Module 4

Handling DOM Events

Creating HTML Dynamically

Updating HTML Content Dynamically

Managing Web Forms with JavaScript

Form Validation

In this lesson we are going to look at:

- regular expressions.
- preventing the default behaviour of forms.
- simple form validation.

JavaScript Regular Expressions

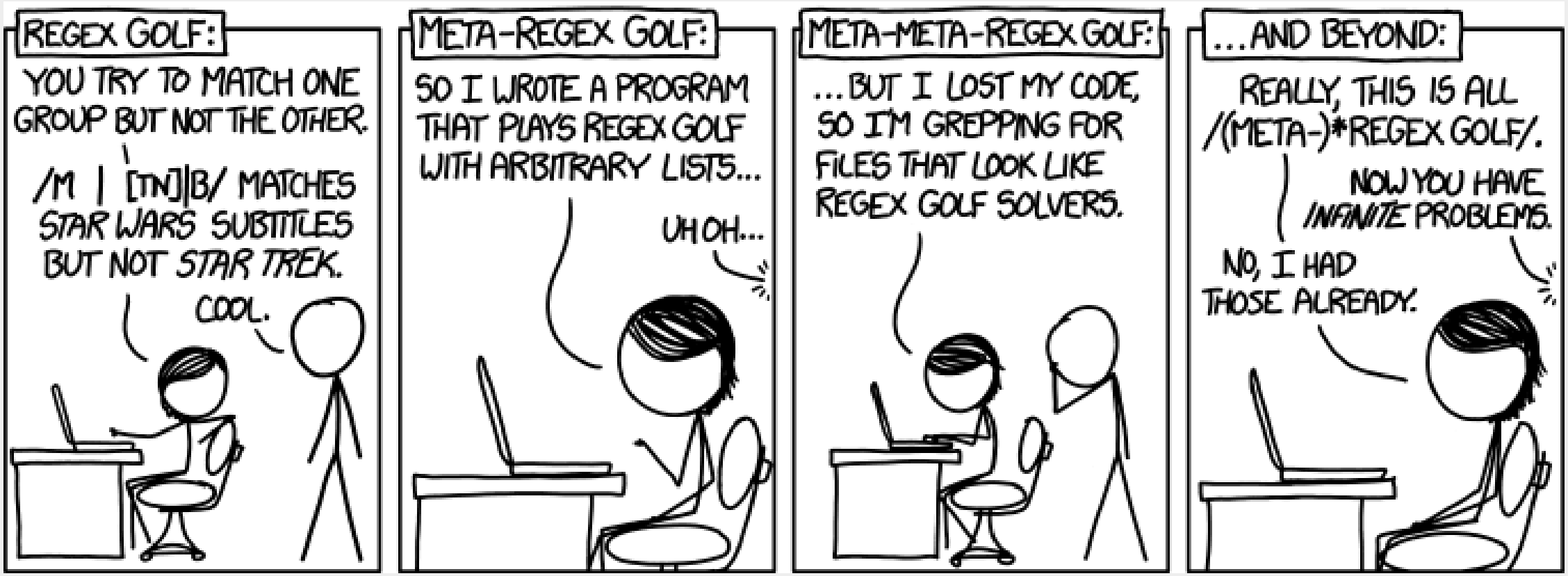
A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to **describe what you are searching for**.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of text search and text replace operations.

Regex golf



1313: Regex Golf Explanation

Regular Expression Syntax

```
/pattern/modifiers;
```

Example

```
let pattern = /noroff/i;
```

`/noroff/i` is a regular expression.

`noroff` is a pattern (to be used in a search).

`i` is a modifier (modifies the search to be **case-insensitive**).

Using String Methods

In JavaScript, regular expressions are often used with the two string methods: `search()` and `replace()`.

The `search()` method uses an expression to search for a match, and returns the position of the match.

The `replace()` method returns a modified string where the pattern is replaced.

Using String `search()` With a String vs RegEx

The `search()` method searches a string for a specified value and returns the position of the match:

Use a string to do a search for "Noroff" in a string:

```
let str = "The students at Noroff are future digital innovators.";
let pos = str.search("Noroff");
console.log(pos); // 16
```

Use a regular expression to do a **case-insensitive** search for "noroff" in a string:

```
let str = "The students at Noroff are future digital innovators.";
let pos = str.search(/noroff/i);
console.log(pos); // 16
```

Using String `replace()` With a String vs RegEx

The `replace()` method replaces a specified value with another value in a string:

```
let str = "Visit Sweden!";
let res = str.replace("Sweden", "Norway");
console.log (res); // Visit Norway!
```

Use a **case insensitive** regular expression to replace Sweden with Norway in a string:

```
let str = "Visit Sweden!";
let res = str.replace(/sweden/i, "Norway");
console.log (res); // Visit Norway!
```

Using `test()`

The `test()` method is a RegExp expression method.

