

Функции и асинхронность в JavaScript



ORTDNIPRO.ORG/JS

1. Функции и функциональные выражения

Функции в JavaScript

```
2
3  function action(a, b, c){
4      let sum = a + b + c;
5      return sum;
6  }
7
8  let process = function(a, b, c){
9      let sum = a + b + c;
10     return sum;
11 }
12
13 let calculate = (a, b, c) => a + b + c;
14
15 typeof action; //function
16 typeof process; //function
17 typeof calculate; //function
18
```

Функции в JavaScript – блоки кода которые возможно вызывать (выполнять) многократно. Синтаксисом JS предусмотрено несколько способов определения функций: Объявление функции (***Function Declaration***) (3), Функциональное выражение (***Function Expression***, она же «анонимная» функция) (8), и стрелочные-функции (***arrow-function***, они же лямбда-функции) (13). Функции в JavaScript – тип данных, функцию мы можем размещать в переменных, как и другие типы данных. Отличие в том, что функции мы можем вызывать.

Подробнее: <https://learn.javascript.ru/function-basics>

Подробнее: <https://learn.javascript.ru/arrow-functions-basics>

Оператор ... и функции

```
2
3   let process = function(a, b, c, ...others){
4       console.log(others);
5       let sum = a + b + c;
6       return sum;
7   }
8
9   process(1,2,3,4,5,6,7); // return 6;
10  // in console: [4,5,6,7];
11
```

Функция может принимать параметры и возвращать результат своей работы для дальнейшего использования (оператор *return*).

Но при помощи оператора ... (в данном случае его называют *rest-оператором*) мы можем принять любое количество параметров и работать с ними как с массивом (**ES2015**).

Подробнее: <https://learn.javascript.ru/rest-parameters-spread-operator>

Параметры по умолчанию в функциях

```
2
3   let process = function(a = 1, b = 2, c = 3){
4       console.log(a, b, c);
5       let sum = a + b + c;
6       return sum;
7   }
8
9   process(1,2); // return 6;
10  // in console 1, 2, 3
11
```

Передача неполного набора параметров не является ошибкой в **JavaScript**, но может создать проблемы при работе функции. При помощи синтаксиса параметров по умолчанию мы можем указать значения которые будут использоваться если тот или иной параметр не будет передан (**ES2015**).

Подробнее: <https://learn.javascript.ru/function-basics#parametry-po-umolchaniyu>

Функция в объекте – метод

```
2
3   let arr = ["Jhon", (name) => alert(`Hello ${name}!`) , "Alice"];
4
5   arr[1]('Bill');
6
7   //-----//
8
9   let ob = {
10       name : "Jhon",
11       city : "Dnipro",
12       action: function(name){
13           alert(`Hello ${name}!`);
14       }
15   }
16
17   ob.action("Maria");
18
```

Функции могут размещаться в ячейках массива (коллекций Set и Map) а также в свойствах объекта. При этом для функций в составе объектов есть отдельный термин – **метод**.

Самовывзывающиеся функции в JavaScript

```
2  
3  ✓ (function(){  
4      console.log("...");  
5  })();  
6
```

Самовывзывающиеся функции – удобный механизм выполнить какие-либо действия автоматически, не создавая переменных и внося в код явных вызовов функций. Другими словами не засоряя глобальную область видимости. Активно используется в сторонних библиотеках.

Замыкания

```
2
3   let user_name = "Jhon";
4
5   function test(){
6       |   console.log(`Hello ${user_name}!`);
7   }
8
9   user_name = "Jane";
10
11   test();
12
```

У функций есть доступ к внешним переменным, этот механизм называют **замыканием**, он позволяет обращаться к внешнему контексту и получать оттуда актуальные данные.

Подробнее: <https://learn.javascript.ru/closure>

2. Таймеры в JavaScript

Таймеры в JavaScript

```
2
3   let f1 = function(){
4       console.log("Function for Timeout called");
5   }
6
7   let f2 = function(){
8       console.log("Function for Interval called");
9   }
10
11   let timeout_id = setTimeout(f1, 1000);
12
13   let interval_id = setInterval(f2, 3000);
14
```

