Chapter 4
Operators, Expressions, and Statements

Adapted from
JavaScript: The Complete Reference 2nd Edition
by
Thomas Powell & Fritz Schneider

Statements

- JavaScript program is made up of statements
- Statements are separated by semi-colons (;) or returns
  - Semi-colons are favored since the scripts are “safer” particularly when whitespace is removed (‘crunched’)
- Statements are fairly whitespace agnostic between operators except for newlines

Groups of Statements

- Groups of statements enclosed within curly braces `{ }` are called **blocks**
- Blocks are often found with conditional and loop statements
  - `if (happy) { }` or `while (count < 100) { }`
- Always used as function bodies
  - `function doit(x,y) { }`
- Blocks may have their own variable scope as in the case of function bodies but otherwise share the global scope

Basic Operators

- Assignment (=)
  - Watch out for comparison (==) typo bug
- Basic Arithmetic
  - +, -, *, /, % (modulus)
- String concatenation also uses +
  - Watch for type conversion here!
- Arithmetic nuances
  - Any number (except 0) / 0 is infinity
  - 0/0 is NaN
  - Once infinite always infinite (infinity – 1 = infinity)

Bitwise Operators

- Used to manipulate bit strings. Turns any number provided to a 32bit integer and then performs a bitwise operation and then back to a 32 bit integer
- Consult the following truth table when using bitwise operators

First Bit	Second Bit	AND (&)	OR (|)	XOR (^)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise Operators Contd.

- Example
  - 3 = 00000011 5 = 00000101
  - 3 & 5 =

```
00000011
00000101
--------
00000001  -> 1
```

- The bitwise not operator (~) invert zeros to ones and ones to zeros which in effect creates a negative value in binary

Bitwise Operators Contd.

- Bitwise shift operators are also supported
  - << shift left and fill with zeros
  - >> shift right with sign (copy 0 or 1)
  - >>> shift right and fill with zero

- Generally you will not find many practical uses of bitwise operators since you do not have direct memory access in browser hosted JavaScript, however, for certain types of storage or cookie usage some developers may find ingenious ways to use these operators.

Advanced Assignment

Shorthand	Expanded	Example
x+=y	x=x+y	var x=5; x+=7 //x now 12
x-=y	x=x-y	var x=5; x-=7; //x now –2
x*=y	x=x*y	var x=5; x*=7; //x now 35
x/=y	x=x/y	var x=5; x/=2; // x now 2.5
x%=y	x=x%y	var x=5; x%=4; // x now 1

Note: The bitwise operators also support a shorthand form if you are so inclined to use those operators.

Increment & Decrement

- ++ = plus one (ex. x++ same as x=x+1)
- -- = minus one (ex. x-- same as x = x –1)
  - Pre increment/decrement ++x; --x;
  - Post increment/decrement x++; x--;

- Subtle difference of when increment decrement happens can cause problems

```
var x=3;
alert(x++)  // next try alert(++x)
```

Comparison Operators

Operator	Meaning	Example	Evaluates
<	Less than	4 < 8	True
<=	Less than equal to	6 <=5	False
>	Greater than	4 > 3	True
>=	Greater than equal to	5>=5	True
!=	Not equal	6!=5	True
==	Equal to	6==5	False
===	Equal to (same type)	‘5’ === 5	False
!==	Not equal to (same type)	5!==‘5’	True

Logical Operators

- && (AND) – both sides of operation must be true to evaluate to true, otherwise false
- || (OR) – either side of operation must be true to evaluate to true, otherwise false
- ! (NOT) – reverse the value
- Be aware of using ( )s to force evaluation properly
- See section on short-circuit evaluation ahead

?: Operator

- Simple conditional branch (same as in C)
- Syntax

  (*expression***) ?** *true-statement* **:** *false-statement***;**
- Example
  - `(x > 5) ? alert("x is bigger") : alert("x is smaller");`
- Commonly used with object detection

```
var doRoll;
(document.images)? doRoll=true : doRoll=false;
```

- Not the same as **if** because **?** operator cannot support multiple statements on a condition
- Do note that in JavaScript this operator is more commonly used than expected, likely due to desire to reduce code size

Comma Operator

- Allows multiple statements to be strung together and executed as one statement. The only value returned is the last one.

```
var a,b,c,d;
a = (b="5", c="7", d="56");
document.write(a,b,c,d);
// a and d will be 56
```

- Comma used in method and function calls to separate parameters

```
document.write(x,y);
myFiddle(x,y,1);
```

Void Operator

- The **void** operator specifies an expression to be evaluated without returning a value

```
var a,b,c,d;
 a = void (b="5", c="7", d="56");
 document.write(a,b,c,d);
 // a will be undefined and d = 56
```

- Most common use of **void** is with pseudo-URLs so as to not accidentally side-effect something in HTML
  - Click me

Typeof Operator

- The **typeof** operator is used to determine the type of a particular literal value or variable. The operator returns a string holding the type of the passed value.

```
var a = 3; name = "Howard";
alert(typeof a); // displays number
alert(typeof name); //displays string
```

- A chart in the previous chapter discussing type conversion shows possible returned values. The typeof null being the most interesting as it returns a value of "object."

