

Объекты в JavaScript / AJAX



JavaScript
Courses

www.courses.dp.ua

Зачем нужны объекты?

Если бы мы решали задачу: в которой необходимо рассчитывать стоимость доставки посылок, каждая посылка – коробка (прямоугольный параллелепипед) у которой заданы высота, длина, ширина (в метрах), масса (в кг.) и идентификатор посылки (просто номер), обрабатываемых **посылок может быть много...** Также необходим расчёт объёма посылки для логистических целей

Расчёт стоимости при этом осуществляется по такой схеме: если объем посылки до 1 куб. метра (включительно) то стоимость доставки рассчитывается по массе исходя из цены 10 грн. за кг. Если объем превышает 1 куб. м. то стоимость рассчитывается по объему исходя из цены 100 грн. за куб. м. Также в цену включается «упаковка» которая рассчитывается исходя из площади поверхности коробки по цене 5 грн. за кв. м.

Сложность была бы не столько в расчётах, сколько в способе хранения информации и её применении в расчётах...

Объекты...

Объекты в JavaScript

```
2
3   let person = {
4       name: "Jhon",
5       lastName: "Smith",
6       sayHello: function(){
7           return `Hello my name is ${this.name} ${this.lastName}`;
8       }
9   }
10
11   console.log( person.sayHello() );
```

Объект в JavaScript представляет собой ассоциативный массив содержащий данные (свойства) и функции (методы) которые эти данные обрабатывают. **Объект** в JavaScript один из шести базовых типов данных.

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

Ключевое слово **this**

```
2
3   let person = {
4       name: "Jhon",
5       lastName: "Smith",
6       sayHello: function(){
7           return `Hello my name is ${this.name} ${this.lastName}`;
8       }
9   }
10
11 console.log( person.sayHello() );
```

Ключевое слово **this** – ссылка на сам объект. Другими словами **this** указывает на тот ассоциативный массив (объект) которому принадлежит функция, в которой **this** используется встречается. **this** используется только в функциях объекта. **Важно: у arrow-функций нет своего this.**

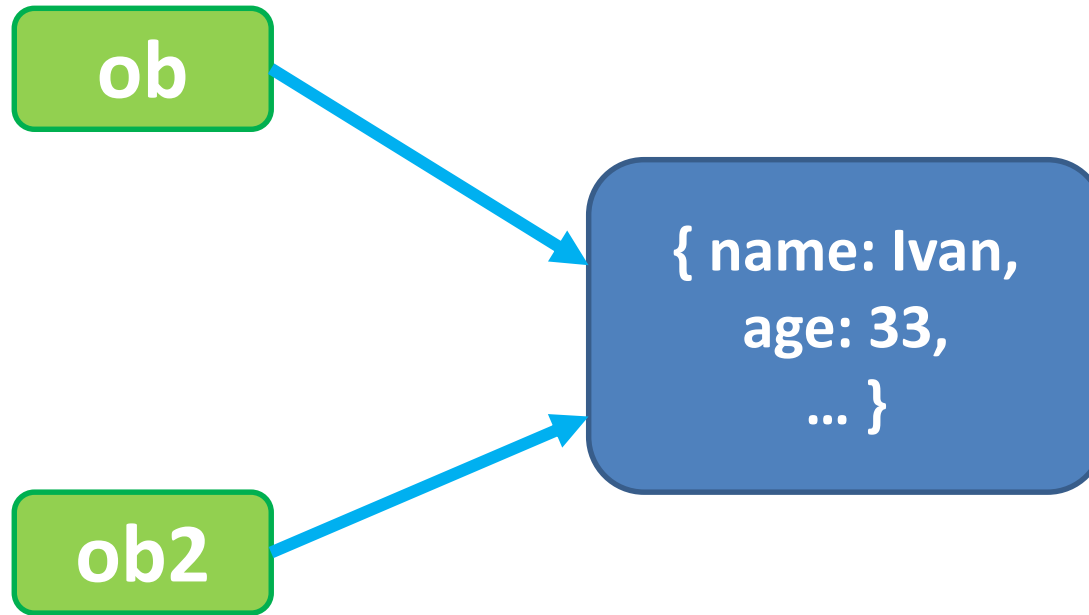
Подробнее: <https://learn.javascript.ru/object-methods>

Объекты в JavaScript

```
2
3   let person = {
4       name: "Jhon",
5       lastName: "Smith"
6   }
7
8   person = null;
9
10  console.log(person, typeof person);
11
```

null – заглушка на случай “когда объекта нет”.

