Programming in JAVASCRIPT

LAB 1

Environmental configuration

1. We use a model project named *NameLabJS.zip*
2. The works operating on the view are performed in the section where *id="Project X".*

1) Script loading control: *async, defer*

Put scripts in the project section:

index.html

```
<script>
            var name = prompt('Jak masz na imię?', '');
</script>


<script>
          document.write('<p>Ćw. 1. Witaj, ' + name + '</p>');
</script>
```

Check how the entire page script is parsed.

Then change the way the scripts are loaded onto the *defer* and check again how the entire page script is parsed:

index.html

```
<script defer>
```

Do the same with the *async* attribute.

Place the variable *name* declaration script where you think it should be.

2. Arrays

A little theory.

Arrays allows to save more than one value in one place. Without arrays each element of the list must correspond to a separate variable.
To create an array and save elements in it, you must first declare the name of the array (similar to creating variables) and then provide a list of comma-separated values.

```
var days = ['N.B.', 'Wt.', 'Śr.', 'Czw.', 'Pt.', 'Sob.', 'Ndz.'];
```

The square brackets - [] - inform the interpreter that he found the array.
An empty array:

```
var playList = [];
```

The tables can store various types of values. This means that there can be numbers, character strings and logical values in one array:

```
var prefs = [1, 223, 'www.javascript.com', false];
```

Even other arrays and objects can be placed in the array. This technique helps to store complex data. Placing the entire instruction in one line makes it difficult to read the program. For the sake of clarity, we create arrays in several lines:

```
var authors = [ 'Stan Lee',
                Simon Bisley,
                Grzegorz Rosiński',
                Todd McFarlane
                ];
```

The browser interpreter omits additional spaces and ambulance characters, so although the code occupies five lines, it is treated as a one-line instruction, as indicated by a semicolon at its end.

Task 1 Adding and removing elements from the array

Declare a table with sample (your own) data, then use the instructions

index.html

```
<script>
	alert(tablica[0]);
</script>
```

Display one of the items (does not have to be the first).

Then modify the **last** element in advance:

index.html

```
<script>
	tablica[tablica.length-1] = 'Nowa wartość';
</script>
```

and display it too.

Now we will increase the number of elements from the script level, using different ways.

Let us assume that the table has 5 elements (the index is counted from 0). So we add:

index.html

```
<script>
	tablica[5] = 'nowy6';
</script>
```

The array automatically increases its length. Or:

index.html

```
<script>
	tablica[tablica.length] = 'nowy7';
</script>
```

Or when adding more elements:

index.html

```
<script>
     tablica.push('nowy8', 'nowy9', 'nowy10', 'nowy11')
</script>
```

Now add an element to the beginning of the array:

index.html

```
<script>
     tablica.unshift('nowy12', 'nowy13')
</script>
```

The choice of the method is important if you want to keep the order of the elements (e.g. months, days of the week, user playlist, etc.).
The *push()* and *unshift()* commands return a value. When a task is completed, they pass a new number of elements in the array:

```
var p = [0,1,2,3];
var totalItems = p.push(4,5);
```

This passage will assign a value of 6 to the *totalItems* variable because in the table *p*
There will be six elements.

The *pop()* and *shift()* commands are used to remove elements from the beginning and end of the table. Both remove one value from the array, with *pop()* *at* the beginning and *shift()* *at the* end.

The *pop()* and *shift()* commands, like *push()* and *unshift()*, return data when the action is complete - this is the value of the element removed from the table. Therefore, the following code removes the value and saves it in the *removedItem* variable:

```
var p = [0,1,2,3];
var removedItem = p.pop();
```

When the code is completed, the value of the variable *removedItem* is *3* and the *p* array contains the numbers *[0,1,2]*.

TASK 2.

Create a second array with your own sample data.

Display in the new paragraph of project section 1:

index.html

```
<script>
     document.write('<p>Pierwszy element to <strong>');
     document.write(nowaTablica[0] + '</strong></p>');
</script>
```

Similarly, display the last element with the variable `.length-1`

After the last line of the display code for the last item, add the item using the `unshift()` method and add the display code for the first item again.

About facilities.

Many elements of JavaScript, like many elements of web pages are *objects:*

So in the context of previous exercises:

Facility	The property	Method
document		write()
['N.B.', 'Wt.', 'Śr.']	length	push()
		pop()
		unshift()
		shift()

Each JavaScript object has its own properties and methods. For example, an array object has the length property, and a document object provides the write() method. To refer to the properties of an object or call its method, write() is used with a dot. Dots connect objects with their properties or methods.

3. DOM

All the HTML elements on the website are contained in the DOM HTML tree *document* object and form the so-called element collections.

The number of elements in the *form* collection is read by the *lenght* property

```
var formsCounter = document.forms.length;
```

TASK 3.

Write down in the project section the number of elements on the page, and write down what exactly was counted (from which section and what elements).

GetElementById(id) method

If an item has an identifier, it can be found using *getElementById(id)*. This is the most commonly used way to access an item.

Ownership of *internalHTML*

The easiest way to download or replace the content of an item:

```
document.getElementById("name").innerHTML = "MyName";
```

TASK 4: Inserting data in the paragraphs using the DOM

- In the section o *id="main-page"* place your data using a script and references to elements according to the DOM model.
- Change the contents of the two new paragraphs you have created and enter any elements from the previously created tables into them.
- *Using the while* loop:

```
while (condition) {
// Repeated JavaScript code.
}
```

Write down **half of the** elements of one of the arrays

- Then do the same with the loop for:

```
for (i = 0; i <????; i++) {
    // Repeated JavaScript code
}
```

- Write a function that will display a window with your data, taken from the elements using objects from the DOM, reacting to the *onClick* event of any *button*.

```
function name Function() {
// JavaScript code activated.
}
```

TASK 5. QUIZ

We will consolidate the previous exercises by creating new functionality.

Design a simple quiz with your own questions and place it in the Project 2 section.

Display the questions one by one in the section, collect the results and display at the end.

Example of a question array:

```
var questions = [
    ['How many moons does the Earth have?', 1],
    ['How many moons has Saturn?', 62],
    ['How many moons does Venus have?', 0]
];
```

Example of a correctness test:

```
if (answer == question[1]) {
    alert('Correct answer!');
    score++;
} else {
    The correct answer is ' + question[1]);
}
```

ZIP THE PROJECT AND PLACE IT ON MSTeams Platform.