Università della Svizzera italiana

Faculty of Informatics

# JavaScript and HTML5

Prof. Cesare Pautasso

http://www.pautasso.info

cesare.pautasso@usi.ch

@pautasso

# Object-Oriented JavaScript



# O-O Java vs. O-O JavaScript

| Java: Class-based | JavaScript: Prototype-based |
|---|---|
| Classes/Objects | Objects (only) |
| Class definitions + Constructors | Protoypes + Constructors |
| Objects created with new | Objects created with new |
| Inheritance of Classes | Inheritance using Prototypes |
| **Cannot change** class definitions at run time | Constructor/Prototype only give initial definition. Objects definition **can be changed** at run time |

# Properties

Object = flexible container of unordered collection of named properties (and methods)

```
var student = {}
```

In Java: `JSObject = Map<String, Object>`

Create an empty object

```
student.name = "Peggy";
student.university = "USI";
student.date_of_birth = new Date();
```

Populate its properties



# Methods

```
student.age = function() {
        return (new Date().getTime()) –
                this.date_of_birth.getTime();
}
```

Declare a method for the object

Use `this` to access the fields of the object

```
 if (student.age() > 18) { ... }
```

Call a method

# this

```
f(x);                    // function call
```

`this` = the global object

```
obj.m(x);                // method call
```

`this` = the object `obj`

```
new C();                 // constructor
```

`this` = the new object

```
onclick="m(this)"      // event handler
```

`this` = the DOM element on which the event occurs

# Constructors

Any function called with new becomes a constructor

```
function Person(name) {
   this.name = name;
   this.age = function() {...};
   return this; //not needed
}
//call the constructor
var me = new Person("Peggy");
me.age();
```

Constructor names typically begin with an uppercase letter

- this inside the constructor refers to the newly created object
- The constructor initializes the properties and the methods of the new object

# Constructors (!)

Only functions called with new become a constructor.

```
function Person(name) {
    this.name = name;
    this.age = function() {...};
    return this; //not needed
}
//call the function
var me = Person("Peggy");
me.age();
```

What happens if you **forget new**?

- `this` inside the function refers to the **global object**
- The function updates the properties and the methods of the **global object**

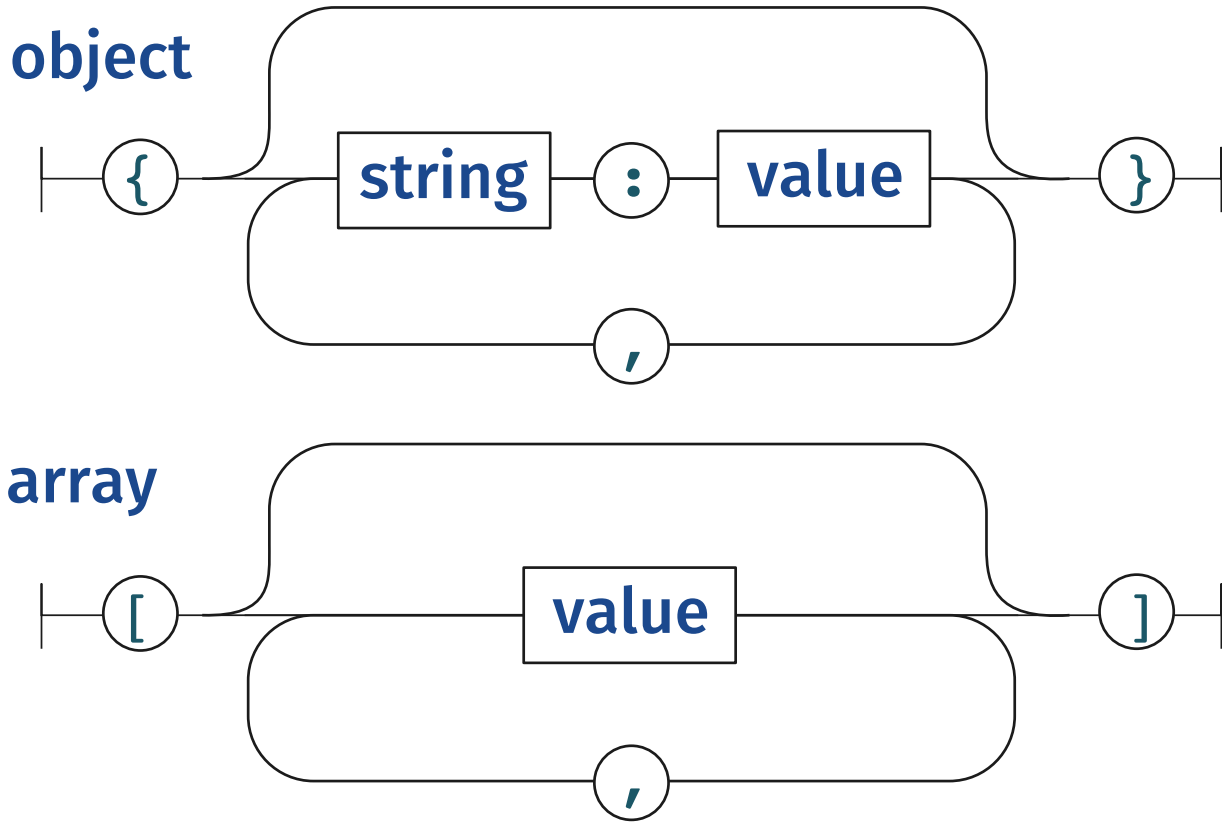
# Object Literals

```
var person = {
  name: "Peggy",
  date_of_birth: new Date(1927,0,31),
  address: {
    street: 'Via Buffi',
    number: 13
  }
}
```

