

JavaScript

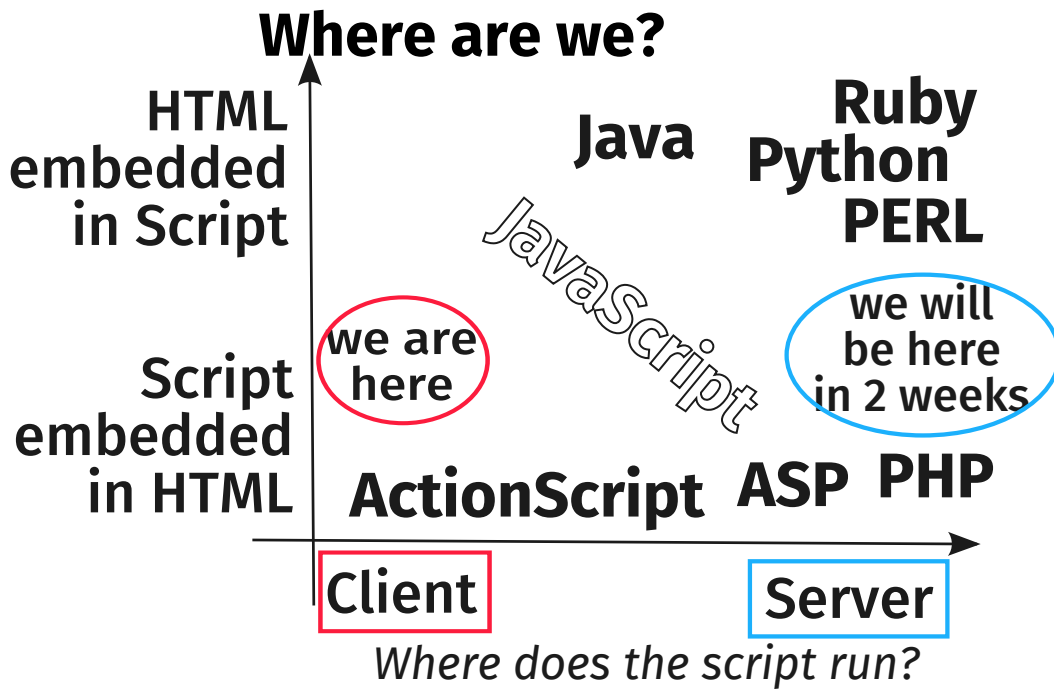
Prof. Cesare Pautasso

<http://www.pautasso.info>

cesare.pautasso@usi.ch

[@pautasso](#)

3 / 39



History

- Sun: Java (1994, Applets running in a Web browser)
- Netscape: Mocha/LiveScript/JavaScript (Navigator 2.0B3 - December 1995)
- Microsoft: JScript (IE3.0 August 1996)
- ECMA Standardization: ECMAScript (1.0 - 1998)
- JavaScript "serious" language since IE6, Firefox1
- Upcoming **ES6 Harmony** release (August 2014 feature freeze)

Scripting, What for?

- Add behavior to static content
- (X)HTML + CSS declarative languages for page/text layout description only
- Dynamically generate Web pages
- Modify the content of a Web page on the fly
- Run Web applications on the browser
 - Minimize latency with user interaction
 - Immediate feedback with input form validation
 - Immediate reaction to user events
 - Control the browser
(history, window, popups, statusbar)

Hello World

```
<script>
  document.body.innerHTML = "Hello World";
</script>
```

1. When does this code run?
2. What is document?
3. Where do you see the output?

9 / 39

When does it run?

1. Immediate Execution

- During page load (<body><script>)
- In the console

2. Event Handler

- Respond to user actions
 - Mouse, Keyboard, Forms
 - Browser Navigation
- Timeout events
- Page load event
- HTTP request events
- Web Workers

Events

Mouse

```
onClick  
ondblclick  
onmousedown  
onmousemove  
onmouseout  
onmouseover  
onmouseup
```

Page

```
onload  
onunload  
onresize
```

Touch

```
ontouchstart  
ontouchmove  
ontouchend
```

Input

```
onblur  
onchange  
onfocus  
onkeydown  
onkeypress  
onkeyup  
onreset  
onsubmit
```

Popup Message

```
<body>
  <h1 onmouseup="alert('Hello World 2')">
    Click Me
  </h1>
  <form onsubmit="alert('Hello World')">
    <input type="submit" value="Click Me"/>
  </form>
</body>
```

alert() blocks the browser and annoys the user (DO NOT USE)

