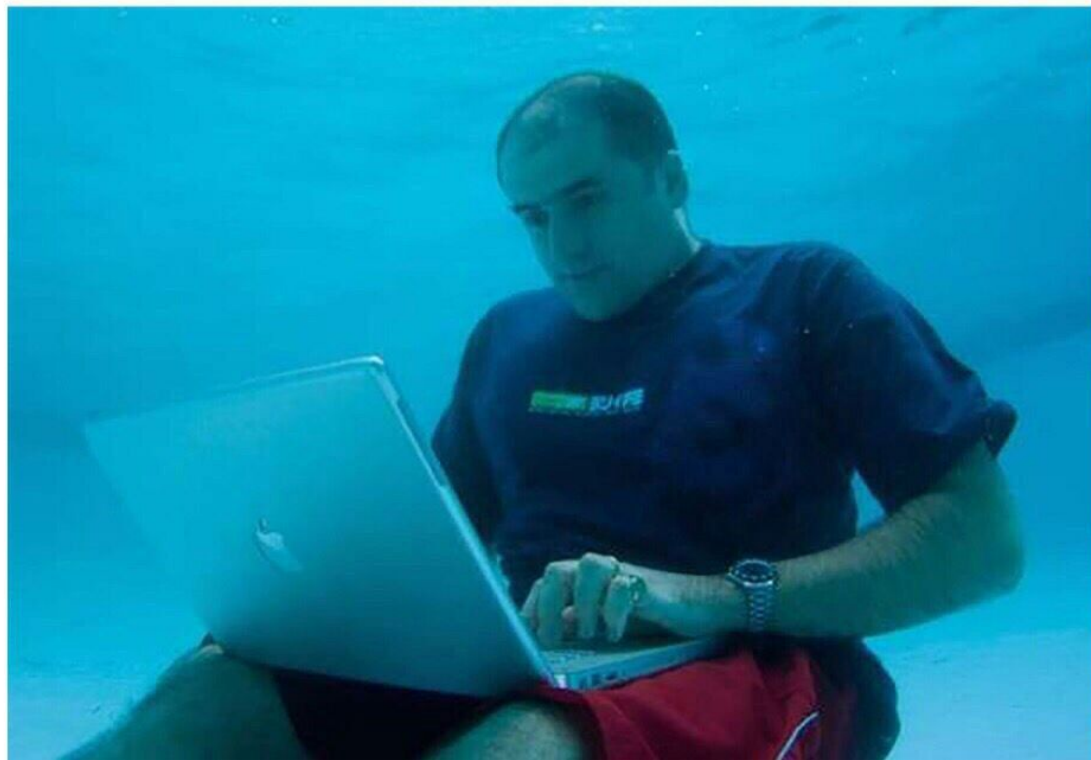




Лекция 5

Массивы

Лайфхак для програмістів:
Якщо кодити на JavaScript під водою,
ніхто не побачить що ви плачете



Цели на сегодня:

- Научиться работать с массивами;
- Узнать о стандартных методах массивов.

Типы данных (Data Types)

1. Прimitives

- a. Числа (Numbers);
- b. Строки (Strings);
- c. Логические (Boolean)
- d. Символ (Symbol);
- e. null;
- f. undefined.

2. Сложные

- a. Объекты (Objects);
- b. Массивы (Arrays);
- c. Функции (Functions).

Массив (Array) –

разновидность объекта, которая предназначена для хранения пронумерованных значений и предлагает дополнительные методы для удобного манипулирования такой коллекцией.

NB!

```
console.log(typeof {name: 'Peter'}) // object
```

```
console.log(typeof [1, 2, 3]) // object
```



```
/*  
    Метод Array.isArray(arg: any) возвращает true, если переданное  
    значение является массивом или false, если оно массивом не является  
*/  
  
const numbers = [1, 2, 3, 4, 5];  
const user = {name: 'Mary'};  
  
console.log(Array.isArray(numbers)); // true  
console.log(Array.isArray(user)); // false
```

Значения в массивах
нумеруются с 0

Как создать массив?



// Первый способ создать массив – через функцию-конструктор

```
const arr = new Array(1, 2, 3, 4);
```

```
console.log(arr) // [1, 2, 3, 4]
```

```
const secondArr = new Array(4);
```

```
console.log(secondArr) // [<4 empty slots>]
```



// Второй способ создать массив – через квадратные скобки

```
const arr = [1, 2, 3, 4];  
console.log(arr) // [1, 2, 3, 4]
```

```
const secondArr = [4];  
console.log(secondArr) // [4]
```

Методы **Array.from()** и **Array.of()** преобразуют полученные элементы в массив.

Различие их в том, что **Array.of()** принимает неограниченное число аргументов и превращает их все в массив, а **Array.from()** принимает callback в качестве второго аргумента.

ES6



```
const first = Array.of(1, 2, 'string');  
console.log(first); // [1, 2, "string"]
```

```
const second = Array.from('abc');  
console.log(second); // ["a", "b", "c"]
```

Как добавить штуки в массив?



// Можно напрямую указать номер элемента в массиве

```
const arr = [];  
arr[1] = 'String';  
arr[3] = true;
```

```
console.log(arr);  
// [ <empty>, "String", <empty>, true ]
```



// А можно воспользоваться специальными методами – push и unshift

```
const arr = [3];
```

// Push добавляет элемент в конец

```
arr.push(10); // [3, 10]
```

// Unshift добавляет элемент в начало

```
arr.unshift(1, 2); // [1, 2, 3, 10]
```

Размер важен



```
/*
```

Свойство `length` возвращает длину.