It searches a string for a pattern, and returns `true` or `false`, depending on the result.

The following example searches a string for the character "e":

```
let str = "The best things in life are free!";
let pattern = /e/;
let x = pattern.test(str);
console.log (x); // true, because str contains an e
```

You don't have to put the regular expression in a variable first. The lines above can be shortened to one:

```
console.log (/e/.test("The best things in life are free!"));
// true
```

Using `exec()`

The `exec()` method is a RegExp expression method.

It searches a string for a specified pattern, and returns the found text as an **object**.

If no match is found, it returns an empty (null) object.

The following example searches a string for the character "e":

```
let obj = /life/i.exec("The best things in life are free!");
// [0: "life", index: 19, input: "The...", length: 1]

let obj2 = /life/i.exec("Life, the Universe and Everything");
// [0: "Life", index: 0, input: "Life...", length: 1]
```

Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches:

Modifier	Description
`i`	Perform case-insensitive matching
`g`	Perform a global match (find all matches rather than stopping after the first match)
`m`	Perform multiline matching

Regular Expression Patterns

Brackets are used to find a range of characters:

Expression	Description
`[abc]`	Find any of the characters between the brackets
`[0-9]`	Find any of the digits between the brackets
`(x|y)`	Find any of the alternatives separated with `|`

Regular Expression Patterns, cont.

Metacharacters are characters with a special meaning:

Metacharacter	Description
`\d`	Find a digit
`\s`	Find a whitespace character
`\S`	Find a non-whitespace character
`\b`	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
`\uxxxx`	Find the Unicode character specified by the hexadecimal number xxxx

Regular Expression Patterns, cont.

Quantifiers define quantities:

Quantifier	Description
`n+`	Matches any string that contains at least one n
`n*`	Matches any string that contains zero or more occurrences of n
`n?`	Matches any string that contains zero or one occurrences of n

Complete JavaScript RegExp Reference

Matching an e-mail with Regular Expression

The (too) simple approach:

```
/\S+@\S+\.\S+/
```

The HTML5 Specification approach (from ca. 2013)**:

```
/^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/
```

A more accurate approach***:

```
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]
+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.
[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/
```

The slightly insane RFC822 approach:

Mail::RFC822::Address: regexp-based address validation

** The forward slash `/` is a bit odd and will fail *** Line-breaks are added for readability, remove before using

The silver lining

You rarely have to make your own (non-trivial) regular expression patterns.

Most of the time you may find the proper patterns online:

- 9 Regular Expressions You Should Know
  - This article has very nice explantions of how the patterns work!
- Top 15 Commonly Used Regex
  - This article has a nice "cheat sheet"

Further reading

- Regular expressions @ mdn web docs
  - includes some nice tools

Event: preventDefault() method

The `preventDefault()` method of the `Event` interface tells the `user agent` that if the event does not get explicitly handled, its default action should not be taken as it normally would be.

One normal use is to stop a `form` from submitting onsubmit, so we can validate the form data, and - if OK - then let JS do the `submit()`;

Syntax

```
event.preventDefault()
```

What is form validation?

Go to any popular site with a registration form, and you will notice that they provide feedback when you don't enter your data in the format they are expecting. You'll get messages such as:

- **"This field is required"** (You can't leave this field blank).
- **"Please enter your phone number in the format xxx-xxxx"** (A specific data format is required for it to be considered valid).
- **"Please enter a valid email address"** (the data you entered is not in the right format).
- **"Your password needs to be between 8 and 30 characters long and contain one uppercase letter, one symbol, and a number."** (A very specific data format is required for your data).

This is called **form validation**. When you enter data, the browser and/or the web server will check to see that the data is in the correct format and within the constraints set by the application. Validation done in the browser is called **client-side validation**, while validation done on the server is called **server-side validation**.

Different types of client-side validation

There are two different types of client-side validation that you'll encounter on the web:

- **Built-in form validation** uses HTML form validation features, which we've discussed in many places throughout this module. This validation generally doesn't require much JavaScript. Built-in form validation has better performance than JavaScript, but it is not as customizable as JavaScript validation.
- **JavaScript validation** is coded using JavaScript. This validation is completely customizable, but you need to create it all (or use a library).

Demos: Using Regular Expression for form validation

Demo of two (very slightly) different approaches to HTML form validation done by JavaScript.

Level 2: Form validation with VueJS

Todos

Github Classroom

JS1 Lesson 4.2 Creating HTML Dynamically

JS1 Lesson 4.3 Updating HTML Content Dynamically

JS1 Lesson 4.4 Managing Web Forms with JavaScript

Mollify

Read Managing Web Forms with JavaScript, and do the Lesson Task

Read

Client-side form validation

Solutions for all the GitHub Classroom lessons will be covered some time next week.