browser might mistake them for tags).

- HTML Entities are used to output these special characters. For example: *<* is the entity code for <.

- You can also use entity numbers, such as: ¥
  - Entity names are recommended as they are more readable.
  - However, some Entity names are not supported by all browsers.

HTML Entities

Here are some sample entities, there are actually many

Entity Output	Description	Code	Number
	non-breaking space		
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
§	section	&sect;	&#167;
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

Head Elements

The Head element contains information about the HTML document:

```
<meta charset="utf-8" />
<meta name="title"       content="Frogi Pets Shop" />
<meta name="description" content="Only the Best for your Pet" />
<meta name="keywords"    content="Frogi, Pets Food, miyahu" />
```

In a professional level HTML, the <head> contains much more, we will get to them later, if you are curious see here…

Styling with CSS

- HTML is about the **structure** of the document
- CSS is about the **design** of the document

Introduction to CSS

- CSS – Cascading Style Sheets
- We use it to *magically* control the look of our apps

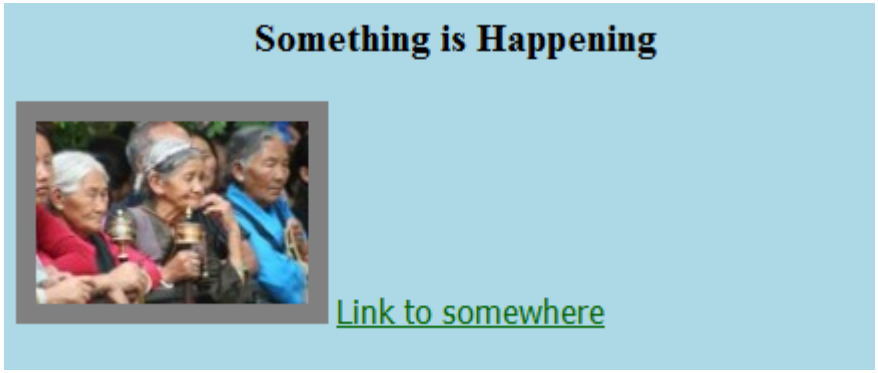
- Be warned, getting hold of the CSS magic will take you further than you think.

Introduction to CSS

- The attribute style can be used on most of the HTML elements.
  - Inside this attribute we can use inline CSS

- Using CSS we can define the look of our web page

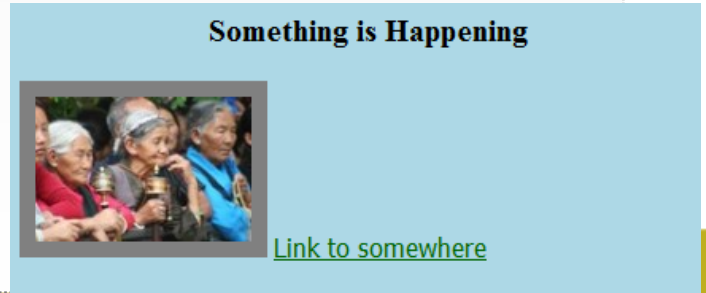
- For example, examine the following Formatting:



Using CSS

Let's have a look at the backing CSS:

```
<html>
<body style="background-color: lightblue;">

<p style="text-align: center; font-size:20px; font-weight:
bold;">
Something is Happening
</p>

<img src="pic1.jpg" style="border: 10px gray solid" />

<a href="http://google.com" style="color: darkgreen;
                                   font-family: tahoma;  ">
Link to somewhere
</a>

</body>
</html>
```



Using the Class Attribute

- Using the style attribute is the wrong way to design the look of your pages:
  - Mix of HTML & CSS is **hard to maintain**
  - We cannot **reuse** previous declaration (hard to keep a consistent look)

- Use the **class** attribute instead:

```
<style>
    .nice-link{
        color: darkgreen;
        font-family: tahoma;
    }
</style>
```

```
<a href="http://google.com" class="nice-link">Go Look</a>
```

CSS Selectors

- Selectors are our way to refer the HTML elements from CSS (and also Javascript)
- Here are some examples:

```
p {
   color: blue;
}

.title {
    text-align: center;
}

/* every  span inside an element with the class title */
.title  span {
 color: red;
}
```

Showing / Hiding

- Use the display:

```
h1 {
   dispaly: none;
}

.hide {
    display: none;
}
```

Selectors

Here is another example.

