

# Программирование на JavaScript

---

**JS**  
**COURSE**  
**ORT DNIPRO**

---

**ORT****DNIPRO**.ORG/**JS**

**Первым делом**

**Наша группа: JS13**

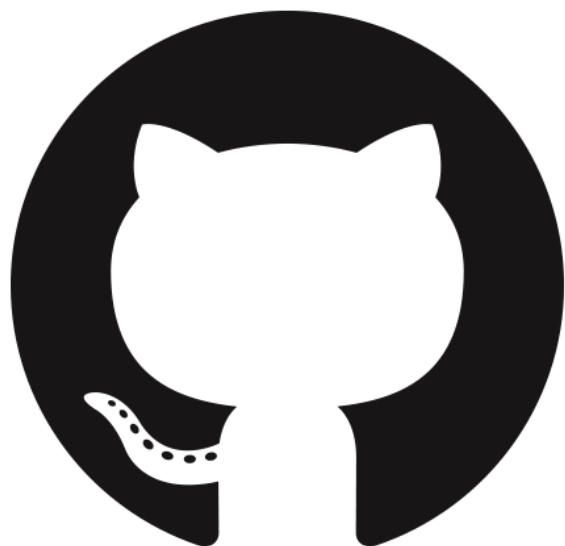
**<https://js13.club>**



Общение при помощи  
мессенджера

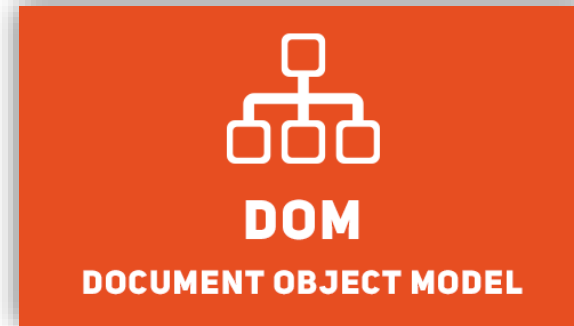
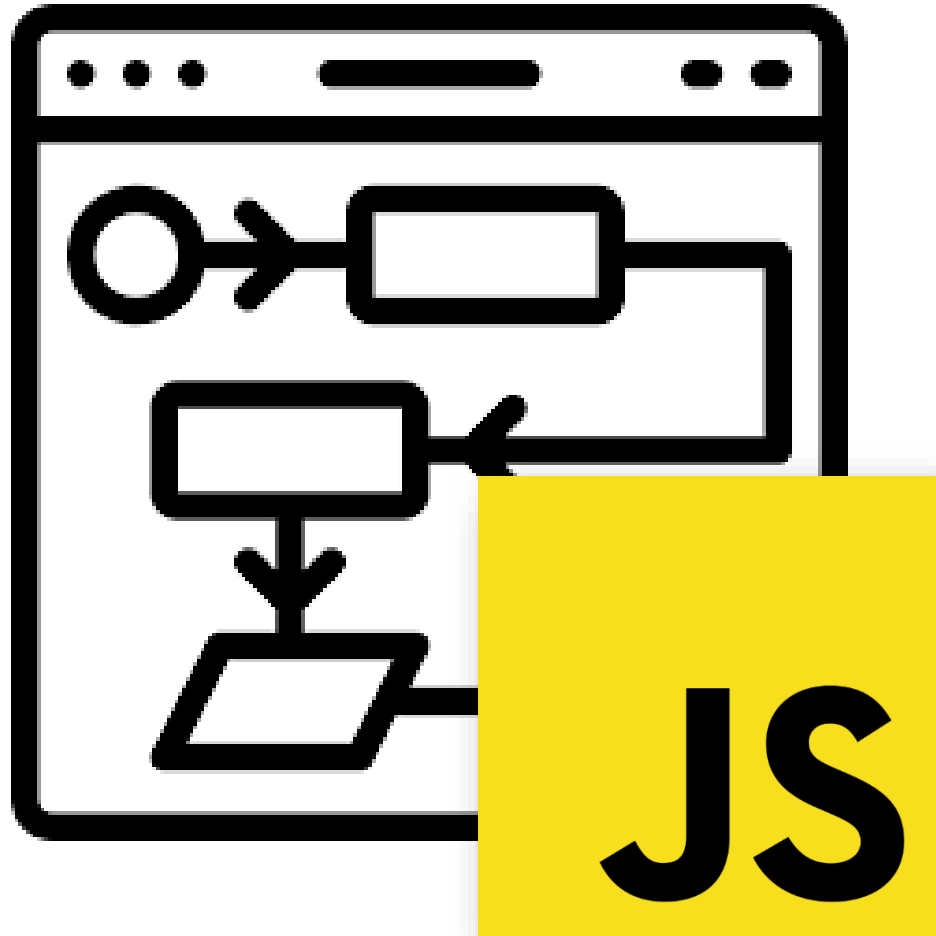
**Telegram**, а для  
обмена материалами  
и домашних заданий  
будем использовать

**GitHub**



**О чём курс?**

# О программировании и веб-разработке с применением языка JavaScript



Firestore



Поехали!

