JavaScript Syntax

Comments

Comments are sections of code that you do not want to run. A single line comment is preceded by //, while multiline comments are surrounded by /* ... */:

```
var x = 100
// This is a comment; it will be ignored when the code is run

var y = 50    // This is a comment on the same line as some code

/* This is a longer
comment, that
spans multiple lines */
```

Line Endings

Lines of JavaScript code are separated by a return character (so each line is actually on its own line) or by a semicolon:

```
var x = 100
var y = 50
var a = 20; var b = 40
```

Either is fine, though it's good to be in the habit of ending lines with the ; symbol. This makes it easier to "minify" your code, and put long scripts on a single line.

Case Sensitivity

JavaScript is a case-sensitive language, which means that the strings 'hello', 'Hello', and 'HELLO' would not be equal. JavaScript keywords, like function and for, must be written in lowercase.

Quotation Marks

Strings in JavaScript can be surrounded by either single or double quotation marks.

```
var newString = "I'm a string with double quotation marks"
var aNewString = 'This is a string with single quotation marks'
var anotherNewString = 'And I'm a string with too many quotation marks'
```

The reason you'd want to use a mix of both can be seen in the color highlighting of the above example; the apostrophe in the word "I'm" was interpreted as the end of the string. Instead, we'll surround the string in double quotes:

```
var anotherNewString = "And I'm a string with too many quotation marks"
```

Variables

A variable in JavaScript can hold one of several data types, including numbers, strings, arrays, booleans, functions, objects, etc. JavaScript is a dynamic language, so we don't need to specify the type of our variable, and we can change it at any time. We assign a value to a variable using the '=' operator.

```
var x = 5
```

You can retrieve the type of a variable using the `typeof` operator:

```
var sampleString = 'Hello world'
var x = false
typeof sampleString       // returns 'string'
typeof x            // returns 'boolean'
```

When naming variables, begin with a lowercase letter. You can use any combination of letters and numbers, and the _ symbol. Do not include a space in the variable name.

JavaScript Control Flow

If and If-Else Statements

If statements are used to execute a block of code, given particular conditions are true.

```
var x = 100
var y = 50
if ( x>0 ) {
  alert('x is positive')
}
if (y<2) {
  alert('y is small')
}
// there will be an alert 'x is positive', but the second alert will not be displayed,
    since y < 2 is false
```

If-else statements are used when you want one block of code to run when a statement is true, and another to run if the statement is false:

```
var x = 4
if (x<0) {
  alert('x is negative')
}
else {
  alert('x is not negative')
}
// the alert will display 'x is not negative', since x < 0 is false
```

For even more options, use `else if` to test for addition conditions:

```
var x = 4
if (x<0) {
  alert('x is negative')
}
else if (x==0) {
  alert('x is zero')
}
else {
  alert('x is positive')
}
// the alert will display 'x is positive', since x<0 and x==0 are both false
```

For Loops

To have a block of code run a particular number of times, use a for loop. A typical for loop takes three arguments: the beginning value, the condition that must be true for the loop to continue, and the increment.

```
for (var i = 0; i < 5; i++) {
  // define a variable i to be 0, and increase i by 1 with each loop
  // the loop will run until i is 5 (but not including i=5)
  console.log(i)
}
// the console will show 0,1,2,3,4

for (var i = 4; i < 20; i=i+2) {
  // define a variable i to be 4, and increase i by 2 with each loop
  // the loop will run until i is 20 (but not including i=5)
  if (i%2==0) {   // check if i is divisible by 2
    console.log(i)    // only print i if i%2==0
  }
}
// the console will show 4,6,8,10,12,14,16,18
```

For loops can also be used to loop over array values or object properties using `for..in`:

```
var sampleArray = [1,3,6,'hi']
for (var i in sampleArray) {
  console.log(i)
}
// the console will show 1,3,6,hi
var sampleObject = { 'first': 1, 'second':3, 'third':'hi' }
for (var i in sampleObject) {
  // i ranges over all property names
  // sampleObject[i] produces the value of the property i
  var string = "The value of property "+i+" is "+sampleObject[i]
  console.log(string)
}
// the console will show:
//  The value of property first is 1
//  The value of property second is 3
//  The value of property third is hi
```

While Loops

While loops are used to execute a code block while a given condition is true. The condition is tested before the code is executed.

```
var x = 2
while (x<1) {
  console.log(x)
  x++
}
// Nothing is printed. The condition is false to begin with, so the code will not be
    executed.

var y = 2
while (y<10) {
  console.log(y)
  y++     // change the value of y, or else the loop will never end
}
// console will show 2,3,4,5,6,7,8,9
```

