Chapter 54: Template Literals

Template literals are a type of string literal that allows values to be interpolated, and optionally the interpolation and construction behaviour to be controlled using a "tag" function.

Section 54.1: Basic interpolation and multiline strings

Template literals are a special type of string literal that can be used instead of the standard `'...'` or `"..."`. They are declared by quoting the string with backticks instead of the standard single or double quotes: `` `...` ``.

Template literals can contain line breaks and arbitrary expressions can be embedded using the `${ expression }` substitution syntax. By default, the values of these substitution expressions are concatenated directly into the string where they appear.

```
const name = "John";
const score = 74;

console.log(`Game Over!

${name}'s score was ${score * 10}.`);
```

```
Game Over!

John's score was 740.
```

Section 54.2: Tagged strings

A function identified immediately before a template literal is used to interpret it, in what is called a **tagged template literal**. The tag function can return a string, but it can also return any other type of value.

The first argument to the tag function, `strings`, is an Array of each constant piece of the literal. The remaining arguments, `...substitutions`, contain the evaluated values of each `${}` substitution expression.

```
function settings(strings, ...substitutions) {
  const result = new Map();
  for (let i = 0; i < substitutions.length; i++) {
    result.set(strings[i].trim(), substitutions[i]);
  }
  return result;
}

const remoteConfiguration = settings`
  label    ${'Content'}
  servers  ${2 * 8 + 1}
  hostname ${location.hostname}
`;
```

```
Map {"label" => "Content", "servers" => 17, "hostname" => "stackoverflow.com"}
```

The `strings` Array has a special `.raw` property referencing a parallel Array of the same constant pieces of the template literal but *exactly* as they appear in the source code, without any backslash-escapes being replaced.

```
function example(strings, ...substitutions) {
  console.log('strings:', strings);
  console.log('...substitutions:', substitutions);
}
```

```
example`Hello ${'world'}.\n\nHow are you?`;
```

```
strings: ["Hello ", ".\n\nHow are you?", raw: ["Hello ", ".\\n\\nHow are you?"]]
substitutions: ["world"]
```

Section 54.3: Raw strings

The `String.raw` tag function can be used with template literals to access a version of their contents without interpreting any backslash escape sequences.

`String.raw`\n`` will contain a backslash and the lowercase letter n, while `` `\n` `` or `'\n'` would contain a single newline character instead.

```
const patternString = String.raw`Welcome, (\w+)!`;
const pattern = new RegExp(patternString);

const message = "Welcome, John!";
pattern.exec(message);
```

```
["Welcome, John!", "John"]
```

Section 54.4: Templating HTML With Template Strings

You can create an `HTML`...`` template string tag function to automatically encodes interpolated values. (This requires that interpolated values are only used as text, and **may not be safe if interpolated values are used in code** such as scripts or styles.)

```
class HTMLString extends String {
  static escape(text) {
    if (text instanceof HTMLString) {
      return text;
    }
    return new HTMLString(
        String(text)
            .replace(/&/g, '&amp;')
            .replace(/</g, '&lt;')
            .replace(/>/g, '&gt;')
            .replace(/"/g, '&quot;')
            .replace(/\\/g, '&#39;'));
  }
}

function HTML(strings, ...substitutions) {
  const escapedFlattenedSubstitutions =
      substitutions.map(s => [].concat(s).map(HTMLString.escape).join(''));
  const pieces = [];
  for (const i of strings.keys()) {
    pieces.push(strings[i], escapedFlattenedSubstitutions [i] || '');
  }
  return new HTMLString(pieces.join(''));
}

const title = "Hello World";
const iconSrc = "/images/logo.png";
const names = ["John", "Jane", "Joe", "Jill"];

document.body.innerHTML = HTML`
  <h1><img src="${iconSrc}" /> ${title}</h1>
```

```
  <ul> ${names.map(name => HTML`
    <li>${name}</li>
  `)} </ul>
`;
```

Section 54.5: Introduction

Template Literals act like strings with special features. They are enclosed by by the back-tick `` and can be spanned across multiple lines.

Template Literals can contain embedded expressions too. These expressions are indicated by a $ sign and curly braces {}

```
//A single line Template Literal
var aLiteral = `single line string data`;


//Template Literal that spans across lines
var anotherLiteral = `string data that spans
         across multiple lines of code`;

//Template Literal with an embedded expression
var x = 2;
var y = 3;
var theTotal = `The total is ${x + y}`;     // Contains "The total is 5"



//Comparison of a string and a template literal
var aString = "single line string data"
console.log(aString === aLiteral)                          //Returns true
```

There are many other features of String Literals such as Tagged Template Literals and Raw property. These are demonstrated in other examples.