# ES

**ECMAScript**

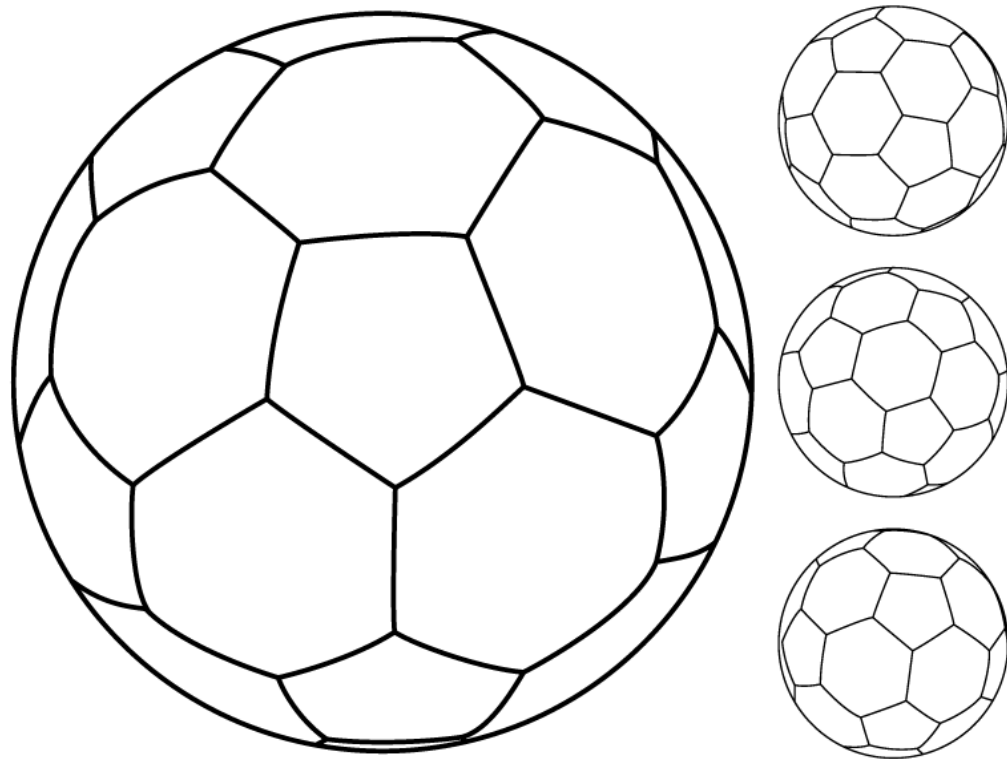
vs

# JS

**JavaScript**



# ECMAScript



*Спецификация...*

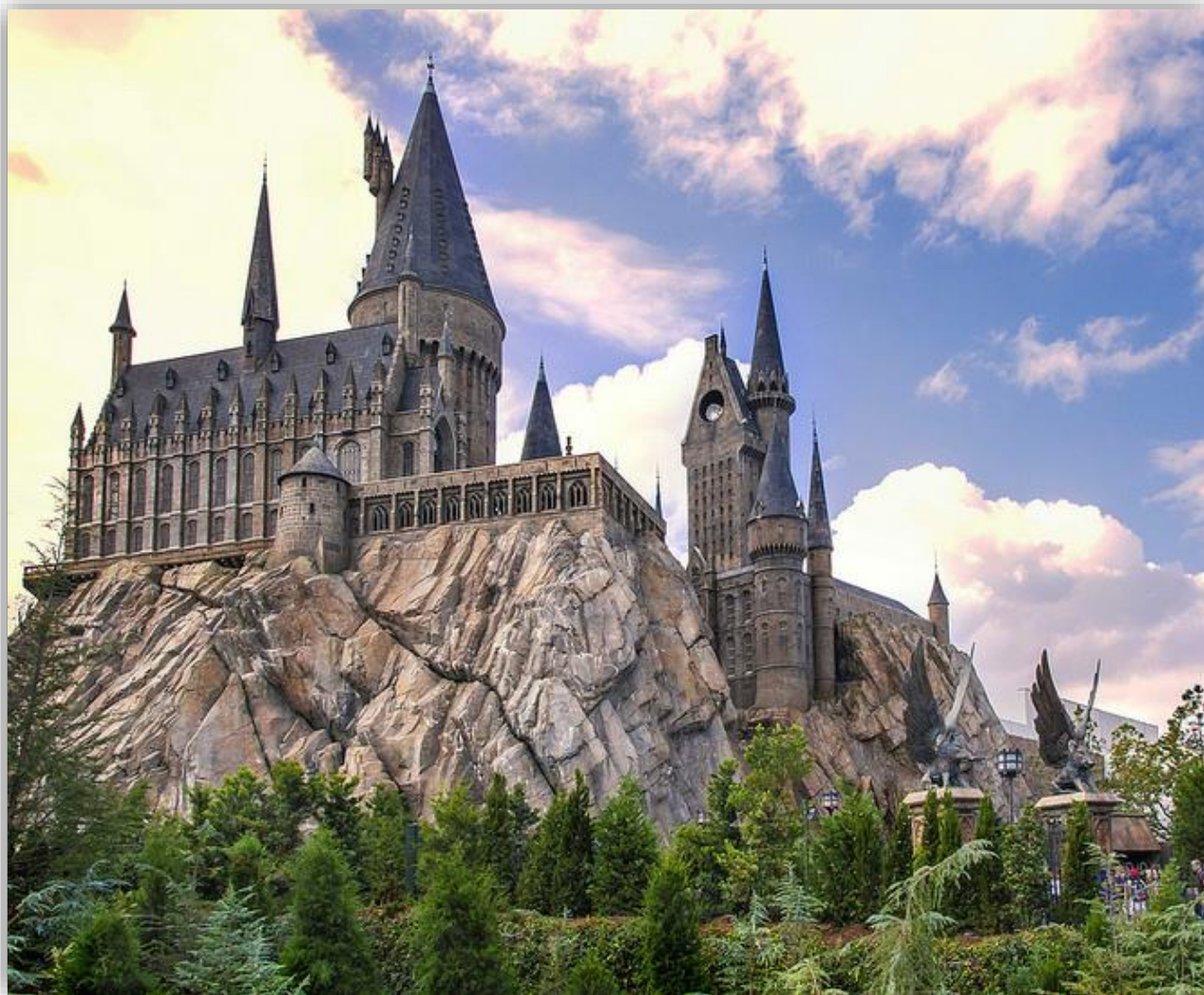
# JavaScript



*...и её реализация*



# Развитие JavaScript...



# 1. Переменные и типы данных

# Переменные в JavaScript

```
1
2  var user_name    = "Elena";
3
4  let user_age     = 27;
5
6  const user_inn   = 3252873450;
7
8  console.log(user_name, typeof user_name);
9  console.log(user_age,  typeof user_age);
10 console.log(user_inn,  typeof user_inn);
11
```

Переменные объявляются при помощи ключевых слов **var**, **let** и **const**. Первые два способа отличаются областью видимости переменной которая создаётся. Третий создаёт переменную у которой нельзя заменить значения после инициализации.

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



# Типы данных в JavaScript

```
3
4 undefined //undefined
5
6 number //42, -35.783, 4e18, NaN, Infinity ...
7
8 string //'Hello', "World", `!!!` ...
9
10 boolean //true, false
11
12 object //null, { prop:'value', ... } ...
13
14 symbol //Symbol('marker'), Symbol.for('label')
15
16 bigint //35n, 9999999999999999999999999999999n
17
18 function //function(...){...}, (...) => ...
19
```

Переменные могут хранить значение одного из поддерживаемых типов данных. В ходе выполнения кода может меняться как содержимое переменной так и его тип.