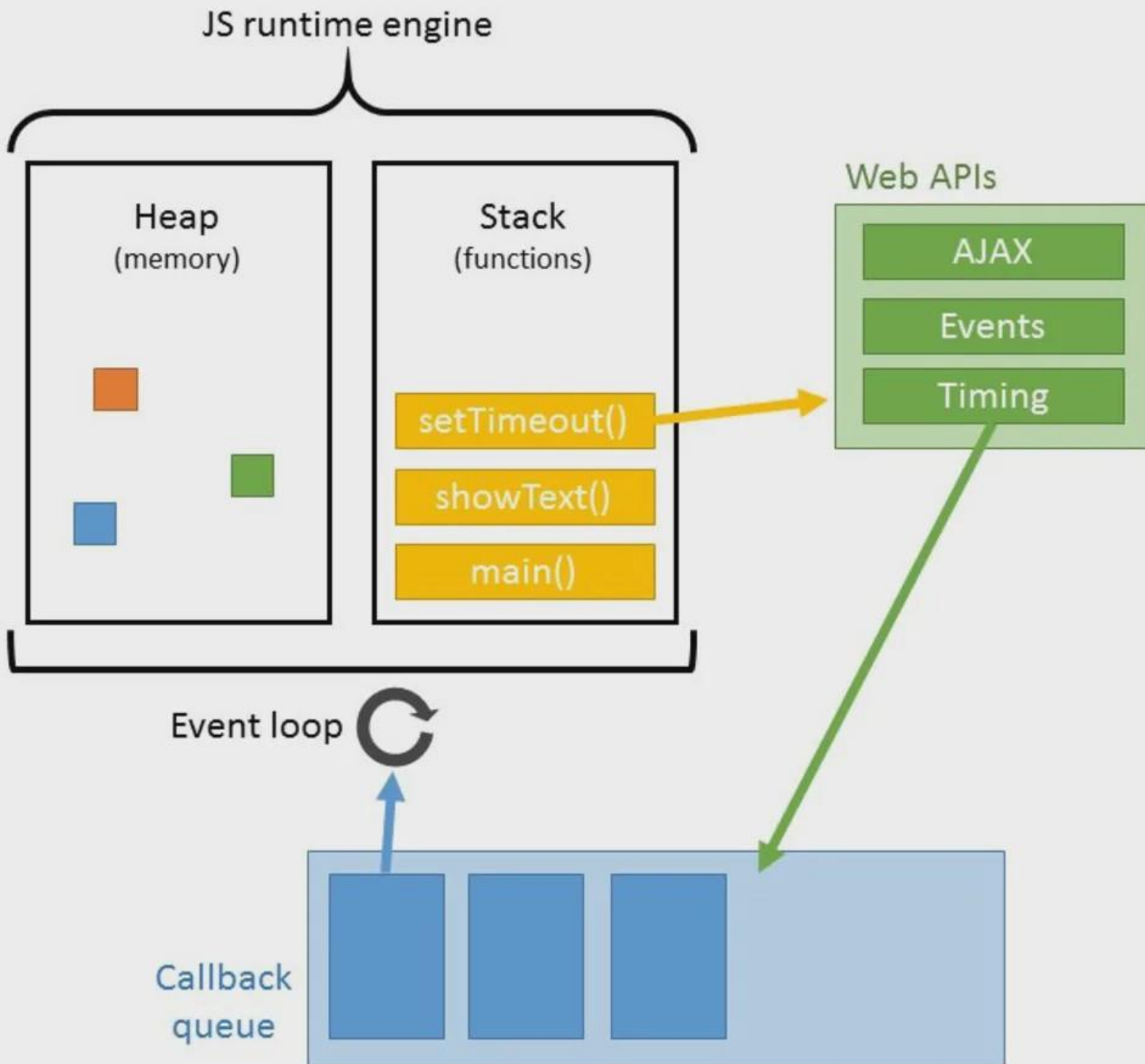
setTimeout(*some_function*, *delay*) – вызовет функцию *some_function* через *delay* миллисекунд. Сделает это один раз.

setInterval(*some_function*, *delay*) – вызовет функцию *some_function* через *delay* миллисекунд. И будет повторять вызов каждые *delay* миллисекунд.

Обе функции возвращают **id** таймера, с помощью которого и функций **clearTimeout(id)** и **clearInterval(id)** уничтожить таймер еще до его вызова. Обе функции можно отнести к инструментам **асинхронности**.

Подробнее: <https://learn.javascript.ru/settimeout-setinterval>

3. Цикл событий / Event Loop



Event Loop

JavaScript однопоточный язык программирования, но тем не менее нам доступны асинхронные инструменты. Доступны они за счёт функционирования механизма **Event Loop** (или **цикла событий**, но **не стоит путать с событиями DOM**).

