Learn web development

Useful string methods

see all contributors

Previous Overview: First steps Next

Now we've looked at the very basics of strings, let's move up a gear and start thinking about what useful operations we can do on strings with built-in methods, such as finding the length **of a text string, joining and splitting strings, substituting one character in a string for another, and more.**

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, an understanding of what JavaScript is.
Objective:	To understand that strings are objects, and learn how to use some of the basic methods available on those objects to manipulate strings.

Strings as objects

We've said it before, and we'll say it again — *everything* is an object in JavaScript. When you create a string, for example by using

```
1 var string = 'This is my string';
```

your variable becomes a string object instance, and as a result has a large number of properties and methods available to it. You can see this if you go to the `String` object page and look down the list on the side of the page!

Now, before your brain starts melting, don't worry! You really don't need to know about most of these early on in your learning journey. But there are a few that you'll potentially use quite often that we'll

look at here.

Let's enter some examples into a fresh console. We've provided one below (you can also open this console in a separate tab or window, or use the browser developer console if you'd prefer).

```
>
```

Finding the length of a string

This is easy — you simply use the `length` property. Try entering the following lines:

```
var browserType = 'mozilla';
browserType.length;
```

This should return the number 7, because "mozilla" is 7 characters long. This is useful for many reasons, for example you might want to find the lengths of a series of names so you can display them in order of length, or let a user know that a username he has entered into a form field is too long, if it is over a certain length.

Retrieving a specific string character

On a related note, you can return any character inside a string by using **square bracket notation** — this means you include square brackets (`[]`) on the end of your variable name. Inside the square brackets you include the number of the character you want to return, so for example to retrieve the first letter you'd do this:

```
browserType[0];
```

Computers count from 0, not 1! To retrieve the last character of *any* string, we could use the following

line, combining this technique with the `length` property we looked at above:

```
browserType[browserType.length-1];
```

The length of "mozilla" is 7, but because the count starts at 0, the character position is 6, hence us needing `length-1`. You could use this to, for example, find the first letter of a series of strings and order them alphabetically.

Finding a substring inside a string and extracting it

1. Sometimes you'll want to find if a smaller string is present inside a larger one (we generally say *if a substring is present inside a string*). This can be done using the `indexOf()` method, which takes a single parameter — the substring you want to search for. Try this:

   ```
   browserType.indexOf('zilla');
   ```

   This gives us a result of 2, because the substring "zilla" starts at position 2 (0, 1, 2 — so 3 characters in) inside "mozilla". Such code could be used to filter strings. For example, say we have a list of web addresses and only want to print out the ones that contain "mozilla".

2. This can be done in another way, which is possibly even more effective. Try the following:

   ```
   browserType.indexOf('vanilla');
   ```

   This should give you a result of `-1` — this is returned when the substring, in this case 'vanilla', is not found in the main string.

   You could use this to find all instances of strings that **don't** contain the substring 'mozilla', or **do,** if you use the negation operator, as shown below. You could do something like this:

   ```
   if(browserType.indexOf('mozilla') !== -1) {
     // do stuff with the string
   }
   ```

3. When you know where a substring starts inside a string, and you know at which character you want it to end, `slice()` can be used to extract it. Try the following:

   ```
   browserType.slice(0,3);
   ```

   This returns "moz" — the first parameter is the character position to start extracting at, and the second parameter is the character position after the last one to be extracted. So the slice happens from the first position, up to, but not including, the last position.

4. Also, if you know that you want to extract all of the remaining characters in a string after a certain character, you don't have to include the second parameter, rather, you only need to include the character position from where you want to extract the remaining characters in a string. Try the following:

```
browserType.slice(2);
```

This returns "zilla" — this is because the character position of 2 is the letter z, and because you didn't include a second parameter, the substring that was returned was all of the remaining characters in the string.

Note: The second parameter of `slice()` is optional: if you don't include it, the slice ends at the end of the original string. There are other options too; study the `slice()` page to see what else you can find out.

Changing case

The string methods `toLowerCase()` and `toUpperCase()` take a string and convert all the characters to lower or upper case, respectively. This can be useful for example if you want to normalize all user-entered data before storing it in a database.

Let's try entering the following lines to see what happens:

```
var radData = 'My NaMe Is MuD';
radData.toLowerCase();
radData.toUpperCase();
```

Updating parts of a string

You can replace one substring inside a string with another substring using the `replace()` method. This works very simply at a basic level, although there are some advanced things you can do with it that we won't go into yet.