**Тип влияет на то какие операции могут быть выполнены с переменной.** Тип переменной можно получить при помощи оператора/функции **typeof**.

# Преобразование типов

Несмотря на наличие механизма автоматического приведения типов может возникать ситуации требующие принудительного преобразования типов (чаще всего **string** к **number**), для этого есть ряд возможностей. В первую очередь при помощи соответствующих функций-конструкторов **Number()**, **String()**, **Boolean()**, **BigInt()**, **Symbol()** и т.д.

Подробнее: <https://learn.javascript.ru/types-conversion>

## 2. Export/Import (ES Modules)

# Процедура экспорта/импорта модулей (ES Modules)

```
JS main.js  x  JS lib.js
assets > js > JS main.js
1
2  import def from './lib.js';
3
4  import { pi as Pi, sum, config } from './lib.js';
5
6  console.log(def, Pi, sum, config);
7
8

10
11  <script src="./main.js" type='module'></script>
12
```

```
JS main.js  JS lib.js  x
assets > js > JS lib.js > ...
1
2  const pi = 3.14;
3
4  function sum(a, b) {
5      return a + b;
6  }
7
8  let config = {
9      enable: true,
10     count: 42,
11     id: 'HX883'
12 }
13
14 export { pi, sum , config };
15
16 export default config;
17
```

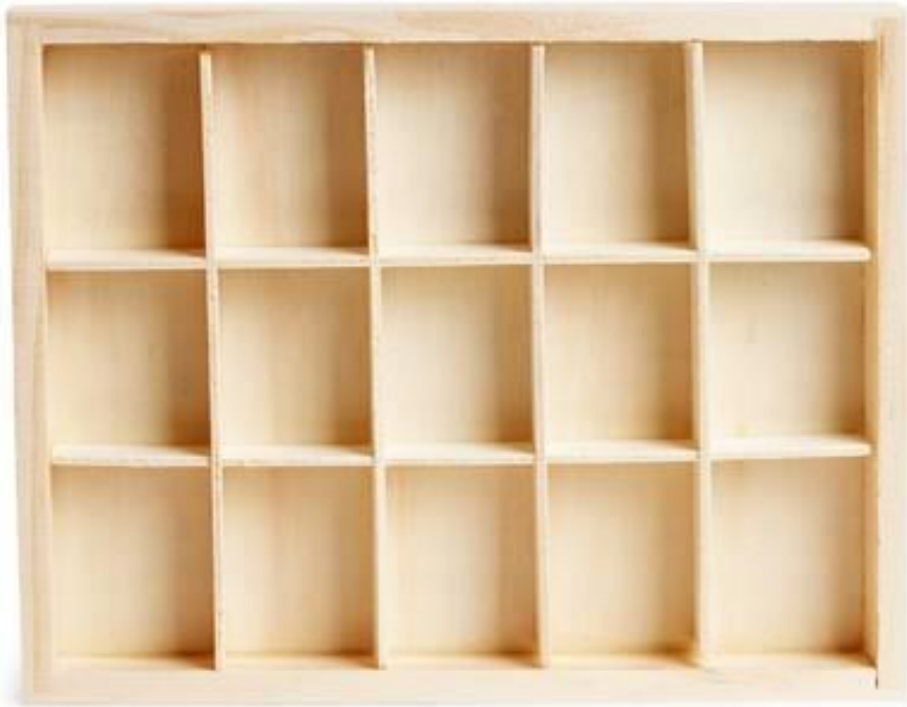
Директивы **export/import** по сути позволяют подключать сторонние (специальным образом подготовленные) *js-файлы* (**ES-модули**) с кодом непосредственно из *js-кода*. Для работы этого механизма первый файл (в котором импортируются другие) должен быть подключен с атрибутом **type='module'**.

