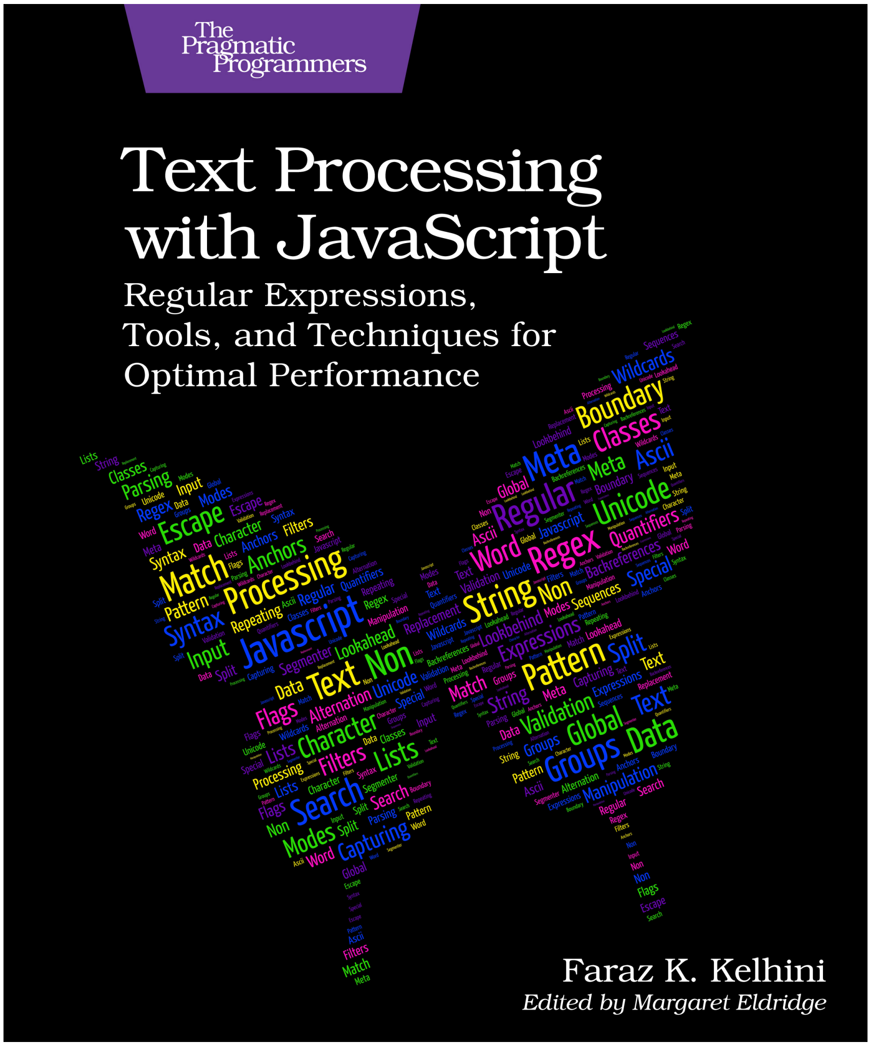

The Pragmatic Programmers

Text Processing with JavaScript

Regular Expressions, Tools, and Techniques for Optimal Performance

Faraz K. Kelhini

Edited by Margaret Eldridge

Recipe 40

Generating Indices for Matches with the d Flag

Task

Suppose you're building a tool that helps detect errors and potential problems in JavaScript code. You need your tool to be able to detect the use of reserved words in variables and functions and warn the user.

Ideally, you want to program your tool to pinpoint the exact part of the code where the reserved word is misused rather than outputting just a line number. So, if the code has a variable assignment with a reserved word like this:

```
let default = 7;
```

You want to indicate the error like this:

```
let default = 7;
    ↑------ Invalid variable name
```

To achieve this task, you need a regex that provides the start and end indices of the match.

