Notizen für Dienstag 09.05.2023

JS Array Methods / npm and Linting Basics

▼ Session 1 —> JS Array Methods

▼ Introduction

There’s certain methods in JavaScript that we can call from array variables. like e.g. `myArrayVariable.forEach()` the `forEach()` here represents an array method (we’ll learn more about `forEach()` in the **Learned Material** section) all array methods take a callback method, and the callback method on all array all can take the same amount and type of parameters `arrayElement`, `index`, `array` (in this exact order), their name could also be `curryWurst0` `curryWurst1`, `curryWurst2` doesn’t matter as long as we remember that the **first** parameter provides us with the iteration’s step element, the **second** parameter provides us the index of the current iteration step, and the **third** parameter provides us a copy of the original array (the one we just used to call the array method).

▼ Learned Material

▼ `forEach()`

There’s an array function named `forEach(callbackFunction)`. It returns nothing, it only iterates through the array and does the action provided in the callback function (which we provide as a parameter)

```
// Define array and initialize a value to it.
const myArray = [
  "Hello",
  "World",
  "How",
  "Are",
  "You"
]

// Iterate through the myArray array variable.
myArray.forEach((myArrayElement) => {
  // Prints "Hello" "World" "How" "Are" "You" (each word in a new line).
  console.log(myArrayElement);
});
```

In the code snippet above we define an array named `myArray` that contains 5 elements all of type `string`. Bellow it we call the array using the `forEach()` method and we pass to it a fat arrow function `() => {}` as the callback function parameter which the `forEach()` function requires. The `forEach()` then provides us through it's higher order function a value to the parameter of the fat arrow function we provided `(someValue) => {}`. Specifically the `forEach()` function provides us at the callback function parameter, the element, that the `forEach()` is at during it's iteration step so on the first iteration it provides the `myArray` element that contains the value `"Hello"` in the second iteration step it provides the `myArray` element that contains the value `"World"` and so on until it reaches the last element which contains the value `"You"`.

> 💡 It doesn't differ much from simply iterating through the array with a loop

▼ `map()`

There's an array function named `map(callbackFunction)`. It returns a new copy of the array, it iterates through the array and does the action provided in the callback function (which we provide as a parameter)

```
// Define array and initialize a value to it.
const myArray = [
  "Hello",
  "World",
  "How",
  "Are",
  "You"
]

// Iterate through the myArray array variable.
const saveHereAnEditedCopyOfMyArray = myArray.map((myArrayElement) => {
  // Transforms the myArrayElement value by adding an "A"
  // at the end of it.
  return (myArrayElement = myArrayElement + "A");
});

// Prints an array object that contains 5 elements with the values
// element 0: "HelloA"....element 4: "YouA".
console.log(saveHereAnEditedCopyOfMyArray);
// Prints an array object that contains 5 elements with the values
// element 0: "Hello"....element 4: "You".
console.log(myArray);
```

In the code snippet above we define an array named `myArray` that contains 5 elements all of type `string`. Bellow it we call the array using the `map()` method and we pass to it a fat arrow function `() => {}` as the callback function parameter which the `map()` function requires. The `map()` then provides us through it's higher order function a value to the parameter of the fat arrow function we provided `(someValue) => {}`. Specifically the `map()` function provides us at the callback function parameter, the element, that the `map()` is at during it's iteration step so on the first iteration it provides the `myArray` element that contains the value `"Hello"` in the second iteration step it provides the `myArray` element that contains the value `"World"` and so on until it reaches the last element which contains the value `"You"`. We than `return` to our map method on each iteration a string value where we simply add an `"A"` string value to the back of every element's value. This returned string is later, (in the background magic of the map method), applied to the corresponding element's value of the copy array that the `map()` method creates and finally this map's return value array is applied to the `saveHereAnEditedCopyOfMyArray` edited variable. Finally we `console.log` the 2 variables namely `saveHereAnEditedCopyOfMyArray` and `myArray`.