Объекты в JavaScript



object - ссылочная структура данных, т.е сам объект находится где-то в памяти, а в переменной находится только ссылка на него, поэтому когда мы копируем такую переменную в другую, то копируются только ссылки, а сам объект остаётся одним и тем же.

this привязывается в динамике

```
2
3   let func = function(){
4       return `Hello my name is ${this.name} ${this.lastName}`;
5   }
6
7   let person_1 = {
8       name: "Jhon",
9       lastName: "Smith",
10      sayHello: func
11  }
12
13  let person_2 = {
14      name: "Alice",
15      lastName: "Gates",
16      sayHello: func
17  }
18
19  console.log( person_1.sayHello() );
20  console.log( person_2.sayHello() );
21
```

this привязывается к объекту в момент вызова метода, поэтому одна и та же функция может входить в состав двух и большего количества объектов.

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

Конструктор – Когда нужно много однотипных объектов

```
2
3   let func = function(){
4       |   return `Hello my name is ${this.name} ${this.lastName}`;
5   }
6
7   function Person(name, lastName){
8       |   this.name      = name;
9       |   this.lastName  = lastName;
10      |   this.sayHello  = func;
11  }
12
13  let person_1 = new Person('Jhon', 'Smith');
14  let person_2 = new Person('Alice', 'Gates');
15  let person_3 = new Person('Bill', 'Roberts');
16
17  console.log(person_1.sayHello());
18  console.log(person_2.sayHello());
19  console.log(person_2.sayHello());
20
```

Функция-конструктор - позволяет создавать много однотипных объектов. Функция конструктор всегда должна использоваться с оператором **new**, иначе у неё не будет доступа к **this** нового созданного объекта.

Использовать оператор **return** не нужно. Конструктор может (и как правило должен) иметь параметры.

Подробнее: <https://learn.javascript.ru/constructor-new>

Прототипы

Прототипы

У объекта может быть объект-предок, в **JavaScript** его называют **прототипом**. Если требуемое свойство (или метод) не найден в объекте, то оно ищется у **прототипа**.

Прототип это объект который «дополняет» своими свойствами и методами другой (дочерний) объект. Установить кто у объекта будет **прототипом** можно при помощи свойства **__proto__**.

Благодаря **прототипам** в JavaScript можно организовать объекты в «**цепочки**» так, чтобы свойство, не найденное в одном объекте, автоматически искалось бы в другом (родительском).

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

Прототипы

```
2
3   let func = function(){
4       return `Hello my name is ${this.name} ${this.lastName}`;
5   }
6
7   let family = {
8       lastName: "Smith",
9       sayHello: func
10  }
11
12  function Person(name){
13      this.name      = name;
14      this.__proto__ = family;
15  }
16
17  let person_1 = new Person('Jhon');
18  let person_2 = new Person('Alice');
19  let person_3 = new Person('Bill');
20
21  console.log(person_1.sayHello());
22  console.log(person_2.sayHello());
23  console.log(person_2.sayHello());
24
```

Свойство или метод не найденные в объекте – будут взяты из **прототипа** (или *прототипа* *прототипа*, если в цепочке прототипов искомое свойство или метод есть).

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

`.toString() / .valueOf()`

Методы `.toString()`/`.valueOf()` у объектов

```
2
3   let auto_1 = {
4       title: "Ford Focus",
5       id: "AE5589BH"
6   }
7
8   let auto_2 = {
9       title: "Honda Accord",
10      id: "CH5633TB",
11      toString: function(){
12          return `${this.title} (${this.id})`;
13      }
14   }
15
16   alert(auto_1);
17   alert(auto_2);
18
```

localhost:5000 says
[object Object]

OK

localhost:5000 says
Honda Accord (CH5633TB)

OK

Метод `.toString()`, если он определен у объекта – позволяет браузеру корректно преобразовать объект **к строке**. Также есть метод `.valueOf()` для преобразования **к числу**.

