COMP284 Scripting Languages
Lecture 14: JavaScript (Part 1)
Handouts (8 on 1)

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

JavaScript: Motivation

- PHP and Perl both allow us to create dynamic web pages
- In web applications, PHP and Perl code is executed on the web server (server-side scripting)
  - allows to use a website template that is instantiated using data stored in a database
  - 'business logic' is hidden from the user:
    the code of an application is not visible to the user/client;
    the user/client only has access to the HTML produced by the code
  - not ideal for interactive web applications:
    too slow to react and too much data needs to be transferred
  - operations that refer to the location of the user/client are difficult,
    for example, displaying the local time

```
echo date('H:i l, j F Y');
```

    displays the local time on the server not the local time for the user

JavaScript

- JavaScript is a language for client-side scripting
  - script code is embedded in a web page (as for PHP),
    but delivered to the client as part of the web page and
    executed by the user's web browser
    ⇝ code is visible to the user/client
  - allows for better interactivity as reaction time is improved and
    data exchange with the server can be minimised
  - a web browser may not support JavaScript or
    the user may have disallowed the execution of JavaScript code
  - different JavaScript engines may lead to different results, in particular,
    results not anticipated by the developer of JavaScript code
  - performance relies on the efficiency of the JavaScript engine and
    the client's computing power (not the server's)
  - operations that refer to the location of the client are easy:

```
document.write("Local time: " + (new Date).toString());
```

JavaScript: History

- originally developed by Brendan Eich at Netscape
  under the name Mocha
- first shipped together with Netscape browser in September 1995
  under the name LiveScript
- obtained its current name in December 1995 under a deal between
  Netscape and Sun Microsystems, the company behind Java,
  in December 1995
- does not have a particularly close relationship to Java,
  it mixes aspects of Java with aspects of PHP and Perl
  and its own peculiarities
- is a dialect of ECMAScript, a scripting language standardised in the
  ECMA-262 specification and ISO/IEC 16262 standard since June 1997
- other dialects include Microsoft's JScript and TypeScript and
  Adobe's ActionScript

Websites and Programming Languages

Website	Client-Side	Server-Side	Database
Google	JavaScript	C, C++, Go, Java, Python, ~~PHP~~	BigTable, MariaDB
Facebook	JavaScript	Hack, PHP, Python, C++, Java, . . .	MariaDB, MySQL, HBase Cassandra
YouTube	~~Flash~~, JavaScript	C, C++, Python, Java, Go	BigTable, MariaDB
Yahoo	JavaScript	PHP	MySQL, PostgreSQL
Amazon	JavaScript	Java, C++, Perl	Oracle Database
Wikipedia	JavaScript	PHP, Hack	MySQL, MariaDB
Twitter	JavaScript	C++, Java, Scala	MySQL
Bing	JavaScript	ASP.NET	MS SQL Server

Wikipedia Contributors: Programming languages used in most popular websites. Wikipedia, The Free Encyclopedia, 20 October 2017, at 11:28. http://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites [accessed 23 October 2017]

JavaScript: Hello World!

```
<html><head><title>Hello World</title></head>
<body>
<p>Our first JavaScript script</p>
<script type="text/javascript">
  document.writeln("<p><b>Hello␣World!</b></p>")
</script>
<noscript>
JavaScript not supported or disabled
</noscript>
</body></html>
```

- JavaScript code is enclosed between `<script>` and `</script>`
- Alternative HTML markup that is to be used in case JavaScript is not enabled or supported by the web browser, can be specified between `<noscript>` and `</noscript>`
- File must be stored in a directory accessible by the web server, for example `$HOME/public_html`, and be readable by the web server
- No particular file name extension is required

JavaScript scripts

- JavaScript scripts are embedded into HTML documents and are enclosed between `<script` and `</script>` tags
- A JavaScript script consists of one or more statements and comments
  ⇝ there is no need for a main function (or classes)
  - Statements do not have to end in a semi-colon but they can
    ⇝ stick to one convention in your code
  - Whitespace before and in-between statements is irrelevant
    (This does not mean it is irrelevant to someone reading your code)
  - One-line comments start with // and run to the end of the line
  - Multi-line comments are enclosed in /* and */
  - Comments should precede the code they are referring to

Types

- JavaScript is a loosely typed language — like PHP and Perl
- JavaScript distinguished five main types:
  - boolean – booleans
  - number – integers and floating-point numbers
  - string – strings
  - function – functions
  - object – objects (including arrays)
- Integers, floating-point numbers, and strings do not differ significantly from the corresponding Perl scalars, including the pecularities of single-quoted versus double-quoted strings
- JavaScript distinguishes between these five types including between the three primitive types boolean, number and string

