

# Функции в JavaScript



[ORTDNIPRO.ORG/JS](https://ORTDNIPRO.ORG/JS)

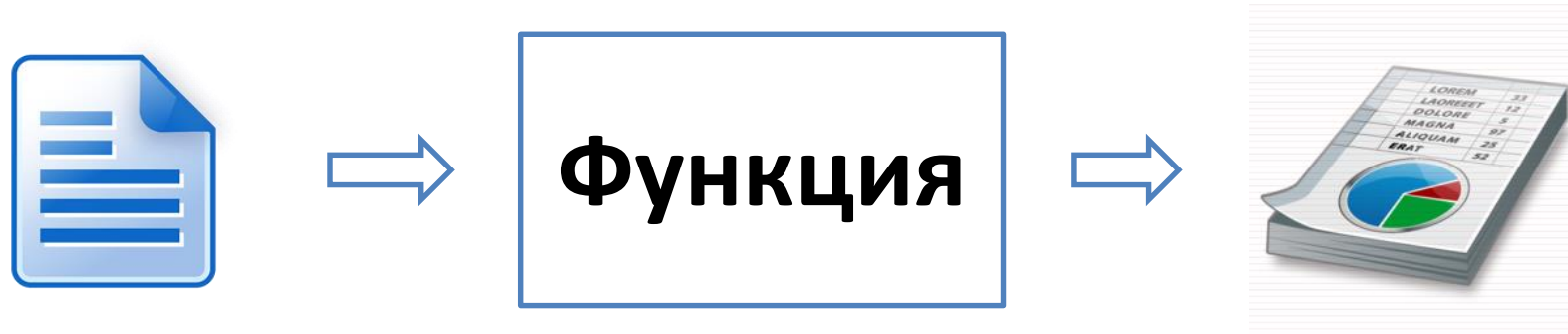
# 1. Функции и функциональные выражения

**Функция** – фрагмент кода, у которого есть имя, который можно вызывать из любого места в программе. **Функции** уменьшают количество кода в программе, код функции пишется один раз, используется многократно.



Идея функций заключается в следующем: **зачем писать многократно одно и тоже, лучше сказать программе: я уже такое писал, возьми и повтори это здесь, там, и еще вот там.**

# Функции в JavaScript



**Функция** также называют «подпрограммами» (программа в программе). Как и у программы в целом задача функции получить данные на входе и дать результат их обработки на выходе (хотя получение данных и/или выдача результатов не является обязательным).

# Какая польза от функций?

1. Уменьшаем дублирование (повторение) кода;
2. Проще вносить изменения;
3. Абстрагирование от деталей;