This selector is formally called the *Descendant combinator*

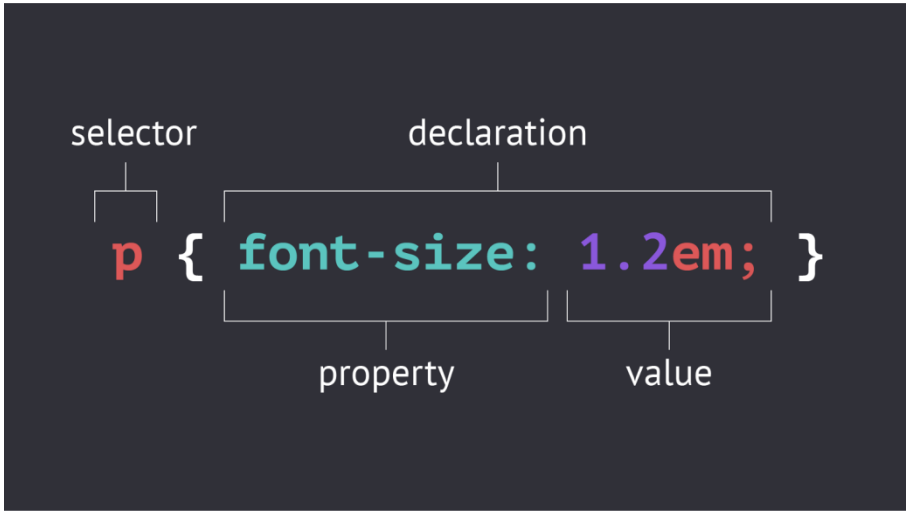
```
<a href="http://google.com" class="nice-link">
   <img src="pic1.jpg"  />
</a>
```

```
.nice-link img {
   border:10px ridge maroon;
}
```

Only images inside a .*nice-link* are affected.

Selectors

- Using selectors effectively is one of the keys for building professional level CSS



- We will later master those selectors
  - If you cannot wait, here is the List, and here is a good place to practice

Pseudo Classes

CSS has basic support for reacting to user interaction:

```
.nice-link{
    text-decoration:none;
    color: green;
}
.nice-link:hover{
    text-decoration:underline;
    color: red;
}
```

Link to somewhere

Link to somewhere

Using a CSS File

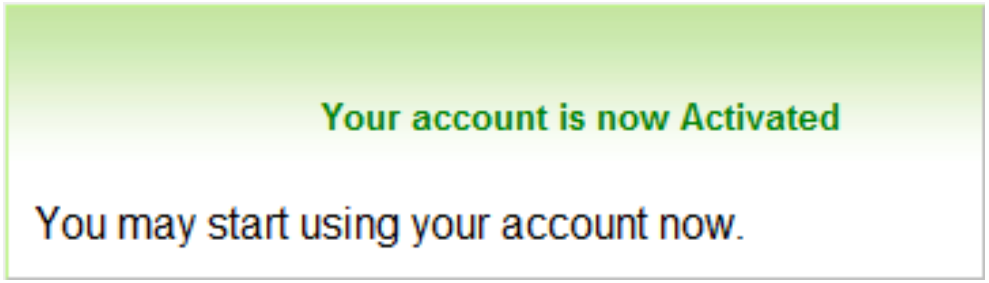
- Usually, we need to share the same styling across several pages
- So our CSS declarations should reside in a separate file - a CSS File.
- Here is how we reference the CSS file from the HTML:

```
<head>
    <link rel="stylesheet" href="main.css" />
</head>
```

*The * Element

- The * * element is commonly used to define a division (area) in the document:
- The is a block element.

```
<div class="nice-box">
    <h5>Your account is now Activated</h5>
    You may start using your account now.
</div>
```