Variables

- JavaScript variable names do not start with a particular character
- A JavaScript variable name may consist of letters, digits, the $ symbol, and underscore, but cannot start with a digit
  ⇝ you can still stick to the PHP and Perl 'convention' that (some) variable names start with a $ symbol
- JavaScript variable names are case sensitive

Variables

- Variables can be declared using one of the following statements:

```
var variable1, variable2, ...
var variable1 = value1, variable2 = value2, ...
```

  - The second statement also initialises the variables
  - Used inside a function definition, a declaration creates a local variable (only accessible within the function)
  - Used outside a function definition, a declaration creates a global variable
- A variable can be inialised without a declaration by assigning a value to it:

```
variable = value
```

  - Both inside and outside a function definition, initialising an undeclared variable creates a global variable
- Note: A declaration does not specify the type of a variable only assigning a value of a certain type gives a variable a type

Variables

- In JavaScript, the use of the value of a variable that is neither declared nor initialised will result in a reference error and script execution stops
- A declared but uninitialised variable has the default value undefined and has no specific type
- JavaScript automatically converts a value to the appropriate type as required by the operation applied to the value (type coercion)
- The value undefined is converted as follows:

Type	Default	Type	Default	Type	Default
bool	false	string	'undefined'	number	NaN

```
myVar1++              // reference error
var myVar2
myVar2++              // myVar2 has value NaN
var myVar3
myVar3 = myVar3 + '!' // myVar3 has value 'undefined!'
```

Assignments

- JavaScript uses the equality sign = for assignments

```
student_id = 200846369;
```

  As in PHP and Perl, this is an assignment expression
- The value of an assignment expression is the value assigned

```
b = (a = 0) + 1;  // a has value 0, b has value 1
```

- JavaScript supports most of the standard binary assignment operators:

Binary assignment	Equivalent assignment
var += *expr*	*var* = *var* + *expr*
var -= *expr*	*var* = *var* - *expr*
var *= *expr*	*var* = *var* * *expr*
var /= *expr*	*var* = *var* / *expr*
var %= *expr*	*var* = *var* % *expr*

  Note: **= is not supported

Constants

- Some JavaScript dialects allow the definition of constants using

```
const variable1 = value1, variable2 = value2, ...
```

  - defines one or more constants
  - constants follow the same scope rules as variables
- However, this construct is not supported by Internet Explorer 6–10 and does not have the desired effect in Safari before version 5.1.7 nor Opera before version 12

Values, Variables and Types

- string typeof *value*
  returns a string representation of the type of *value*

Boolean	"boolean"	Number	"number"
String	"string"	Object	"object"
undefined	"undefined"	null	"object"
NaN	"number"	Infinity	"number"

  Future versions of JavaScript may have an option to change typeof null to "null" (as in PHP)

```
document.writeln("Type of 23.0: " + typeof(23.0) + "<br />"
document.writeln("Type of \"23\": " + typeof("23") +"<br />"
var a
document.writeln("Type of a:    " + typeof(a) + "<br />"
```

```
Type of 23.0: number<br />
Type of "23": string<br />
Type of a:    undefined<br />
```

Typecasting

JavaScript provides several ways to explicitly type cast a value

- Apply an identity function of the target type to the value

"12" * 1	⇝	12	!!"1"	⇝	true
12 + ""	⇝	"12"	!!"0"	⇝	true
false + ""	⇝	"false"	!!""	⇝	false
[12,[3,4]] + ""	⇝	"12,3,4"	!!1	⇝	true
			[12,13] * 1	⇝	NaN
			[12] * 1	⇝	12

Typecasting

JavaScript provides several ways to explicitly type cast a value

- Wrap a value of a primitive type into an object
  ⇝ JavaScript has objects `Number`, `String`, and `Boolean` with unary constructors/wrappers for values of primitive types
  (JavaScript does not have classes but prototypical objects)

`Number("12")`	⇝	`12`	`Boolean("0")`	⇝	`true`
`String(12)`	⇝	`"12"`	`Boolean(1)`	⇝	`true`
`String(false)`	⇝	`"false"`	`Number(true)`	⇝	`1`

- Use parser functions `parseInt` or `parseFloat`

`parseInt("12")`	⇝	`12`	`parseFloat("2.5")`	⇝	`2.5`
`parseInt("2.5")`	⇝	`2`	`parseFloat("2.5e1")`	⇝	`25`
`parseInt("E52")`	⇝	`NaN`	`parseFloat("E5.2")`	⇝	`NaN`
`parseInt("␣42")`	⇝	`42`	`parseFloat("␣4.2")`	⇝	`4.2`
`parseInt("2014Mar")`	⇝	`2014`	`parseFloat("4.2end")`	⇝	`4.2`

