Lab 7 – Working with JS RegEx

1 Introduction

Welcome to Week 7!

Last week we introduced JavaScript event. This week we will expand on that understanding by looking at how JavaScript can be integrated into our websites for providing interactivity and client-side validation. The difficult level is ramping up from what we have done in previous weeks, so make sure that you take your time to understand the concepts and ask your tutor for help where necessary.

2 Learning Objectives

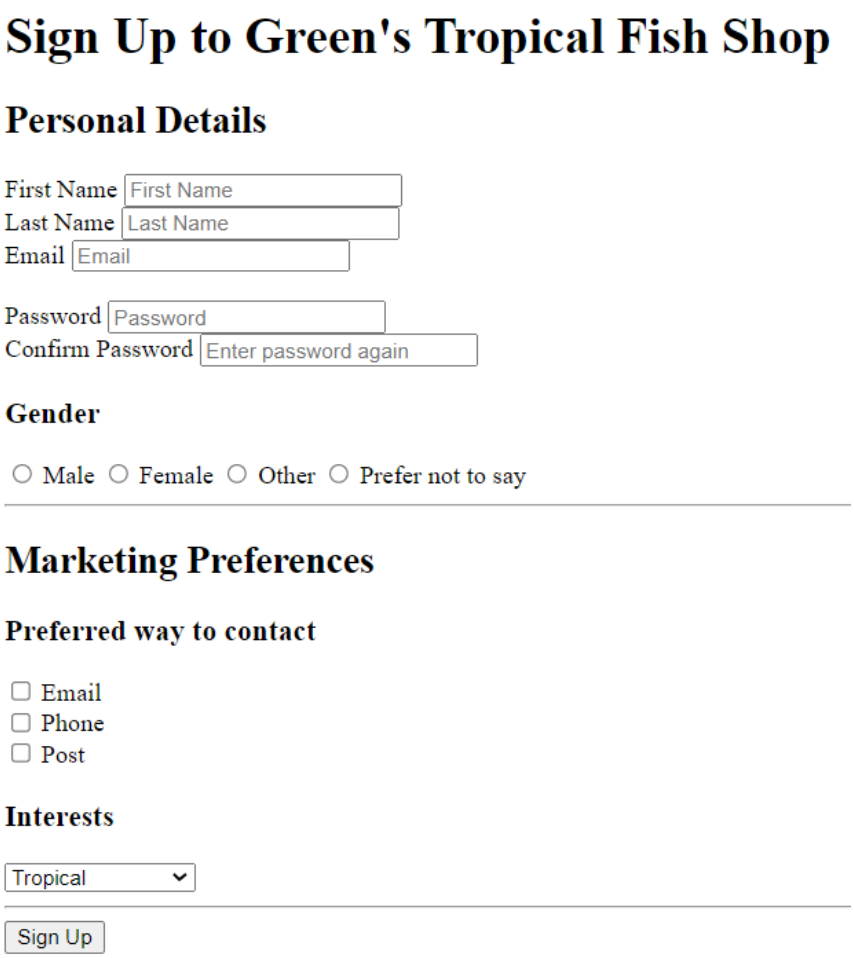
- To understand what JS is used for in web applications
- To understand the importance of input validation and how it differs from verification
- To understand what regular expressions are, when to use them and how to use them

3 Create a form for us to validate

3.1 Exercise 1: Create a HTML form

Create a new HTML document for today's exercises. Styling is optional, but you will need to link it to a JS file for later exercises. Create form that mimics the below screenshot. Make sure you include IDs for all fields as they will be needed later.

- You can add a horizontal line to your site with the self-closing <hr /> element (hr for horizontal rule)
- The options in the combo box can be "Tropical", "Coldwater", "Saltwater", "Ponds & Garden"



4 Practicing regular expressions

4.1 Exercise 2: Introduction to Regular expressions

Using the lecture notes and internet to help you, can you explain what each of the following regular expressions do. Use the code from the lecture notes to test each one.

#	RegEx
1	`tropical`
2	`^tropical$`
3	`^[A-Za-z]+$`
4	`^[A-Za-z ]+$`
5	`^[0-9]{11}$`

#	RegEx
6	`?=.{8,}`
7	`?=.*[A-Z]`
8	`?=.*[a-z]`
9	`?=.*[0-9]`
10	`[^A-Za-z0-9]`

Can you write a single RegEx that combines numbers 6 to 10 in the table above? Why would we do this?

4.2 Exercise 3: Read an article

This short Medium article contains a good introduction to Regular Expressions. Read through the article, and then use the tables as a quick reference when completing the rest of this week's activities.

https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285

4.3 Exercise 4: Let's write some functions

In this exercise, we will write a function for each input within the form. Each function will take the relevant input as a parameter and return true if the input is valid, and false otherwise. Let's do the first one together.