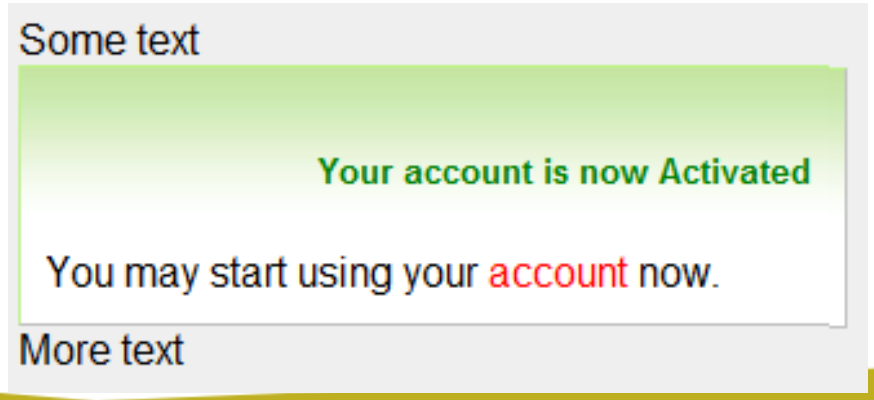

```
.nice-box {
    padding:10px;
    width:400px;
    border-top:1px #BEF488 solid;
}
.nice-box h5{
    text-align:center;
    color:green;
}
```

The <*span*> Element

The <*span*> element is used to define an **inline** span (area) in the document:

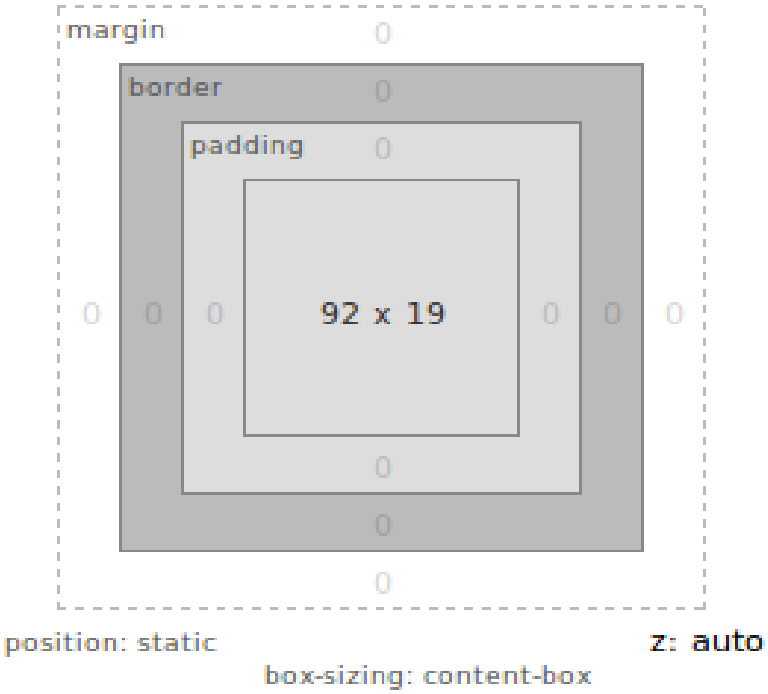
```
Some text
<div class="nice-box">
   <h5>Your account is now Activated</h5>
    You may start using your
    <span>account</span> now.
</div>
More text
```

```
.nice-box span {
   color:red;
}
```



Box Model

- The box model is consistent of:
- The content size (**width, height**)
- The **padding**
- The **border**
- And the **margin**



Victorious!