Comparison operators

JavaScript distinguishes between (loose) equality `==`
and strict equality `===` in the same way as PHP:

expr1 `==` *expr2*	Equal	TRUE iff *expr1* is equal to *expr2* after type coercion
expr1 `!=` *expr2*	Not equal	TRUE iff *expr1* is not equal to *expr2* after type coercion

- When comparing a number and a string, the string is converted to a number
- When comparing with a boolean, the boolean is converted to 1 if `true` and to 0 if `false`
- If an object is compared with a number or string, JavaScript uses the `valueOf` and `toString` methods of the objects to produce a primitive value for the object
- If two objects are compared, then the equality test is true only if both refer to the same object

Comparison operators

JavaScript distinguishes between (loose) equality `==`
and strict equality `===` in the same way as PHP:

expr1 `===` *expr2*	Strictly equal	TRUE iff *expr1* is equal to *expr2*, and they are of the same type
expr1 `!==` *expr2*	Strictly not equal	TRUE iff *expr1* is not equal to *expr2*, or they are not of the same type

`"123" == 123`	⇝	`true`	`"123" === 123`	⇝	`false`
`"123" != 123`	⇝	`false`	`"123" !== 123`	⇝	`true`
`"1.23e2" == 123`	⇝	`true`	`1.23e2 === 123`	⇝	`false`
`"1.23e2" == "12.3e1"`	⇝	`false`	`"1.23e2" === "12.3e1"`	⇝	`false`
`5 == true`	⇝	`false`	`5 === true`	⇝	`false`

Comparison operators

JavaScript's comparison operators also applies type coercion to their operands and do so following the same rules as equality `==`:

expr1 `<` *expr2*	Less than	true iff *expr1* is strictly less than *expr2* after type coercion
expr1 `>` *expr2*	Greater than	true iff *expr1* is strictly greater than *expr2* after type coercion
expr1 `<=` *expr2*	Less than or equal to	true iff *expr1* is less than or equal to *expr2* after type coercion
expr1 `>=` *expr2*	Greater than or equal to	true iff *expr1* is greater than or equal to *expr2* after type coercion

`'35.5' > 35`	⇝	`true`	`'35.5' >= 35`	⇝	`true`
`'ABD' > 'ABC'`	⇝	`true`	`'ABD' >= 'ABC'`	⇝	`true`
`'1.23e2' > '12.3e1'`	⇝	`false`	`'1.23e2' >= '12.3e1'`	⇝	`false`
`"F1" < "G0"`	⇝	`true`	`"F1" <= "G0"`	⇝	`true`
`true > false`	⇝	`true`	`true >= false`	⇝	`true`
`5 > true`	⇝	`true`	`5 >= true`	⇝	`true`

Equality

Why do we care whether `5 == true` is true or false?
⇝ it influences how our scripts behave
⇝ it influences whether more complex objects are equal or not

PHP:

```
if (5) print("5 is true");
else   print("5 is not true");
print(" and ");
if (5 == true) print("5 is equal to true");
          else print("5 is not equal to true");
```

Output: `5 is true and 5 is equal to true`

JavaScript:

```
if (5) document.writeln("5 is true");
  else document.writeln("5 is not true")
document.writeln(" and ")
if (5 == true) document.writeln("5 is equal to true")
          else document.writeln("5 is not equal to true")
```

Output: `5 is true and 5 is not equal to true`

Equality

Why do we care whether `5 == true` is true or false?
⇝ it influences how our scripts behave
⇝ it influences whether more complex objects are equal or not

PHP:

```
$array3 = array("1.23e2",5);
$array4 = array("12.3e1",true);
if (($array3[1] == $array4[1]) && ($array3[2] == $array4[2]))
     print("The two arrays are equal");
else print("The two arrays are not equal");
```

Output: `The two arrays are equal`

JavaScript:

```
$array3 = ["1.23e2",5]
$array4 = ["12.3e1",true]
if (($array3[1] == $array4[1]) && ($array3[2] == $array4[2]))
     document.writeln("The two arrays are equal")
else document.writeln("The two arrays are not equal")
```

Output: `The two arrays are not equal`

Equality

Note: The way in which more complex data structures are compared also differs between PHP and JavaScript

PHP:

```
$array3 = array("1.23e2",5);
$array4 = array("12.3e1",true);
if ($array3 == $array4)
     print("The two arrays are equal");
else print("The two arrays are not equal");
```

Output: `The two arrays are equal`

JavaScript:

```
$array3 = ["1.23e2",5]
$array5 = ["1.23e2",5]
if ($array3 == $array5)
     document.writeln("The two arrays are equal")
else document.writeln("The two arrays are not equal")
```

Output: `The two arrays are not equal`

Revision

Read

- Chapter 14: Exploring JavaScript

of

R. Nixon:
Learning PHP, MySQL, and JavaScript.
O'Reilly, 2009.