5. JavaScript

Frank Stephan

February 5, 2014

JavaScript

- JavaScript is a scripting language mostly used in HTML pages and PDF documents.
- JavaScript is mostly used to program the client side of an application and runs in Web browser, on mobile devices, and in desktop widgets.
- JavaScript is a prototype-based scripting language that mixes imperative, object-oriented and functional programming styles.

Oral Explanation

The World Wide Web

Browser

Web Server

Oral Explanation

ECMAScript

In November 1996, Netscape announced that it had submitted JavaScript to Ecma International for consideration as an industry standard, and subsequent work resulted in the standardized version named ECMAScript. JavaScript was formalized in the ECMAScript language standard. "JavaScript" is a trademark of Oracle Corporation.

Oral Explanation

JavaScript in HTML

JavaScript can be directly embedded into an HTML page using the element `<script>`.

```
1  <HTML>
2  <HEAD><TITLE>JavaScript</TITLE></HEAD>
3  <BODY>
4  <H1>Time</H1>
5  <SCRIPT>
6  //This is a javaScript Program
7  document.write("<H2>Lord Byron</H2>");
8  </SCRIPT>
9  </BODY>
10 </HTML>
```

Load this HTML file into a browser | Oral Explanation

JavaScript in HTML

There are other ways to embed JavaScript in HTML:

- With the `src` attribute of the `<script>` element
- With the JavaScript hyperlink
- With an HTML event handler

```
1 <HTML>
2 <HEAD><TITLE>Button</TITLE></HEAD>
3 <BODY>
4 <SCRIPT SRC="js.js"> </SCRIPT>
5 <A HREF="javascript: alert('Oooch!');">Click Me</A>
6 <INPUT TYPE="button" VALUE="Click Me" ONCLICK="alert('
      Oooch!');">
7 </BODY>
8 </HTML>
```

Load this HTML file into a browser | Oral Explanation

Literal Values

JavaScript manipulates numbers, strings, Booleans, nulls, objects and functions. Primitive types are Boolean, numeric, strings. Complex types are array, object and function. JavaScript is weakly typed. That means that a variable has a type that is inferred from its value and that the type can change dynamically.

```
1 8
2 -1024
3 -3.1451516
4 "Hello World"
5 "<b>Hello World</b>, <i>I know HTML tags</i>!<br>"
6 true
7 false
8 null
9 undefined
```

Oral Explanation

Variables

Variables provide a means to store and manipulate values. A variable name can be any sequence of letters, digits, underscore and dollar sign. It cannot start with a digit.

```
1 var age;
2 var current_year = 2014;
3 var year_of_birth = 1990;
4 age = current_year - year_of_birth;
```

Javascript 1.5 and Unicode

Since Javascript 1.5, any Unicode letter can be used in a variable name. While this gives birth to interesting variable names, it is very impractical to write and maintain the code with most text editors. Do NOT do it!

Oral Explanation

Keywords and other Reserved Words

Keywords are special words that are interpreted by the JavaScript interpreter. Keywords and other reserved words cannot generally be used as identifiers for variables, functions and so on.

```
1 break case continue default delete do else
2 export false for function if import in new
3 null return switch this true typeof var void while with
```

```
1 abstract boolean byte catch char class const debugger
2 double enum extends final finally float goto implement
3 instanceof int interface long native package private
4 protected public short static super synchronized throw
5 throws transient try
```

Oral Explanation

Input and Output

JavaScript can input values from different widgets: forms, prompt, files and read the HTML.
JavaScript can output values in different widgets: the HTML, files, the console, buttons and canvas.

```
1  <HTML>
2  <HEAD><TITLE>JavaScript</TITLE></HEAD>
3  <BODY>
4  <H1 id="title">To Time</H1>
5  <SCRIPT>
6  document.write("<H2>Lord Byron</H2>");
7  alert("Lord Byron");
8  var newname= window.prompt("Give a new title", "");
9  document.getElementById("title").innerHTML = newname;
10 </SCRIPT>
11 </BODY>
12 </HTML>
```