<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/export>  
<https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Statements/import>



# **3. Коллекции (структуры данных)**

# Коллекции (структуры данных)



Коллекциями в языках программирования называют **структуры данных** предназначенные для хранения множества значений. Коллекции в **JavaScript** можно разделить на те которые хранят пары **ключ => значение** (массив **Array**, ассоциативные массивы **Object** и **Map**) и просто хранящие значения без индексации (множество – **Set**).

# Коллекции в JavaScript

*Большинство (но не все) коллекций построено по принципу хранения пар: ключ-значение и такие коллекции называют **словари**...*

- |  |        |
|--|--------|
| 1. Массивы (с числовыми индексами)                 | Array  |
| 2. Ассоциативные массивы (со строковыми индексами) | Object |
| 3. Словари (с ключом произвольного типа)           | Map    |
| 4. Множество (без ключей, элементы не повторяются) | Set    |

*Тип данных всех коллекций – **object**, все они построение на базе объектов.*

## 4. Массивы (Array)



## Массив / Array

**Массив** (с числовыми индексами) – коллекция хранящая неограниченное количество элементов (ячеек), у каждого из которых есть порядковый номер. Типы данных хранимых в ячейках массива не ограничены, в рамках одного массива в разных ячейках могут храниться разные типы данных, в том числе и другие (вложенные) массивы.

# Базовые действия с массивом

```
2
3   let arr = [10, 23, 167, 32, 77];
4   //let arr = new Array(10, 23, 167, 32, 77);
5
6   arr[2] = 787;
7
8   console.dir(arr); //All structure of object.
9
10  console.log("Array length: ", arr.length);
11  console.log("Array: ", arr[0], arr[1], arr[2], arr[3], arr[4]);
12  console.log("Out of...", arr[42]); //undefined;
13
14  arr.push(100);
15  let last = arr.pop();
16
17  arr.unshift(200);
18  let first = arr.shift();
19
20  delete arr[2]; //WARNING!
21
```

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

## Полезные методы массива

```
2
3   let arr = [10, 23, 167, 32, 23, -56, 0, 77];
4
5   arr.indexOf(23);           //1;
6   arr.lastIndexOf(23);       //4;
7   arr.indexOf(99);           //-1 - It's mean NotFound;
8
9   arr.includes(32);           //true;
10  arr.includes(88);           //false;
11
12  arr.reverse();              //Previous order is lost;
13  console.log(arr);           //[77, 0, -56, 23, 32, 167, 23, 10];
14
15  let arr_2 = arr.slice(2, 5);
16  console.log("Sliced:", arr_2); //[-56, 23, 32];
17
18  arr.splice(2, 5, 'a', 'b', 'c');
19  console.log("Spliced:", arr); //[77, 0, "a", "b", "c", 10]
20
```

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



# Псевдомассивы на примере строкового типа

```
2
3   let str = 'Joan Peter Michelle Laura Stiven';
4
5   console.log('String length:', str.length);
6   console.log(str[0], str[1], str[2], str[3]);
7
8   let arr = str.split(' '); //Create Array;
9   console.dir(arr);    //[ "Joan", "Peter", "Michelle", "Laura", "Stiven" ];
10
11  let new_str = arr.join(', ');
12  console.log(new_str); // 'Joan, Peter, Michelle, Laura, Stiven'
13
```

**Псевдомассивами** называют структуры у которых есть возможность обратиться к элементами при помощи синтаксиса [...], а также возможность узнать количество элементов (**.length**), но, при этом, не являющиеся массивами и не обладающие функциональностью массивов. В частности строки не позволяют менять символы строки.

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



## 5. Оператор ...

(**spread** оператор, оператор деструктуризации)

# Оператор ... (spread оператор)

```
2
3   let arr_1 = [1,2,3];
4   let arr_2 = [4,5,6, ...arr_1];
5
6   console.log(arr_1, arr_2); //[1, 2, 3] > [4, 5, 6, 1, 2, 3];
7
8   let maximun = Math.max(...arr_2);
9   console.log("Maximum", maximun); //6
10
11  let arr_copy = [...arr_2]; //One level copy of arr_2
12  console.log(arr_copy); //[4, 5, 6, 1, 2, 3];
13
```

Оператор ... (**spread** оператор) находясь по правую сторону от оператор присвоения (или при передаче параметров функции) позволяет подставить всё содержимое массива или любого другого итерируемого (перебираемого), объекта.