Подробнее: <https://learn.javascript.ru/object-toprimitive>

try/catch/finally/throw

или

Обработка ошибок

Обработка ошибок (исключений)

```
2
3   let f = function(a, b){
4       |   return a + b;
5       |
6       |
7       f = 42;
8
9       try{
10          |   let result = f(2, 3);
11          |   }catch(e){
12             |   console.log("This code if we have error", e);
13             |   }finally{
14                |   console.log("This code work always");
15                |
16                }
```

Если в блоке **try** произойдёт ошибка, выполнение блока прекратится и перейдёт к блоку **catch**, в котором могут быть выполнены какие-либо действия направленные на нивелирования влияния ошибки на работу скрипта. Если в блоке **try** ошибка не произошла, то блок **catch** не выполняется. Независимо от того произошла ошибка или нет, после **try-catch** скрипт пойдёт выполняться дальше, как ни в чём не бывало. Блок **finally** выполняется в любом случае. В этом блоке обычно размещается код который должен при любом варианте развития событий завершить те или иные действий (например убрал иконку-лоадер с экрана независимо от того успешна ли была загрузка).

Подробнее: <https://learn.javascript.ru/try-catch>

Генерация ошибки | оператор **throw**

```
2
3     try{
4
5         throw new Error("Info about error!");
6
7     }catch(e){
8         console.log("This code if we have error", e);
9     }finally{
10        console.log("This code work always");
11    }
12
```

При помощи оператора **throw** мы можем «выбросить» свою ошибку, для этого оператору достаточно передать любое значение, но хорошей практикой является использование для этих целей объекта **Error** или производного от него.

Подробнее: <https://developer.mozilla.org/uk/docs/Web/JavaScript/Reference/Statements/throw>

AJAX

Asynchronous Javascript and XML

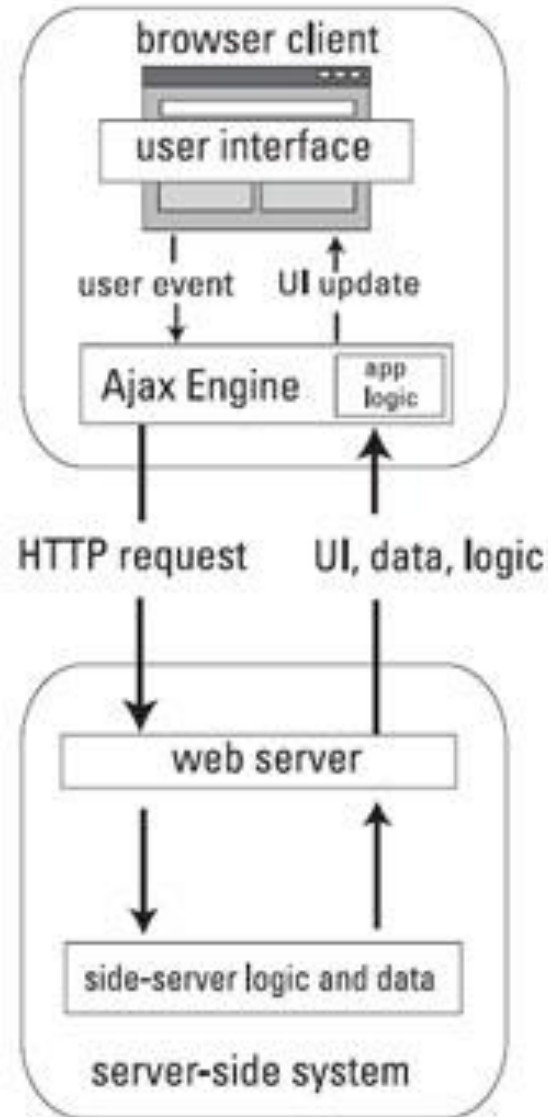
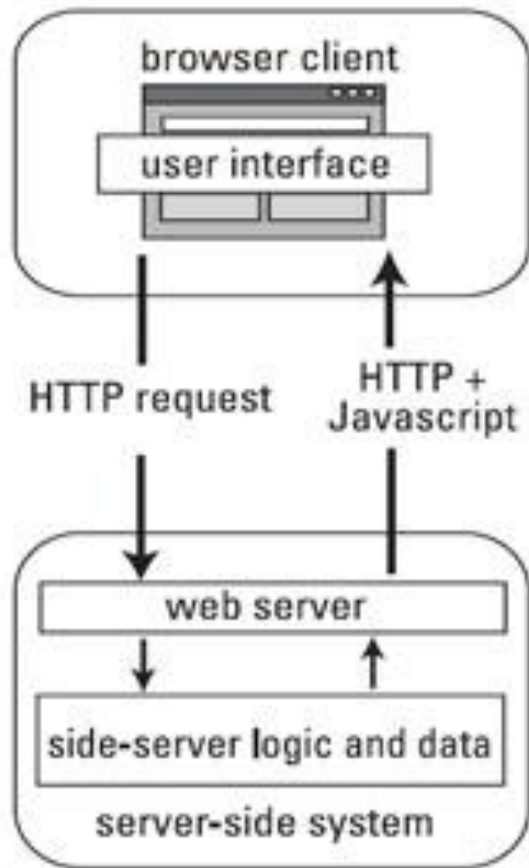
Asynchronous JavaScript And XML