Load this HTML file into a browser | Oral Explanation

Comments

Comments explain the code.

```
1 /* Comments open with a slash followed by a start
2 and close with a start followed by a slash.
3 They can span several lines or be embedded  in
4 a line of code */
5
6 // Single line comments open with a double slash
7 // and close with the line
```

Oral Explanation

Arithmetic Operators

- Addition: +
- Subtraction: -
- multiplication: *
- Division: /
- Modulus: %
- Increment: ++
- Decrement: --

```
1 var x = 39;
2 var y = x * x + x + 41;
3 document.write("<H1>Euler said that this number is
      prime: " + y +"<H1>");
```

Load the HTML into a browser Oral Explanation

The Math Object

Other mathematical constants and functions are available as properties and methods of the `Math` object: Euler's number, the natural logarithm of 2, absolute value, floor and ceiling, power, random number generation, etc.

```
1 var x = Math.pow(2,31) -1;
2 document.write("<H1>Euler said that this number is
      prime: " + x +"<H1>");
3 document.write("<H1>Euler's number is: " + Math.E +"<H1
      >");
4 document.write("<H1>This is a random number: " + Math.
      random() +"<H1>");
```

Load the HTML into a browser | Oral Explanation

Conditional Branching

The `if` statement allows conditional branching based on general conditions.

```
1 if (age <11) {category = 'child';}
2 else if (age <20) {category = 'teenager';}
3 else {category = 'adult';}
```

Conditional branching

The `switch` statement allows a conditional branching based on the value of a variable.

```
1 switch (age)
2 { case 0: {category = 'baby'; break;}
3   case 1: case 2: {category = 'toddler'; break;}
4   default: {category = 'child'; break;}}
```

Oral Explanation

Comparison Operators

- Equality: ==, ===
- Not equal to: !=
- Greater than: >
- Lower than: <
- Greater than or equal to: >=
- lower than or equal to: <=

Oral Explanation

Boolean Operators

- Conjunction: &&
- Disjunction: ||
- Negation: !

```
1 var x =  Math.random();
2 var y = Math.random();
3 if ((x <= 0.5 && y <= 0.5) || !(0.5 >= x || 0.5 >= y))
4   {document.write( "Tail");}
5 else {document.write( "Face");}
```

Load the HTML into a browser | Oral Explanation

Bounded and Conditional Iteration

The `for`, `while` and `do-while` loops iterate on a general condition. The `for` loop syntax simplifies the programming of the simple bounded iteration.

```
1  var i;
2  for (i=0;i<100; i++)
3  {document.write("<I> I shall write this first line a
       hundred times</I>.<br>"); }
4  i=0;
5  while (i < 100)
6  {document.write("<B> I shall write this second line a
       hundred times</B>.<br>"); i=i+1; }
7  i=0;
8  do
9  {document.write("<U> I shall write this third line a
       hundred times</U>.<br>"); i=i+1; }
10 while (i < 100)
```

Load the HTML into a browser | Oral Explanation

Jump

JavaScript also supports `break` `continue` statements and labels. It is generally considered that they should be avoided as much as possible (bad style, not structured).

Oral Explanation

Functions

Functions are subroutines that can be called. They have parameters to take input values when called. They return an output value. They can call themselves (recursively).

```
1 function fibonacci(n) {
2 var f;
3     if (n == 0){f = 0;}
4     else if (n == 1){f = 1;}
5     else  {f = fibonacci(n - 1) + fibonacci(n - 2);}
6     return f;
7 }
```

Load the HTML into a browser Oral Explanation

Objects

Objects have properties. The values of these properties can be accessed using the dot notation.

```
1  var myComputerScientist = {
2    firstName: "John",
3    lastName: "Von Neumann",
4    birth: 1903,
5    death: 1957};
6  document.write(myComputerScientist.lastName);
7
8  var myOrGate = {
9    input1: true,
10   input2: false,
11   output: true};
12 document.write(myOrGate.output);
```