Do While Loops

Do-while loops are used to execute a code block while a given condition is true. The condition is tested after the code is executed, in contrast to while loops.

```
var x = 2
do {
  console.log(x)
  x++
}
while (x<1)
// The console will show 2, and x is increased to 3. Then the condition is tested and
    found to be false, so the loop ends.

var y = 2
do {
  console.log(y)
  y++     // change the value of y, or else the loop will never end
}
while (y < 10)
// console will show 2,3,4,5,6,7,8,9,10
```

Break

The `break` command is used to stop a loop. For instance, the following code creates an array. Then we create a for loop that tests if the given value of x is in the array. If it is, we'll set a boolean value equal to true:

```
var testArray = [1,10,2,8,3,15,11,7]
var x = 2
var xHasBeenFound = false
for (var i in testArray) {
  if (i==x) {
    xHasBeenFound = true
  }
}
// xHasBeenFound is now true
```

Since x is actually found at index 2, there was no need to continue the search. Adding a break command will stop the loop when it's no longer necessary to run.

```
var testArray = [1,10,2,8,3,15,11,7]
var x = 2
var xHasBeenFound = false
for (var i in testArray) {
  if (i==x) {
    xHasBeenFound = true
    break
  }
}
// xHasBeenFound is now true, and the loop stopped after only 3 runs instead of 8
```

Switch

Switch statements can be more convenient that a series of if-then-else statements. Blocks of code are executed depending on the value of an expression.

```
var breed = prompt("What kind of dog do you have?")
// a pop-up will display on screen asking the question
// the value of breed will be a string equal to whatever the user types as their
    answer

switch ( breed ) {
  case "English Bulldog":   // if the user typed English Bulldog:
    alert("Bulldogs are courageous but gentle and were bred to bait bulls.")
    break;                  // end the switch statement, since we found the case
  case "Clumber Spaniel":
    alert("Clumber spaniels are calm but affectionate and originated in France.")
    break;
  case "Labrador Retriever":
    alert("Labs are athletic and are excellent swimmers.")
    break;
  case "Papillon":
    alert("Papillons are small but intelligent dogs with long hair and big ears.")
    break;
  default:      // the user entered an expression that wasn't on our list
    alert("Sorry, I don't know about your dog!")
}
```

JavaScript Strings

Strings in JavaScript can be any combination of letters, numbers, and symbols, surrounded by single or double quotes.

```
var sampleString = "I am a string"
```

All strings have a length property that returns the number of characters (including whitespace) in the string:

```
var sampleString = "I am a string"
sampleString.length   // returns 13
```

Concatenation

Two strings can be concatenated (combined into a single string) using the + symbol:

```
var sampleString = "I am a string"
var longerString = sampleString + " and now I am longer"
// longerString has value "I am a string and now I am longer"
```

You can concatenate two strings or a string and a variable or number. Anything surrounded by quotes will be interpreted literally.

```
var x = 4
var y = 2
var firstTry = "The sum of x and y is x+y"
// firstTry value is "The sum of x and y is x+y"
var secondTry = "The sum of "+x+" and "+y+" is x+y"
// secondTry value is "The sum of 4 and 2 is x+y"
var thirdTry = "The sum of "+x+" and "+y+" is "+(x+y)
// secondTry value is "The sum of 4 and 2 is 6"
```

Useful string methods

Each character in a string is given an index, starting at 0. To find the character at a particular index, use the `charAt` method:

```
var sampleString = "I am a string"
sampleString.charAt(0)    // returns "I"
sampleString.charAt(7)    // returns "s"
```

To find the index of a particular character or string, use the `indexOf` method. This will only return the first index where the character or string appears. The method returns -1 if the string is not found.

```
var sampleString = "I am a string"
sampleString.indexOf('a')   // returns 2
sampleString.charAt('str')    // returns 7
sampleString.charAt('d')    // returns -1
```

To create a substring, use the `substr` method, which takes two parameters: the starting index and the length of the substring:

```
var sampleString = "I am a string"
var subString = sampleString.substr(4, 8)
// subString has value "a string"
```

To split a string into array elements using a particular separator, use the `split` method:

```
var students = "Anne,Claire,Derek,John,Natalie"
var studentArray = students.split(",")    // split with separator ,
// studentArray is now ["Anne","Claire","Derek","John","Natalie"]
```