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

# Деструктуризация массива

```
2
3   let arr = ['Alfa', 'Beta', 'Gamma', 'Delta', 'Epsilon'];
4
5   let [a, b] = arr;
6
7   console.log(a, b); //Alfa Beta;
8
9   let [c, d, ...e] = arr;
10
11  console.log(c, d, e); //Alfa Beta ["Gamma", "Delta", "Epsilon"];
12
```

**Деструктуризация массива** – способ извлечь элементы массива для присваивания их значений отдельным переменным.

Подробнее: <https://learn.javascript.ru/destructuring-assignment>

## 6. Ассоциативный массив (Object)

# Базовые действия с объектом (ассоциативным массивом)

```
2
3  let parcel = {
4      title: "Gift",
5      width: 200,
6      height: 300,
7      length: 100,
8      price: 199
9  }
10
11  parcel.price      = 119;
12  parcel.fragile    = true;
13  parcel['city code'] = '49000';
14
15  console.dir(parcel);
16
17  let {title, price, ...others} = parcel;
18
19  console.log(title, price); //Gift 199
20  console.log(others); /* { width: 200, height: 300,
21                        length: 100, fragile: true,
22                        city code: "49000" } */
23
```

**Ассоциативный массив** это также коллекция вида ключ-значение, но в отличие от массивов **ключом выступает** не число, а **строка**. В JavaScript в качестве ассоциативных массивов выступают **объекты (object)** - одноимённый тип данных). Можно сказать также, что объекты в JavaScript построены на базе концепции ассоциативных массивов. Объекты также могут быть подвержены **деструктуризации**. Понятие длины (**length**) и последовательности элементов в ассоциативных массивах не применяется.

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

Подробнее: <https://learn.javascript.ru/destructuring-assignment>

# Объект - ссылочная структура

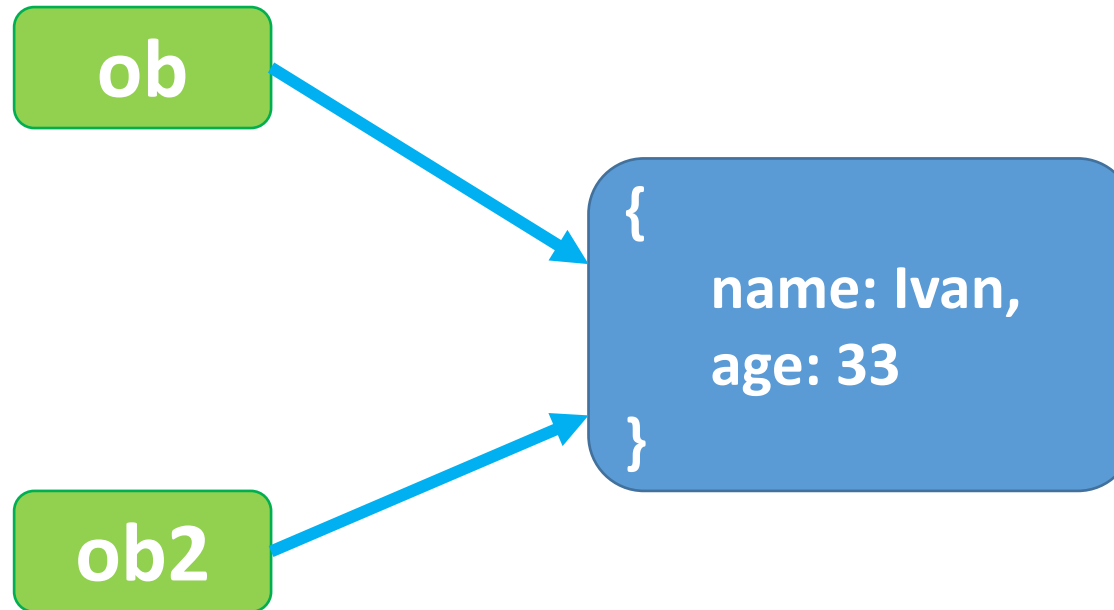
```
2
3 let person_1 = { name: 'Jhon', age: 35 };
4 let person_2 = person_1;
5
6 person_1.name = "Helen";
7 person_1.age = 27;
8
9 console.log(person_1); // {name: "Helen", age: 27} ?!?!?!
10 console.log(person_2); // {name: "Helen", age: 27}
11
12 //let person_3 = Object.assign({}, person_1);
13 let person_3 = {...person_1};
14
15 person_1.name = "Bill";
16 person_1.age = 51;
17
18 console.log(person_1); // {name: "Bill", age: 51}
19 console.log(person_3); // {name: "Helen", age: 27}
20
```

В переменных хранятся не сами объекты а ссылки на области памяти где они расположены, поэтому при «копировании» переменной присваивается ссылка на объект. И обе переменные позволяют работать с одним и тем же объектом. Если необходимо создать копию объекта, то помочь может оператор ... или же метод **Object.assign(...)**.