You have successfully grasped the basics of

HTML & CSS

Now lets dance

Javascript

In The Browser

Calling Functions from HTML

Very often, JS is used to handle user events:

```
<button onclick="doIt()" >Do It!<button>

<script>
function doIt() {
    if (confirm('Are you sure?')) {
        // Do it!
    }
}
</script>
```

Handling Events

- In the browser, Javascript is mainly used to react to events.
- Here are some examples:
  - The Page finished loading.
  - Mouse click.
  - Keystroke in a textbox.
  - Dragging, pinching…

Passing the event

Lets review the event object

```
<button onclick="doIt(event)" >Do
It!<button>

<script>
function doIt(ev) {
    console.log('Event:', ev);
}
</script>
```

The document

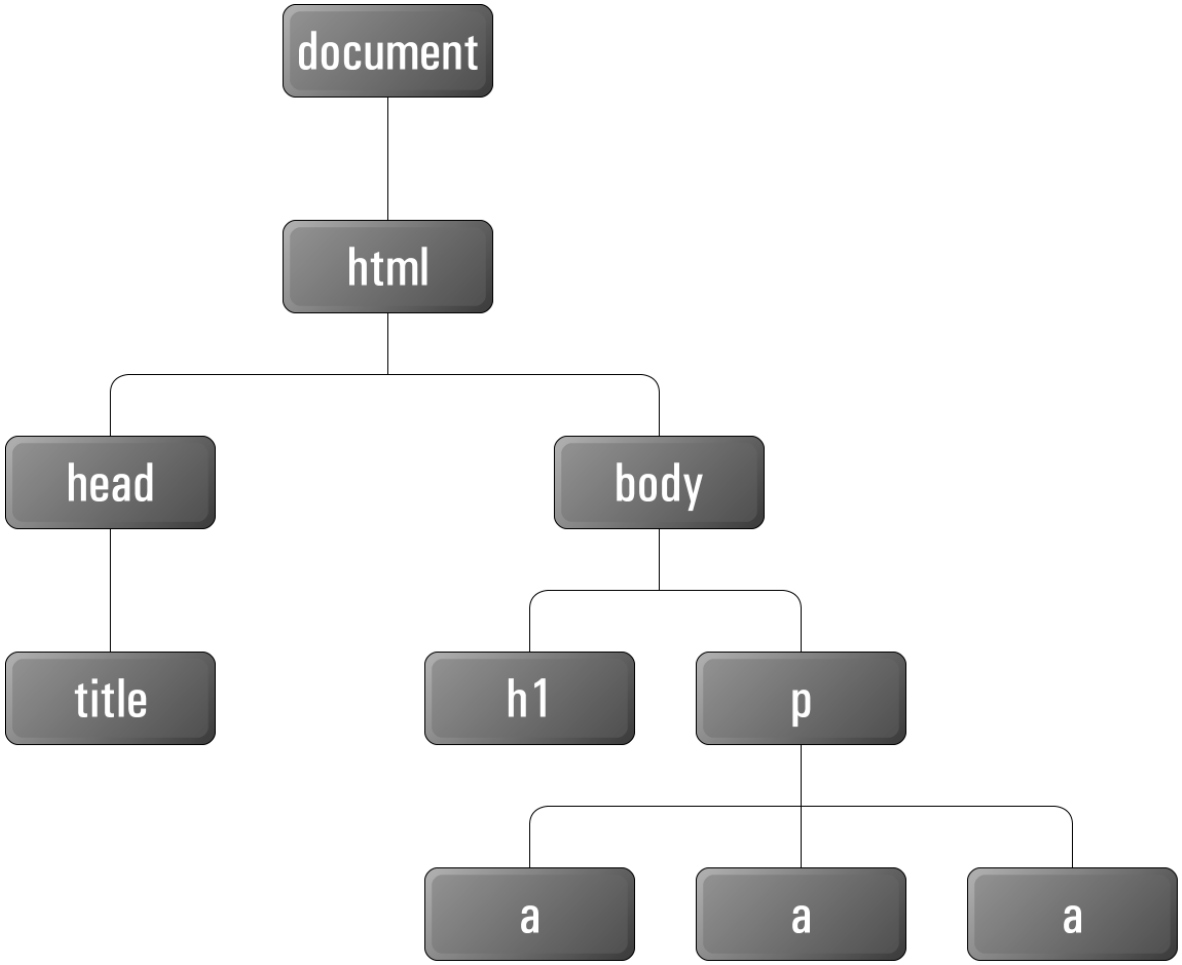
The document object provide access to the current document.

```
document.getElementById('myBox');
document.querySelector('#myBox');

document.querySelectorAll('span');
```

Its recommended to stick to querySelector

DOM – Document Object Model

The DOM is a tree representation of our HTML document.