▼ `filter()`

There's an array function named `filter(callbackFunction)`. It returns a new array of the array, it iterates through the array and does the action provided in the callback function (which we provide as a parameter)

```
// Define array and initialize a value to it.
const myArray = [
  "Hello",
  "World",
  "How",
  "Are",
  "You"
];

// Iterate through the myArray array variable.
const saveHereAFilteredCopyOfMyArray = myArray.filter((myArrayElement) => {
  // Filters by the criteria that the myArrayElement value
  // contains at least 1 lowercase "w" in it.
  return myArrayElement.includes("w");
});

// Prints an array object that contains 1 element with the value
// element 0: "How".
console.log(saveHereAFilteredCopyOfMyArray);
// Prints an array object that contains 5 elements with the values
// element 0: "Hello"....element 4: "You".
console.log(myArray);
```

In the code snippet above we define an array named `myArray` that contains 5 elements all of type `string`. Bellow it we call the array using the `filter()` method and we pass to it a fat arrow function `() => {}` as the callback function parameter which the `filter()` function requires. The `filter()` then provides us through it's higher order function a value to the parameter of the fat arrow function we provided `(someValue) => {}`. Specifically the `filter()` function provides us at the callback function parameter, the element, that the `filter()` is at during it's iteration step so on the first iteration it provides the `myArray` element that contains the value `"Hello"` in the second iteration step it provides the `myArray` element that contains the value `"World"` and so on until it reaches the last element which contains the value `"You"`. We than `return` to our `filter()` method on each iteration a condition that needs to `return` `true` in order to supply the element we are at during the `filter()` method's iterations in new array copy that the `filter()` method returns at the end. Finally this `filter()`'s return value array is applied to the `saveHereAFilteredCopyOfMyArray` variable. Finally we `console.log` the 2 variables namely `saveHereAFilteredCopyOfMyArray` and `myArray`. The reason the `saveHereAFilteredCopyOfMyArray` only returns one element in it's array with the value `"How"` is because the `myArrayElement.includes("w");` will only `return` `true` when an element's value contains a lowercase `w`.

▼ Array method stacking combinations

You can combine array functions endlessly so you could say e.g. first transform elements with `map()` and then filter through them by certain criteria with `filter()` and then transform them again or filter them again if you want. Here's an example

```
// Define array and initialize a value to it.
const myArray = [
  "Hello",
  "World",
  "How",
  "Are",
  "You"
];

const saveHereAnEditedAndFilteredCopyOfMyArray = myArray
  .map((myArrayElement) => {
    // Transforms the myArrayElement value by converting it's
    // value to uppercase letters.
    return myArrayElement.toUpperCase();
  })
  .filter((myArrayElement) => {
    // Filters by the criteria that the myArrayElement value
    // contains at least 1 uppercase "L" or uppercase "E" in it.
    return myArrayElement.includes("L") || myArrayElement.includes("E");
  });

// Prints an array object that contains 3 elements with the values
// element 0: "HELLO",
// element 1: "WORLD",
// element 2: "ARE".
console.log(saveHereAnEditedAndFilteredCopyOfMyArray);
// Prints an array object that contains 5 elements with the values
// element 0: "Hello"....element 4: "You".
console.log(myArray);
```

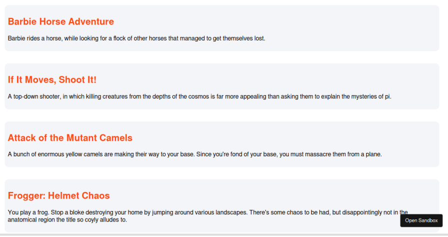
▼ Handouts and Challenges

Example code during session:

dreamy-brown-62q8bo - CodeSandbox

dreamy-brown-62q8bo by actyvystom

https://codesandbox.io/s/dreamy-brown-62q8bo?file=/js/index.js



Barbie Horse Adventure

Barbie rides a horse, while looking for a flock of other horses that managed to get themselves lost.

If It Moves, Shoot It!

A top-down shooter, in which killing creatures from the depths of the cosmos is far more appealing than asking them to explain the mysteries of pi.

Attack of the Mutant Camels

A bunch of enormous yellow camels are making their way to your base. Since you're fond of your base, you must massacre them from a plane.

Frogger: Helmet Chaos

You play a frog. Stop a bloke destroying your home by jumping around various landscapes. There's some chaos to be had, but disappointingly not in the anatomical region the title so coyly alludes to.

