

Strings and Regular Expressions

Relevant Links

- Flanagan's book, sections 3.2, 10.1, 10.2, (10.3 optional)
- MDN's guide on RegExps¹
- MDN's RegExp reference²
- MDN's String object reference³

Strings in Javascript

- String literals are formed by surrounding the text in single or double quotes: "a string", 'another string'.
- Escape characters are combinations that hold special meaning. Refer to table 3.1 in the book, or in MDN's String reference.
- The plus operator is overloaded to cause string concatenation. Other values will be coerced to strings if necessary.
- Strings can be accessed via array indexing: "hi there"[4] == "h", "hello".length == 5. There is no separate type for single characters.
- Strings are immutable: You cannot change their value, you can only create a new string.

String methods

charAt⁴ An alternative for accessing a specific location in the string

charCodeAt⁵ Returns the numeric Unicode value of the character at a specific location.

concat⁶ Concatenates strings

indexOf⁷ Search within the string for the first occurrence of a substring.

lastIndexOf⁸ Similar to indexOf, but finds the last occurrence instead.

split⁹ Split a string into an array of strings based on a separator parameter.

substr¹⁰ Extract a section of a string

substring¹¹ Very similar to substr, but slightly different arguments

toLowerCase¹² Converts to lower case

toUpperCase¹³ Converts to upper case

Regular Expressions in Javascript

- Regular expressions are patterns that allow us to express very intelligent searches in strings.

¹https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

²https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp

³https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

- The full language of regular expressions is quite rich and will take some time to fully digest. We will only scratch the surface here.
- A regular expression literal is marked by two forward slashes: `/stuffHere/`. The second slash may optionally be followed by the letters “i”, “g”, and/or “m”, which affect how the expression behaves.
 - i** the regular expression ignores case, so an “a” would match both lower case and upper case A’s.
 - g** the regular expression becomes “global”, so it will keep on searching for more matches after the first one.
 - m** the regular expression possibly matches across multiple lines.
- Between the two forward slashes, a number of elements can be present, each with its meaning. Here are the most important ones:
 - c** Any character except for a few special that we will mention further down matches itself. This includes numbers, letters, some punctuation etc.
 - \t** Matches a tab character.
 - \n** Matches a newline character.
 - \r** Matches a carriage return. Often appears before the newline character to denote end of line.
 - .** A dot matches any character at all, except for newlines. Precede the dot with a backslash if you want to match a literal dot.
 - \d** Matches a digit.
 - \D** Matches any non-digit.
 - \w** Matches an alphanumeric character or underscore.
 - \W** Matches any character that is not alphanumeric or underscore.
 - \s** Matches a whitespace character (spaces, tabs, newlines and a couple more).
 - \S** Matches a non-whitespace character.
 - ** Matches an actual backslash.
 - \c** For a character that has a special meaning by itself, escape it with a backslash if you want its literal meaning. e.g. `\\. \\?, *, \\[`.
 - [xyz]** Matches any of the characters inside the brackets. You can also indicate ranges, like `[a-z.0-9]`, which matches any lowercase character or digit or dot.
 - [^xyz]** Matches anything BUT what follows the caret.
 - ^** Doesn’t match an actual character, but it must match the beginning of a line.
 - \$** Doesn’t match an actual character, but it must match the end of a line.
 - \b** Doesn’t match an actual character, but it must match a word boundary. Useful for matching whole words. For instance `/\\bcat\\b/` will match in “12cat” but not in “cats”.
 - (...)** Treats the contents as a group. In many regular expression searches, you would be able to access the matched pieces. Also called *capturing parentheses*.
 - \n** where n is a number. Matches a previously matched group.
 - (?:...)** *Non-capturing parentheses*. Useful when you simply want to group some elements together, but do not care for capturing the result separately.
 - x*** Matches the expression x zero or more times.

Returns a boolean indicating whether there was a match or not.

Returns an array of the matched results, or null.

Returns the index of the first successful match, or -1 if there is no match.

Replaces a match with s2. Look at the documentation for details on this second argument.

Use the matches of r as separators to split the string at.

~~r.exec(s)~~ ^{15016, 18, 19, 20, 21, 22} methods operate on one item at a time, unless the “g” flag has been added to the regular expression.

- The array returned by these methods varies by method, and often contains extra properties. Consult the documentation for details.
- When working with regular expressions, it is always a good idea to create a list of test strings first, then see how the regular expression behaves on those strings.