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

Подробнее: <https://learn.javascript.ru/destructuring-assignment>

# 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.

# Прототипы объектов

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 о прототипах мы поговорим в контексте «Объектно-ориентированного программирования» в JavaScript.*

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



# 7. 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 в **JavaScript**. Формат поддерживается практически во всех современных языках программирования.

```
[ { "name": "Jane",  "age": 23 },  
  { "name": "Max",   "age": 16 },  
  { "name": "Maria", "age": 34 },  
  { "name": "Alex",  "age": 20 },  
  { "name": "Cate",  "age": 45 } ]
```

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

Для работы с форматом **JSON** у нас есть два метода: **JSON.stringify(data)** – который преобразует структуру данных в строковое представление, и метод **JSON.parse(str)** который делает обратное действие.

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

## WebAPI построенные на обмене данными в формате JSON

Разработчикам доступно огромное количество сервисов которые предоставляющие доступ к данным в формате **JSON**. Такого рода сервисы носят название **WebAPI**.



```
[{"ccy": "USD", "base_ccy": "UAH", "buy": "28.05000", "sale": "28.25000"},  
{"ccy": "EUR", "base_ccy": "UAH", "buy": "31.95000", "sale": "32.45000"},  
{"ccy": "RUR", "base_ccy": "UAH", "buy": "0.41500", "sale": "0.43500"},  
{"ccy": "BTC", "base_ccy": "USD", "buy": "6143.7724", "sale": "6790.4852"}]
```

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

# 8. Циклы в JavaScript

```

2 //while - цикл с проверкой условия на входе;
3 while(a > b){
4     //.....
5 }
6
7
8 //do-while - цикл с проверкой условия на выходе;
9 do{
10     //.....
11 }while(a != b);
12
13 //for - цикл со счётчиком;
14 for(var i = 0; i < 10; i++){
15     //.....
16 }
17
18 //for-of - цикл перебора значений массивов и псевдомассивов;
19 let arr = [10, 35, 70, 90, 120];
20 for(let value of arr){
21     //.....
22 }
23
24 //for-in - цикл перебора ключей объекта.
25 let ob = { name: "Jhon", lastName: "Smith", age: 28, city: "Dnipro" };
26 for(let key in ob){
27     //.....
28 }
29

```

## Циклы в JavaScript

**JavaScript** содержит большой набор из 5 циклов (в классическом понимании цикла как средства повторения фрагмента кода) и десятков «цикло-подобных» конструкций. Циклы в **JavaScript** ориентированны на широкий спектр задач: циклы по условию (на входе и на выходе), цикл со счётчиком, циклы для перебора ключей и значений в структурах данных.

# Цикл for-await-of

```
23  
24     let promises = [new Promise(), new Promise(), new Promise(), ];  
25  
26     for await(p of promises){  
27         |     console.log( p.someResultData );  
28     }  
29
```

Цикл **for-await-of** позволяет перебрать итерируемую (перебираемую, массив или псевдомассив) состоящий из объектов типа **Promise**. Цикл будет ожидать когда разрешится каждый из **Promis'ов** и только тогда начинать выполнение каждого шага цикла.

Подробнее о Promise и асинхронности далее по ходу курса

# Операторы break и continue

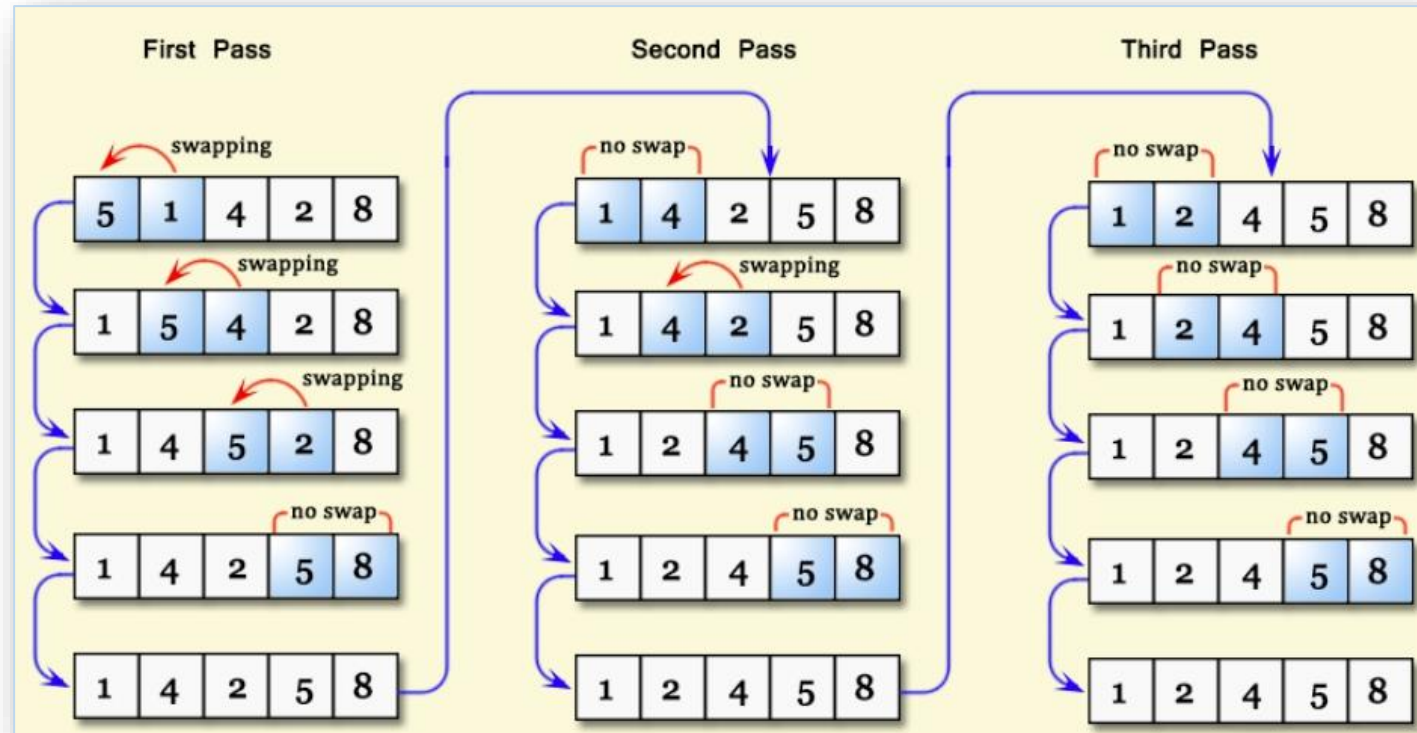
```
2
3  ✓ while(a > b){
4      //...
5  ✓   if(a == 100){
6       continue;
7   }
8
9  ✓   if(b == 100){
10      break;
11  }
12 }
13
```