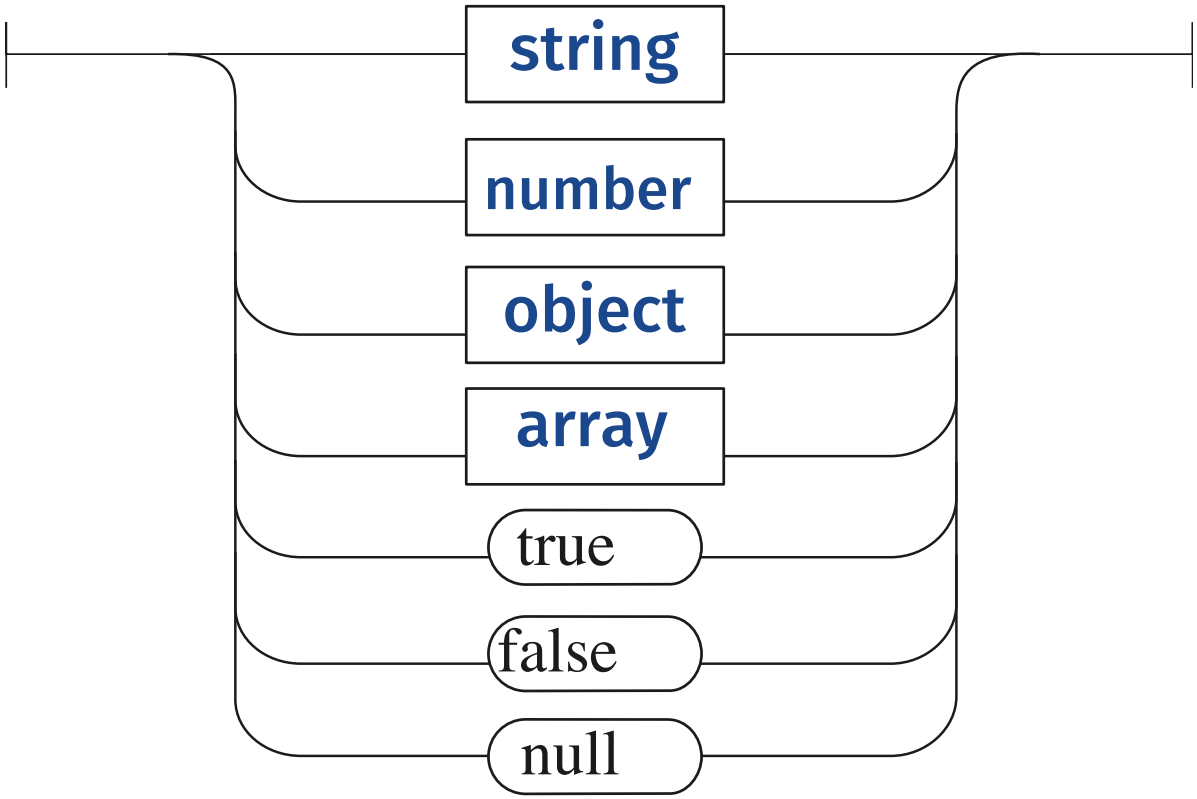
Objects can also be created directly using object literals following the JavaScript Object Notation (JSON)



# JSON Syntax



# value



# Constructor with Literals

```
function person(name, dob) {
  return {
    name: name,
    date_of_birth: dob,
    age: function() {
       return...
    }
  }
}

var p = person("Peggy", new Date(...));
```

lowercase function name

new not needed here!

