Advanced Web Programming

Lab4 - data driven

Array, objects, Higher-order function and JSON

Goals: The topic of today's classes is data processing in java script. Let's focus on processing data stored in arrays and objects. We will learn about the possibilities of the Higher-order function for processing data collections. In the second part of the class we will deal with the problem of feeding data from external files. Using AJAX and the json format, we will load and display in our application data from an external json file.

Section 1. Array methods (Manipulation and Searching)

In Javascript, there are only 6 data types defined – the primitives (boolean, number, string, null, undefined) and object (the only reference type). Arrays do not belong to this list because they are objects as well. This is a common confusion among developers who assume that arrays are a special data type in Javascript.

The items in the array are nothing more than properties of that objects. You can use any name as property for an objects, including numbers or even empty string. However, if the property name is not a valid identifier (doesn't start with a letter), you can only access it through `obj[property_name]` and not `obj.property_name`.
So nothing stop us from taking an object and declaring properties like 0, 1, 2 and so on and accessing through `obj[0]`, `obj[1]` and so on.
An array has, alongside its elements, a special property named `length` and a generous **collection of methods for manipulating** the elements. They represent the real power of arrays. You can search, sort, delete, insert and even simulate behavior of other collection data types, like queue and stack.

Remember:

- arrays are normal objects and their elements are properties with names 0, 1, .. etc.
- arrays also have a special properties lenght and many function that can manipulate the elements

More about Array -> https://www.w3schools.com/js/js_array_methods.asp or my lecture nr 3.

Exercise 1. In catalog Example 1 you find three files. File app.js contains array of objects. Details description what to do you can find in app.js file. Execute the command.

Exercise 2. In catalog Example 2 you find three files. File app.js contains array of objects. Details description what to do you can find in app.js file. Execute the command.

Exercise 3. In catalog Example 3 you find three files. File app.js contains array of objects. Details description what to do you can find in app.js file. Execute the command.

Exercise 4. In catalog Example 4 you find three files. File app.js contains array of objects. Details description what to do you can find in app.js file. Execute the command.

Section 2. JSON – How to load data from external file

Exercise 5. In Catalog Example 5 you can find some files. In html file you find two select options. Write a script which load data from file people.json to app and display data on the page.

solution with explanation

I show you how to fetch and display data from a JSON file using vanilla JavaScript.

First, we will fetch the JSON data by using the fetch API. This will return a promise with our JSON data. Then we will append the data dynamically by creating HTML elements on the fly. We will then append our JSON data to those elements.

Getting JSON data from an API and display it on a web page is a common thing you will do quite often.

You have a people.json file. It is fill with the following data:

```
[
  {
    "id": "1",
    "firstName": "John",
    "lastName": "Doe"
  },
  {
    "id": "2",
    "firstName": "Mary",
    "lastName": "Peterson"
  },
  {
    "id": "3",
    "firstName": "George",
    "lastName": "Hansen"
  }
]
```

Save this file in the same directory as our index.html file.

Fetching the JSON data

To be able to display this data in our HTML file, we first need to fetch the data with JavaScript.

We will fetch this data by using the fetch API. We use the fetch API in the following way:

```
fetch(url)
  .then(function (response) {
    // The JSON data will arrive here
  })
```

```
  .catch(function (err) {
    // If an error occured, you will catch it here
  });
```

The url parameter used in the fetch function is where we get the JSON data. This is often an http address. In our case it is just the filename people.json. We don't have to drill down to any directory since the json file is in the same directory

The fetch function will return a promise. When the JSON data is fetched from the file, the then function will run with the JSON data in the response.

If anything goes wrong (like the JSON file cannot be found), the catch function will run.

Let us see how this will look in out example:

```
fetch('people.json')
  .then(function (response) {
    return response.json();
  })
  .then(function (data) {
    appendData(data);
  })
  .catch(function (err) {
    console.log(err);
  });
```

Here we are fetching our people.json file. After the file has been read from disk, we run the then function with the response as a parameter. To get the JSON data from the response, we execute the json() function.

The json() function also returns a promise. This is why we just return it and chain another then function. In the second then function we get the actual JSON data as a parameter. This data looks just like the data in our JSON file.

Now we can take this data and display it on our HTML page. Notice that we are calling a function called appendData. This is where we create the code which will append the data to our page

Notice that in our catch function, we are just writing the error message to out console. Normally you would display an error message to the user if something went wrong.

Displaying the JSON data

Before we display our JSON data on the webpage, let's just see how the body of our index.html file looks like.

```
<body>
  <div id="myData"></div>
</body>
```

Our plan is to fill our JSON data inside this div dynamically.

Our goal is to just simply display the full name of the people in our JSON file.

Step 1 – Get the div element from the body

Remember the div with the myData id from our index.html? We want to fetch that div using JavaScript. We need this because we are going to fill the div with the people in our JSON file.