За изменение страницы в браузере пользователя отвечает JavaScript, но до этого момента JavaScript изменял страницу только на основе данных полученные еще при загрузке страницы в браузер и/или в зависимости от действий пользователя. Получить какие-то новые (дополнительные) данные JavaScript не мог.

С появлением в браузерах специального объекта **XMLHttpRequest** у **JavaScript** появилась возможность делать HTTP-запросы к сайтам, и изменять страницу уже на основе данных которых не было при загрузке странице. Т.е. дозагружать HTML и/или другие данные и вставлять их на страницу.

Asynchronous JavaScript And XML



Идея заложенная в **AJAX** - не перезагружая страницу полностью, запросить у сервера данные и вставить их в дерево документа.

XMLHttpRequest

или

AJAX на практике

XMLHttpRequest

Объект **XMLHttpRequest** позволяет использовать функциональность HTTP-клиента, а по простому – делать HTTP-запросы когда страница уже в браузере.

*Несмотря на наличие **XML** в названии объекта, с его помощью можно передавать и другие форматы данных.*

Подробнее: <http://xmlhttprequest.ru>

XMLHttpRequest

```
2
3   let url = 'https://api.privatbank.ua/p24api/pubinfo?exchange&json&coursid=11';
4
5   let xhr = new XMLHttpRequest();
6
7   xhr.open("GET", url);
8
9   xhr.onload = function(){
10       let data = JSON.parse(this.response);
11       console.log(data);
12   }
13
14   xhr.send();
15
```

Объект **XMLHttpRequest** позволяет использовать функциональность HTTP-клиента, а по простому – делать HTTP-запросы когда страница уже в браузере.