Oral Explanation

Methods

Objects have methods. The methods can be invoked using the dot notation.

```
1 var myOrGate = {
2   input1: true,
3   input2: false,
4   output: function () {return this.input1 || this.
        input2;}};
5 document.write(myOrGate.output());
```

Oral Explanation

Constructors

Object of the same kind can be created with constructors.

```
1 function OrGate(input1, input2) {
2   this.input1 = input1;
3   this.input2 = input2;
4   this.output = function () {return this.input1 || this
        .input2;}};
5
6 var myOrGate = new OrGate(true, false);
7 document.write(myOrGate.output());
```

Oral Explanation

Arrays

Arrays are data structures that help organize data.

```
1  var myOrGate1 = [true, true, true];
2  document.write(myOrGate1[0]);
3  document.write(myOrGate1[1]);
4  document.write(myOrGate1[2]);
5  document.write(myOrGate1.length);
6
7  var myOrGate2 = new Array(false, true, true);
8
9  var myOrGate3 = new Array();
10 myOrGate3[input1] = false;
11 myOrGate3[input2] = false;
12 myOrGate3[output] = false;
```

Oral Explanation

Arrays

Arrays objects in JavaScript have several constructors and methods to manipulate them.

- `length` gives the length of the array.
- `sort()` sorts the elements of an array.
- See also `reverse()`, `slice()`, `splice()` etc.

Oral Explanation

Stack

An array can be manipulated as a stack.

- `pop()` removes the last element of an array, and returns that element.
- `push()` adds a new element to the end of an array.

Last In First Out (LIFO)

The resulting list is `[1,2,3,4,5]`

```
1 var A = new Array();
2 A.push(1); A.push(2); A.push(3); A.push(5); A.push(4);
3 var x = A.pop(); var y = A.pop();
4 A.push(x); A.push(y);
```

Oral Explanation

Queue

An array can be manipulated as a queue.

- `shift()` removes the first element of an array, and returns that element.
- `push()` adds a new element to the end of an array.
- See also `unshift()`.

First In First Out (FIFO)

The resulting list is `[1,2,3,4,5]`

```
1 var A = new Array();
2 A.push(0); A.push(1); A.push(2); A.push(3);
3 A.push(4); A.push(5);
4 var x = A.shift();
```

Oral Explanation

Multidimensional Arrays

Multidimensional arrays are arrays of arrays.

```
1 var A = [[1, 2],[4, 5],[7, 8],[7, 8]];
2 var B = [[1, 2, 3, 2],[3, 2, 1, 3]];
3 var R = new Array();
4 for (i=0;i<A.length;i++)
5   {R[i] = new Array();
6   for (j=0;j<B[0].length;j++)
7     {R[i][j] = 0;
8     for (k=0;k<B.length;k++)
9       {R[i][j] = R[i][j] + A[i][k]*B[k][j];}}}
```

Load the HTML into a browser Oral Explanation

References

Objects, arrays and functions are passed by reference (not by value).

Oral Explanation

```
1  function input (value) {
2    this.value = value;
3    this.output = function () {return this.value;}};
4
5  function OrGate(input1, input2) {
6    this.input1 = input1;
7    this.input2 = input2;
8    this.output = function () {return this.input1.output
         () || this.input2.output();}};
9
10 var v1 = new input(false);
11 var v2 = new input(true);
12 var myOrGate = new OrGate(v1, v2);
13 document.write(myOrGate.output() +"<BR>");
14 v2.value = false;
15 document.write(myOrGate.output()+"<BR>");
```

Load the HTML into a browser | Oral Explanation

Syntax Errors

The program must be written according to the rules of the language (keywords, order etc.).

```
1 {for (i = 0;i < 5)
2     {document.write("The number is: "+ i + "<BR>");}}
```

There must be three parts in the “for” statement.

```
1 {for (i = 0;i < 5;i = i+1)
2     {document.write("The number is: "+ i + "<BR>");}}
```

Oral Explanation

Type Errors

Discrepancies between the types of variables and the expected types of functions and operations.

```
1 var i = "hello";
2 i = Math.abs(i);
3 document.write("The number is: "+ i + "<BR>");
```

JavaScript is dynamically and weakly typed. It is flexible but does not help preventing typing errors.

