Chapter 78: Escape Sequences

Section 78.1: Entering special characters in strings and regular expressions

Most printable characters can be included in string or regular expression literals just as they are, e.g.

```
var str = "ポケモン"; // a valid string
var regExp = /[A-Ωα-ω]/; // matches any Greek letter without diacritics
```

In order to add arbitrary characters to a string or regular expression, including non-printable ones, one has to use *escape sequences*. Escape sequences consist of a backslash ("\\") followed by one or more other characters. To write an escape sequence for a particular character, one typically (but not always) needs to know its hexadecimal character code.

JavaScript provides a number of different ways to specify escape sequences, as documented in the examples in this topic. For instance, the following escape sequences all denote the same character: the *line feed* (Unix newline character), with character code U+000A.

- `\\n`
- `\\x0a`
- `\\u000a`
- `\\u{a}` new in ES6, only in strings
- `\\012` forbidden in string literals in strict mode and in template strings
- `\\cj` only in regular expressions

Section 78.2: Escape sequence types

Single character escape sequences

Some escape sequences consist of a backslash followed by a single character.

For example, in `alert("Hello\nWorld");`, the escape sequence `\n` is used to introduce a newline in the string parameter, so that the words "Hello" and "World" are displayed in consecutive lines.

Escape sequence	Character	Unicode
`\b` (only in strings, not in regular expressions)	backspace	U+0008
`\t`	horizontal tab	U+0009
`\n`	line feed	U+000A
`\v`	vertical tab	U+000B
`\f`	form feed	U+000C
`\r`	carriage return	U+000D

Additionally, the sequence `\0`, when not followed by a digit between 0 and 7, can be used to escape the null character (U+0000).

The sequences `\\`, `\'` and `\"` are used to escape the character that follows the backslash. While similar to non-escape sequences, where the leading backslash is simply ignored (i.e. `\?` for `?`), they are explicitly treated as single character escape sequences inside strings as per the specification.

Hexadecimal escape sequences

Characters with codes between 0 and 255 can be represented with an escape sequence where `\x` is followed by the 2-digit hexadecimal character code. For example, the non-breaking space character has code 160 or A0 in base 16, and so it can be written as `\xa0`.

```
var str = "ONE\xa0LINE"; // ONE and LINE with a non-breaking space between them
```

For hex digits above 9, the letters `a` to `f` are used, in lowercase or uppercase without distinction.

```
var regExp1 = /[\x00-xff]/; // matches any character between U+0000 and U+00FF
var regExp2 = /[\x00-xFF]/; // same as above
```

4-digit Unicode escape sequences

Characters with codes between 0 and 65535 (216 - 1) can be represented with an escape sequence where `\u` is followed by the 4-digit hexadecimal character code.

For example, the Unicode standard defines the right arrow character ("?") with the number 8594, or 2192 in hexadecimal format. So an escape sequence for it would be `\u2192`.

This produces the string "A ? B":

```
var str = "A \u2192 B";
```

For hex digits above 9, the letters `a` to `f` are used, in lowercase or uppercase without distinction. Hexadecimal codes shorter than 4 digits must be left-padded with zeros: `\u007A` for the small letter "z".

Curly bracket Unicode escape sequences

`Version ≥ 6`

ES6 extends Unicode support to the full code range from 0 to 0x10FFFF. In order to escape characters with code greater than 216 - 1, a new syntax for escape sequences was introduced:

```
\u{???}
```

Where the code in curly braces is hexadecimal representation of the code point value, e.g.

```
alert("Look! \u{1f440}"); // Look! ????
```

In the example above, the code `1f440` is the hexadecimal representation of the character code of the Unicode Character *Eyes*.

Note that the code in curly braces may contain any number of hex digits, as long the value does not exceed 0x10FFFF. For hex digits above 9, the letters `a` to `f` are used, in lowercase or uppercase without distinction.

Unicode escape sequences with curly braces only work inside strings, not inside regular expressions!

Octal escape sequences

Octal escape sequences are deprecated as of ES5, but they are still supported inside regular expressions and in non-strict mode also inside non-template strings. An octal escape sequence consists of one, two or three octal digits, with value between 0 and 3778 = 255.

For example, the capital letter "E" has character code 69, or 105 in base 8. So it can be represented with the escape sequence `\105`:

```
/\105scape/.test("Fun with Escape Sequences"); // true
```

In strict mode, octal escape sequences are not allowed inside strings and will produce a syntax error. It is worth to note that `\0`, unlike `\00` or `\000`, is *not* considered an octal escape sequence, and is thus still allowed inside strings (even template strings) in strict mode.

Control escape sequences

Some escape sequences are only recognized inside regular expression literals (not in strings). These can be used to escape characters with codes between 1 and 26 (U+0001–U+001A). They consist of a single letter A–Z (case makes no difference) preceded by `\c`. The alphabetic position of the letter after `\c` determines the character code.

For example, in the regular expression

```
`/\cG/`
```

The letter "G" (the 7th letter in the alphabet) refers to the character U+0007, and thus

```
`/\cG`/.test(String.fromCharCode(7)); // true
```