Object Operators (., [ ], new, and delete)

- Period operator (**.**) used to access properties and methods of an object (ex. `document.write(); `)
- Objects can be accessed via associated array style with `[]` operators (ex. `document["lastModified"];` is the same as `document.lastModified; `)
- **new** operator used to create objects (ex. `var myArray = new Array(2,45,67,45,52); `)
- **delete** operator used to delete items from an object. Primarily used with arrays (ex. `delete myArray[1];`)

Operator Precedence

- Operators are from left to right except when operator precedence takes over
- The highest precedence are those to access objects (.) and arrays ([ ]) as well as group objects (). Then comes increment/decrement, miscellaneous object operators, math, comparisons, logical operators, and lastly assignments
- Complete chart shown in book

If Statements

- Syntax

if (*expression***)**
*statement***;**

if (*expression***)**
{
statement(s)
}

Example

```
var x = 3;
 if (x > 2)
   alert("It's bigger than 2! ");
```

Expressions can be anything that evaluates eventually into a Boolean

```
if ((weather == "Sunny") && (day == "Saturday"))
   {
    alert("To the beach…");
   }
```

If Statements Contd.

- You can also add a second condition with an **else**
- Syntax:

if (*expression***)**
statement or block
else
statement or block

```
var x = 5;
if (x > 1)
 {
  alert("x is greater than 1.");
  alert("Yes x really is greater than 1.");
 }
else
 {
   alert("x is less than 1.");
   alert("This example is getting old.");
  }

alert("moving on …");
```

If Statements Contd.

- You can combine **if-else** statements sequentially
- Syntax:

if (*expression***)**
statement or block
else if (*expression***)**
statement or block
else
(*expression*)

- The number of **if** statements strung together is up to you but as you add more and more you may find that the **switch** syntax discussed in a few moments provides a better solution

Short-circuit Evaluation

- Recall with a logical operator that JavaScript will only evaluate what is necessary to determine the outcome.
- When using (AND) **&&** a single false statement results in false so you may not need to look at both statements
- Using (OR) **||** a single true statement results in a true evaluation.
- If a condition can bail out early it will—this is termed short circuit evaluation and may result in unexpected results
- The example on the next slide shows short circuit evaluation in action

Short-Circuit Evaluation Example

```
document.write("<pre>");
document.writeln("No short circuit evaluation\n");

var age = 31;
if ((document.writeln(" Left expression evaluates"),
    (age >= 13))  &&    (document.writeln(" Right expression evaluates"), (age <= 19)))
   document.writeln("Result: You are a teenager.");
else
  document.writeln("Result: You are not a teenager.");

document.writeln("\n");
document.writeln("With short circuit evaluation\n");

var age = 31;
if ((document.writeln(" Left expression evaluates"),  (age <= 19)) &&
    (document.writeln(" Right expression evaluates"), (age >= 13)))
  document.writeln("Result: You are a teenager.");
else
  document.writeln("Result: You are not a teenager.");
document.write("</pre>");
```

Purposeful Short-circuit Use

- Short-circuit evaluation can be put to good use
  - Example: use object detection with if statements and type conversion to see if a browser supports a particular object

```
if (document.all)
 { /* do interesting stuff */ }
```

- In some cases if you a long object path you cannot do this if you assume the existence of a parent object as it will throw an error

```
if (window.screen.height)
// errors if screen does not exist
```

- So you purposefully use short-circuit evaluation like so

```
if ((window.screen) && (window.screen.height))
```

Switch

- JS 1.2 and greater supports **switch**
- Syntax

```
switch (expression)
{
    case condition 1: statement(s)
            break;
    case condition 2: statement(s)
            break;
      …
    case condition n: statement(s)
            break;
    default: statement(s)
}
```

Switch Example

```
 var yourGrade='A';
switch (yourGrade)
{
  case 'A': alert("Good job.");
            break;
  case 'B': alert("Pretty good.");
            break;
  case 'C': alert("You passed!");
            break;
  case 'D': alert("Not so good.");
            break;
  case 'F': alert("Back to the books.");
            break;
  default: alert("Grade Error!");
}
```

Switch Example 2

- No need to use blocks { } with switch

```
var yourGrade='C', deansList = false, academicProbation = false;
switch (yourGrade)
{
  case 'A': alert("Good job."); deansList = true;
            break;
  case 'B': alert("Pretty good."); deansList = true;
            break;
  case 'C': alert("You passed!"); deansList = false;
            break;
  case 'D': alert("Not so good."); deansList = false;
academicProbation = true;
              break;
  case 'F': alert("Back to the books."); deansList = false;
academicProbation = true;
            break;
  default: alert("Grade Error!");
}
```

