STRING INTERPOLATION / TEMPLATE LITERALS

TEMPLATE STRINGS

Syntactically these are strings that use backticks (i.e. `) instead of single (') or double (") quotes. The motivation of Template Strings is three fold:

- String Interpolation
- Multiline Strings
- Tagged Templates

STRING INTERPOLATION

Another common use case is when you want to generate some string out of some static strings + some variables. For this you would need some **templating logic** and this is where **template strings** get their name from. Here's how you would potentially generate an html string previously:

```
var lyrics = 'Never gonna give you up';
var html = '<div>' + lyrics + '</div>';
```

Now with template strings you can just do:

```
var lyrics = 'Never gonna give you up';
var html = `<div>${lyrics}</div>`;
```

Note that any placeholder inside the interpolation (${ and }) is treated as a JavaScript expression and evaluated as such e.g. you can do fancy math.

```
console.log(`1 and 1 make ${1 + 1}`);
```

Lets take this real life practical example where we want to build a general SQL query to retrieve some table(s) rows .We need the table name to be variable ,not hard coded on the query string so we have two options :

The old way ,by using concatenation where we just append the table name variable to the static query string with the concatenation operator or +

```
buildSqlQuery(tableName):string
{
    let query = "select * from " + tableName ;
    return query;
}
```

Obviously ,this is will just work as you can expect but imagine if you have multiple variables that you need to put on the string ,the process will become cumbersome and error prone .

The modern approach is to use template strings that were created mainly for the sake of building strings out of static and variable parts .

Now lets modify our previous method to use the elegant template strings instead .

```
buildSqlQuery(tableName):string
{
    let query = `select * from ${tableName}`
```

```
        return query;
    }
```

As you can see we need to use two things, backticks and ${} interpolation.

Anything inside ${} will be evaluated as a TypeScript expression .

So strings are not static and dump container of characters anymore, TypeScript can now look, inside the strings, for any expressions and evaluate them .

MULTILINE STRINGS

Ever wanted to put a newline in a JavaScript string? Perhaps you wanted to embed some lyrics? You would have needed to ESCAPE THE LITERAL NEWLINE using our favorite escape character `\`, and then put a new line into the string manually `\n` at the next line. This is shown below:

```
var lyrics = "Never gonna give you up \
                  \n Never gonna let you down";
```

With TypeScript you can just use a template string:

```
var lyrics = `Never gonna give you up
Never gonna let you down`;
```

TAGGED TEMPLATES

You can place a function (called a `tag`) before the template string and it gets the opportunity to pre process the template string literals plus the values of all the placeholder expressions and return a result. A few notes:

- All the static literals are passed in as an array for the first argument.
- All the values of the placeholders expressions are passed in as the remaining arguments. Most commonly you would just use rest parameters to convert these into an array as well.

Here is an example where we have a tag function (named `htmlEscape`) that escapes the html from all the placeholders:

```
var say = "a bird in hand > two in the bush";
var html = htmlEscape `<div> I would just like to say : ${say}</div>`;

// a sample tag function
function htmlEscape(literals, ...placeholders) {
    let result = "";

    // interleave the literals with the placeholders
    for (let i = 0; i < placeholders.length; i++) {
        result += literals[i];
        result += placeholders[i]
            .replace(/&/g, '&amp;')
            .replace(/"/g, '&quot;')
            .replace(/'/g, '&#39;')
```

```
                .replace(/</g, '&lt;')
                .replace(/>/g, '&gt;');
        }

        // add the last literal
        result += literals[literals.length - 1];
        return result;
    }
```

STRING OR TEMPLATE TAGS

Another common use of template strings are tags. If you are not familiar with templating engines concepts from template strings are borrowed, then a tag is simply a function that can pre-process a string to present it in another format. For example, you can create tags for common things such as

- Capitalize a string.
- Escape some characters such as HTML tags.
- Convert it to uppercase.
- Convert it to lowercase.
- Format dates etc.
- There are countless uses of tags.

In TypeScript to create a tag, we just create a normal function and place it before the string

The tag function takes two parameters, an array of static strings and a variadic array of evaluated expressions

```
function uppercase(strings, ...values){
    //you can now convert the sub strings to uppercase ,
    //concatenate them and return an uppercase string
    //of the original
    //We are just going to return the set of substrings and evaluated
expressions
    return `[ ${strings} ] :  (${values})`;
}
```

Now let's use it

```
var upperCaseString = uppercase `i want this string to be converted to
uppercase`;
```

CONCLUSION

Template strings in TypeScript are borrowed from templating engines used mainly with server-side web frameworks to create templates. In this tutorial we have seen common and practical uses for template strings and how they can replace old methods for writing powerful utilities with clear and more readable code