Lab 02 - Javascript

✨ Intro

Introduction video [https://drive.google.com/file/d/125SZ_EqMi5FIPCbiNWinSkYq7xrHzhh1/view?usp=sharing]

JavaScript is the programming language of not only the browser, but also the server, native applications and even Arduino boards! [https://www.youtube.com/watch?v=6CmIidOxc2g]

In the browser

JavaScript in the browser has many uses:

- you can add **event listeners** and run code based on certain things happening - the user moving a mouse, clicking a button or resizing the window
- you can **interact and change HTML nodes** from simple things such as changing their content up to completely generating a page using only JavaScript. That's how modern web frameworks such as React.js work!
- you can **make network requests** and access resources on other websites
- you can **send native notifications** with various content
- you can **control media** such as audio on a page

And many, many more things. We'll go through each one of these things in the tasks, but first let's see how the language looks like.

JavaScript language basics

JavaScript is a language that looks very similar to languages that you might have used in the past, such as Java, C++ or Racket. As you'll see, some concepts are indeed very similar, although some things are a bit different.

Variables: let vs const vs var

In JavaScript there are three ways to declare a variable:

- `let x = 'hello'`
- `const y = "I won't change"`
- `var z = 'never use me like this'`

JavaScript doesn't have types - this means that a variable can take a value of any type, and can change types at will. This gives us a great deal of flexibility when working with the language. For example, the following is a valid JavaScript code:

```
let x = 'I am a string'
x = 5 + 5
```

In the list above we also see the **const** variable type. As you might expect, this type of variable is a *constant*, and its value won't change. For example, this code will fail:

```
const x = 5
x = 6 // Uncaught TypeError: Assignment to constant variable.
```

Constants are very useful when we want to ensure that a certain variable won't change (like in C, where we declare global constants) or when we want to maintain the answer from a certain function untouched.

Semicolons are optional

JavaScript does something called Automatic Semicolon Insertion [http://www.bradoncode.com/blog/2015/08/26/javascript-semi-colon-insertion/]. This feature means that we don't have to worry about semicolons, as the JavaScript interpreter will insert them automatically, but we can also use them if we feel like it.

Why is var so introverted ?

You may have noticed that `var` above isn't the most friendly type. This is because `var` isn't scope-limited, and this can cause some nasty things. For example, `var` allows you to declare the same variable twice:

```
var x = 5
var x = undefined
console.log(x) // Prints 'undefined'
```

`var` also does some nasty things such as not being block scoped [https://hackernoon.com/why-you-shouldnt-use-var-anymore-f109a58b9b70], which can cause a whole lot of problems, but we won't deal with those as they are not our focus here.

The moral here is simple: > #*** Don't use `var`! ***

Special values

As in any language, there are a couple of values that you might see around and that have a special meaning to them:

- `undefined` - this means that the variable has been declared but not initialized (`let x`)

```
var x
console.log(x) // undefined
```

- `NaN` - stands for *not a number*. This is set when doing invalid conversion operations:

```
let x = 'hello' * 3
console.log(x) // NaN
```

- `null` - returned from some functions when no response can be given.

```
let x = 'hello'.match('bye')
console.log(x) // null
```

Functions

Functions in JavaScript have a couple of interesting properties:

- they can take any number of arguments
- they can be passed around as variables
- they can be declared as variables
- they can be called asynchronously

Functions are usually declared in one of two ways:

- With the `function` keyword:

```
function add (a, b) {
  return a + b
}
```

- As an arrow function:

```
let sum = (a, b) => a + b
```

Arrow functions are a more compact form of writing a function, without declaring it explicitly. They have some interesting properties:

- if they have only one argument, the parantheses can be omitted:

```
let inc = a => a + 1
```

- they return anything that's after the arrow if there are no brackets:

```
let withoutBrackets = (a, b) => a + b
let withBrackets = (a, b) => { let sum = a + b }

console.log(withoutBrackets(1,2)) // logs 3
console.log(withBrackets(1,2)) // logs undefined because we
                               // don't return anything
```

Callbacks

A function that is passed as a parameter is usually called a callback. This is because that function is usually *called back* at a later time. For example:

```
function sayHi () {
  console.log('Hi!')
}

function iHaveACallback(callback) {
  setTimeout(
    () => callback(), // The function we want to call
    1000 // Miliseconds to wait before calling the function
  )
}

iHaveACallback(sayHi) // Will print 'Hi!' after 1 second
```

Try running the code above in your browser's developer console and see what happens.

Network requests using fetch

In order to do network requests, we're going to use the Fetch API [https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API]. This API is based on Promises. It looks weird at first, but promises are just another way to write callback functions. For example, a simple Fetch request looks like this:

```
fetch('https://jsonplaceholder.typicode.com/posts/1') // Make a GET request
  .then(response => response.json()) // Treat the response as JSON
  .then(json => console.log(json)) // Prints a JSON object
  .catch(e => console.log('Uh oh! An error occured'))
```

You can see that the Fetch promise has three parts:

- `fetch('https:...')` - This is the main call of the function. It specifies the URL we want to request
- `.then(...)` - This function specifies what we do after the request is done. We can have multiple chained `then` calls, and each function sent as a parameter will get the previous function's return value
- `.catch` - This function will run whenever an error occurs

Browser APIs

The browser allows us to use JavaScript in order to interact with the DOM. Let's see how we can do some simple operations with it!

Manipulating the DOM

We can manipulate the DOM by using functions available in the `document` global object:

```
const body = document.querySelector('body') // Get the page body, or any other HTML element
let testDiv = document.createElement('div') // Creates a div element

testDiv.textContent = 'I am a new div!' // Set the inner content of a div

body.appendChild(testDiv) // Adds the div as a child to the page
```

Try running the code above on a blank page and see what happens!

Event listeners

Apart from adding elements to the DOM, we can also listen to certain events. For example, assuming we have this HTML:

```
<body>
  <div class='clickme'>
    Click me!
  </div>
</body>
```

We can use the following JavaScript code to trigger a message to the user when the div is clicked:

```
let div = document.querySelector('.clickme') // Get the Div
div.addEventListener('click', () => alert('Hello!))
```

Tasks

For today, you'll going to have to do the following:

1. Download the zip file containing the tasks.
2. Go to the *tasks*/ folder, and complete all the functions marked with *TODO:* in the *index.js* file
3. In the *api*/ folder, you have the necessary functions of a command line API client for the typicode API. Fill in the functions and make it work by calling the addPost, getPosts and deletePost functions from the browser's command line
4. In the apiInterface/ folder, you will have to reuse the api/ files in order to make a Postman-like interface for the API. Add event listeners to the buttons and output the result from the API in the div in the HTML.
5. In the imageGif/ folder, you have an array of image locations and an img object. Can you make a GIF out of them?
6. Optional: In the notifyMe/ folder, you already have a setup for a notification request. Trigger a notification when pressing the button.

Feedback

Please take a minute to fill in the **feedback form [https://forms.gle/NuXCJktudGzf4rLg6]** for this lab.

se/labs/02.txt · Last modified: 2021/10/25 10:25 by marius.calapod