This is how we will do it:

```
var mainContainer = document.getElementById("myData");
```

We get the element by executing the getElementByID function. This will find the element with the id myData.

Step 2 – Loop through every object in our JSON object

Next step is to create a simple loop. We can then get every object in our list of JSON object and append it into our main div.

```
for (var i = 0; i < data.length; i++) {
  // append each person to our page
}
```

Step 3 – Append each person to our HTML page

Inside the for-loop we will append each person inside its own div. This code will be repeated three times for each person.

First, we will create a new div element:

```
var div = document.createElement("div");
```

Next we will fill that div with the full name from our JSON file.

```
div.innerHTML = 'Name: ' + data[i].firstName + ' ' + data[i].lastName;
```

Lastly, we will append this div to our main container:

```
mainContainer.appendChild(div);
```

That's it. Now we have finished appending the JSON data to our index.html page. The full appendData function looks like this:

```
function appendData(data) {
  var mainContainer = document.getElementById("myData");
  for (var i = 0; i < data.length; i++) {
    var div = document.createElement("div");
    div.innerHTML = 'Name: ' + data[i].firstName + ' ' + data[i].lastName;
    mainContainer.appendChild(div);
  }
}
```

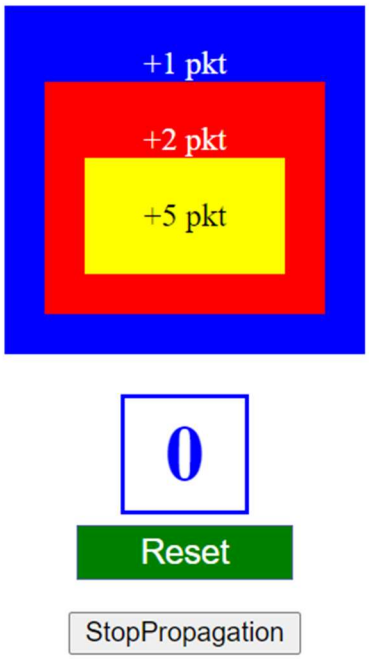
Try to copy this code in your own editor. As an exercise, you can try to style the output to look nicer. Remember to include the people.json file as well. This file must be in the same directory as your index.html file for this to work.

Exercise 6. In Catalog Example 6 you can find some files from ready project. In this project we don't use local file but remote fake backend server. Review the code and analyze the applied solutions.

Exercise 7 In Catalog Example 7 you can find template to work. In app.js you can find url adres to fetch data. Take and display in your page random yoke about chuckNoris.

Homework exercises.

Exercises 1 . Create a page with 3 elements, e.g. divas / photos positioned as shown in the picture below. (3points)



Let pressing any of them display the appropriate message:

Yellow - "you pressed yellow with a value of 5"

Red - "you pressed red with a value of 2"

Blue - "you pressed blue with a value of 1"

If the sum of the pressed values exceeds 30, the option of adding by the object 2 should be turned off. If 50, the event handler is additionally removed by the object 3.

Let there be 2 more buttons on the page: StopStartPropagation, Reset.

StopStartPropagation - turns propagation on and off globally for all divs.

Reset - returns to the initial state - restoring the point counter and connecting the click event handling function.

Exercises 2. (3points) Write a page that displays a balloon (using the balloon emoji, 🎈). When you press the up arrow, it should inflate (grow) 10 percent, and when you press the down arrow, it should deflate (shrink) 10 percent. You can control the size of text (emoji are text) by setting the font-size CSS property (style.fontSize) on its parent element. Remember to include a unit in the value—for example, pixels (10px).

The key names of the arrow keys are "ArrowUp" and "ArrowDown". Make sure the keys change only the balloon, without scrolling the page.

When that works, add a feature where, if you blow up the balloon past a certain size, it explodes. In this case, exploding means that it is replaced with an 💥 emoji, and the event handler is removed (so that you can't inflate or deflate the explosion).

// HTML code

🎈

// JS code

// Your code here

You'll want to register a handler for the "keydown" event and look at event.key to figure out whether the up or down arrow key was pressed.

The current size can be kept in a binding so that you can base the new size on it. It'll be helpful to define a function that updates the size—both the binding and the style of the balloon in the DOM—so that you can call it from your event handler, and possibly also once when starting, to set the initial size.

You can change the balloon to an explosion by replacing the text node with another one (using replaceChild) or by setting the textContent property of its parent node to a new string.

Exercises 3 . **(4 points) Under url address https://api.unsplash.com there is a cloud backend that give you example of images.**

'https://api.unsplash.com/search/photos?query='+input.value+'&per_page=30&client_id=xxxxxxxxxx''

Create a simple search engine that will allow you to upload photos of the subject matter specified in the search engine's input field. Example page is shown below.