Подробнее:

https://youtu.be/j4_9BZezSUA

Тут докладчик еще более странный...

4. Геолокация и callback'и

Геолокация в теории



Широта == Latitude

Долгота == Longitude

```
{ ..., latitude: 48.4767, longitude: 35.0543, ... };
```

Геолокация на практике

```
2
3 // 'Classic' version
4 navigator.geolocation.getCurrentPosition( position => {
5     console.log('Your position: ', position.coords);
6 }, error => {
7     console.log('Geolocation error:', error);
8 })
9
```

У браузера есть возможность узнать координаты пользователя на местности. Для этого мы можем воспользоваться методом **navigator.geolocation.getCurrentPosition()** который принимает **callback** функции для получения координат и информации об ошибке. Но важно **проверять поддерживает ли браузер геолокацию** проверяя наличие свойства **geolocation** объекта **navigator**.

Подробнее: <https://developer.mozilla.org/ru/docs/Web/API/Geolocation/getCurrentPosition>

Немного о статических карта на примере Here Map

```
https://image.maps.api.here.com/mia/1.6/mapview?app_id=oZmMWRV4tAjQmgkxBvF0&app_code=x5pKHqifhw1mnS_zBTIFsA&z=11&w=600&h=600&c=48.4608,35.0501
```

Сервис **Here Map** предоставляет возможность размещать на наших страницах картографические материалы, управляя позицией и масштабом отображения.

Вы можете воспользоваться шаблоном в репозитории [./src/template-geolocation/](#)

5. Объект Promise

Объект Promise

```
2
3  ✓ let promise = new Promise((resolve, reject) => {
4
5  ✓   setTimeout( () => {
6     let result = 2 ** 10;
7     resolve(result);
8     //reject('Operation imposible');
9   }, 5000);
10
11 });
12
13 promise.then( result => console.log('Successful, result is', result) );
14
15 promise.catch( error => console.log('Failed, error is', error) );
16
17 promise.finally( () => console.log('Promise Finished'));
18
19 console.log("After Promise");
20
```

Promise – механизм позволяющий писать асинхронный код последовательно (насколько это возможно), избегая вложенности **callback**’ов. **Promise** – объект который принимает функцию, в которой запускается асинхронная операция, при помощи параметров функции есть возможность из асинхронного кода сообщить об успешном или не успешном завершении операции (параметры **resolve**, **reject** – функции вызов которых приведёт к завершению работы **Promise**’а).

