TWeb

Regular Expressions and Web Applications

Bertil Chapuis

Overview of Today's Class

- Quiz about last week's lecture
- Correction of last week's assignment
- Regular Expressions
- HTML Forms
- Web Applications with Express
- Introduction of next week's assignment

Quiz

Feeling the Quiz Fatigue?

Speakup will **comeback**...

Sooner or later!

Coding Challenge

Ecrivez un programme qui, étant donné les titres de livres suivants, crée un index inversé.

```
var stopwords = ["the", "and", "a", "in"]
var books = ["Fantastic Mr Fox", " The Fox and the Star", "The Fox and the Hound", "Fox in Socks", "Maybe a Fox"];

var index = books.map().flatMap().reduce();

var stopwords = ["the", "and", "a", "in"]
var books = ["Fantastic Mr Fox", " The Fox and the Star", "The Fox and the Hound", "Fox in Socks", "Maybe a Fox"];

// Solution presented in class
var index = books
    .map(book => book.toLowerCase().split(" "))
    .map(words => words.filter(word => word != '' && !stopwords.includes(word)))
    .flatMap((words, position) => words.map(word => [word, position]))
    .reduce((dict, [word, position]) => {
        dict[word] = dict[word] || [];
        dict[word].push(position);
        return dict;
    }, {});

index["fox"] // [0, 1, 2, 3, 4]
index["fantastic"] // [0]
index["the"] // undefined
```

Correction

✋ Ex5: Express and Form Validation

Questions ?

Regular Expressions

JS Regular Expressions *

Regular expressions are patterns used to **match** and **extract** character combinations in strings.

It is usefull for validating inputs, parsing files, extracting information from free text.

For instance, given the format for registration plates in Switzerland:

- Validate that the given string a valid registration plate
- Extract all the registration plates listed in a unstructured text

Format for registration plates in Switzerland:

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

Building Regular Expressions *

The following notations can be used to define a regular expression in Javascript.

```
var re = /ab+c/;
var re = new RegExp(/ab+c/);
```

A regular expression is usually built in a **recursive** fashion with the following constructs:

- **Character Classes** (`.`, `\s`, `\d`) that distinguish types of chararters (letters or digits)
- **Character sets** (`[a-z]`) that match any of the enclosed characters
- **Either operator** (`x|y`) that match either the left or right handside values
- **Quantifiers** (`*`, `+`, `?`, `{n}`, `{n,m}`) that indicate the number of times an expression matches
- **Boundaries** (`^`, `$`) that indicate the beginnings and endings of lines and words
- **Groups** (`()`, `(?<name>)`, `(?:)`) that extracts and remember (or not) information from the input
- **Assertions** (`x(?=y)`) that helps at defining conditional expressions

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

JS Fun with Flags *

Regular expressions have optional flags that allow for functionality like global and case insensitive searching.

```
var re = /ab+c/; // no flag
var re = /ab+c/g; // global search
var re = /ab+c/i; // case-insensitive search
var re = /ab+c/m; // multi-line search

var re = /ab+c/gi // global case-insensitive search
```

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

JS Executing Regular Expressions *

The following notations can be used to execute regular expressions.

```
var re = /ab+c/g;

re.test("ac"); // false
re.test("abbc"); // true

re.exec("ac"); // null
re.exec("abbc"); // ["abbc"]

"abc abbc".matchAll(re); // [["abc"], ["abbc"]]
```

In addition to `matchAll`, a string comes with the `match`, `replace`, `search` and `split` methods.

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

Hands-on with Regular Expressions

- Try the afformentionned constructs with an online regex evaluator.
- Write a regular expression that extracts the canton and the number of a Swiss registration plates.
- Write a regular expression that extracts a list of Swiss registration plates from a free text.

* https://regexr.com

HTML Forms

HTML Forms *

HTML Forms are one of the main points of interaction between a user and a web site or application. Forms allow users to enter data, generally sending that data to the web server.

The `<form>` element defines a form.

```
<form action="/articles/" method="POST">
    <label for="email">Author's e-mail:</label>
    <input type="email" id="email" name="email">
    <label>Title:
        <input type="text" id="title" name="title">
    </label>
    <label for="content">Content:</label>
    <textarea id="content" name="content"></textarea>
    <button type="submit">Save</button>
</form>
```

- The `action` attribute defines the location (URL) where the form should be sent when it is submitted.
- The `method` attribute defines which HTTP method to send the data with (it can be "get" or "post").

* https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Your_first_HTML_form

JS HTML Input Element *

How an `<input>` works varies considerably depending on the value of its type attribute.

Some of the available types are: `text`, `password`, `submit`, `hidden`, `radio`, `checkbox`, `file`, `email`, `url`, `date`, `url`, etc.

- The `name` attribute specifies a name for the value, which is used when the form is submitted.
- The `value` attribute specifies the default value of the `<input>`.
- The `paceholder` attribute lets you specify a text that appears within the `<input>` element's content area itself when empty.
- The `required` attribute indicates that the `<input>` is mandatory.
- The `readonly` attribute indicates that the `<input>` cannot be edited.

* https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

ExpressJS

ExpressJS *

Fast, unopinionated, minimalist web framework for Node.js.

Web frameworks usually come with:

- A **middleware** layer that helps at customizing the behavior of the framework.
- A **router** that helps at defining the HTTP endpoints.
- A **template engine** that helps at rendering webpages.

Multi-page applications (MPA) are decreasing in popularity, but they almost always do the job. As the Web (as a platform) evolves slowly, MPAs are often cheaper and simpler to maintain.

* https://expressjs.com/

Let's play with these concepts *

```
const express = require('express')
const app = express()
const port = 3000

// Serve static files
app.use(express.static("public"));

// Define a Middleware
app.use('/form/', (req, res, next) => {
    console.log(req);
    next();
});

// Handle get requests
app.get('/form/', (req, res) => {
    res.send('Hello World!');
});

// Handle post requests
app.post('/form/', (req, res) => {
    res.send('Post saved!')
});

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

* https://github.com/tweb-classroom/example-express