Sometimes called **factory methods**



# Object Augmentation

```
person.height = "120cm";
person.hello = function() { ... };
```

You can add properties and methods to an object even **after** it has been created

```
delete person.height;
```

Properties and methods can be removed from an object with the `delete` operator

# Testing Object Fields

Never assume an object's field has been defined

```
var o = {};
if (o) { //o is defined
}
o.field = x;
if (o.field) { // o.field is defined
}
if (o && o.field) {
// both o and o.field are defined
}
```

# Parasitic Inheritance

```
function person(name, dob) {
  return {
    name: name,
    date_of_birth: dob,
    age: function(){...}
  };
}
function student(name, dob, uni) {
  var that = person(name, dob);
  that.uni = uni;
  that.canEnroll = function() {...};
  return that;
}
```

1. return a new JSON object

2. Create the “super” object

*3. Augment it*

*new not needed here!*

```
var s = student(“Peggy”, new Date(...), “USI”);
if (s.age() > 18 && s.canEnroll() ) { ... }
```

# Object Composition

```
function person(name, dob) {
  return {
    name: name,
    date_of_birth: dob,
    age: function(){...}
  };
}
function student(name, dob, uni) {
  var that = {};
  that.person = person(name, dob);
  that.uni = uni;
  that.canEnroll = function() {...};
  return that;
}
var s = student(“Peggy”, new Date(...), “USI”);
//s = {person: {name: "Peggy", date_of_birth: , age: },
//     uni: "USI", canEnroll: }
```

# Power Constructors

```
function class(a,b) {
  // initialize the object from the superclass
  var that = superclass(a);
  // declare private properties
  var private_c;
  // declare private methods
  function private_method() {...}
  // declare public properties
  that.public_d = b;
  // declare public methods
  that.public_method = function(p) {
    this.public_d ... ;
    private_c;
    private_method();
  }
  return that;
}
```

# Namespaces

```
var namespace = {};
```

Define a prefix to protect all your objects and functions so that their names do not conflict with others

```
namespace.o = {};
namespace.f = function() {};
```

Everything within the namespace is always visible. Use packages to separate the external public interface from the private implementation

# Packages

```
var package = function () {
  // declare private properties
  var private_c;
  // declare private methods
  function private_method() {...}
  // declare public interface
  return {
    //declare public properties
    public_d: 100,
    // declare public methods
    public_method: function(p) {
      var x = p || public_d;
      private_c;
      private_method();
    }
  }
} ();
```

Syntax reminder:

```
var x = function()
{}();
```

Warning: remember to call the function to instantiate the package

# Dynamic HTML



## What is dynamic HTML?

Manipulate an HTML document from the JavaScript code

- Add new elements
- Remove existing elements
- Change the position of elements in the tree
- Modify element content (`innerHTML`)
- Control the element CSS style (formatting, visibility, position, layout)
- Respond to user events

