**Description**

AngularJS lets us format data or manipulate array collections through **filters**. In this example we'll take a look at the **formatting filters**.

To specify a filter in the HTML template, we can use the common data biding notation {{*expression*}} and add a pipe (|) after the expression to format its result, so we'll have something like this {{*expression* | *filter*}}. If the filter accepts some parameters, we can specify them in sequence with the colon (:) character {{*expression* |*filter:param1:param2*}}.

In point 1 we see a simple filter. It's the built-in uppercase formatting filter that formats the content of stringData turning it to uppercase.

In point 2 we show that we can chain multiple filters together. Each time a pipe character is encountered, the preceding result will be processed by the filter specified after the pipe. So, in the example, stringData is first turned to lowercase and then the limitTo filter trims the lowercase string. The limitTo filter accepts a parameter that in this case says that we want to keep the first 7 characters of the string.

If we prefer to apply our filter in JavaScript, point 3 shows how it can be done. We see that we can apply the date filter in three different ways, the first one with the usual syntax for the HTML template and the other two directly in JavaScript. Let's take a look at the two JavaScript filter functions. formatDate1 retrieves the date filter using the $filter variable. That variable is available for us using Dependency Injection and allows to retrieve any available filter. In the example we're saying: give me the datefilter function and apply the date and format parameters to the function.formatDate2 is a different way to achieve the same result. We can retrieve the datefilter through Dependecy Injection also specifying as a module function argument (the controller function in this case) the name of the filter with a Filter suffix. Here we've written dateFilter as a function argument because we want the filter nameddate. In formatDate2 then we simply apply to the dateFilter function the parameters that the filter needs (the date and a string representing the desired output format).

There might be cases where the built-in filters are not enough for us of course, but there's a solution. Point 4 shows how we can write a custom formatting filter. At the time of writing, the built-in currency filter is pretty useless since it has very limited options, so it's a good time to write our own custom currency formatting filter and we're going to call it customCurrency. We can register a custom filter with the module function named filter. The parameter of filter is a function that returns the actual filtering function. In this example we specify also the numberFilter argument because we want to retrieve the built-in number filter that is going to be useful in our formatting function. Our filter function accepts 5 parameters: the first one is the data that has to be formatted, while the others are easy to understand just reading their names. We've made all the parameters optional so at the beginning of the filter function we provide some default values in case the parameters are not specified in input. The description of the inner workings of the function is available in the comments inside the code. We also have an isNumeric utility function that helps us to check if a variable is actually a number or not. In the HTML template we can then use the customCurrency formatting filter with the default parameter values or specify our own separating them with the colon character.