

Advanced JavaScript Programming

Quick JavaScript Recap

Lesson 1, Activity 2: Primitive data types

JavaScript comes with a number of data types that we can use in our variables. Let's look at them.

Null

Null is a data type that has only one possible value, `null`, which is also a reserved word in the language. We use `null` to represent a value that we don't know or that is missing.

```
var name = "Homer"; var ssn = null;
```

In the above example we know what to put in the `name` variable, but we don't know yet what to put in the `ssn` variable. Maybe we will know what to put in there later in our program, but maybe not.

Undefined

The Undefined type also has a single value, `undefined`, and it is similar to `null` but not exactly the same thing.

JavaScript uses `undefined` as the default value for any variable that has not been initialized yet. Let's modify our previous example.

```
var name = "Homer"; var ssn;
```

Now the value of `ssn` is `undefined` because it is no longer initialized to `null` or anything else.

The `undefined` type is used a lot when we want to detect if a variable has already been declared.

```
//Check if we already have a start time
if (startTime === undefined) {
    startTime = new Date();
}
```

Boolean

Boolean is a very common data type in every language. It has only two values, `true` and `false`, which are reserved words and, I hope, self-explanatory.

```
var enabled = true;
var disabled = false;
```

Number

The `Number` data type can represent two types of numeric values: 32-bit integers or 64-bit floating point numbers.

Number values are created using number literals, which are shown in the following demo.

Code Sample:

QuickRecap/Demos/numbers.html

```

---- C O D E   O M I T T E D ----

var numbers = {};
numbers['age'] = 25; // simple, decimal, integer
numbers['price'] = 45.95; // floating point
numbers['letter'] = NaN; // not a number

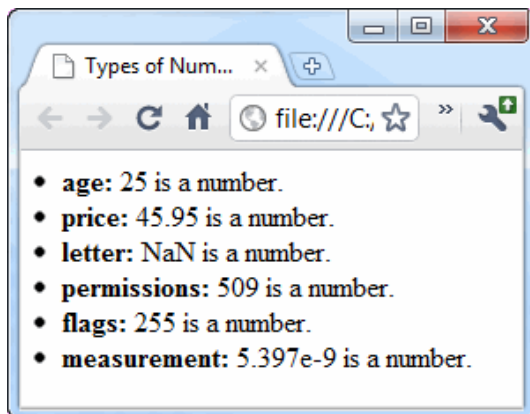
//SPECIAL CASES
// (note leading 0) integer in octal, 509 in decimal:
numbers['permissions'] = 0775;
// integer in hexadecimal, 28 in decimal:
numbers['flags'] = 0x00ff;
// floating point in scientific notation:
numbers['measurement'] = 5.397e-9;

---- C O D E   O M I T T E D ----

for (var i in numbers) {
  document.write("<li><strong>" + i + ": </strong>");
  document.write(numbers[i]);
  document.write(" is a " + typeof numbers[i]);
  document.write("<./li>");
}
---- C O D E   O M I T T E D ----

```

This will display as follows:



- The first two, 25 and 45.95, are self-explanatory.
- The third, NaN, stands for "Not a Number," which means that JavaScript cannot convert the value to a number.
- The final three (octal, hexadecimal, and scientific notation) are special cases that you can use, but are unlikely to run into.

String

The `String` is used to represent text. Programmers spend a lot of time manipulating strings in pretty much any programming language.

We create strings using literal values enclosed in single or double quotation marks. JavaScript also supports a few special encodings for common characters such as *new line* (`\n`) and *tab* (`\t`).

