2 How to Represent Data

JavaScript has the usual types of data found in other programming languages.

Values, Literals, and Variables

In programming, a value is any piece of data that your programming language lets you create, store, and use. There are many kinds of values in JavaScript, but we'll start with the simplest: numbers and strings. These are described below.

For any kind of value, there are two ways it can appear in code. One form of a value is called a **literal**. That means you see the value itself. For example, **12** is the literal for the number 12. **'Hello, World!'** is the literal for the string "Hello, World"

The other form of a value is a **variable**. A variable is like a variable in mathematics. It is the name of a value. In JavaScript, one way to create a variable and give it a value is to use the form **var** *name* **=** *value***;** e.g.,

```
var dozen = 12;
```

```
var country = 'Spain';
```

Any place where you need a value, you can either write a literal or a value... or an **expression**, which is a formula that combines values to return a new value. We'll show many examples of expressions.

Use Numbers For Calculations

If you need to do calculations, represent your data in JavaScript with integers, like 12, and floating point numbers, like 58.71. Floating point is what programmers call numbers with decimal points.

```
var normalTemp = 98.6;
```

Numbers can be added, subtracted, multiplied, divided, and so on. Unlike some programming languages, JavaScript uses the same arithmetic rules for integers and floating point. Dividing 1 by 3 has the same answer as dividing 1.0 by 3.0.

```
var height = 5;
```

```
var width = 10;
```

```
var area = height * width;
```

```
var perimeter = 2 * height + 2 * width;
```

Like most languages, most floating point numbers are **inexact**, that is, they are correct only for about 7 decimal places. This is because of how numbers are represented internally as binary fractions.

Use Strings for Text Data

Like most programming languages, JavaScript has strings for holding text data. There are two ways to write a string:

- with double quotes, like this: "a string with double quotes"
- with single quotes, like this: 'a string with single quotes'

There is no difference internally. You can use either notation when entering strings. The only difference is that you can put a double quote inside a single-quoted string, and vice versa. But if you want to put a double quote inside a double-quoted string, or vice versa, you must **escape** the quote with a backslash, like this

```
var str1 = "Don't be silly!";
```

```
var str2 = 'Don\'t be silly!';
```

```
var str3 = 'John said "OK".';
```

```
var str4 = "John said \"OK\"";
```

Most modern JavaScript style guides recommend using single quotes consistently.

JavaScript has operations for concatenating strings, searching strings, trimming spaces from strings, and so on. Concatenation uses the addition operator.

```
var weather = 'Hot and dry';
```

```
var forecast = 'Today will be ' + weather;
```

You can concatenate numbers to strings. When JavaScript sees addition with a string and a number, it treats the number like a string.

```
var temp = 17;
```

```
var forecast = 'Today will be ' + temp + 'degrees Celsius';
```

Use Arrays for Lists

Like other language, JavaScript has **arrays**. An array is just a list of zero or more pieces of data. When written literally, a pair of square brackets delimits an array, and the **elements** are separated by commas.

```
var monthLengths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];
```

```
var monthNames = ["January", "February", "March", "April", "May", "June",
   "July", "August", "September", "October", "November", "December"];
```

The data can be anything, including numbers, strings, other arrays, or objects.

JavaScript has array operations for creating, combining, extracting elements, and so on.

Use Objects for Structured Data

Unlike languages like C, C++, or Java, an array can contain mixed types. You could have an array with numbers, strings, and arrays. Normally, this isn't a good idea, because it leads to hard to maintain code. For example, the following, while legal JavaScript, would be a bad way to represent data about months of the year.

```
var monthLengths = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];
```

```
var monthNames = ["January", 31, "February", 28, "March", 31, "April", 30,
  "May", 31, "June", 30, "July", 31, "August", 31,
  "September", 30, "October", 31, "November", 30, "December", 31];
```

In our code, we'd have to remember that to get information about the 4th month, we need to look at the 7th element for the name and the 8th element for the length.

Grouping the names and lengths together would be slightly better, but still not great.

```
var months = [["January", 31], ["February", 28], ["March", 31], ["April", 30],
```

```
  ["May", 31], ["June", 30], ["July", 31], ["August", 31],
  ["September", 30], ["October", 31], ["November", 30], ["December", 31]];
```

Now at least we can get information about the 4th month from the 4th element, but that information is in an subarray. The fact that the name is in the first element of that subarray and the length is in the second element would have to be documented for clarity.

The right way to represent structured data, i.e., data with parts with different roles, is to use **objects**. An object is a list of **key-value** pairs. The key is a simple string and the value can be anything, including an array or another object. A colon is used to separate the key from its value, and a comma is used to separate key-value pairs. The object is wrapped inside curly braces.

```
var person = { "name": "John", "age": 25};
```

Not all languages have an object data type,. Those that do use various names. Python calls them **dictionaries**. Ruby calls them **hashes**. Java has something similar, called a **Map**, but they're not as simple to create and use.