Handouts

https://github.com/neuefische/ffm-web-23-3/blob/main/sessions/js-array-methods/js-array-methods.md

JS Array Methods Session Handouts

Handout Resources:

- MDN web docs: Array forEach
- MDN web docs: Array map
- MDN web docs: Array filter
- MDN web docs: NodeList

Additional Resources:

- `join()` (from challenge 4)
- `document.querySelectorAll()` from last challenges

Challenges

https://github.com/neuefische/ffm-web-23-3/blob/main/sessions/js-array-methods/challenges-js-array-methods.md

JS Array Methods Session Challenges

Challenge	Solution
1. Color Boxes (`forEach`)	1. Color Boxes (`forEach`)
2. `map`	2. `map`
3. `filter`	3. `filter`
4. Repetition: `map` and `filter`	4. Repetition: `map` and `filter`

1. SPA 1: a working solution

- SPA (Single-page application) from last challenges

2. SPA 2: simplified with `forEach`
3. SPA 3: `forEach` within `forEach`

Last challenges require no work, it's only for observation and study

---

▼ Session 2 —> npm and Linting Basics

▼ Introduction

NPM stands for Node Package Manager, it's a collection of packages and modules which we can install in our applications, it's something like an app-Store for your developing your applications. The packages are open source but there is a possibility to restrict the access to those packages for specific users.
In this session we'll be learning how to install packages and how to work with it.
Initially we have to install node ideally the LTS (which stands for Long Term Support) the LTS versions are usually the most stable and hereby reliable ones, but the MacBook's we've received already have this installed so we can safely skip this step for now.

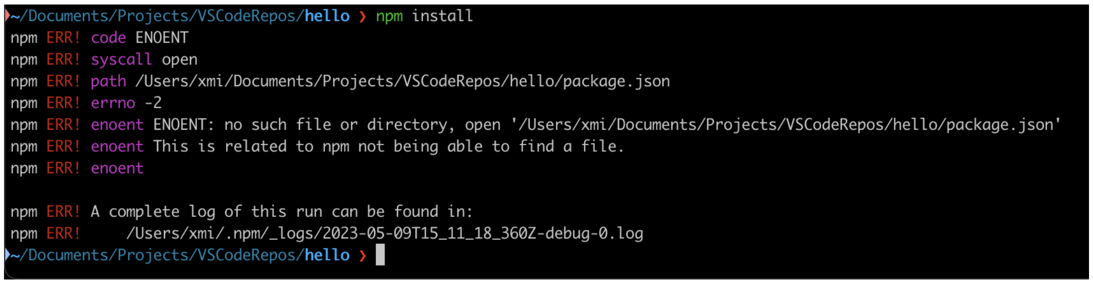
▼ Learned Material

▼ `npm install`

Is the core command to install all the package dependencies in the `node_modules` folder, (which is automatically created if previously didn't exist), within your project files which need to contain a `package.json` file.

E.g. for the session we've received this one to later work with in the challenges is this:

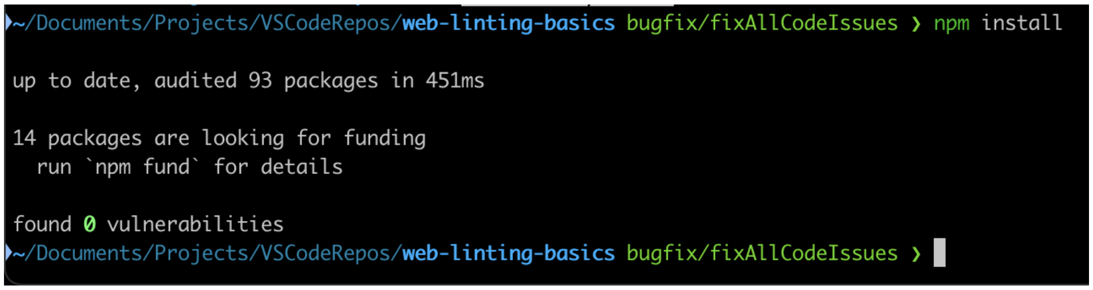
```
https://github.com/actyvystom/web-linting-basics-template
```



```
~/Documents/Projects/VSCodeRepos/hello ❯ npm install
npm ERR! code ENOENT
npm ERR! syscall open
npm ERR! path /Users/xmi/Documents/Projects/VSCodeRepos/hello/package.json
npm ERR! errno -2
npm ERR! enoent ENOENT: no such file or directory, open '/Users/xmi/Documents/Projects/VSCodeRepos/hello/package.json'
npm ERR! enoent This is related to npm not being able to find a file.
npm ERR! enoent

npm ERR! A complete log of this run can be found in:
npm ERR!     /Users/xmi/.npm/_logs/2023-05-09T15_11_18_360Z-debug-0.log
~/Documents/Projects/VSCodeRepos/hello ❯
```

If you would attempt to run `npm install` in a folder that does **not** contain a `package.json` file, you should a similar error message to this one.



```
~/Documents/Projects/VSCodeRepos/web-linting-basics bugfix/fixAllCodeIssues ❯ npm install

up to date, audited 93 packages in 451ms

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
~/Documents/Projects/VSCodeRepos/web-linting-basics bugfix/fixAllCodeIssues ❯
```

And here is the the case when you run `npm install` in a folder which **does** contain a `package.json` file.