Оператор **break** позволяет прервать цикл, оператор **continue** позволяет завершить текущий шаг (итерацию) цикла и перейти к следующей. Могут применяться во всех 5-ти видах циклов.

Подробнее: <https://learn.javascript.ru/while-for#metki-dlya-break-continue>

# Сортировка данных (массивов)

Когда необходимо внести изменения в существующий набор данных.

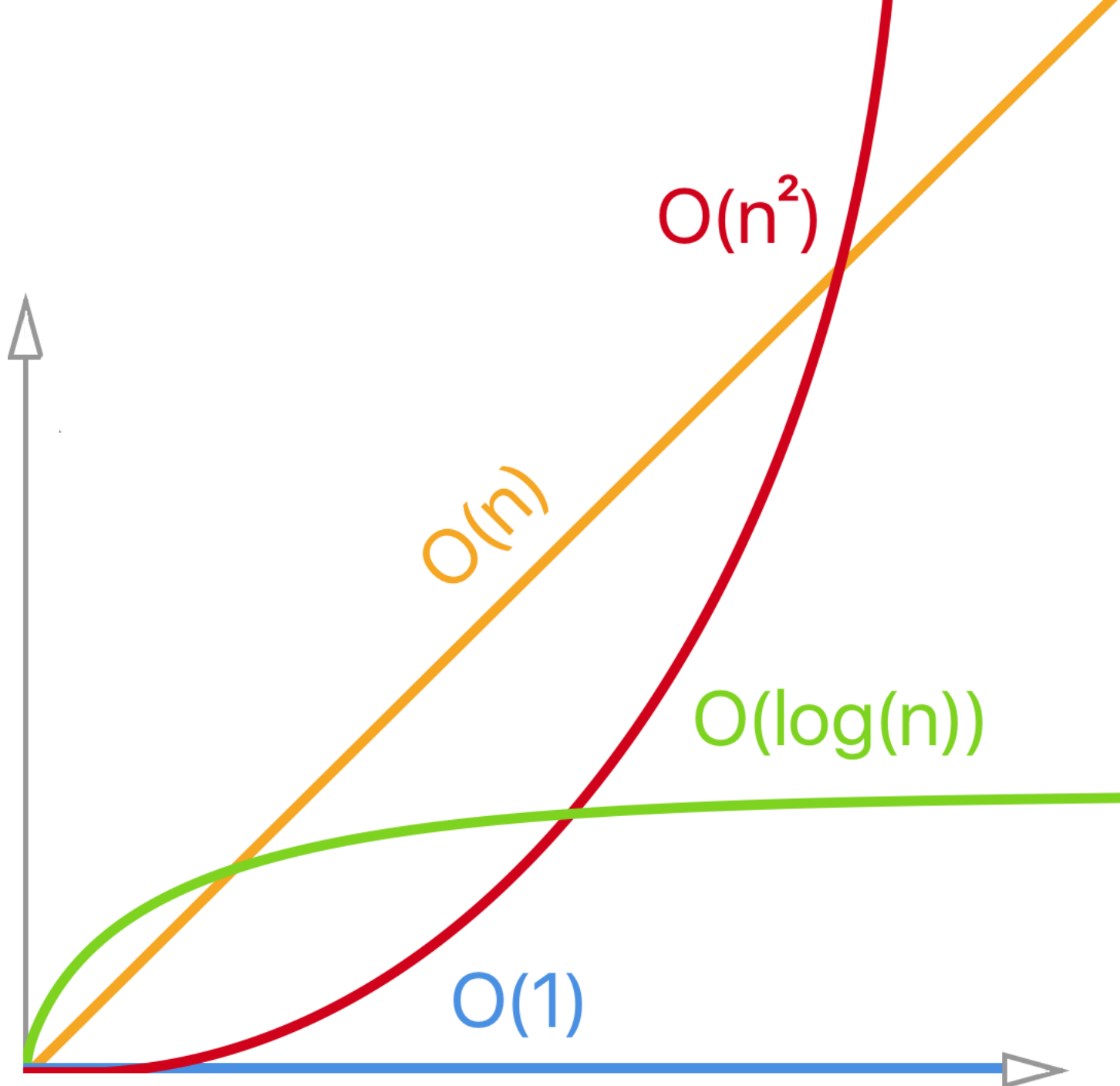


Классический алгоритм «пузырьковой» сортировки.