Подробнее: <http://xmlhttprequest.ru>

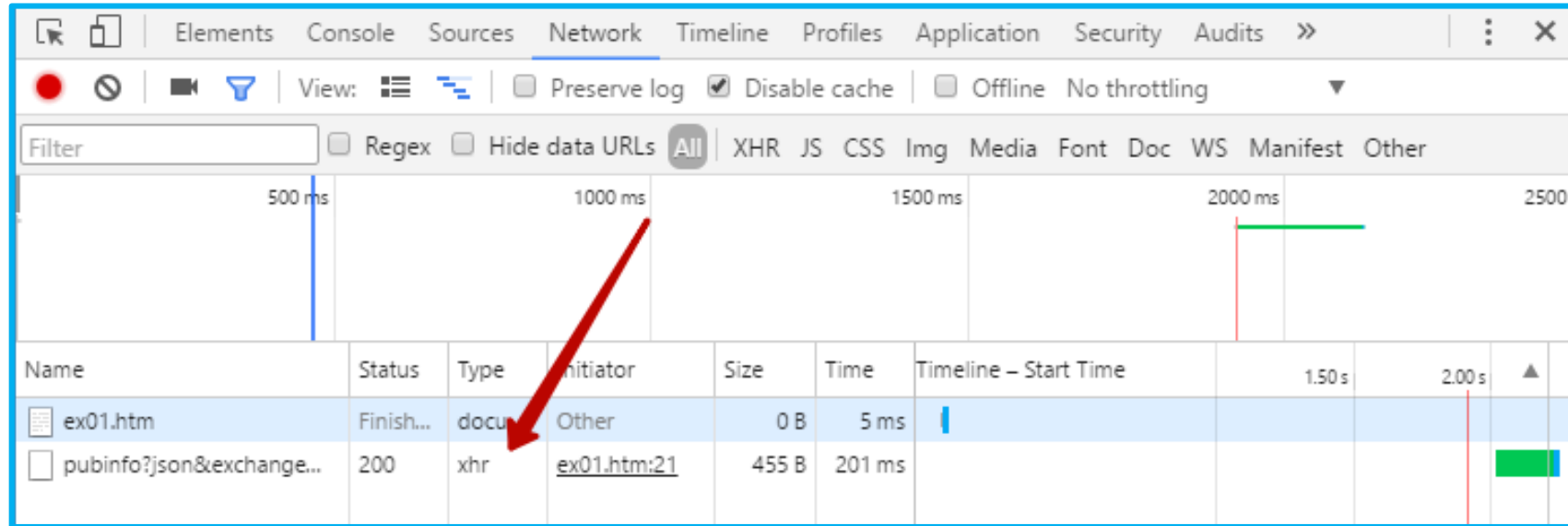
XMLHttpRequest

XMLHttpRequest – поддерживает событийную модель, и в зависимости от развития ситуации генерирует те или иные события.

Синхронный запрос – при котором браузер ждёт ответа, скрипт при этом «замирает» до прихода ответа.
Асинхронный – скрипт продолжает выполняться, при поступлении ответа будет вызвана функция зарегистрированная как обработчик события **onload**.

Подробнее: <http://xmlhttprequest.ru>

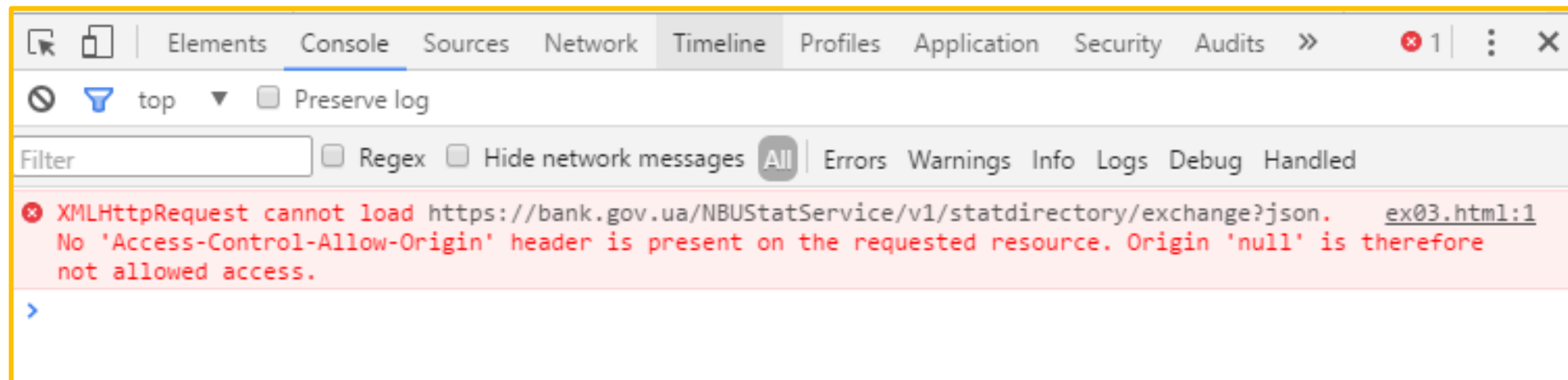
XMLHttpRequest



В консоли разработчика хорошо заметны запросы которые делались через **XMLHttpRequest** – по характерной метке **type** равной **xhr**.