Code Sample:

QuickRecap/Demos/special-chars.html

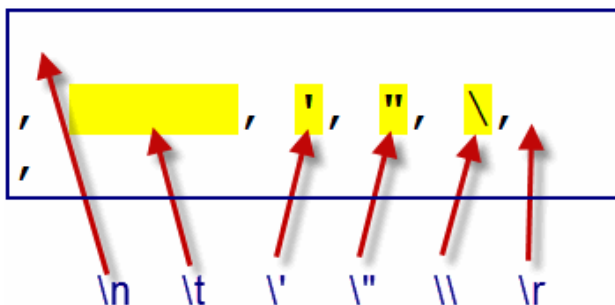
```

---- C O D E   O M I T T E D ----

<script type="text/javascript">
  /*
   * the backslash (\) is used as an escaped character in JS
   * \n = new line
   * \t = tab
   * \' = single quote
   * \" = double quote
   * \\ = backslash
   * \r = carriage return
   */
  var chars = ['\n', '\t', '\'', '\"', '\\', '\r'];
</script>
---- C O D E   O M I T T E D ----
<pre>
<script type="text/javascript">
for (var i in chars) {
  document.write("<span>");
  document.write(chars[i]);
  document.write("</span>, ");
}
</script>
</pre>
---- C O D E   O M I T T E D ----

```

We use the `<pre>` tag so that the browser displays the new lines and tabs. The screenshot below shows how this will render. We've added arrows to indicate the location of the different characters:



Every value in JavaScript can be converted to a string by using the `toString()` method, like: `var s = myValue.toString();`

Lesson 1, Activity 3: Native Types

In addition to primitive data types, JavaScript provides a few other data types, which are implemented as objects.

Date

We can store date values using `Date` objects. The `Date` object stores the date and time information internally as the number of milliseconds since January 1, 1970.

There aren't date literals in the language, so we have to explicitly create a `Date` object when we need one.

Code Sample:

[QuickRecap/Demos/dates.html](#)

```
---- C O D E   O M I T T E D ----

// current date and time:
var rightNow = new Date();
// 4th of July, note the 0-based month number:
var holiday = new Date(1776, 6, 4);
// 4th of July, format varies with browser locale (avoid this):
var holiday2 = Date.parse('7/4/1776');
---- C O D E   O M I T T E D ----
```

There two important pitfalls in the above example:

1. The month is a number from 0 to 11 when passed as a parameter.
2. The parseable string formats vary by browser implementation and by user locale, so avoid the `parse()` method.

Array

Arrays are very powerful in JavaScript.

The `Array` object can be instantiated using a constructor call or a literal. The array indices must be positive integers, but the values stored in an array can be of any type or reference to any object.

We can create new arrays using the `new Array` constructor or a literal (using square brackets).

Code Sample:

[QuickRecap/Demos/arrays.html](#)

```
---- C O D E   O M I T T E D ----

//constructor call
var cities = new Array();
cities[0] = "Albuquerque";
cities[1] = "Syracuse";
alert(cities.length); //cities.length will be 2
cities[9] = "Tampa"; //cities.length will be 10
```

```
//literal syntax
var teams = [ "Cubs","Yankees","Mariners" ];
alert("Go " + teams[1] + "!"); //Will alert the string: "Go Yankees!"
alert(teams.length); // length will be 3

//literal syntax with not initial values
var dogs = [];
alert(dogs.length); // length will be 0
---- C O D E   O M I T T E D ----
```

Things to note:

1. The array index starts at 0 in JavaScript
2. Arrays can be initialized with values or without values using the literal syntax.
3. The array length is always one greater than the highest numerical index. You should ensure your indices are continuous positive numbers if you are going to loop over the values in the array using the `for (var i=0; i<myarray.length; ++i) {}` approach.

Object

The `Object` type serves as the base for all the objects in JavaScript, regardless of their data type or how they were created.

The `Object` type is also used when we want to create custom objects. As with arrays, we can create new objects using a constructor call or a literal (using curly brackets).

Code Sample:

[QuickRecap/Demos/objects.html](#)

```
---- C O D E   O M I T T E D ----

//constructor call
var employee = new Object();
employee.name = "Homer Simpson";
employee.badgeNumber = 35739;

//literal syntax
var boss = {
  "name" : "Montgomery Burns",
  "badgeNumber" : 1
};

employee.reportsTo = boss;
alert(employee.name + " reports to " + employee.reportsTo.name);
---- C O D E   O M I T T E D ----
```

Regular Expressions

Regular Expressions are used to find occurrences of a string pattern inside a larger string.