JavaScript Arrays

An array in JavaScript is an ordered list of values, which could be strings, numbers, other arrays, objects, booleans, or even functions. An array is created by listing its values between square braces, separated by commas:

```
var sampleArray = [1,3.5,[2,4,6],"hello world",true,83,-2]
```

To access a value from the array, you'll need its index. Indices are integer values beginning at 0. To access a value of the array at index n, use the format `tableName[n]`. Append additional brackets to find values of nested arrays:

```
sampleArray[0]      // returns 1
sampleArray[2]      // returns the array [2,4,6]
sampleArray[4]      // returns true
sampleArray[2][1]   // returns 4, the value at index 1 of the array at index 2
```

To find the index of a value in an array, use the `indexOf` method. If the value is not found in the array, the method will return -1. If a value appears more than once, only the first index is returned.

```
var week = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
week.indexOf("Monday")    // returns 1
week.indexOf("January")   // returns -1, because "January" is not in the array
```

Finding the number of values in an array

Every array has a length property that returns the number of values it contains. Note that if the array contains another subarrays, they'll be counted as a single value, regardless of how many values they contain:

```
sampleArray.length    // returns 7
var newSampleArray = [[1,2,3],[4,[5,6,7,8]]]
newSampleArray.length       // returns 2
newSampleArray[0].length    // returns 3, the number of values in [1,2,3]
newSampleArray[1].length    // returns 2, the number of values in [4,[5,6,7,8]]
newSampleArray[1][1].length // returns 4, the number of values in [5,6,7,8]
```

Adding and removing values

To add values to the beginning or end of an array, use the `unshift` and `push` methods, respectively:

```
var evenArray = [2,4,6,8]
evenArray.push(10)        // evenArray is now [2,4,6,8,10]
evenArray.unshift(0)      // evenArray is now [0,2,4,6,8,10]
```

To add a value at a particular index, with the option of removing other values, use the `splice` method. Splice takes three arguments: the first is the index where you'd like the new value to be added, the second is the number of values you'd like removed starting at the same index, and the third is the value you'd like added to the array.

```
var x = [1,2,3,4,6,7]
var x.splice(4,0,5) // add 5 at index 4, but don't remove any values
// x is now [1,2,3,4,5,6,7]
var y = [1,2,3,4,'five','six',7]
var y.splice(4, 2, 5) // add 5 at index 4, and remove 2 values starting at index 4
// y is now [1,2,3,4,5]
```

The `shift` and pop methods remove and return the first and last entries of an array:

```
var x = [1,2,3,4,5]
var firstx = x.shift()
// x is now [2,3,4,5], and firstx is 1
var lastx = x.pop()
// x is now [2,3,4], and lastx is 5
```

Other useful array methods

To create a subset of a particular array, use the `slice` method. The method takes two parameters: the index where the subarray should begin, and the index just after you'd like to end:

```
var week = ["Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"]
var workWeek = sampleArray.slice(1,6)
// workWeek is ["Monday","Tuesday","Wednesday","Thursday","Friday"]
```

If your array contains only numerical values, you can arrange them into increasing order. Strings in an array are sorted according to their unicode representation, which may not always be alphabetical.

```
var numberArray = [4,1,8,2,10]
numberArray.sort()    // numberArray is now [1,2,4,8,10]
```

Array values can be reversed:

```
var numberArray = [4,1,8,2,10]
numberArray.reverse()   // numberArray is now [10,2,8,1,4]
```

To combine two arrays into one, use the `concat` method:

```
var evenArray = [2,4,6,8]
var oddArray = [1,3,4,7]
var numberArray = evenArray.concat(oddArray)  // numberArray is now [1,2,3,4,5,6,7,8]
```

To apply a function to each value in an array (changing the values themselves), use the `forEach` method:

```
function addTwo( e, i, a ) {      // e is a value at index i in array a
  a[i] = e + 2
}
var sampleArray = [1,2,3]
sampleArray.forEach( addTwo )     // sampleArray is now [3,4,5]
```

To create a string from your array values with an optional separator, use the `join` method:

```
var x = [4,"hello",9,-2,3]
var xString = x.join(separator="..")
// xString is now 4..hello..9..-2..3
```

Loops and arrays

We can easily loop through array elements by using their indices:

```
var myArray = [1,2,3,4,5]
for (var i = 0; i < myArray.length; i++) {
  console.log( myArray[i] )
}
```

JavaScript Objects

An object in JavaScript has properties, much like objects in real life have characteristics. The properties are variables attached to the object and define its characteristics.