Кросс-доменные запросы

Кросс-доменные запросы



Не все AJAX запросы безопасны, браузер бдит 😊

Сравните результаты AJAX-запросов по адресам:

http://www.courses.dp.ua/demo/ajax_json_1/

http://www.courses.dp.ua/demo/ajax_json_2/

Кросс-доменные запросы

Кросс-доменные запросы (т.е. запросы к другому домену, не к тому с которого загружен скрипт) проходят контроль безопасности (**который осуществляет браузер**).

Чтобы страница могла быть доступна через кросс-доменные запросы (читай **AJAX** запросы к страницам других сайтов), страница должна сама сказать об этом, а именно установить в **HTTP** ответе заголовок **Access-Control-Allow-Origin**.

Подробнее: <https://learn.javascript.ru/xhr-crossdomain>

JSON

JavaScript Object Notation

JSON (JavaScript Object Notation)

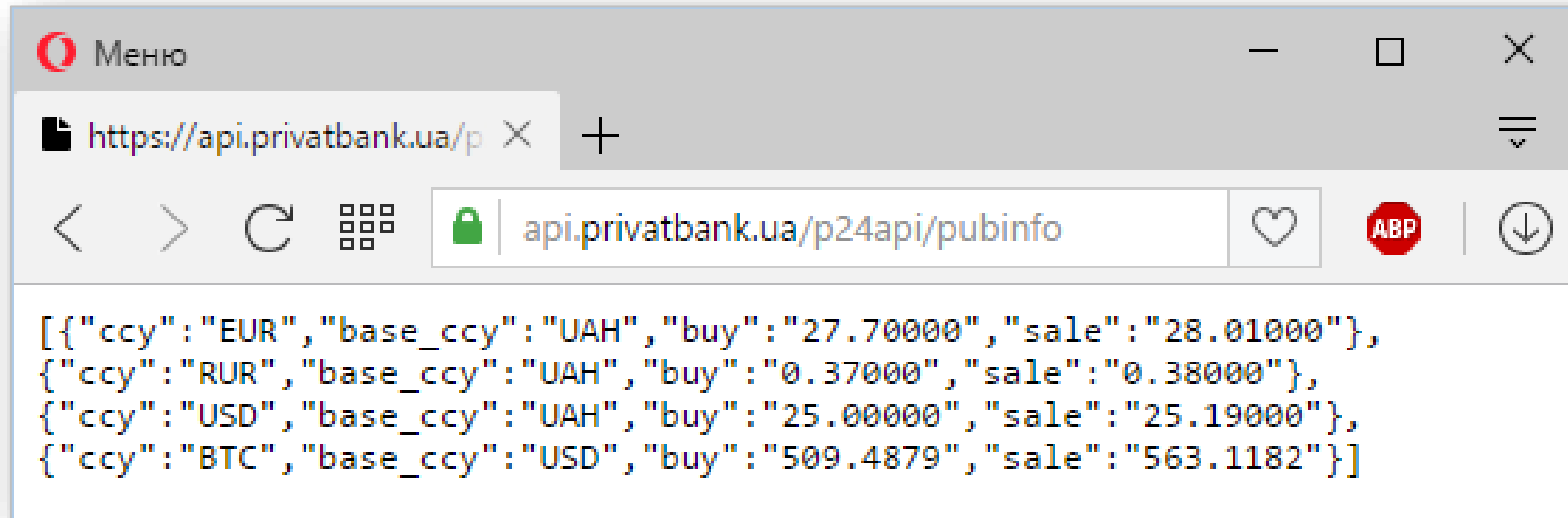
***JSON** - текстовый формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Основан на синтаксисе (правилах записи) массивов и объектов в **ECMAScript**. Формат поддерживается практически всеми современными языками программирования.*

```
' { "name": "Вася", "age": 35, "isAdmin":  
  false, "friends": [0,1,78,99] } ';
```

*Сохранение объекта/массива в строковом виде (с последующим восстановлением) также называют **сериализацией**.*

<http://www.json.org/json-ru.html>

JSON в интернете



JSON является популярным форматом для обмена информацией в Интернете. Большое количество сервисов позволяют получить информацию в формате **JSON** для более удобной её обработки. В JavaScript преобразование в **JSON** выполняется при помощи метода **JSON.stringify()**, а преобразование из **JSON** в объект методом **JSON.parse()**

API


(Application Programming
Interface)

Application Programming Interface

API (интерфейс программирования приложений, интерфейс прикладного программирования)
([англ.](#) *application programming interface*) — По сути набор правил, которые определяют как необходимо общаться со сторонним сайтом/программой/системой если мы хотим запросить у него данные или передать ему данные.