Learning regular expressions can have a big payoff depending on the type of work you do.

JavaScript implements regular expressions with `RegExp` objects. It also supports the Perl-style literals.

Code Sample:

[QuickRecap/Demos/regex.html](#)

```
---- C O D E   O M I T T E D ----

var text = "Webucator";
var pattern = new RegExp('cat', 'g');
var samePattern = /cat/g; //using the literal syntax
alert( pattern.test( text ) );// shows 'true'
alert( samePattern.test( text ) );// shows 'true' again
---- C O D E   O M I T T E D ----
```

We will study regular expressions in a later lesson.

Lesson 1, Activity 4: Functions

Functions in JavaScript are more than just static blocks of code. They are `Function` objects that we can use just like any other data type value, e.g. we can pass functions to other functions, we can store a function in a variable, we can modify a function, etc.

We will discuss functions in detail later. For now let's just remember how we declare and call functions.

Code Sample:

[QuickRecap/Demos/functions.html](#)

```
---- C O D E   O M I T T E D ----

//declare the function
function sayHowMuch(name, price, quantity) {
  var finalPrice = price * quantity;
  alert('The price for ' + quantity + ' ' +
    name + '(s) is $' + finalPrice);
}

//call the function with arguments
sayHowMuch('ice cream cone', 1.99, 3);
sayHowMuch('Movie ticket', 10.00, 5);
---- C O D E   O M I T T E D ----
```


Lesson 1, Activity 5: The DOM

The DOM is not JavaScript

It's important to understand that the DOM is a standard that is separate from JavaScript. It was created by the W3C to normalize the browser vendors' implementations of Dynamic HTML.

The DOM enables programmatic access to the HTML document structure for reading or modification purposes through an API. When we write code like `document.getElementById('abc')` we are accessing the DOM using JavaScript.

We can traverse our entire HTML document looking for specific HTML elements, which are called *nodes*, in the DOM. We can even create and append new elements to the DOM.

JavaScript is not the DOM

Just as the DOM is not written specifically for JavaScript, JavaScript is not written specifically for the DOM.

JavaScript is an interpreted language with a runtime execution engine. It needs a host environment to instantiate the engine and forward the JavaScript code to it. The browser is one of many hosts for JavaScript. Other hosts are Adobe Flash plugins (via ActionScript), desktop widgets (like Yahoo! Widgets, MS Gadgets, OS X Dashboard Widgets), Firefox browser add-ons, and even some kinds of electronic equipment.

All that said, the browser was the originally intended host for JavaScript and by far the most common one.

The window object

In browser scripts, the `document` object is actually a property of the `window` object, which is the default (or global) object of JavaScript in that environment. So typing `window.document.body` is the same as typing `document.body`. The DOM starts at the `document` object.

There are other things one may think are part of JavaScript when, in fact, they're browser-specific features, like the `alert()`, `prompt()`, `setTimeout()`, and `open()` functions. These are just methods of the `window` object, not part of JavaScript itself.

Lesson 1, Activity 6: The XMLHttpRequest object

Another important object that we use a lot in JavaScript these days is the XMLHttpRequest object. This is the object that powers the Ajax functionality in a lot of web pages.

This object is also *not* part of JavaScript. It can be used from JavaScript but it isn't part of the language.

The XMLHttpRequest object allows our scripts to initiate a request to a URL and collect the server response without the need to reload the entire page.

We will provide a video presentation on the XMLHttpRequest object later in this lesson.