# 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>

# rest-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**.

# Самовывзывающаяся функция

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

**Самовывзывающаяся функция** – 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>

# 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
```

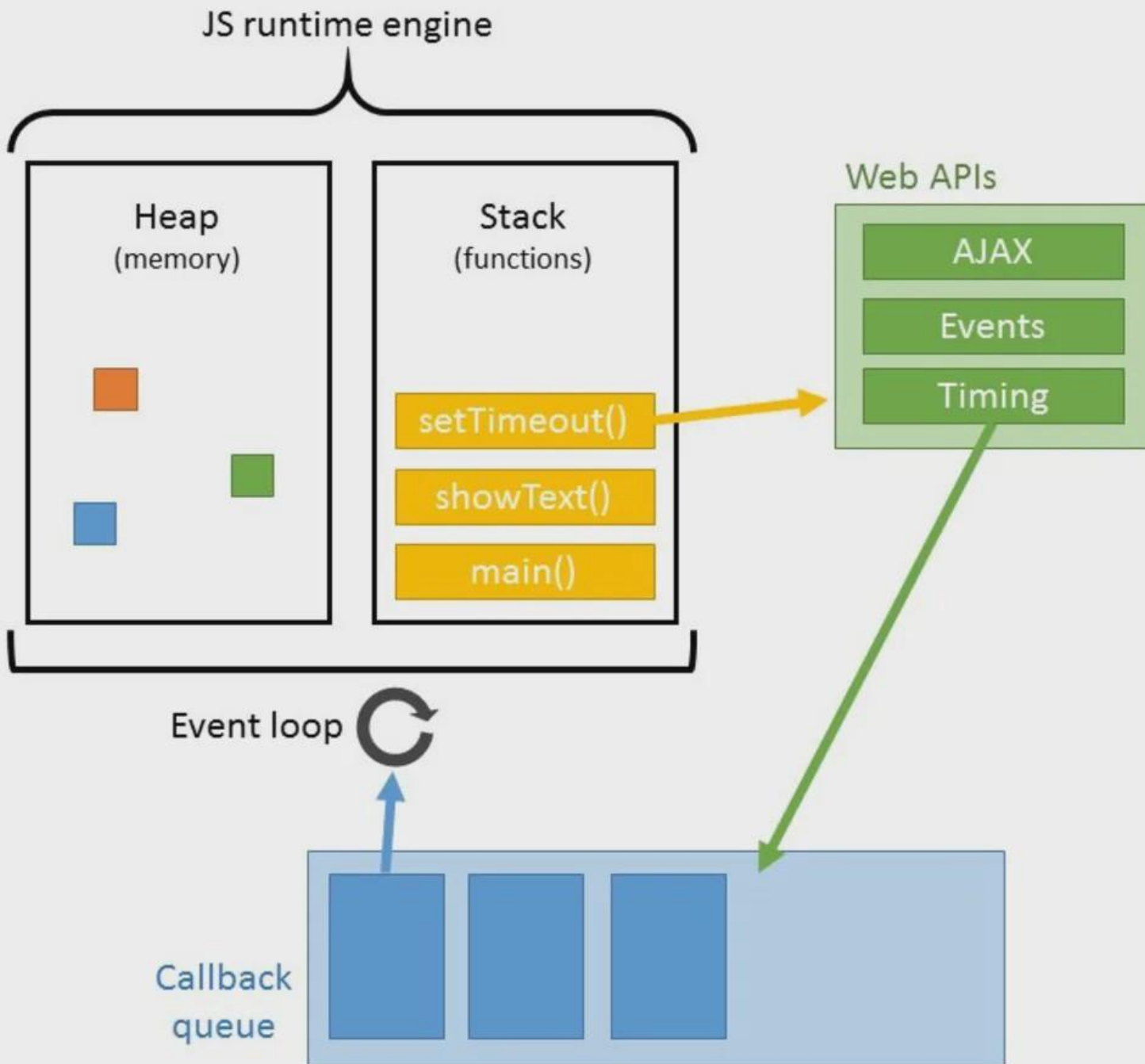
**setTimeout**(*some\_function*, *delay*) – вызовет функцию *some\_function* через *delay* миллисекунд. Сделает это один раз.

**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](https://youtu.be/j4_9BZezSUA)

*Тут докладчик еще более странный...*

## **2. 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>



# Перебирающий методы массива .forEach()