Oral Explanation

Semantic Errors

A program with no syntax error may not do what was intended.

```
1 {for (i = 0;i = i+1;i < 5)
2     {document.write("The number is: "+ i + "<BR>");}}
```

"i = i+1" returns the value assigned to i also as the value of a condition. This value is interpreted as "true" if it is positive and as "false" if it is 0. So "i = i+1" is a syntactical legal condition. Furthermore, there are no constraints how the third condition updates i, anything is permitted in JavaScript.
Different programming languages are more or less permissive.

Oral Explanation

Bad Style

Bad style does not help track bugs.

```
1 for (k=10;k=k-1;document.write((10-k)+" "));
```

The loop above outputs 1 2 3 4 5 6 7 8 9

Oral Explanation

Infinite Loops

Program execution may not terminate! Sometimes by design.

```
1 {for (i = 0;i >= 0; i = i+1)
2     {document.write("Wait for me!");}}
```

```
1 {var something;
2 while (true)
3     {something = wait_for();
4     take_action(something);}}
```

Note: See JavaScript event handling.

Oral Explanation

Example: Prime Triple

Sometimes not: A Prime Triple is a triple of three prime numbers, p, $p+2$ and $p+4$.

```
1 var p=4;
2 while (! (isprime(p) && isprime(p+2) && isprime(p+4)))
3 {p = p+1;}
4 document.write("("+p+", "+(p+2)+", "+(p+4)+") is a
      prime triple. <BR>");
```

(3, 5, 7) is the only prime triple.

Oral Explanation

Most likely, the bug is in your code...

How to Deal with the Bug?

- **Prevent the bug**: follow disciplined, rigorous software engineering (design and development) approach (software analysis, software verification, correctness proofs);
- **Track the potential bug**: test the software;
- **Find the bug**: instrument the code (put print statements to trace the execution; you may have used defensive programming, design by contract or other software engineering approaches; use debugging tools, analysis tools and other software engineering tools);
- **Fix the bug**: remove the bugs;

Oral Explanation

Techniques for Analyzing Programs

Some techniques can be used by compilers, tools or programmers to analyze programs and eliminate bugs, such as:

- Syntax analysis
- Type checking
- Type inference
- Abstract interpretation
- Formal methods and code verification

Oral Explanation

"In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selections amongst them [...] One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation."
attributed to
Ada Lovelace (10 December 1815 - 27 November 1852), who helped Charles Babbage (26 December 1791 - 18 October 1871) with his machines.

Oral Explanation

Normalizing Scores

The following function normalizes a list of scores, for instance grades, by dividing every score by the maximum score.

```
1 function normalize_score(A)
2 {var max = 0;
3 for (i=0;i <= A.length-1; i = i+1)
4     {if (A[i] > max) {max = A[i]}}
5 for (i=0;i <= A.length-1; i = i+1)
6      {A[i] = A[i] / max * 100;}
7 return A;}
```

Oral Explanation

Normalizing Scores

The following function is (sometimes) faster.

```
1 function normalize_score(A)
2 {var max = 0; var factor;
3 var l = A.length -1;
4 for (i=0;i <= l; i++)
5     {if (A[i] > max) {max = A[i]}}
6 factor =  100/max;
7 for (i=0;i <= l; i++)
8      {A[i] *= factor;}
9 return A;}
```

Oral Explanation

How to Become a Good Programmer

Experience! Experience! Experience! (and Perseverance ...)

- Choose your language(s);
- Program;
- Read manuals, guides, tutorials and other articles;
- Follow and contribute to developers' forums.

Oral Explanation

Attribution

Some of the examples are adapted from Stanford's "CS101 - Introduction to Computing Principles" (`http://www.stanford.edu/class/cs101`) and W3 Schools JavaScript tutorial (`http://www.w3schools.com`).