Switch Example 3

- Make sure you understand fall-thru and **breaks** with **switch**

```
var yourGrade='B', deansList = false,
    academicProbation = false;

switch (yourGrade)
{
  case 'A':
  case 'B': alert("Pretty good."); deansList = true;
            break;
  case 'C': alert("You passed!"); deansList = false;
            break;
  case 'D':
  case 'F': alert("Back to the books.");
            deansList = false;
            academicProbation = true;
            break;
  default: alert("Grade Error!");
}
```

While Loops

- While loops run statements over and over again
- Syntax

  while (*expression*)

  statement or block of statement to execute

- Example

```
var count = 0;
while (count < 10)
 {
  document.write(count+"<br />");
  count++;
 }
document.write("Loop done!");
```

While Loop Example

- Loops can count up or down and skip in any fashion you can come up with.

```
var count = 100;
while (count > 10)
{
  document.write(count+"<br />");
  if (count == 50)
    count = count – 20;
  else
    count = count - 10;
}
```

- Loops may never even execute

```
 var count = 0;
while (count > 0)
{
// do some statements
}
```

Infinite Loops

- A loop can be created so poorly as to never end—a so called “infinite loop”

- Browsers have a variety of ways for handling this

```
var count = 0;
 while (count < 10)
  {
   document.write("Counting down forever: " + count +"<br
/>");
   count--;
  }
 document.write("Never reached!");
```

Do-while Loops

- The **do-while** loop is just like the **while** loop except that the condition is checked at the end of the loop, thus the loop always runs at least once.

- Syntax

do
{
statement(s);
}
while *(expression);*

- Example

```
var count = 0;
  do
  {
    document.write("Number " + count + "<br />");
    count = count + 1;
  } while (count < 10);
```

For Loops

- For loops are compact and include loop initialization, test statement and iteration in one line
- Syntax
  - **for** (*initialization*; *test condition*; *increment or decrement*)
    looped statement or block
- Example

```
for (var i = 0; i < 10; i++)
    document.write ("Loop " + i + "<br />");
```

- Watch out for semi-colon placement with **for** loops

```
for (var i = 0; i< 10; i++);
   {
    document.write("value of i="+i+"<br />");
   }
document.write("Loop done");
```

Loop control: continue

- When the **continue** statement is encountered a loop continues on (go back to the top of the loop and maybe iterate depending on the type of loop)
- The example here continues on when the loop reaches 8

```
var x=1;
while (x < 20)
{
  x=x+1;
  if (x == 8)
    continue;
    // continues loop at 8 without print
  document.write(x+"<br />");
}
```

Loop control: break

- When the **break** statement is encountered a loop may be stopped. Break effectively leaves an enclosing program block { }
- Example here breaks out of the loop when 8 is reached

```
var x = 1;
while (x < 20)
{
  if (x == 8)
     break;  // breaks out of loop completely
  x = x  + 1;
  document.write(x+"<br />");
}
```

Labels and Flow Control

- A label can be used with **break** and **continue** to direct flow control more precisely.
- A label is just an identifier followed by a **:**
- Consider the example here showing a label used with break

```
outerloop:
  for (var i=0; i < 3; i++)
  {
        document.write("Outerloop: "+i+"<br />");
        for (var j = 0; j < 5; j++)
          {
             if ( j == 3)
               break outerloop;
             document.write("Innerloop: "+j+"<br />");
           }
  }
  document.write("All loops done"+"<br>");
```

Object Statements: With

- The **with** statement provides programmers with a short hand notation when referencing objects
- Given

```
document.write("Hello from JavaScript");
document.write("<br />");
document.write("You can write what you like here");
```

we could use a with to shorten things

```
with (document)
  {
   document.write("Hello from JavaScript");
   document.write("<br />");
   document.write("You can write what you like here");
  }
```

- Syntax

with (*object*)
{
statement(s);
}

Object Statements: for..in

- The **for…in** statement is used to loop through the properties of an object (if they can be enumerated)
- Syntax

  for (*variablename* **in** *object*)

  statement or block to execute

- Example

```
var aProperty;
 document.write("<h1>Navigator Object Properties</h1>");
 for (aProperty in navigator)
 {
   document.write(aProperty);
   document.write("<br />");
 }
```

Summary

- This chapter covered the basics of JavaScript from operators and expressions to control statements

- We found the syntax to be very much like modern programming languages like C with the addition of features to support objects (**with**, **for…in**)

- Some features seemed somewhat out of place (e.g. bitwise operators) given the context of JavaScript's use but it may hint at the more general purpose the language is becoming