Solution

Send the supplied code one line at a time to a function that looks for an invalid variable/function name. Use the `d` flag to obtain the start and end indices of the name:

part_2/flag_indices/indices_ex1.js

```
// The js code you want to check.
// In production, you'll likely use the FileReader API
// or a textarea to grab the code.
const code = `
  let a = 123;
  let b = 456;
  let default = 7;
`;

// A short list of js reserved words.
// A full list is available here:
// https://mzl.la/3XG92DO
const reserved = ["class", "default", "this", "case", "if"];
```

```
// Build a regex pattern with the reserved words
const re = new RegExp(`(?:var|let|const|function)\\s+(${reserved.join("|")})`,
                      "d");

// Find and display the location of the reserved word
function locateReservedWord(line) {
  const match = line.match(re);

  // If no match is found, return
  if (match === null) {return;}

  // Assign the start and end indices using the destructuring assignment.
  // indices[0] holds the indices of the matched string.
  // indices[1] holds the indices of the first capturing group.
  const [start, end] = match.indices[1];

  // Build the error message
  const error =
    " ".repeat(start) +    // Add spaces before the arrow
    "↑" +
    "-".repeat(end - start - 1) +
    " Invalid name (reserved word)";

  console.log(line);
  console.log(error);
}

// Split the code into separate lines,
// then send each line to locateReservedWord()
code.split(/\n|\r|\r\n/).forEach(line => {
  locateReservedWord(line);
});

// Logs:
// → let default = 7;
// →     ↑------ Invalid name (reserved word)
```

Your code can now indicate the exact position of a reserved word in a variable or function name.

Browser Compatibility

Despite being a newcomer to the regex family, the d Flag is supported by all modern browsers.[1] In the Node environment, you'll need a minimum version of 16.0.0 (Released 2021-04-20). To support older browsers, you can use a polyfill available in the regexp-match-indices package on NPM.[2]

1. https://mzl.la/3u78Y6w
2. https://www.npmjs.com/package/regexp-match-indices

Discussion

The hasIndices flag (d) indicates that the matching result should provide additional information about the start and end positions of each matched substring. The information will be stored in a property named indices. Consider this example:

part_2/flag_indices/indices_ex2.js
```
const str = "word1 word2";
const re = /word/dg;

console.log(re.exec(str).indices[0]);    // → [ 0, 4 ]
console.log(re.exec(str).indices[0]);    // → [ 6, 10 ]
```

When we set the d flag in a regex, an indices property will be available in the result of exec(), match(), and matchAll(). Here, we're using the exec() method, which is similar to match() except that it provides indices in the global mode too (see Appendix 2, Implementing Regex in JavaScript, on page ?).

The regex in this recipe requires using the RegExp() constructor because we're constructing the pattern dynamically with an array of reserved words. Any backslash in RegExp() must be escaped with another backslash. So, we write the shorthand character class to match whitespaces in the form of \\s rather than \s. Remember, if your dynamically created list contains a backslash, you must escape it too.

Also, pay attention to the second parameter of RegExp(). The RegExp() constructor uses a different approach to set the flags: it takes an optional second parameter containing the letters of the flags to set. Here, we want to set the hasIndices flag, so we pass d. As with the first argument, the second argument must be a string. Do not wrap it in slashes.

Let's analyze the regex in more detail:

```
(?:var|let|const|function)\\s+(${reserved.join("|")})
```

```
● (?:var|let|const|function) non-capturing group
  ○ 1st alternative: matches the characters "var" literally
  ○ 2nd alternative: matches the characters "let" literally
  ○ 3rd alternative: matches the characters "const" literally
  ○ 4th alternative: matches the characters "function" literally
● \\s matches any whitespace character
  ○ + matches the previous token between one and unlimited times
● (${reserved.join("|")}) 1st Capturing Group
  ○ ${reserved.join("|")} retrieves the array of reserved words and joins its
    items with a vertical bar, resulting in class|default|this|case|if
● Flags
  ○ d provides information about the start and end indices
```

Take advantage of the hasIndices flag to obtain information about the start and end positions of matches. Remember, when using the `RegExp()` constructor, you can't append flags to the regex pattern the way you typically do with regex literals. Instead, you should pass a string containing the flags as the second argument of the constructor.