▼ `package.json`

Looking into the `package.json` file we should something similar to this..

```
{
  "name": "web-linting-basics-template",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "npm run htmlhint && npm run prettier:check && npm run eslin
t",
    "fix": "npm run htmlhint && npm run prettier:write && npm run eslin
t",
    "htmlhint": "npx htmlhint \"**/*.html\"",
    "prettier:check": "npx prettier --check .",
    "prettier:write": "npx prettier --write .",
    "eslint": "npx eslint \"**/*.js\""
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/neuefische/web-linting-basics-templat
e.git"
  },
  "keywords": [],
  "author": "",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/neuefische/web-linting-basics-template/iss
ues"
  },
  "homepage": "https://github.com/neuefische/web-linting-basics-template#
readme",
  "devDependencies": {
    "eslint": "^8.16.0",
    "htmlhint": "^1.1.4",
    "prettier": "^2.6.2"
  }
}
```

For the time all we need to care about is the `"scripts"` and the `"devDependencies"` no need to worry about everything else for now.
In the

In the `scripts` we can see properties like `test`, `fix`, `htmlhint`, and so on. These are variables representing commands we can call that contain a bunch of other commands as values, in other words they are shortcuts. You can run a script command with the keywords `npm run` followed by the shortcut command you want to run, e.g. `npm run htmlhint` which in turn runs the assigned value of `htmlhint` which is `npx htmlhint \"**/*.html\"`. This specific command would call for the linter to run all your project's html files which then checks for error and reports them back to you. This uses the `htmlhint` which we can also find down at the `devDependencies` section.

In the `devDependencies` we define packages which will assist us in the development, like those linters (`eslint` and `htmlhint`) and other tools like the `prettier` which only checks for the formatting for the code. Each of the written `devDependencies` packages has a number written next to them, this represent the version of the given package we seek for during the run of `npm install` so the right versions of dependencies are installed for our project in the `node_modules` folder.

▼ `node_modules`

Here is the location where all our installed packages are landing in, make sure you mention `node_modules` in your `.gitignore` file because this folder is or can be very heavy and literally all the person who clones the project has to do while setting it up is run the command `npm install` on it to have those packages installed.

▼ VS Code Extensions

Here are some useful extensions that will make your life easier also known as QoL (Quality of Life)

- Prettier - Code formatter
- ESLint
- HTMLHint

💡 I highly recommend reading the handouts for more details

▼ Handouts and Challenges

REQUIRED FOR CHALLENGES:

https://github.com/actyvystom/web-linting-basics-template

💡 Make a new repo out of this for your own GitHub, and `git clone` it with the respective link to your git repo containing a copy of this repo.

Handouts

Challenges

https://github.com/neuefische/ffm-web-23-3/blob/main/sessions/npm-and-linting-basics/npm-and-linting-basics.md

npm and Linting Basics Session Handouts

Handout Resources:

- About npm:
  - npm website
  - package.json specification
  - npm install documentation
  - Semantic Versioning specification
- Linters:
  - HTMLHint
  - Prettier
  - ESLint
- VSCode Plugins for Linters:
  - HTMLHint
  - Prettier
  - ESLint

https://github.com/neuefische/ffm-web-23-3/blob/main/sessions/npm-and-linting-basics/challenges-npm-and-linting-basics.md

npm and Linting Basics Session Challenges

Challenge

1. Instead of the “this link” use the template repo provided above. or here (it’s the same link)

https://github.com/actyvystom/web-linting-basics-template

Solution

The solution should look like this

https://github.com/dimitriosxmi/web-linting-basics

or as a deployed GitHub page, (which you don’t have to do, it’s not part of the challenge), should look like this sample-title

https://dimitriosxmi.github.io/web-linting-basics/

if you wanna deploy it like this, Thomas just made a screenshot guide you can find it here

https://github.com/neuefische/ffm-web-23-3/tree/main/week4/pages-guide