Подробнее: <https://learn.javascript.ru/promise/>

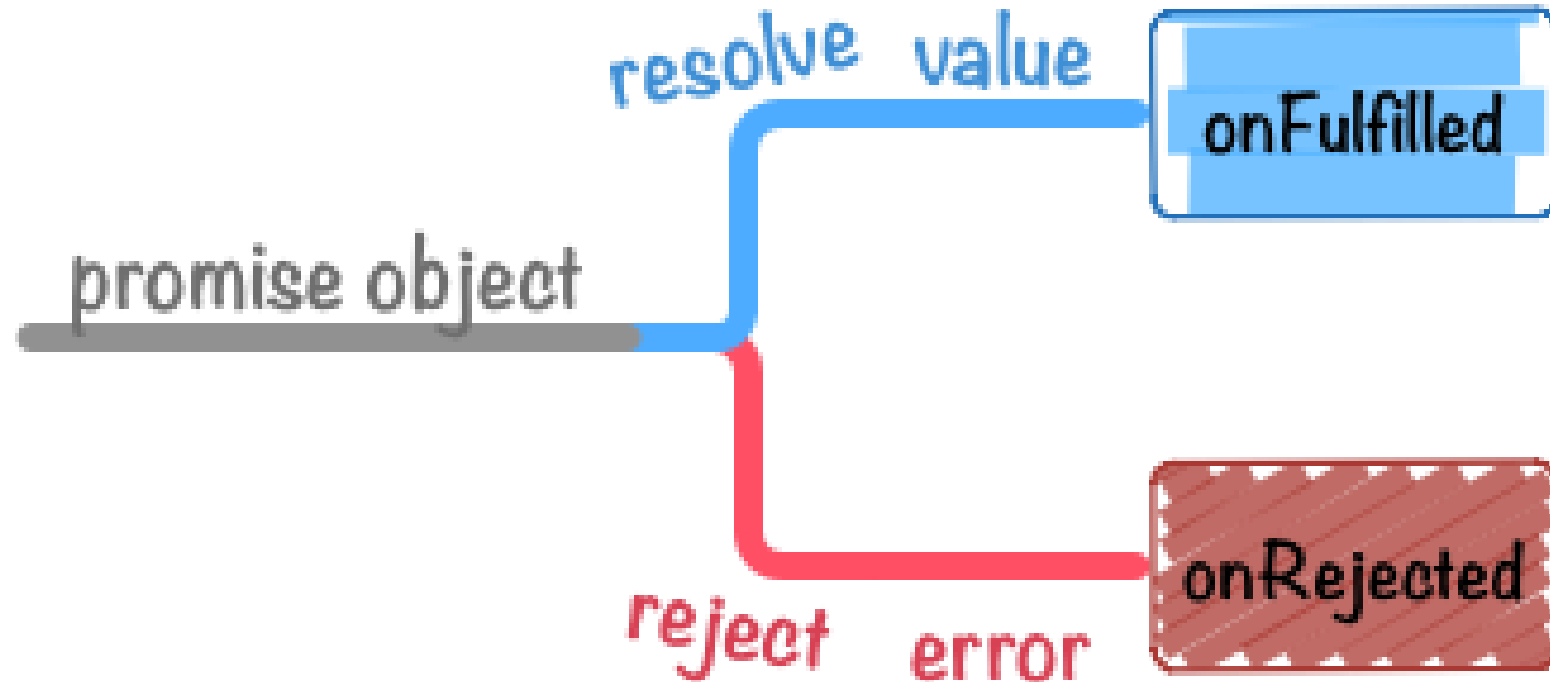
Объект Promise

```
2
3 ✓ let promise = new Promise((resolve, reject) => {
4
5 ✓   setTimeout( () => {
6     let result = 2 ** 10;
7     resolve(result);
8     //reject('Operation imposible');
9   }, 5000);
10
11 });
12
13 promise
14   .then( result => console.log('Successful, result is', result) )
15   .catch( error => console.log('Failed, error is', error) )
16   .finally( () => console.log('Promise Finished') );
17
18 console.log("After Promise");
19
```

У объекта **Promise** есть 3 полезных метода для возможности зарегистрировать функции на случай успешного завершения **Promise**'а, для случая завершения с ошибкой, и для ситуации когда код нужно выполнить как бы **Promise** не завершился (успешно или нет). Эти методы соответственно **.then()**, **.catch()** и **finally()**. Эти методы могут быть вызваны цепочкой т.к. эти методы возвращают ссылку на сами **Promise**. Функция переданная **.then()** может вернуть результат (в т.ч. другой **Promise**) и цепочка может опять включать **.then()** для его обработки.

Подробнее: <https://learn.javascript.ru/promise/>

Жизненный путь Promise



Жизненный путь **Promise** всегда завершается одним из двух состояний:
Fulfilled – успешное завершение, либо **Rejected** – неудачное завершение.

6. `async/await`

async/await – упрощение кода Promise'ов

```
2
3 let url = 'https://bank.gov.ua/NBUStatService/
4           v1/statdirectory/exchange?json';
5
6 (async function(){
7     let result = await fetch(url);
8     result     = await result.json();
9
10    console.log(result);
11
12 })();
13
```

async/await – надстройка над **Promise** позволяющая писать код в полностью привычном синхронном стиле, при этом откладывая ожидания завершения операций до тех пор пока её результат действительно понадобится;

async – отмечает функцию как асинхронную (результат такой функции оборачивается в **Promise**);