It takes two parameters — the string you want to replace, and the string you want to replace it with. Try this example:

```
browserType.replace('moz','van');
```

Active learning examples

In this section we'll get you to try your handing at writing some string manipulation code. In each

exercise below, we have an array of strings, and a loop that processes each value in the array and displays it in a bulleted list. You don't need to understand arrays or loops right now — these will be explained in future articles. All you need to do in each case is write the code that will output the strings in the format that we want them in.

Each example comes with a *Reset* button, which you can use to reset the code if you make a mistake and can't get it working again, and a *Show solution* button you can press to see a potential answer if you get really stuck.

Filtering greeting messages

In the first exercise we'll start you off simple — we have an array of greetings card messages, but we want to sort them to list just the Christmas messages. We want you to fill in a conditional test inside the `if( ... )` structure, to test each string and only print it in the list if it is a Christmas message.

1. First think about how you could test whether the message in each case is a Christmas message. What string is present in all of those messages, and what method could you use to test whether it is present?
2. You'll then need to write a conditional test of the form *operand1 operator operand2*. Is the thing on the left equal to the thing on the right? Or in this case, does the method call on the left return the result on the right?
3. Hint: In this case it is probably more useful to test whether the method call *isn't* equal to a certain result.

- Happy Birthday!
- Merry Christmas my love
- A happy Christmas to all the family
- You're all I want for Christmas
- Get well soon

```
var list =
document.querySelect
or('.output ul');
list.innerHTML = '';
var greetings =
['Happy Birthday!',

'Merry Christmas my
love',
                  'A
happy Christmas to
all the family',

'You\'re all I want
for Christmas',

'Get well soon'];

for (var i = 0; i <
greetings.length;
```

Reset Show solution

Fixing capitalization

In this exercise we have the names of UK cities, but the capitalization is all messed up. We want you to change them so that they are all lower case, except for a capital first letter. A good way to do this is to:

1. Convert the whole of the string contained in the `input` variable to lower case and store it in a new variable.
2. Grab the first letter of the string in this new variable and store it in another variable.
3. Using this latest variable as a substring, replace the first letter of the lowercase string with the first letter of the lowercase string changed to upper case. Store the result of this replace procedure in another new variable.
4. Change the value of the `result` variable to equal to the final result, not the `input`.

> **Note**: A hint — the parameters of the string methods don't have to be string literals; they can also be variables, or even variables with a method being invoked on them.

- lonDon
- ManCHESTer
- BiRmiNGHAM
- liVERpoOL

```
var list =
document.querySelect
or('.output ul');
list.innerHTML = '';
var cities =
['lonDon',
'ManCHESTer',
'BiRmiNGHAM',
'liVERpoOL'];
for(var i = 0; i <
cities.length; i++)
{
  var input =
cities[i];
  // write your
code just below here
```

Reset Show solution

Making new strings from old parts

In this last exercise the array contains a bunch of strings containing information about train stations in the North of England. The strings are data items that contain the three letter station code, followed by some machine-readable data, followed by a semi-colon, followed by the human-readable station name. For example:

```
MAN675847583748sjt567654;Manchester Piccadilly
```

We want to extract the station code and name, and put them together in a string with the following structure:

```
MAN: Manchester Piccadilly
```

We'd recommend doing it like this:

1. Extract the three-letter station code and store it in a new variable.
2. Find the character index number of the semi-colon.
3. Extract the human-readable station name using the semi-colon character index number as a

reference point, and store it in a new variable.

4. Concatenate the two new variables and a string literal to make the final string.
5. Change the value of the `result` variable to equal to the final string, not the `input`.

- MAN675847583748sjt567654;Manchester Piccadilly
- GNF576746573fhdg4737dh4;Greenfield
- LIV5hg65hd737456236dch46dg4;Liverpool Lime Street
- SYB4f65hf75f736463;Stalybridge
- HUD5767ghtyfyr4536dh45dg45dg3;Huddersfield

```
var list =
document.querySelect
or('.output ul');
list.innerHTML = '';
var stations =
['MAN675847583748sjt
567654;Manchester
Piccadilly',

'GNF576746573fhdg473
7dh4;Greenfield',

'LIV5hg65hd737456236
dch46dg4;Liverpool
Lime Street',

'SYB4f65hf75f736463;
Stalybridge',
```

Reset Show solution

Conclusion

You can't escape the fact that being able to handle words and sentences in programming is very important — particularly in JavaScript, as web sites are all about communicating with people. This article has given you the basics that you need to know about manipulating strings for now. This should serve you well as you go into more complex topics in the future. Next, we're going to look at the last major type of data we need to focus on in the short term — Arrays.

Previous Overview: First steps Next