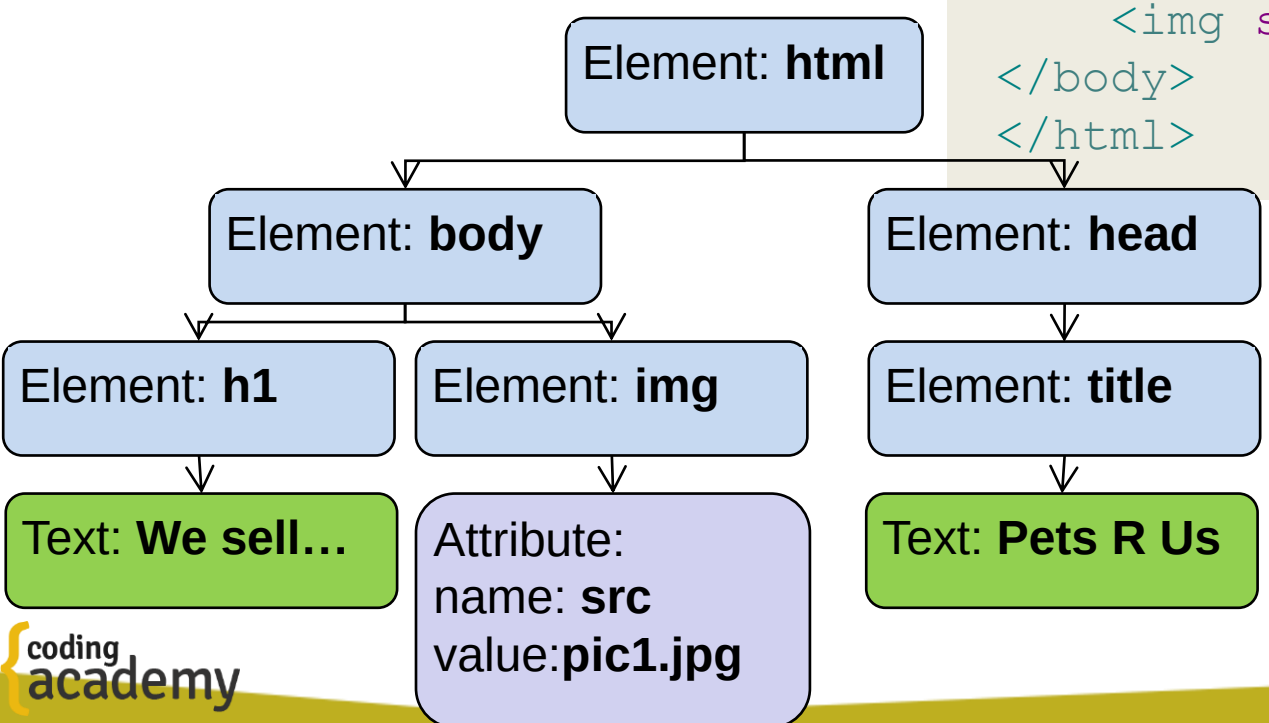


DOM – Document Object Model

The DOM is a tree of objects representing each element in the page:

```
<html>
  <head>
    <title>Pets R Us</title>
  </head>
  <body>
    <h1>We sell Pets</h1>
    <img src="pic1.jpg"/>
</body>
</html>
```

- It is composed from nodes:



innerText

We can get a hold of a DOM element and manipulate it:

```
var elBox = document.querySelector('.box');
elBox.innerText = 'Hello!'
```

Using the innerText property you may alter the textual content of an element.

innerHTML

Using the innerHTML attribute you may alter the HTML content of an element.