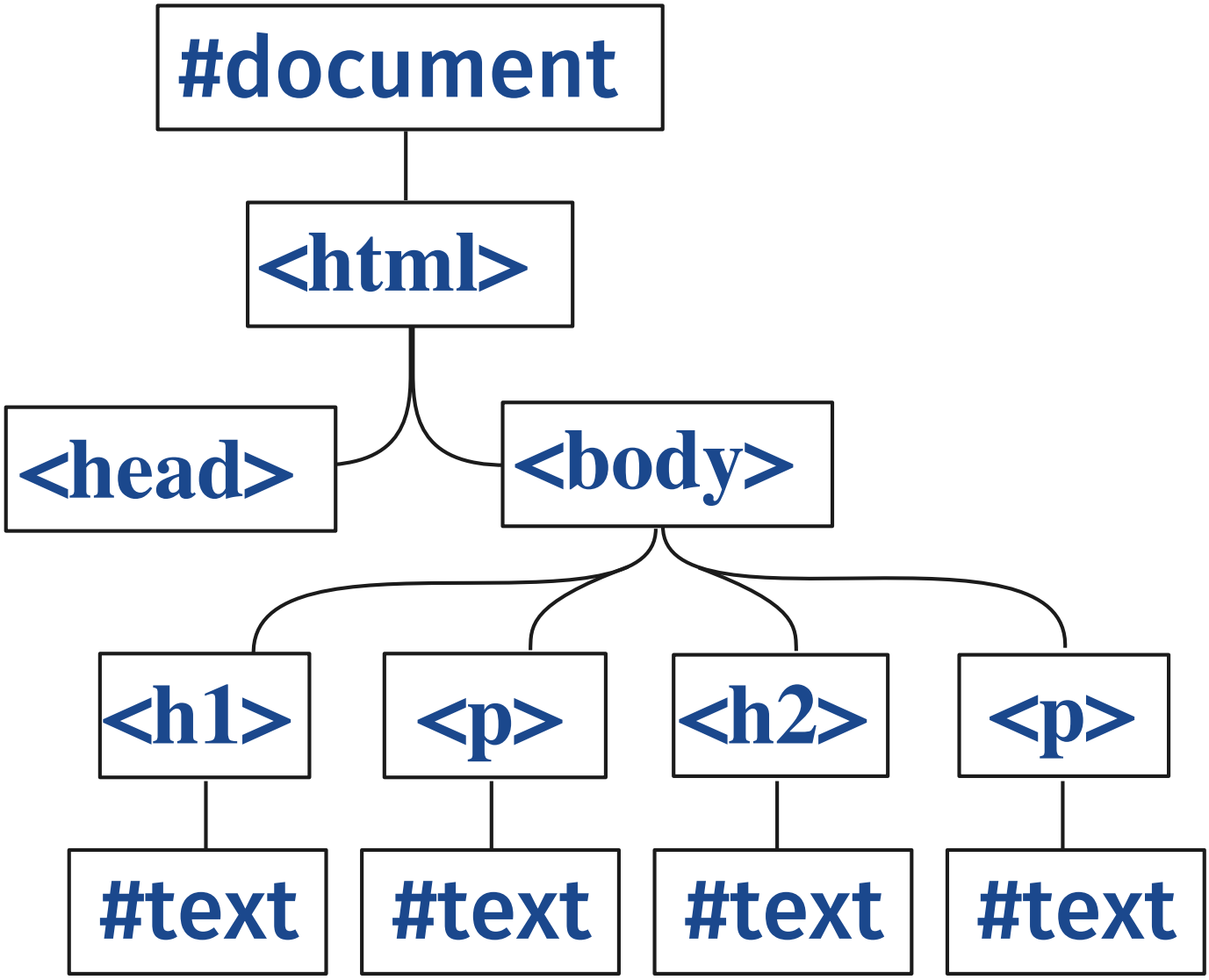
# DOM

## Document Object Model

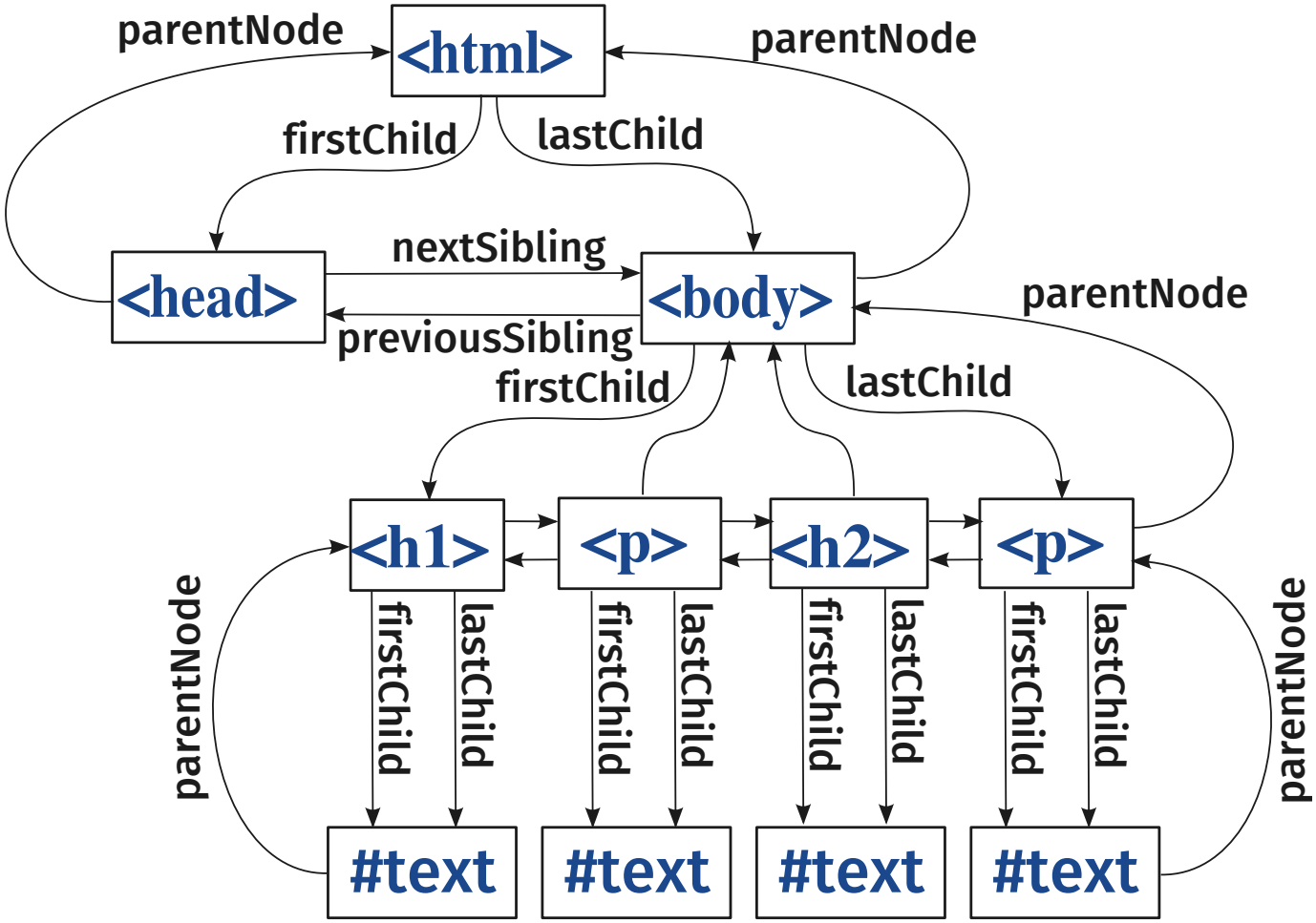
```
<html>
  <head></head>
  <body>
    <h1>Heading 1</h1>
    <p>Paragraph.</p>
    <h2>Heading 2</h2>
    <p>Paragraph.</p>
  </body>
</html>
```

This HTML string is parsed by the browser and represented as objects that can be accessed from JavaScript code

# DOM Tree



# Traversing the DOM






# Traversing the DOM

```
document //root of the tree
document.body //the body element node
document.body.firstChild
document.body.lastChild
document.body.childNodes[] // array of children
```

Navigate through the DOM tree

# Finding DOM Elements

```
document.getElementById('section1');
document.getElementsByTagName('div');
```

Direct, random-access to specific elements

```
document.getElementsByClassName("section");
document.querySelector("#section1");
document.querySelectorAll("div");
document.querySelectorAll(".section");
document.querySelectorAll("table > tr > td");
```

New HTML5 API (Use CSS selectors in JavaScript)



# Creating DOM Elements

```
document.createElement(tagName)
document.createTextNode(text)

//clone the element
node.cloneNode()

//clone the element and its descendents
node.cloneNode(true)
```

The newly created elements are not connected to the document tree

# Adding elements to the tree

```
//Add neu as the lastChild of node
node.appendChild(neu)

//Add to the children of node before sibling.
node.insertBefore(neu, sibling)
node.insertBefore(neu, node.firstChild)

// Swap the old child element with neu.
node.replaceChild(neu, old)
old.parentNode.replaceChild(neu, old)
```



# Removing DOM elements

```
//remove the old child node and return it
node.removeChild(old)

//remove the old node itself
old.parentNode.removeChild(old)
```

# Working with CSS Styles

```
//read-write the style class of a node element
node.className

//for multi-class elements (HTML5)
node.classList

//low-level access to the style properties
node.style.property
```

CSS properties map 1:1 with JavaScript properties (except property names that contain “-”. z-index → zIndex, background-color → backgroundColor, etc.)

