1DA1611:A

Advanced Internet Programming

Project nr. 6 Report: 2023-03-30

Illia Priadko (309062)

Plan & required work:

- Learn JSON and its application in JavaScript;

- Run and explain the examples;

1. Resources to learn JSON

Taking advantage of the resources provided in the task, I was able to investigate and learn more about JSON, and the way JS handles data in general.

I have used w3schools and MDN. In short, it is very well summed up in this sentence:

“JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax. It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).”

2. Examples analysis

Examples 4, 6, and 9 are assigned to me to analyse. Here they are, one-by-one:

Example 4:

```
<!DOCTYPE html>
<html>
<body>

<h2>Store and retrieve data from local storage.</h2>
<p id="demo"></p>

<script>
// Storing data:
const myObj = { name: "John", age: 31, city: "New York" };
const myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
let text = localStorage.getItem("testJSON");
let obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
</script>

</body>
</html>
```

In short, this code demonstrates how to store and retrieve data using the JSON format and local storage in a web browser.

The code first creates an object *myObj,* with three properties: name, age, and city. It then uses the *JSON.stringify()* method to convert *myObj* into a string, which is stored in local storage using the *localStorage.setItem()* function (1st argument is the identifier, second argument is value)

To retrieve data, *localStorage.getItem()* function is used to get the JSON-formatted string that was earlier stored in local storage by key "*testJSON*". It is later converted back to a JavaScript object using the *JSON.parse()* method, giving us a JavaScript object instead of input string.

Finally, the code sets the value of the HTML element with the ID "*demo*" to the value of the name of the retrieved object. This displays the name "John" in the webpage.

Example 6:

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON.stringify() will remove any functions from an object.</h2>
<p>Convert functions into strings to keep them in the JSON object.</p>

<p id="demo"></p>

<script>
const obj = {name: "John", age: function () {return 30;}, city: "New York"};
obj.age = obj.age.toString();
const myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>


</body>
</html>
```

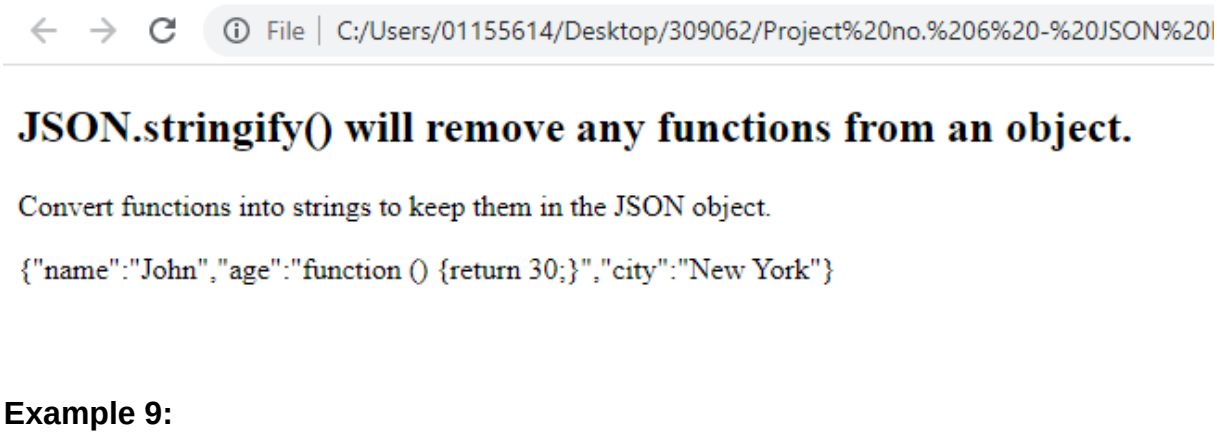
In this example we could see how the *JSON.stringify()* method removes any functions from an object. The code first creates an object *obj*, which has the same three properties: name, age, and city. However, here, the age is set to a function that returns 30.

Next, the *toString()* method is called on the age property of the obj object. This converts the age function into a string, which can be then included in the JSON-formatted string. *JSON.stringify()* is then called on the *obj* object to convert it into a JSON-formatted string. The age function's output was converted to a string using the *toString()* method, it can now be included in the JSON-formatted string.

Now, the value of the *myJSON* string is assigned to an HTML *<p>* element to be shown on the webpage itself.

To sum it up, this code snippet demonstrates how to include functions in a JSON-formatted string by first converting them to strings using the *toString()* method. This is because the *JSON.stringify()* method cannot handle inlined functions inside an object and will actually remove them from the JSON-formatted string. Here's what happens when we exclude the line where *toString()* is called:

Instead of expected:



Example 9:

```
<!DOCTYPE html>
<html>
<body>

<h2>Access a JavaScript Object</h2>
<p id="demo"></p>

<script>

/* You can access object values by using dot (.) notation: */

const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
</script>

<p id="demo1"></p>
```

```
<script>
const myJSON1 = '{"name":"Jack", "age":50, "car":null}';
const myObj1 = JSON.parse(myJSON1);
document.getElementById("demo1").innerHTML = myObj1["name"];
</script>

<p id="demo2"></p>

<script>
const myJSON2 = '{"name":"Peter", "age":60, "car":null}';
const myObj2 = JSON.parse(myJSON2);
document.getElementById("demo2").innerHTML = myObj2["age"];
</script>

</body>
</html>
```

This example demonstrates different ways to access values of properties of a JS object parsed from a JSON string.

The snipped provided in example 9 consists of three parts (blocks) of JS code, each block, like in previous examples, parses a JSON string into a JS object and then displays the value of one of the properties of the object in an HTML element using two different ways of accessing (referring to) it.

The first part uses **dot** notation to access the value of the *name* property of the *myObj* object and displays it in the HTML element with the ID "*demo*":

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

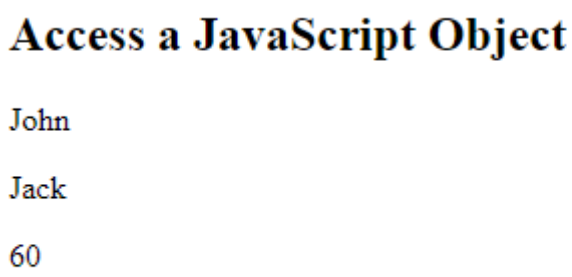
The second part uses **square bracket** notation to do essentially the same thing, except for using *myObj1* to output to "*demo1*":

```
const myJSON1 = '{"name":"Jack", "age":50, "car":null}';
const myObj1 = JSON.parse(myJSON1);
document.getElementById("demo1").innerHTML = myObj1["name"];
```

The third part uses the same **square bracket** notation to access the value of *age* in *myObj2* and displays it in "demo2":

```
const myJSON2 = '{"name":"Peter", "age":60, "car":null}';
const myObj2 = JSON.parse(myJSON2);
document.getElementById("demo2").innerHTML = myObj2["age"];
```

To sum up, these code snippets show us different ways to values of properties of a JS object parsed from a string using **dot notation** or **square bracket notation**. Here is how the output looks like, as expected:



Conclusion:

While doing this project, I had learned about JSON, and how it is used inside JS code to store, retrieve, and handle data. *stringify, getItem, setItem, parse* functions inside the global JSON object allow programmers to make handling hierarchical data like that much easier. The three provided examples explained to me in detail how these operations are done.