- So we can add new elements on the fly!

```
function changeLink(){
    var elMyLink = document.querySelector('.myLink')
    elMyLink.innerHTML='<img src="pic1.jpg"/>';
}


<a class="myLink" href="http://g.com"> Go Search</a>
```

Show / Hide element

Its very common to hide / show an element:

```
var elMsg = document.querySelector(‘.msg');

// HIDE:
elMsg.style.display = ‘none’;
// REMOVE THE HIDE (SHOW):
elMsg.style.display = ‘’;

// And sometimes:
elMsg.style.display = ‘block’;

// Also…
elMsg.hidden = true;
```

Adding / Removing Classes

Its very common to manipulate the display of elements by using classes:

```
var elBox = document.querySelector('.box');
var elMsg = document.querySelector('.msg');

if (elBox.classList.contains('selected')) {
   elMsg.classList.add('success')
   elBox.classList.remove('selected’)

}

 elBox.classList.toggle(‘match’)
```

DOM Subtrees

Note that every node in the tree has a querySelector function

- That will search for matching elements only in that subtree

```
var elBox = document.querySelector('.box');
var elP = elBox.querySelector('p')
elP.innerText = 'Hello!'

// Same as:
var elP = document.querySelector('.box p');
elP.innerText = 'Hello!'
```

Passing this

It is sometimes useful to send to the event handler function a reference to the evented element :

```
<button onclick="send(this)">Send</button>

function send(elBtn) {
    elBtn.innerText = 'Sending...'
}
```

HTML DOM Objects

- Every DOM element is a node object in the DOM tree
  - So they have common properties such as: id, class, parentElement and functions
- But **each type** of DOM element has its own unique functions
- For examples:
  - **<img>** has an src property
  - **<a>** has an href property
  - **<input>** has a value property

Racing Cars

Lest work with the following code:

```
var gCars = [{ id: 1, distance:0, speed: 10 },
             { id: 2, distance: 0, speed: 15 }];

function renderCars(cars) {
    var strHTML = ''
    for (var i = 0; i < gCars.length; i++) {
        strHTML += '<div class="car car' + (i + 1) + '"
                    onclick="speedUp('+i+')"></div>';
    }
    var elRoad = document.querySelector('.road');
    elRoad.innerHTML = strHTML;
}
```

Move them using margin-left

DOM – node's properties

- Every node in the DOM tree supports the following attributes:
  - e.parentNode - the parent node of e.
  - e.childNodes - the child nodes of e.
  - e.attributes - the attributes nodes of e.
  - e.innerHTML - the inner text value of e.
  - e.nodeName – read-only, the name of e.
    - For element – the tag name.
    - For attribute – the attribute name.
    - For text - #text.
  - e.nodeValue - the value of e.
    - For element – undefined.
    - For attribute – the attribute value.
    - For text – the text itself.
  - e.nodeType
    - The most useful types: 1 – element, 2 – attribute, 3 – text.

DOM – node's functions

- Every node in the DOM tree supports the following methods:
  - e.querySelector(selector)
  - e.getElementById(id) - get the element with a specified id under e.
  - e.getElementsByTagName(name) – get all elements of a specified tag under e.
  - e.appendChild(node) – adds a child node.
  - e.removeChild(node) – removes a child node.

Use *console.dir(el)* to examine the element

Data- attributes

Sometimes its useful to keep some of our model's **data** on the DOM elements, this is done by setting "data-" attributes:

Use

```
function foo(el){
    var inside = el.getAttribute('data-inside'); //17
    var data = el.dataset; // {inside: 17}
}


<div class="box" data-inside="17" onclick="foo(this)">
    See what in Me!
</div>
```

Body onload

Access the document only after it is loaded

```
<body onload="init()">
```

Handling Events

- Have a look at the following HTML code:

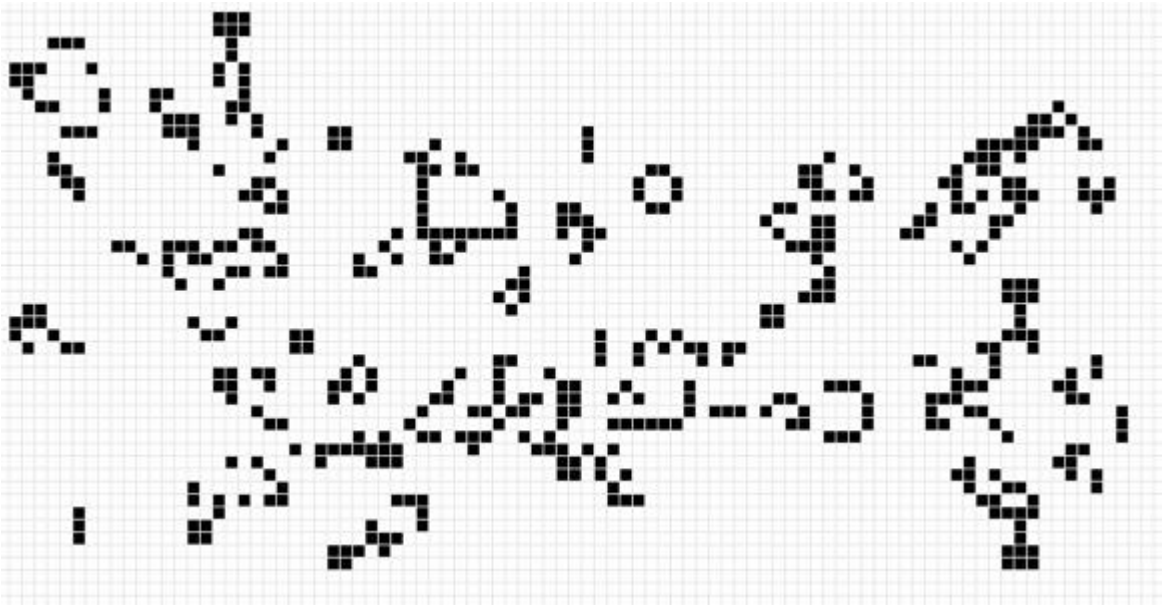
```
<body onload="init()">
   Your name: <input type="text"  id="userName" onkeyup="echoInput()" />
   <input type="button" value="Do It!" onclick="confirmAndDo()" />
   <div id="echoArea" style="color:green" ></div>
</body>
```

- It is safe to refer to elements only after the document has loaded, so we used the **onload** event to focus on an element.

```
function init () {
   var elUserName = document.querySelector("#userName");
   elUserName.focus();
}

function echoInput() {
   var elUserName = document.querySelector("#userName");
   var elEchoArea = document.querySelector("#echoArea");

   elEchoArea.innerHTML = elUserName.value;
}
```

Matrix as a <table>

Lets display Game of Life in an HTML Table



Web Board Games

Render a game board

We may use HTML Table to present a game board that is described in a matrix. Lets review:

```
function renderBoard(board) {
  var elBoard = document.querySelector('.board');
  var strHTML = '';
  for (var i = 0; i < board.length; i++) {
    strHTML += '<tr>\n';
    for (var j = 0; j < board[0].length; j++) {
      var currCell = board[i][j];
      var cellClass = 'cell-' + i + '-' + j + ' ';

      strHTML += '\t<td class="cell ' + cellClass +
                 '" onclick="moveTo(' + i + ',' + j + ')" >\n';
      strHTML += currCell;
      strHTML += '\t</td>\n';
    }
    strHTML += '</tr>\n';
  }
elBoard.innerHTML = strHTML;
}
```

Summary

- Javascript is the one and only scripting language for the web.
- Use Javascript to:
  - Code HTML events,
  - Create a dynamic and responsive GUI,
  - Dynamically manipulate your HTML elements.
- HTML documents are available as DOM,
  defining a standard way to access and modify elements.

Victorious!