If an object key doesn't contain spaces or punctuation, it can be written without string quotes, e.g., `{ name: "John", age: 25 }`

> This is not true of JavaScript Object Notation (JSON) used to pass information between a web server and client. Only string keys are allowed in JSON.

With objects, we have a much clearer, if somewhat longer, way to represent month information.

```
var months = [ {name: "January", length: 31}, {name: "February", length: 28},
  {name: "March", length: 31}, {name: "April", length: 30}, {name: "May", length: 31},
  {name: "June", length: 30}, {name: "July", length: 31}, {name: "August", length: 31},
  {name: "September", length: 30}, {name: "October", length: 31},
  {name: "November", length: 30}, {name: "December", length: 31} ];
```

Now, when we get the information for the 4th month, we have data labeled with the keys **name** and **length**.

For more on objects, see the Objects section.

Converting Objects to Strings and Back

Occasionally, you need to convert a JSON object into string form. For example,

- To display a JSON object on a web page for debugging purposes, you need to make a string version.
- To send JSON data to a server over the web, you or some function you use has to create a string that can be sent to the server.
- To store JSON data in a browser's local storage, you need to save it as a string.

Fortunately, JavaScript provides a function, **JSON.stringify()** to do the work for you. There are two common ways to call it.

First, suppose our data is stored in the variable **bookData**. Try executing the following in the developer console:

```
var bookData = {
    "book_id": "79141967",
    "title": "Ysabel",
    "author_lf": "Kay, Guy Gavriel",
    "author_fl": "Guy Gavriel Kay",
    "author_code": "kayguygavriel",
    "ISBN": "0143016695",
    "ISBN_cleaned": "0143016695",
    "publicationdate": "2007",
    "entry_stamp": "1318989423",
    "entry_date": "Oct 18, 2011",
    "copies": "1",
    "dateacquired_date": "Dec 31, 1969"
};
```

If you just want to create a string for storing **bookData** or sending it to a server, do

```
JSON.stringify(bookData);
```

In the Chrome developer console, this will return the following string.

```
"{"book_id":"79141967","title":"Ysabel","author_lf":"Kay, Guy Gavriel","author_fl":"Guy Gavriel
Kay","author_code":"kayguygavriel","ISBN":"0143016695","ISBN_cleaned":"0143016695","publicationdate":"2007","entry_stamp":"1318989423","entry_date":"Oct
18, 2011","copies":"1","dateacquired_date":"Dec 31, 1969"}"
```

Though Chrome is working correctly, it's output is misleading. The above is not a valid string. If you try the same thing in Firefox, you will see more accurate output that quotes the nested double quotes:

```
"{\"book_id\":\"79141967\",\"title\":\"Ysabel\",\"author_lf\":\"Kay, Guy Gavriel\",\"author_fl\":\"Guy Gavriel
Kay\",\"author_code\":\"kayguygavriel\",\"ISBN\":\"0143016695\",\"ISBN_cleaned\":\"0143016695\",\"publicationdate\":\"2007\",\"entry_stamp\":\"1318989423\",\"entry_date\":\"Oct
 18, 2011\",\"copies\":\"1\",\"dateacquired_date\":\"Dec 31, 1969\"}"
```

This is all you need for many purposes. If however you want a string that a developer could read quickly, you want something with indentation. To do this you call **JSON.stringify()** like this:

```
JSON.stringify(bookData, null, 2)
```

The **null** parameter says you don't want to do any special processing. That's the normal case. The **2** parameter says to use 2 spaces when indenting. That's also the normal case. The output of this call in Chrome is

```
"{
  "book_id": "79141967",
  "title": "Ysabel",
  "author_lf": "Kay, Guy Gavriel",
  "author_fl": "Guy Gavriel Kay",
  "author_code": "kayguygavriel",
  "ISBN": "0143016695",
  "ISBN_cleaned": "0143016695",
  "publicationdate": "2007",
  "entry_stamp": "1318989423",
  "entry_date": "Oct 18, 2011",
  "copies": "1",
  "dateacquired_date": "Dec 31, 1969"
}"
```

Again, note that Chrome is not showing the necessary backslashes on the nested double quotes.

These strings are not JSON objects. If you did

```
var str = JSON.stringify(bookData);
```

then **bookData.ISBN** would return "0143016695", but **str.ISBN** would be undefined.

To convert a string with JSON data back to a real JavaScript object, use **JSON.parse()**. For example,

```
var dataFromStr = JSON.parse(str);
```

will set **dataFromStr** to an object that is a copy of **bookData**. **dataFromStr.ISBN** will return "0143016695".