12 / 39

A better Hello World

```
<html>
  <head>
    <script>
      function start() {
        document.body.innerHTML = "Hello World";
      }
    </script>
  </head>
  <body onload="start()">
    ...
  </body>
</html>
```

- Runs after the whole page has been loaded

Functional JavaScript

Functions

```
function name(parameters) {  
    var x;  
    ...  
    return [expression];  
}
```

Function Scoping Variables defined inside a function are not visible outside of the function

```
return; = return undefined;
```

Functions - Example

```
<html>
  <head>
    <script>
      function product(a,b) {
        return a*b;
      }
    </script>
  </head>
  <body>
    <script>
      document.body.innerHTML = product(6,7);
    </script>
  </body>
</html>
```

Optional Arguments

```
function f(a,b,c) {
  if (a===undefined) { a = 1 };
  if (!b) { b = 1 };
  c = c || 1;
  return a+b*c;
}
var y = f(1,2,3);    //y = 7
var x = f(1);       //x = 2
var z = f(1,2,3,4); //z = 7
```

Always test if parameters are defined and set them to default values if they are not (defensive programming).

Note: Use the first expression with `===` to distinguish undefined from `null`, `"`, `0`, `false`, since they all evaluate to `false` in the second expression.

Counting Arguments

```
function f(a,b,c) {
  var actual = arguments.length;
  var d = arguments[3];
  return a*b+c-d;
}
var expected = f.length; //3

var z = f(1,2,3,4);    //z = 1
```

You can check how many arguments a function expects (`f.length`) and how many arguments it actually gets (`arguments.length`).

Warning: out-of-bounds arrays simply return undefined.

Anonymous Functions

```
function f(x) { return 42 + x; };  
var y = f(z);
```

Named Function Declaration

```
var f = function(x) { return 42 + x; };  
var y = f(z);
```

Anonymous Function Expression Assigned to Variable

```
var f = function f(x) { return 42 + x; };  
var y = f(z);
```

Named Function Expression Assigned to Variable

```
var y = function(x) { return 42 + x; } (z);
```

Assign Result of Anonymous Function Call

More information: [Function Declarations vs. Function Expressions](#)

Functional Parameters

```
function map(f, a) {
  var result = new Array;
  for (var i = 0; i != a.length; i++)
    result[i] = f(a[i]);
  return result;
}
var square = function(x) {
  return x * x;
}
var s = map(square, [0,1,2,3,4,5]);
//s = [0,1,4,9,16,25]
```

Functions can be passed as parameters

Inner Functions

```
function outer(X,Y) {
  var local = 'Result: ';
  var inner = function(K) {
    var z = X + Y + K;
    alert(local + z);
    return z;
  }
  return inner(1);
}
var a = outer(-1,100); //a = 100
```

- Basic Scope Rule: An inner function has access to the variables and parameters of all functions that it is contained within.

Closures

```
function outer(X,Y) {  
  var local = 'Result: '  
  var inner = function(K) {  
    var z = X + Y + K;  
    alert(local + z);  
    return z;  
  }  
  return inner;  
}  
  
var f = outer(-1,100);  
var a = f(1); //a = 100
```

- Basic Scope Rule: An inner function has access to the variables and parameters of all functions that it is contained within.
- Closures: The scope that an inner function enjoys continues even after the parent functions have returned.
- A closure is a function together with its context: a snapshot of all variables accessible from it.

Closures - Useful Example

```
function makeConverter(factor, offset) {
  offset = offset || 0;
  return function(input) {
    return ((offset+input)*factor).toFixed(2);
  }
}

var milesToKm = makeConverter(1.60936);
var poundsToKg = makeConverter(0.45460);
var F2C = makeConverter(0.5556, -32);

var km = milesToKm(10);    //16.09
var kg = poundsToKg(2.5); //1.14
var c = F2C(98);          //36.67
```

Adapted from: [Understanding JavaScript Closures](#)

Closures Simple Example

```
function outer(x) {
  var inner = function(y) { return x * y; }
  return inner;
}
var one = outer(1);
//one = function(y) { return 1 * y; }
var two = outer(2);
//two = function(y) { return 2 * y; }
var a = one(10); //a = 10
var b = two(10); //b = 20
```

- Multiple closures are created every time the outer function is called. Each has its own copy of the context.

Counting with Closures

```
function outer(z) {
  var x = z || 0;
  var inner = function() { x++; return x; }
  return inner;
}
var count = outer(1);
var a = count(); //a = 2
var b = count(); //b = 3
```

- Warning: The context within a closure can be modified.