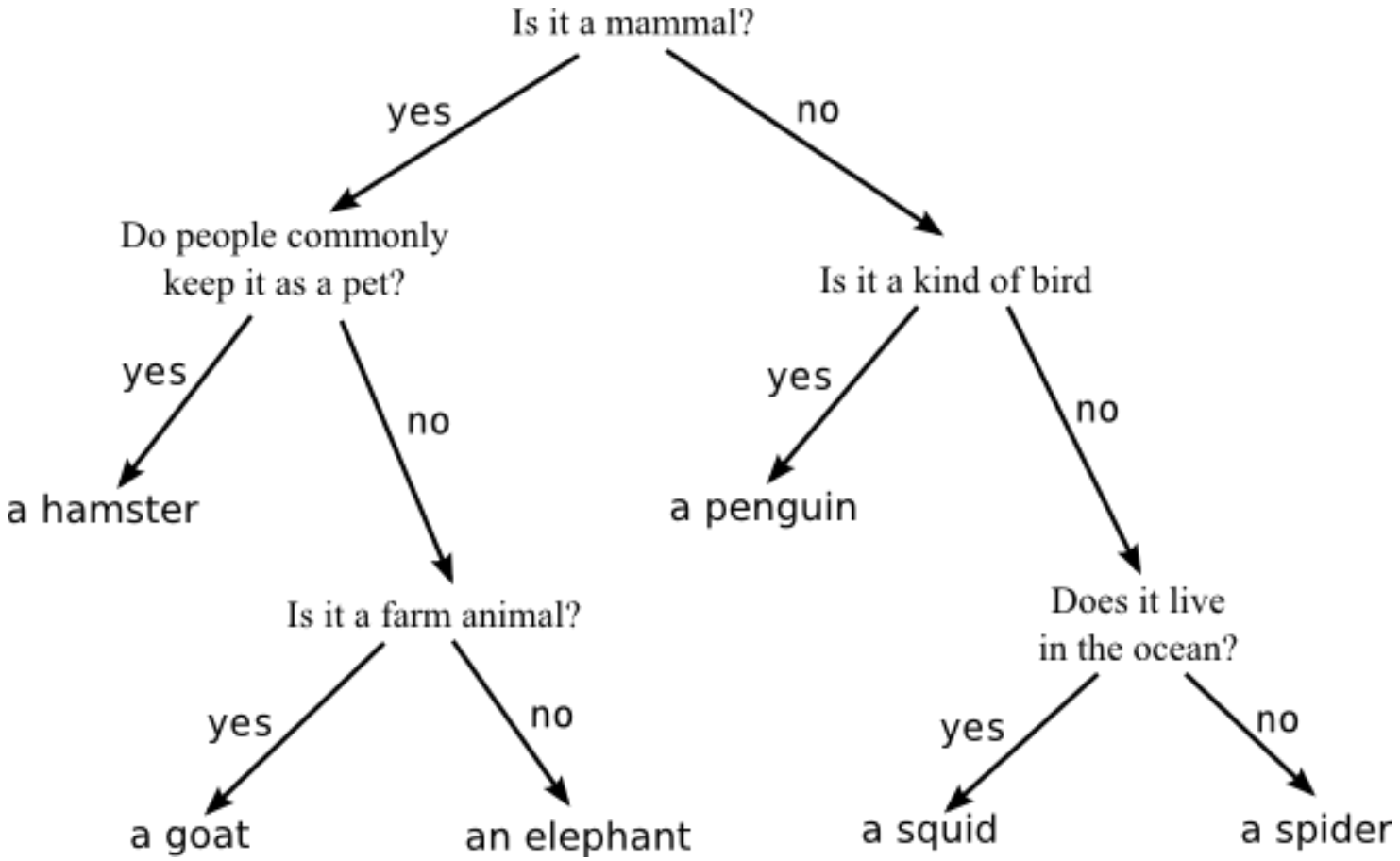
You have successfully grasped the basics of

HTML & CSS & JS in the Browser!

Now lets build something great

Guess Me Game

Lets build a simple game, based on a tree




```
var quest = {
    txt: 'a goat',
    yes : null,
    no : null
};
```