For instance, we can create a JavaScript object called 'dog' with several properties. The value of each property can be any variable type, such as a string, number, array, or boolean value.

```
var dog = new Object ()
  dog.breed = "papillon"
  dog.name = "Damien"
  dog.age = 9
  dog.birthday = ['December', 12, 2006]
  dog.color = "red and white"
  dog.good = true
```

To access the property, use dot-notation. Undefined properties are returned as undefined, not null.

```
dog.breed            // returns 'papillon'
dog.weight           // returns undefined
```

Objects can also be created by listing the properties in curly braces, separating the properties and their value with a colon. Properties and values are separated with a comma.

```
var dog = {
  breed: "papillon",
  name: "Damien",
  age: 9,
  birthday: ['December', 12, 2006],
  color: "red and white",
  good: true,
}
```

Finding the number of properties in an object

Unlike arrays, objects have no length property. However, we can find the number of properties given to an object by getting the length of the array containing its properties:

```
var numberOfProperties = Object.keys(dog).length    // numberOfProperties is now 6
```

Loops and objects

The values stored in an array are indexed by numbers: the first value has index 0, the next has index 1, and so on. Properties of objects are not stored numerically, so instead of looping over an index (such as i) we instead loop through the properties:

```
for (var p in dog) {
  console.log( "The value of "+p+" is "+dog.p)
}
// the console will show
// The value of breed is papillon
// The value of name is Damien
// The value of age is 9
// The value of birthday is  ['December', 12, 2006]
// The value of color is red and white
// The value of good is true
```

Equivalently, you can use brackets to get the value of a property. This is advised, as it works with objects in standard and JSON format:

```
for (var p in dog) {
  console.log( "The value of "+p+" is "+dog[p])
}
```

Combining Arrays and Objects

Creating an array whose values are objects is a common strategy when working with data. For instance, the array cityData below contains several (unnamed) objects, each with the same four properties, that we can access using indices and properties:

```
var cityData = [      // square brackets for array
  {                   // curly braces for object
    city: "Erie",
    state: "PA",
    population: 100671,
    medianIncome: 32508
  },                  // separate array values (objects) with commas
  {
    city: "Belmar",
    state: "NJ",
    population: 5736,
    medianIncome: 56633
  },
  {
    city: "Milwaukee",
    state: "WI",
    population: 599164,
    medianIncome: 35186
  }
]
cityData[2].state          // returns WI
cityData[0].population     // returns 100671
```

When looping, remember that arrays have numerical indices and length, while objects have properties:

```
for (var i = 0; i < cityData.length; i++) {
  var locString = cityData[i].city+" has population "+cityData[i].population
  console.log(locString)
}
// The console will display:
//  Erie has population 100671
//  Belmar has population 5736
//  Milwaukee has population 599164
```

JavaScript Object Notation (JSON)

JSON is not a new kind of object, but simply a particular syntax for storing objects that works particularly well with JavaScript and AJAX. An object in JSON format is a JavaScript object, only all properties are string values themselves:

```
var dog = {
  "breed": "papillon",
  "name": "Damien",
  "age": 9,
  "birthday": ['December', 12, 2006],
  "color": "red and white",
  "good": true,
}
```

To access the value of a property, use notation similar to finding the value at a particular index of an array. However, instead of specifying a numerical index to look for, you just give the property name, also as a string:

```
dog["breed"]             // returns 'papillon'
dog["birthday"][1]       // returns 12
```

We can still loop over the properties of an object in JSON format:

```
for (var k in dog) {
  console.log( dog[k] )
}
// prints papillon, Damien, 9, ['December', 12, 2006], red and white, true in console
```

Operators and Mathematics

Math Properties & Predefined Constants

Property	Description	Example
Math.E	Euler's constant, e	2.718
Math.LN2	Natural (base e) logarithm of 2	2.303
Math.LN10	Natural (base e) logarithm of 10	1.443
Math.LOG10E	Base 10 logarithm of e	0.434
Math.PI	Ratio of circle's circumference to its diameter	3.14159
Math.SQRT1_2	Square root of $1/2$	0.707
Math.SQRT2	Square root of 2	1.414