Functional Closures

```
function outer(z) {  
  var x = 0;  
  return function() { x = z(x); return x; };  
}  
  
var inc = outer(function(y){return y+1;});  
var a = inc(); //a = 1  
var b = inc(); //b = 2  
  
var dec = outer(function(y){return y-1;});  
var c = dec(); //c = -1  
var d = dec(); //d = -2
```

Closures - Timeout Example

```
function fadeElement(id) {
  var dom = document.getElementById(id),
      level = 1;
  function step () {
    var h = level.toString(16);
    dom.style.backgroundColor = '#FFFF' + h + h;
    if (level < 15) {
      level += 1;
      setTimeout(step, 100);
    }
    setTimeout(step, 100);
  }
}
```

Why is this a bug?

```
var dom = [ /* DOM Elements */ ];
var N = dom.length
for (var i = 0; i < N; i++) {
  dom[i].onclick = function() {
    alert( 'You clicked on #' + i );
  };
}
```

Bug: no matter which element you click on, the alert message is always the same (You clicked on #N).

How to fix this?

Fixing the bug with a closure

```
function setupClick(num) {
  return function() {
    alert( 'You clicked on #' + num );
  };
}
function setupEventHandlers(dom) {
  for (var i = 0; i < dom.length; i++) {
    dom[i].onclick = setupClick(i);
  }
}
```


Compact Version

```
function setupEventHandlers(dom) {
  for (var i = 0; i < dom.length; i++) {
    dom[i].onclick = function(num) {
      return function() {
        alert( 'You clicked on #' + num );
      };
    } (i);
  }
}
```

1. Declare the outer anonymous function with parameters, which returns the event handler anonymous function using the parameter
2. Then then call the outer anonymous function passing the actual parameter

Scripting the Web Browser

innerHTML

Access the HTML Parser

A much better way to produce HTML from your JavaScript without breaking the existing page structure

```
<head>
  <script>
function getTime() {
  document.getElementById('clock').innerHTML =
    '<b>' + new Date() + '</b>';
}
  </script>
</head>
<body>
  <p>Time: <span id='clock'?</span> </p>
  <input type='button' onclick='getTime()'
    value='What time is it?'/>
</body>
```

eval

Access the Browser VM

```
eval(string)
```

Compile and execute a string of JavaScript code and return the result

This is what the browser does to run JavaScript code strings embedded in the HTML for running them

Warning: Can open your code to **security risks** if you do not fully trust the source of the code string

open

Control the Browser Windows

```
window.open('http://www.usi.ch', 'usi',  
            'width=400,height=400');
```

```
window.close();
```

```
history.back();
```

```
history.forward();
```

Warning: The last three functions may render the current page invalid and never return!

debugger

Jump into the debugger

```
if (something is wrong) {  
    debugger;  
}
```

The debugger statement can be used as a programmable breakpoint

Hint: Do not use in production code

console

Log and trace your code

```
console.log("message");
```

Print a message into the console

```
console.log(variable)
```

Structured dump of the content of a variable

```
console.time("bottleneck");  
//slow code  
console.timeEnd("bottleneck");
```

Profile your code bottlenecks

Tools

- [Firebug \(https://www.getfirebug.com/\)](https://www.getfirebug.com/) (Firefox debugger)
- [JS Lint \(http://www.JSLint.com/\)](http://www.JSLint.com/) (Style checker)
- [QUnit.js \(http://qunitjs.com/\)](http://qunitjs.com/) (Unit Testing)
- [JS Compress \(http://jscompress.com/\)](http://jscompress.com/) (Minifier)
- [Greasemonkey \(http://www.greasespot.net/\)](http://www.greasespot.net/)
- [JavaScript Shell \(http://www.squarefree.com/shell/shell.html\)](http://www.squarefree.com/shell/shell.html)

References

- Douglas Crockford, JavaScript: The Good Parts, O'Reilly, May 2008
- Marijn Haverbeke, [Eloquent JavaScript \(http://eloquentjavascript.net/\)](http://eloquentjavascript.net/) : A Modern Introduction to Programming, July 2007
- Danny Goodman, Michael Morrison, JavaScript Bible, 6th Edition, Wiley, April 2007
- David Flanagan, JavaScript: The Definitive Guide, Fifth Edition, O'Reilly, August 2006
- Mark Pilgrim, Greasemonkey Hacks, O'Reilly, 2006
- Stoyan Stefanov, JavaScript Patterns: Build Better Applications with Coding and Design Patterns, O'Reilly, 2010