Code Sample:

[QuickRecap/Demos/xmlhttprequest.html](http://localhost/JavaScript/ClassFiles/QuickRecap/Demos/xmlhttprequest.html)

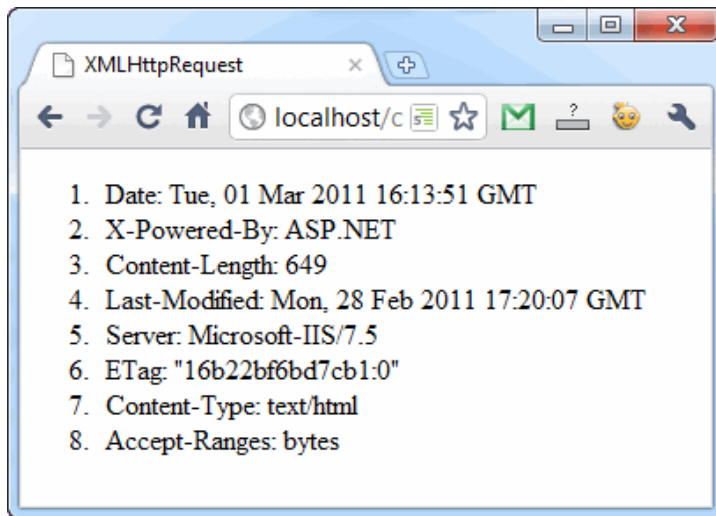
```

---- C O D E   O M I T T E D ----

window.onload = function() {
  var ajax = new XMLHttpRequest();
  var headers;
  var output = document.getElementById("headers");
  var response = document.getElementById("response");
  var temp = "";
  ajax.open("GET", "arrays.html", true);
  ajax.onreadystatechange = function() {
    if ( this.readyState > 2 && typeof headers == "undefined" ) {
      headers = this.getAllResponseHeaders().split("\n");
      temp = "<ol>";
      for (var i in headers) {
        if (headers[i].trim().length > 0) {
          temp += "<li>" + headers[i] + "</li>";
        }
      }
      temp += "</ol>";
      output.innerHTML=temp;
      response.value=this.responseText;
    }
  }
  ajax.send();
}
</script>
</head>
<body>
<output id="headers"></output>
<textarea id="response" cols="50" rows="20"></textarea>
---- C O D E   O M I T T E D ----

```

Note that this code will only work if it runs through a web server (e.g., <http://localhost/JavaScript/ClassFiles/QuickRecap/Demos/xmlhttprequest.html>), which you may not have configured locally. The image below shows how the page appears when run through a web server:



As an aside, see [XMLHttpRequest readyState bug in Opera](#) for an explanation of why we use `this.readyState > 2 && typeof headers == "undefined"` instead of just `this.readyState == 2`.

Lesson 1, Activity 7: JSON

JSON is a lightweight format for exchanging data between the client and server. It is often used in Ajax applications because of its simplicity and because its format is based on JavaScript object literals.

Take a look at this plain text page:

Code Sample:

[QuickRecap/Demos/json.txt](#)

```
{
  "1" : {
    "name" : "Montgomery Burns"
  },
  "35739" : {
    "name" : "Homer Simpson",
    "reportsTo" : 1
  }
}
```

The file above uses JSON syntax to hold employee records. Now imagine that the file was generated dynamically using a call to a database that retrieved all the employees from our company and marked them up in a similar way. We could use Ajax to grab and query that data. Here is a simple example:

Code Sample:

[QuickRecap/Demos/json.html](#)

```
---- C O D E   O M I T T E D ----

<script type="text/javascript">
function showBoss(badgeNo) {
  var ajax = new XMLHttpRequest();
  var employees;
  ajax.open("GET", "json.txt", true);
  ajax.onreadystatechange = function() {
    if ( this.readyState > 2 && typeof employees == "undefined" ) {
      document.getElementById("output").innerHTML = this.responseText;
      employees = JSON.parse(this.responseText);
      alert(employees[employees[badgeNo].reportsTo].name);
    }
  }
  ajax.send();
}
</script>
</head>
<body>
<button onclick="showBoss(35739)">Show Homer's Boss</button>
<pre id="output"></pre>
---- C O D E   O M I T T E D ----
```

Note that, like with [xmlhttprequest.html](#), this code must also run through a web server (e.g,

<http://localhost/JavaScript/ClassFiles/QuickRecap/Demos/json.html>).

Pre-HTML5, the `XMLHttpRequest` and `JSON` objects were not part of the official HTML specification. In HTML5, however, they are.

For a deeper review of how JSON works in JavaScript, see [JSON in JavaScript](#).