Both the first and last names can be validated using the same function as they both validated in the same way. We need to write a function that:

- Checks the name is not null or empty (i.e. that the user has actually entered a name)
- Checks that the name is a string of characters with no numbers
- The name may contain a hyphen
- Names should be no more than 32 characters long (we do this as we need to allocate exact sizes when saving into a database)

First, we need to write a function that will take a string as a parameter and return a Boolean value. We can test our function by calling it and logging the output to the console:

```
const validate_name = (name) => {
    let isValid = true;

    return isValid;
}

console.log(validate_name('Yanlong'))
```

Before we go any further, let's add some more tests. We can create tests for a series of bad and good names. If we refresh the page to rerun the function, some of the tests will give the wrong output (because at the moment we return true every time). Writing our tests before we write our code is called "Test Driven Development" and is very important in industry. Once we have our tests, we write code until all the tests are passing.

```
console.log(validate_name('Yanlong') + ', Expected: true')
console.log(validate_name('yanlong') + ', Expected: true')
console.log(validate_name('yanlong01') + ', Expected: false')
console.log(validate_name('yanlong-zhang') + ', Expected: true')
console.log(validate_name('12345') + ', Expected: false')
console.log(validate_name('') + ', Expected: false')
console.log(validate_name(null) + ', Expected: false')
console.log(validate_name('yanlongzhangyanlongzhangyanlongzhang') + ', Expected: false')
```

As mentioned, with us just returning true, some of the tests fail:

```
true, Expected: true
true, Expected: true
true, Expected: false
true, Expected: true
true, Expected: false
true, Expected: false
true, Expected: false
true, Expected: false
```

So, we need to fix it. Let's start by making sure that any input that is undefined/null, empty or not a string returns false:

```
const validate_name = (name) => {
    let isValid = true;

    if(name === null || name === "" || typeof name !== "string"){
        isValid = false;
    }

    return isValid;
}
```

```
true, Expected: true
true, Expected: true
true, Expected: false
true, Expected: true
false, Expected: false
false, Expected: false
false, Expected: false
true, Expected: false
```

We're passing some of the tests now. Let's add a regular expression check to ensure the name contains no numbers, is between 1 and 32 characters, and can contain hyphens:

```
}

let re = /^[a-zA-Z\-]{1,32}$/;
if(re.test(name) === false){
    isValid = false;
}

return isValid;
```

Hyphens are special characters in regular expressions and often used (e.g., a-z). We specify that the string can contain hyphens by adding an escape backslash before it. Now, all the test should pass:

```
true, Expected: true
true, Expected: true
false, Expected: false
true, Expected: true
false, Expected: false
false, Expected: false
false, Expected: false
false, Expected: false
```

Now implement functions to validate other inputs from the form. Your validation should adhere to the requirements specification in the table below. Use Test Driven Development so that you can have confidence in your results.

#	Function name	Parameters	Validation Checks
1	`validate_email`	`email`	• Email can't be null, empty and has to be a string • Email can't be longer than 64 characters • Email must contain an @ and at least one '.'
2	`validate_password`	`password, confirm password`	• Password and confirm password must be present. Can't be null, empty and must be a string • Password and confirm password must match • Password must be at least 8 characters and no more than 32 • Password must contain at least one: lower case letter; uppercase letter, number and special character (Hint: use Exercise 2 to help you)
3	`validate_gender`	`gender`	• Gender must exist (can't be null) • Gender must be one of (Male, Female, Other, Prefer not to say)
4	`validate_contact_method`	`email, phone, post`	• At least one must be present (can be more than one)

4.4 Exercise 5: Regexone

Like we saw with Flexbox, there are a lot of resources online for learning different web technologies. Regexone is a website that contains a series of challenges. Each challenge requires you to write a RegEx pattern to progress onto the next challenge.

There are 15 challenges in total, can you do the first ten? Work with the person next to you.

https://regexone.com/ (hint: make sure you read the instructions on each challenge)

5 Now validate the form with JavaScript

5.1 Exercise 6: Validate the form input

Use the functions from Exercise 4 to validate the entire form in Exercise 3 when the user clicks the submit 'sign up' button. The following steps should guide you through the process:

1. Add an event listener that executes a function when the button is clicked
2. Inside the function, get the values from each of the inputs using the ID attributes
3. Using the functions from Exercise 4, validate each of your inputs to check if the form is valid
4. Use "e.preventDefault()" to stop the page from refreshing
5. Use the lecture slides to feedback to the user whether the form was valid or not
6. Can you provide specific error messages for each invalid input? This will require you to edit your functions so that they return a message as well as the isValid variable