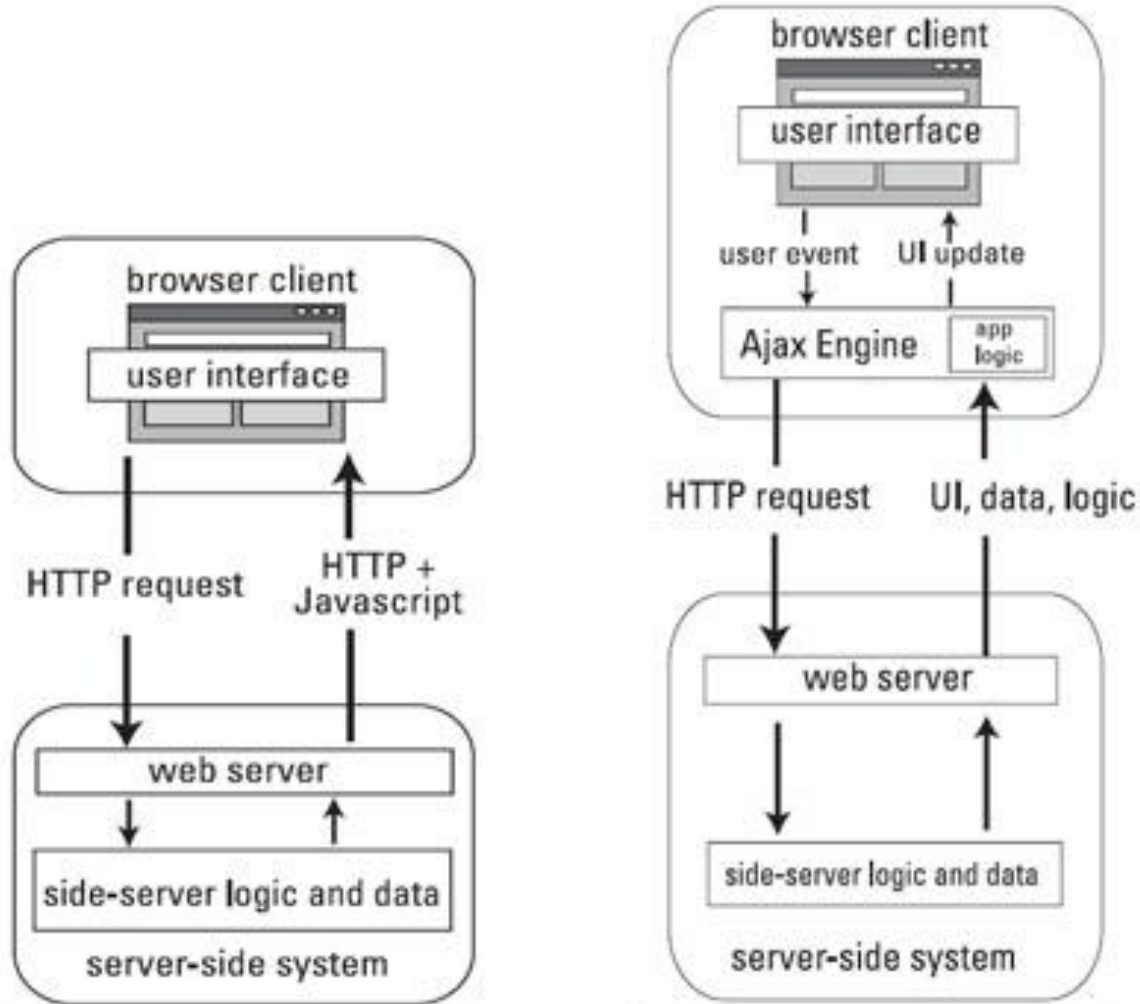
await – при вызове асинхронных функций указывает, что не нужно ждать результата сейчас

Подробнее: <https://learn.javascript.ru/async-await>

7. fetch()

AJAX на **Promise**'ax

Asynchronous JavaScript And XML



Идея заложенная в **AJAX** – не перезагружая страницу, запросить (или передать) у сервера новые данные и использовать их в документе.

fetch() – Promise «обёртка» для выполнения AJAX-запросов

```
2  
3 let url_nbu = 'https://bank.gov.ua/NBUStatService/v1/statdirectory/ovdp?json';  
4  
5 fetch(url_nbu)  
6   .then(result => result.json())  
7   .then(result => console.log(result))  
8   .catch(error => console.log("Error:", error));  
9  
10
```

Функция **fetch()** – выполняет AJAX-запросы, возвращая **Promise**, который завершится с поступлением ответа на запрос или завершится с ошибкой, если запрос будет неудачный.

Подробнее: <https://learn.javascript.ru/fetch/>

API Национального Банка Украины



**НАЦІОНАЛЬНИЙ
БАНК УКРАЇНИ**

Валютные API, информация о финансовом рынке и банковском секторе

<https://bank.gov.ua/ua/open-data/api-dev>

8. Перебирающие методы массивов

Метод .sort() и функция-компаратор

```
2
3 let arr = [23, 4, 67, 117, 34, 0, 55, 78, 5, 9];
4
5 arr.sort(function(a, b){
6     if(a > b){
7         return 1;
8     }else if(a < b){
9         return -1;
10    }else{
11        return 0;
12    }
13 });
14 //arr.sort((a,b) => a - b);
15
16 console.log(arr);
17 //[0, 4, 5, 9, 23, 34, 55, 67, 78, 117]
18
```

Методу **.sort()** массивов можно передать функцию (т.н. функцию-компаратор) которая «подскажет» браузеру как сравнивать два элемента между собой. Функция принимает 2 элемента и должна вернуть 0 если они равны, отрицательное число если второй элемент больше или положительное если первый элемент больше.