Math Methods

Method	Description	Example	Example Returns
Math.abs(x)	Absolute value of x	Math.abs(-6.5)	6.5
Math.acos(x)	Angle θ (in rad) such that $\cos(\theta) = x$	Math.acos(0.6)	0.9272952180016123
Math.asin(x)	Angle θ (in rad) such that $\sin(\theta) = x$	Math.asin(0.6)	0.6435011087932844
Math.atan(x)	Angle θ (in rad) such that $\tan(\theta) = x$	Math.atan(10)	1.4711276743037345
Math.atan2(x,y)	Angle θ (in rad) such that $\tan(\theta) = x/y$	Math.atan2(20,2)	1.4711276743037345
Math.ceil(x)	Smallest integer greater than x	Math.ceil(4.23)	5
Math.cos(x)	$\cos(x)$	Math.cos(Math.PI)	-1
Math.exp(x)	e^x	Math.ceil(4.23)	4
Math.floor(x)	Greatest integer less than x	Math.ceil(4.23)	4
Math.log(x)	Natural logarithm of x	Math.log(1)	0
Math.max(list)	Largest value in list	Math.max(4,6,1,9,2,-5,2)	9
Math.min(list)	Smallest value in list	Math.min(4,6,1,9,2,-5,2)	-5
Math.pow(x,y)	x^y	Math.pow(4,2)	16
Math.random()	Pseudo-random value between 0 and 1	Math.random()	0.37454011430963874
Math.round(x)	Closest integer to x	Math.round(3.14159)	3
Math.sin(x)	$\sin(x)$	Math.sin(Math.PI/2)	1
Math.sqrt(x)	Square root of x	Math.sqrt(25)	5
Math.tan(x)	$\tan(x)$	Math.tan(6)	-0.29100619138474915

Arithmetic Operators

Operator	Description	Example	Example Returns
+	Addition, String concatenation	3+6	9
		"hello "+"world"	"hello world"
-	Subtraction	12-8	4
/	Division	12/3	4
*	Multiplication	2*5	10
%	Modulus (remainder)	17%5	2
++	Increase by one	var x = 5; x++	x is now 6

Random Values

The Math.random() method returns a pseudorandom value in the interval $[0, 1)$, meaning it could be any value between 0 and 1, or 0, but not 1. The function below returns a random integer between two integers a and b:

```
function randomBtwn(a,b) {
  return Math.floor(Math.random()*(b-a+1)+a)
}
var r = randomBtwn(2,80)    // r is now a random integer between 2 and 80
```

Comparison Operators

Operator	Description	Example	Example Returns
x == y	Returns true if x and y are same type and value	5 == 5	true
		5 == '5'	false
x != y	Returns true if x and y are of different types or values	5 != 2	true
		5 != 5	false
x === y	Returns true if x and y are the same type and value	5 === 5	true
		5 === '5'	false
x > y	Returns true if x is strictly greater than y	5>2	true
x >= y	Returns true if x is greater than or equal to y	5>=7	false
x < y	Returns true if x is strictly less than y	3<1	false
x <= y	Returns true if x is at most y	1<=1	true

Assignment Operators

Operator	Description	Example	Example Returns
x = value	Assign a value to a variable x	var x = 5	x is now 5
x += y	Addition assignment, shorthand for x = x+y	x = 7; x += 5	x is now 12
x -= y	Subtraction assignment, shorthand for x = x-y	x = 7; x -= 5	x is now 2
x *= y	Multiplication assignment, shorthand for x = x*y	x = 7; x *= 5	x is now 35
x /= y	Division assignment, shorthand for x = x/y	x = 10; x /= 5	x is now 2
x %= y	Remainder assignment, shorthand for x = x%y	x = 10; x %= 3	x is now 1

Logical Operators

Operator	Description	Example	Example Returns
x && y	Returns true if both x AND y are true	(3 < 4) && (4 == 4)	true
		(3 < 4) && (4 > 4)	false
x || y	Returns true if either x OR y is true	(8 < 4) || (4 == 4)	true
		(3 > 4) || (4 > 4)	false
!x	Returns true if x is false	!(8 < 4)	true
		!(4 == 4)	false

Conditional Operators

The conditional operator takes three operands: a condition that can be evaluated to true or false, an expression to return if the condition is true, and a condition to return if the condition is false. The syntax of the conditional operator is:

```
(condition to test) ? expression_if_true : expression_if_false
```

For example,

```
var x = prompt( "Pick a number" )

// Decide if provided value is larger than 10, and choose a message accordingly:
var returnMessage = ( x > 10 ) ? " larger than 10" : " less than or equal to 10"
alert( "The number " + x + " is " + returnMessage )
```

and

```
var x = prompt( "Find the absolute value of any number: " )
var absX = ( x >= 0 ) ? x : -x
alert( "The absolute value of " + x + " is " + absX )
```