Длина рассчитывается, как: индекс последнего элемента + 1

```
*/
```

```
const arr = [1, 2, 3, 4, 5];
```

```
console.log(arr.length); // 5
```

```
const arr = [];
```

```
arr[1000] = 'Some value';
```

```
console.log(arr.length); // 1001
```



// Свойство `length` позволяет укорачивать массив.

```
const numbers = [1, 2, 3, 4, 5];
```

```
numbers.length = 2;
```

```
console.log(numbers); // [1, 2]
```

Удоли(



```
/*
```

Тут работает уже знакомый вам оператор delete, однако, при его использовании, на месте элемента остается дырка.

```
*/
```

```
const data = ['a', 'b', 'c'];
```

```
delete data[0];
```

```
console.log(data); // [<1 empty slot>, 'b', 'c']
```



```
/*
```

Лучше всего использовать методы `pop` и `shift`, если нужно удалить последний или первый элемент массива

```
*/
```

```
const data = ['a', 'b', 'c'];
```

```
data.pop(); // 'c'
```

```
data.shift() // 'a'
```



```
/*
```

```
    Для удаления значения, отличного от первого или последнего, можно  
    вызывать метод splice
```

```
*/
```

```
const userIds = [1, 2, 3, 4, 5];
```

```
/*
```

```
    Синтаксис:
```

```
    arrayName.splice(начальный индекс, сколько удалить?);
```

```
*/
```

```
userIds.splice(2, 1);
```

```
console.log(userIds); // ?
```



// Дополнительные аргументы в splice замещают удаленные элементы

```
const userIds = [1, 2, 3, 4, 5];
```

```
userIds.splice(2, 2, 35, 61);
```

```
console.log(userIds); // [1, 2, 35, 61, 5]
```

Самостоятельная работа

Напишите программу “Список покупок”. Программа будет состоять из функции/метода **addItem(thing)**, который добавляет продукты в массив **items**.

Функция/метод добавляет в массив только строчные элементы, есть проверка на пустую строку.

Ну это уже перебор



// Для перебора массива можно использовать цикл for.

```
const numbers = [1, 2, 3, 4, 5];
```

```
for (let i = 0; i < numbers.length; i++) {  
    console.log(`${i} элемент массива — ${numbers[i]}`);  
}
```

// Но не нужно!

Перебирающие методы массивов

filter() – выполняет какое-то действие на каждой итерации и возвращает только те объекты, для которых callback вернул *true*;

map() – превращает один массив элементов в другой;

forEach() – циклически обходит массив и выполняет заданное действие на каждой итерации;

reduce() – сворачивает все элементы массива слева направо в одно значение;

reduceRight() – сворачивает все элементы массива справа налево в одно значение.

Функция обратного вызова (callback) –

функция, которая передается в качестве аргумента другой функции и выполняется после выполнения основного кода.



```
/*
```

Метод `Array.prototype.forEach()` перебирает все элементы массива и выполняет на каждой итерации `callback`.

```
*/
```

```
const numbers = [1, 2, 3, 4, 5];
```

индекс

```
numbers.forEach(элемент(element, элементаindex, массивarray) => {  
  console.log(element * 2); // 2, 4, 6, 8, 10  
  console.log(array[index] * 2); // 2, 4, 6, 8, 10  
});
```

```
console.log(numbers); // Array(5) [1, 2, 3, 4, 5]
```



```
/*  
    Метод Array.prototype.filter() перебирает все элементы массива  
    и возвращает новый массив значений, вернувшихся из cb-функции  
*/
```

```
const arr = [1, 2, 3, 4, 5, 6];
```

```
const newArr = arr.filter((element, index, array) => {  
    if (element % 2 === 0) {  
        return element;  
    }  
});
```

```
console.log(newArr); // Array(3) [ 2, 4, 6 ]
```



```
/*  
    Метод Array.prototype.map() создает новый измененный массив,  
    выполняя cb-функцию для КАЖДОГО элемента исходного массива.  
*/  
const numbers = [1, 2, 3, 4, 5];  
  
const result = numbers.map((element, index, array) => {  
    return element * 2;  
});  
  
console.log(numbers); // Array(5) [1, 2, 3, 4, 5]  
console.log(result); // Array(5) [2, 4, 6, 8, 10]
```


/*

Map отличается от filter тем, что map возвращает в новый массив столько значений, сколько было и в исходном массиве

*/

```
const integers = [1, 2, 3, 4, 5, 6];
```

```
const result = integers.map((element, index, array) => {  
  if (element % 2 === 0) {  
    return element;  
  }  
})
```

```
console.log(result);
```

```
// Array(6) [ undefined, 2, undefined, 4, undefined, 6 ]
```




```
/*
```

Метод `Array.prototype.reduce()` применяет указанную функцию к каждому элементу, сворачивая массив в одно значение.

```
*/
```

```
const numbers = [1, 2, 3, 4, 5];
```

```
const result = numbers.map((prev, cur, index, array) => {  
    return prev + cur;  
});
```

```
console.log(result); // 15
```

Некоторые другие методы массивов

Array.prototype.concat() – объединяет несколько массивов;

Array.prototype.indexOf() – возвращает первый индекс, по которому можно найти элемент в массиве;

Array.prototype.every() – проверяет, удовлетворяют ли все элементы указанному условию;

Array.prototype.some() – проверяет, удовлетворяют ли какие-то элементы указанному условию;

Некоторые другие методы массивов

ES6 **Array.prototype.find()** – возвращает значение первого найденного элемента, который соответствует условию;

Array.prototype.join() – объединяет все элементы массива в строку, можно указать разделитель в качестве параметра;

Array.prototype.toString() – преобразует массив в строку;