Полезные API

API Приватбанка





 Разработчикам

API ПриватБанкаAPI юр.лицAPI LiqPay

[Главная](#)
[Регистрация](#)
[Платежные](#)
[Информационные](#)
[Публичные](#)

Доступные API ПриватБанка

ПриватБанк стал первым банком в мире, открывшим публичное API (в сентябре 2009 года). Сегодня его используют уже более 4 900 партнеров и не только в Украине.

ПлатежныеИнформационныеПубличныеAPI для юр.лиц

Архив курсов валют ПриватБанка, НБУ

API позволяет получить информацию о наличных курсах валют ПриватБанка и НБУ на выбранную дату. Архив хранит данные за последние 4 года

[Документация](#)

Инфраструктура ПриватБанка. Банкоматы

Информация о размещении банкоматов ПриватБанка

[Документация](#)

Отделения

Информация о размещении отделений ПриватБанка

[Документация](#)

Курсы валют ПриватБанка

API предоставляет информацию о наличных, безналичных курсах валют ПриватБанка

[Документация](#)


Терминалы самообслуживания

Информация о размещении терминалов самообслуживания ПриватБанка

Оплата частями

API позволяет получить информацию о доступной сумме для покупок по программе "Оплата частями"

© 2014-2019 ПриватБанк



<https://api.privatbank.ua/>

API Национального Банка Украины



**НАЦІОНАЛЬНИЙ
БАНК УКРАЇНИ**

Валютные API, информация о финансовом рынке и банковском секторе

https://bank.gov.ua/control/uk/publish/article?art_id=38441973

Информация по платёжной карте

BINLIST.NET

4571 7360

Enter the first digits of a card number (BIN/IIN)

SCHEME / NETWORK

Visa

TYPE

Debit / Credit

BANK

Jyske Bank, Hjørring

www.jyskebank.dk

+4589893300

BRAND

Visa/Dankort

PREPAID

Yes / No

CARD NUMBER

LENGTH

16

LUHN

Yes / No

COUNTRY

DK Denmark

(latitude: 56, longitude: 10)

Сервис позволяют получить информацию в формате **JSON**. Но необходимо зарегистрироваться и получить ключ

<https://binlist.net> | <https://lookup.binlist.net/536354>

Альтернативный сервис: <https://www.bincodes.com/api-bin-checker/>

Домашнее задание
/узнать

Современный учебник Javascript

Перед вами учебник по JavaScript, начиная с основ, включающий в себя много тонкостей и фишек JavaScript/DOM.

[смотреть на Github](#)

Поделиться:

[НАЙТИ](#)

Содержание

Первые две части посвящены JavaScript и его использованию в браузере. Затем идут дополнительные циклы статей на разные темы.

Предварительные знания – лучший помощник в обучении, поэтому к следующему занятию жду, что **пройдёте разделы 1-й части 11.1-11.6**

<http://learn.javascript.ru/>

+1 Книга



get/set методы

*В составе объектов в JavaScript могут использоваться т.н. геттеры и сеттеры (**get** и **set** методы) – узнайте о них по подробнее.*

Домашнее задание
/сделать

Домашнее задание #D.1

Воспользуйтесь API Национального Банка Украины и выведите в консоль последовательный список курсов доллара за ноябрь 2019 г. по дням.

По такой структуре:

```
01.11.2019 - 24.56 грн.  
02.11.2019 - 24.86 грн.  
03.11.2019 - 25.01 грн.  
...  
29.11.2019 - 24.21 грн.  
30.11.2019 - 24.98 грн.
```