PDP UNIVERSITY

Website development

Lecture 8

Array

Array - special variable that can hold multiple values.

Declaring an array

We declare an array in the following way:

```
const array_name = [element1, element2, ...];
```

Example:

```
const cars = [“Cobalt", “Spark", “Matiz"];
```

We can declare the name of the array first and then insert the elements :

```
const mashinalar = [];
mashinalar[0]= “Cobalt";
mashinalar[1]= “Spark";
mashinalar[2]= “Tiko";
```

Declaring an array

Arrays in Javascript can accept different data types.

Example:

```
const person = [“Eshmat", “Toshmatov", 73];
```

Reference to an array element

Array elements are referenced by index number.

Array element numbering starts from 0:

Example:

```
const cars = [“Cobalt", “Spark", “Matiz"];
let car = cars[0]; // Cobalt
```

Array update

To update a certain element of an array, select this element by its index number and assign a new value to it.

Example:

```
const cars = [“Cobalt", “Spark", “Matiz"];
cars[0] = ”Lacetti”; // Cobalt => Lacetti
```

Array: `pop()` & `push()`

`pop()` – removes the last element of an array. Returns the removed element.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
// Banana,Orange,Apple
```

`push()` – adds an element to the end of an array. Returns new length of the array.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push('Kiwi');
// Banana,Orange,Apple,Mango,Kiwi
```

Array: `splice()`

`splice()` – changes content of an array by **removing**, **adding**, or **replacing** elements.

The return value is an array containing deleted elements.

```
splice(start)
splice(start, deleteCount)
splice(start, deleteCount, item1)
splice(start, deleteCount, item1, item2, itemN)
```

Array: `splice()`

In the following code, we add Lemon and Kiwi starting from the 2nd index of the array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
// Banana,Orange,Lemon,Kiwi,Apple,Mango
```

In the following code, we remove Apple and Mango and add Lemon and Kiwi instead.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 2, "Lemon", "Kiwi");
// Banana,Orange,Lemon,Kiwi
```

Array: `splice()`

In the following code, we delete the 3rd element of the array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 1);
// Banana,Orange,Mango
```

Array: `slice()`

`slice()` – splits part of an array into a new array.

The return value is a new array.

```
slice()
slice(start)
slice(start, end)
```

Array: `slice()`

In the following code, we can extract part of the array from the 2nd to the 4th element (the 4th element is not included) into a new array:

```
const fruits=["Banana", "Apple", "Lemon", "Orange", "Mango"];
const citrus = fruits.slice(2, 4);
// Lemon, Apple
```

Array: `splice()` & `slice()`

`splice()` **modifies original array.**

`slice()` **does not modify original array**.

Array: `indexOf()`

`indexOf()` – searches for a specific element and returns the index at which it is located.

If the same element occurs more than once, it returns the index of the first one.

If the element is not found it returns -1.

```
array.indexOf(item, start) // start is optional
```

Array: `includes()`

`includes()` – checks if an element exists in an array. Returns TRUE if exists, FALSE otherwise.

```
array.includes(search-item)
```

Array: `sort()`

`sort()` – sorts an array. By default it sorts in an alphabetical order. To sort numbers we have to pass callback function.

Modifies original array.

```
// No parameters. Sorts in an alphabetical order
array.sort()
// Sorts using the callback function (compareFn)
array.sort(compareFn)
```

Array: `sort()`

We use `sort()` without parameters to sort string elements, .

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
// Apple,Banana,Mango,Orange
```

To sort the numeric elements, we need to pass a callback function as a parameter to the `sort()` function.

```
const numbers = [4, 2, 5, 1, 3];
numbers.sort(compareNumbers);
function compareNumbers(a, b) { return a - b; }
```

Array: `map()`

`map()` – performs a certain (callback) function for each element of the array.

```
numbers.map(callbackFunction);
```

The callback function accepts 3 parameters:

1. **`element`** – current element being processed in the array.
2. **`index`** – index of the current element being processed in the array.
3. **`array`** – The array `map()` was called upon.

```
function callbackFunction(element, index, array) {}
```

Array: `filter()`

`filter()` – creates a new array with the elements that pass the condition. The condition is checked using the callback function.

(The callback function has the same structure as described in slide 18)

```
numbers.filter(callbackFunction);
```

Array: `find()`

`find()` – returns the value of the first array element that passed the condition.

The condition is checked using the callback function.

(The callback function has the same structure as described in slide 18)

```
numbers.find(callbackFunction);
```

Array: `findIndex()`

`findIndex()` – returns the index of the first array element that passed the condition. The condition is checked using the callback function.

(The callback function has the same structure as described in slide 18)

```
numbers.findIndex(callbackFunction);
```

Thank you for your attention