```
2  
3 let arr = [23, 4, 67, 117, 34, 0, 55, 78, 5, 9];  
4  
5 arr.forEach((item, index, array) => console.log(index, item));  
6
```

Функция переданная методу **.forEach()** массива будет применена к каждому элементу. Функция принимает три параметра, которые получают сам элемент (для которого вызывается функция), его индекс в массиве, и ссылка на сам массив. **С появлением цикла for-of востребованность этого метода упала.**

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>

**Немного практики #1**

## По мотивам: Домашнее задание #С.2



Національний  
банк України

...  
2021-01-01: 145, ...  
2021-02-03: 50, ... грн;  
2021-03-23: 17900000 грн;  
...

С перебирающими методами

Составьте список платежей (отсортированный от прошлого к будущему), когда ожидаются платежи по обязательствам госзайма, с суммой всех платежей которые в этот день должны быть выполнены (на одну дату могут приходиться несколько платежей, тогда на эту дату считаем сумму платежей). (Платежи, которые НЕ в гривне, пересчитайте в гривню).

Вы можете воспользоваться шаблоном  
в репозитории [./src/template-nbu/](https://github.com/nbu-template)

Немного практики #2

или

«О **callback**'ах»

# 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 **проверить 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/](#)



**Будет полезным**

# Перебирающие методы

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

**К следующему занятию будет  
полезно почитать о...**

# К следующему занятию...

1. **Объекты** и ключевое слово **this**;
2. Функция-**конструктор** объектов;
3. **Классы** в JavaScript;
4. Объекты типа **Promise**.

**Домашнее задание**  
**/сделать**

# Домашнее задание #D.1 | «Проверка ИНН»

КАРТКА  
фізичної особи - платника податків

повідомляє, що \_\_\_\_\_  
одержав(ла) ідентифікаційний номер  
наданий Державною податковою адміністрацією України  
згідно з даними, заповненими ним (нею) в обліковій картці.  
Дата занесення до Державного реєстру фізичних осіб - 06/02/1998  
(картка видана для пред'явлення до органів державної реєстрації,  
установ банків та інших).

 М.П. Для довідок

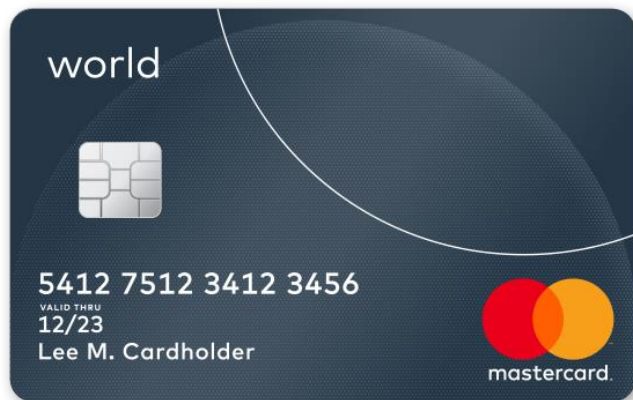
 \_\_\_\_\_  
( підпис ) ( прізвище та ініціали посадової особи  
органу Державної податкової служби )

\_\_\_\_\_ ( дата видачі картки )

*Пользователь вводит ИНН  
(физ. лица Украины),  
Необходимо определить:  
корректен ли код (нет ли  
в нём ошибки), и пол  
(М/Ж) владельца номера.*

**Для проверки: 3463463460; 2063463479.**

# Домашнее задание #D.2 | «Проверка номера карты»



Наш сайт принимает платёжные карты систем: **Visa, Mastercard, Maestro**. Пользователь вводит номер платёжной карты (*payment card number*) – 16 цифр (цифры могут быть разделены пробелами, или дефисами или записаны слитно, возможны пробелы в начале и в конце строки).






**Задача:** Проверить номер на корректность и определить платёжную систему. **Скрипт должен содержать функцию** которая принимает номер карты в виде строки, а результат работы выдаёт объект следующей структуры:

```
{  
    card: "2235778899000016",  
    correct: true, //or false  
    paymentSystem: "visa", //or another  
    accepted: true //or false  
}
```



Если карта относится к платёжной системе которую сайт не принимает, то поле **paymentSystem** оставляем пустым, а **accepted** устанавливаем в **false**. Если номер карты не корректный то **accepted** устанавливаем в **false** и поля **paymentSystem** и **accepted** не задаём.

## К домашнему заданию #D.2

 Visa	 MasterCard	 Discover	 AmericanExpress	 JCB
✓ 4412530595659632	✓ 5287324989755118	✓ 6011139619422678	✓ 340849911182813	✓ 3539584124038594
✓ 4813431262431071	✓ 5369658110635785	✓ 6011117040432748	✓ 345673843441369	✓ 3588422734539547
✓ 4381493988886337	✓ 5153000135610537	✓ 6011406220044898	✓ 345616475358716	✓ 3538044621974255
✓ 4739306813042299	✓ 5327520507510974	✓ 6011774117039986	✓ 375103335418603	✓ 3528852467705472
✓ 4464941706819170	✓ 5155034861872910	✓ 6011069037122495	✓ 375423401400255	✓ 3579534744222947
<a href="#">Generate Visa ➤</a>	<a href="#">Generate MasterCard ➤</a>	<a href="#">Generate Discover ➤</a>	<a href="#">Generate AmEx ➤</a>	<a href="#">Generate JCB ➤</a>

В помощь, генератор номеров банковских карт (**используйте для проверки**):

<https://www.freeformatter.com/credit-card-number-generator-validator.html>