Подробнее: <https://learn.javascript.ru/array-methods>

Полезнейшие методы преобразования массивов

.filter();

Метод **.filter()** формирует новый массив занося в него элементы из старого, но только те которые «одобрит» функция переданная методу в качестве параметра.

.map();

Метод **.map()** формирует новый массив занося в него элементы из старого, но предварительно пропуская каждый элемент через функцию переданную методу в качестве параметра. Эта функция может любым образом преобразовать элемент.

.reduce();

Метод **.reduce()** позволяет хранить при переборе элементов какое-либо промежуточное значение, оно передаётся в первом параметре функции (передаваемой методу). При каждом вызове то что возвращает функция становится этим самым «промежуточным» значением для следующего вызова функции. В результате **.reduce()** возвращает самое последнее «промежуточное значение»

Подробнее: <https://learn.javascript.ru/array-methods#preobrazovanie-massiva>

Будет полезным

Перебирающие методы

В **JavaScript** есть еще ряд методов массивов, а именно: **.every()**, **.some()**, **.find()**, **.findIndex()** узнайте чем они могут быть полезны.

На следующем занятии

На следующем занятии

1. **Объекты**, ключевое слово **this**, принципы ООП;
 2. Функция-**конструктор** объектов;
 3. **Классы** в JavaScript;
- ...

Домашнее задание
/сделать

Домашнее задание #B.1



«Азбука пилотов» (или официально **фонетический алфавит ИКАО**) - стандартизированный способ прочтения букв алфавита английского языка в авиации. Каждая буква кодируется словом, которое при плохой связи позволяет с высокой вероятностью распознать букву которая передаётся. Ваша задача, написать скрипт, который будет переводить буквенно-цифровую комбинацию в набор слов из «азбуки пилотов».

Например: пользователь вводит комбинацию буквенно-цифровую, (буквы **только латинские**) (например: **KL1386**), а скрипт выдает «расшифровку» в соответствии с алфавитом (например: **Kilo Lima One Three Eight Six**). Регистр вводимой комбинации не должен влиять на результат (т.е. большие и маленькие буквы дают один и тот же результат).

Домашнее задание #B.2

ПриватБанк

Дублікат квитанції № 0.0.818663319.1
Код квитанції 0942-7857-4727-1069

Відправник ФЕТИСОВ АНДРІЙ ІГОРОВИЧ
Код -
Банк відправника ПРИВАТБАНК Код банку -

Одержувач ГУ ГОСУДАРСТВЕННОЙ МИГРАЦИОННОЙ СЛУЖБЫ УКРАИНЫ
Код 37764460
Банк одержувача ДЕРЖКАЗНАЧЕЙСЬКА СЛУЖБА УКРАЇНИ, М. КИЇВ

Період оплати 201611 - 201611
Дані лічильників № попер. поточ.

Особовий рахунок 548815
Сума літерами вісімсот десять грн 32 коп.
Призначення платежу *6313;548815;1;* ОФОРМЛ ТА ОБМІН ПАСПОРТА ГР. У
СТРОК: НЕ ПІЗНІШЕ НІЖ 7 РОБ ДНІВ, ФЕТИСОВ АНДРІ

Підпис відправника
Час формування документа 21:50 Дата валютування
Час проведення документа 21:50 Підпис банку (ЕЦП)

Щоб переглянути цей документ в електронній формі:
1. Зайдіть на сторінку rb.ua/check. 3. Введіть код квитанції та нат
2. Оберіть тип документа «Квитанція про оплату». 4. Ви зможете завантажити ел

Написать скрипт который будет словами записывать сумму заданную числом которое ввёл пользователь в пределах от 1 до 999 (включительно). Например **643** => «**шестьсот сорок три гривны**» (не забывая добавлять слово **гривен**, **гривна** и т.д. в зависимости от необходимого склонения).

Если задача решилась быстро и просто, то – расширяем диапазон от 1 до **999 999 999** **гривен**.