ADVANCED JAVASCRIPT FOR WEB SITES AND WEB APPLICATIONS

Session 7 a

1. Regular expressions
2. Form validation

REGULAR EXPRESSIONS

Regular expressions are patterns that you can use to search in strings.

For example, you can check if a string contains certain characters or where they appear in the string. You can check if the value entered by a user an input field is a phone number or an email address.

You can define a regular expression the following way:

```
var reg = /web/;
```

This regular expression would find a match in the strings:

`"JavaScript is used to build wesites"`
and
`"spider's web"`.

It would not find a match in the following string:

`"we are bound to succeed"`

Regular expressions are built using simple patterns, as seen previously, using characters (letters, numbers).

You can also use special characters, that have special meanings in this context.

Special characters include:

- `^` matches the begining of a string
- `$` matches the end of a string
- `+` matches the preceding character 1 time or more
- `?` matches the preceding character 0 time or more
- `.` matches any single character
- `\d` matches any digit character
- `\D` matches any non-digit character

You will notice that some special character have to be escaped (`\`), so they can be used (`\D`, `\d`).

And if you wish to use special characters as literals, you need to escape them.

For example, if you want to find a question mark in a string, you need to look for: `/\?/`.

`/^W/` finds a match in "Web sites" but not "Sites on the web"

`/r$/` finds a match in "spider" but not "rest"

`/a+/` would match the a's in "blaaaaaaa", in "bla" but not in "spider"

`/a?/` would match the a's in "blaaaaaaa", in "bla" AND in "spider"

`/c.n/` finds a match in the strings "can", "con", "accent"

`/\d/` finds a match in "123", "32", "asc8765asd", but not "abc"

`/\D/` finds a match in "abc", but not "123"

To test a regular expression against a string, you can use the `test` method.

This method returns `true` if it finds a match or `false` if it doesn't.

```
var reg = /web/;

var myString = "websites";

reg.test(myString); // returns true
```

Exercise 1

Build a simple function that takes two arguments, a regular expression and a string, and returns `true` or `false`, if the expression finds a match in the string or not.

```
function regExMatch (myRegex, myString) {
  return myRegex.test(myString);
}
```

Exercise 2

Using the function from exercise 1, what do these statements return?

- `regExMatch(/We.?b/, "Web");`
- `regExMatch(/We.?b/, "web");`
- `regExMatch(/We.b/, "Wemb");`
- `regExMatch(/G.+\d+/, "Gilles 33");`
- `regExMatch(/G.+\d+/, "Gilles Guegan");`
- `regExMatch(/G.+(\d+)?/, "Gilles Guegan");`

- `regExMatch(/We.?b/, "Web"); // returns true`
- `regExMatch(/We.?b/, "web"); // returns false`
- `regExMatch(/We.b/, "Wemb"); // returns true`
- `regExMatch(/G.+\d+/, "Gilles 33"); // returns true`
- `regExMatch(/G.+\d+/, "Gilles Guegan"); // returns false`
- `regExMatch(/G.+(\d+)?/, "Gilles Guegan"); // returns true`

FORM VALIDATION EXAMPLE

Let's build a simple form validation plugin and test it on the html in validation.html

This is our HTML:

```
<form id="form" class="validate" action="" method="post">
  <legend>Test form</legend>
  <fieldset>
    <label for="name">First name: </label>
    <input name="name" type="text" class="required name" />
    <label for="email">Email: </label>
    <input type="text" class="required email" name=" email" />
    <input type="submit" class="submit" />
  </fieldset>
</form>
```

See a **working example**.

Using the revealing module pattern, with an init function, we can look for the form on the page, find the input fields that have a class of "required" and make sure these are none empty. This means the html fields we want to validate need to have a class of "required".

If a field is empty, we'll add a class of "invalid" to it, otherwise, we'll remove that class.

In the end, the script should check if there are input fields with an "invalid" class or not and submit the form based on this.

For the time being, we will only check if the fields are empty or not, so the regular expresison we'll need will be simple.

Validation - part 1

Start with the revealing module pattern below:

```
var VALIDATE = (function () {

  var
    init = function () {

    };

  return {
    init: init
  };

}());

VALIDATE.init();
```

Validation - part 2

Select the form on the page, by its class "validate".

Select the input fields in this form, that have a class of "required".

Create an object that will hold our regular expression pattern. For now, this object will only have one property, checking if a field is empty or not. The expression needs to match any character.

```
 var
    form = document.querySelectorAll("form.validate")[0],
    inputs =  form.querySelectorAll(".required"),
    expressionMatch = {
      notEmpty: /.+/
    },

[...]
```

Validation - part 3

Add the function we created earlier to the lsit of variables. The regExMatch function.

```
regExMatch = function (myRegex, myString) {
  return myRegex.test(myString);
},
```

Validation - part 4

In the `init` function, declare an event handler on the submit button, when the submit button in the form is clicked, a `validate` function should be called. We'll create that function in the next step.

```
form.querySelectorAll(".submit")[0].addEventListener(
  "click",
  validate,
  false
);
```

Validation - part 5

Create a `validate` function as another variable.

This function is used as the event handler for the handler we declared earlier, when the submit button is clicked.

The function needs to prevent the default behaviour (form submission); loop through the inputs we've previously selected; check if the field value is empty and add or remove a class of "invalid" based on this.

Remember that our `regExMatch` function returns `true` of `false`. We can use it directly in an `if` statement condition.

Once the loop is finished, the function will check the input fields again, and if there are none with a class of "invalid", will submit the form.

You can submit a form doing:

```
form.submit();
```

```
validate = function (event) {

  event.preventDefault();

  for (i = 0; i < inputs.length; i++) {

    if (regExMatch(expressionMatch.notEmpty, inputs[i].value)) {
      inputs[i].classList.remove("invalid");
    }
    else {
      inputs[i].classList.add("invalid");
    }

  }

  if(form.querySelectorAll(".invalid").length === 0) {
    form.submit();
  }

},
```

Validation - part 6

Let's improve the validation slightly and add two regular expressions to our regex object. One will check that a string matches an email address. The other will check that a string does not contain any digits, just characters (to match a name).

The first expression needs to look for characters, an `@`, more characters, a "." and more characters.

The second expression needs to look for any digit. We'll assume a name should not contain any digit.

```
regExpressions = {
  notEmpty: /.+/,
  name: /\d/,
  email: /.+@.+\..+/
},
```

Validation - part 7

Change the `validate` function we built previously to now also check the class on the input fields. If there is a class "name", use the inverse of the regular expresison matching a number. If the class is "email", use the regular expression to match against an email address.

```
for (i = 0; i < inputs.length; i++) {

  inputClasses = inputs[i].classList;

  if (inputClasses.contains("name")) {
    if(!regExMatch(expressionMatch.name, inputs[i].value)) {
      inputClasses.remove("invalid");
    }
    else {
      inputClasses.add("invalid");
    }
  }
  else if (inputClasses.contains("email")) {
    if(regExMatch(expressionMatch.email, inputs[i].value)) {
      inputClasses.remove("invalid");
    }
    else {
      inputClasses.add("invalid");
    }
  }
}
```