# DOM Node Properties

```
N.nodeName
N.attributes
N.id
N.name
N.className
N.classList
N.style
N.innerHTML
N.textContent
```

## DOM Tree Traversal

```
N.childNodes
N.firstChild
N.lastChild
N.nextSibling
N.ownerDocument
N.parentNode
N.previousSibling
```

---

https://developer.mozilla.org/en-US/docs/DOM/Node

## Event Listeners

```
//add or remove event listeners
N.addEventListener(type, listener, capture)
N.removeEventListener(type, listener, capture)
```

- type (string identifying event without the on prefix)
- listener (call-back function triggered by the event)
- capture (true, will prevent other nodes to receive the same event)

```
//more primitive
N.onclick = listener
```

# Examples (with anonymous listener):

```
document.addEventListener("click",
   function(event) { alert(event); }, false);
document.onclick = function(event) { ... }
```



## innerHTML

```
N.innerHTML = '<b>Text</b>';
```

## Access the HTML Parser

```
//prepare the content
content = ...
//add it to the document (once)
N.innerHTML = content;
```

Construct the content first and later add it to the document

once to minimize page redraw operations by the browser



# HTML5 JavaScript



- Canvas
- GeoLocation
- Local Storage
- Web Workers
- Drag and Drop

# canvas

## Draw on the page from JavaScript code

```
<canvas id="cid" width="640" height="480"></canvas>
```

Define a canvas element for drawing

```
var c=document.getElementById("cid");
var ctx=c.getContext("2d");
```

Get access to the 2d drawing context

```
ctx.fillStyle
ctx.strokeStyle
ctx.rect()
ctx.fillRect()
ctx.arc()
ctx.moveTo()
ctx.lineTo()
ctx.fillText()
ctx.drawImage()
ctx.getImageData()
```

---

More information: https://developer.mozilla.org/en-US/docs/Canvas_tutorial

# localStorage

## Key-value persistent storage on the browser

```
localStorage.key = value; //value is a string
var value = localStorage.key;
if (localStorage.key) //check if key exists
```

```
localStorage.setItem('key',value);
var value = localStorage.getItem('key');
```

Alternative Syntax

## What if you want to store objects?

```
localStorage.key = JSON.stringify(object);
```

Convert the object into a JSON string before storing it

```
var object = JSON.parse(localStorage.key)
```

Parse the stored JSON string back into the object

## sessionStorage

Key-value **temporary** storage on the browser

```
sessionStorage.key = value; //value is a string
var value = sessionStorage.key;
if (sessionStorage.key) //check if key exists
```

Stored values are lost when the browser tab is closed, but will survive a page refresh

**Security Note**: `localStorage` and `sessionStorage` are implicitly scoped by the document origin. Pages downloaded from different websites cannot share data stored on the browser

# geolocation

Find out where the user is on the planet

```
navigator.geolocation.getCurrentPosition(
  function(position) {
         position.coords.latitude;
         position.coords.latitude;
   }, function(error) {
     // error.code can be:
     //   0: unknown error
     //   1: permission denied
     //   2: position unavailable
     //   3: timed out
   });
```

The first callback is called when the position has been computed. The second error handling callback is optional.

```
navigator.geolocation.watchPosition(function(position) {
  //track the position as it changes
});
```

# Web Workers

main.js

```
var worker = new Worker('worker.js');
worker.onmessage = function(event) {
    console.log(event.data);
}
worker.postMessage('start');
```

worker.js

```
self.onmessage = function(event) {
        //do some work
        var data; //output
        self.postMessage(data);
}
```

## Useful links

- Modernizr (http://modernizr.com/) (Browser Detection)
- JS Lint (http://www.JSLint.com/) (Style checker)
- Bootstrap (http://twitter.github.com/bootstrap/) (Predefined template)
- HTML5 Boilerplate (http://html5boilerplate.com/) (Predefined template)
- Web Platform Reference (http://platform.html5.org/)
- HTML5 Landscape Overview (http://dret.typepad.com/dretblog/html5-api-overview.html)
- HTML5 Validator (http://html5.validator.nu/)
- HTML5 & CSS3 Readiness (http://html5readiness.com/)

# References


- Douglas Crockford, JavaScript: The Good Parts, O'Reilly, May 2008
- Danny Goodman, Michael Morrison, JavaScript Bible, 6th Edition, Wiley, April 2007
- David Flanagan, JavaScript: The Definitive Guide, Fifth Edition, O'Reilly, August 2006
- Jeremy Keith, HTML5 for Web Designers, A Book Apart, 2010
- Mark Pilgrim, Dive into HTML5 ()