Подробнее о алгоритмах сортировки: <https://habrahabr.ru/post/204600/>



# 9. Сложность алгоритма



## Оценка сложности алгоритма (концепция **Big O**)

Зависимость времени  
выполнения (а по сути  
**количества операций**) от  
количества  
обрабатываемых данных

Подробнее:

<https://habr.com/ru/post/444594/>  
<https://www.youtube.com/watch?v=ZRdOb4yR0kk>

# Замеры времени выполнения кода

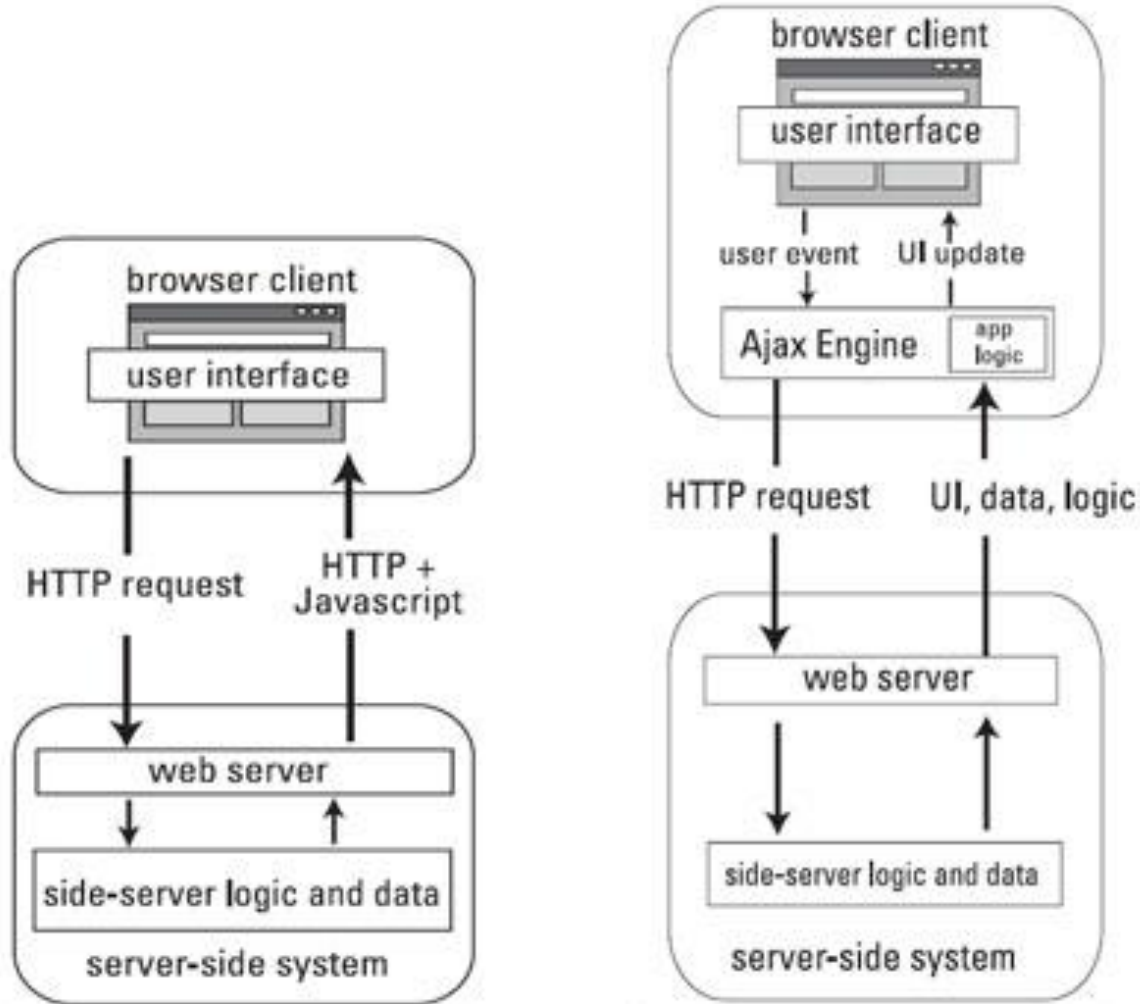
## `performance.now()`

```
2
3   let t1 = performance.now();
4
5   for(let i = 0; i < 1000000; i++){
6       //...do something HARD
7   }
8
9   let t2 = performance.now();
10
11  console.log('Time for HARD work (ms):', t2 - t1);
12
```

Метод **performance.now()** возвращает в миллисекундах временную метку. При сравнении двух и более временных меток можно определить время прошедшее между их получением.

# 10. Основы AJAX

# 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

```
1
2
3  const URL = 'https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json';
4
5  let response = await fetch(URL);
6
7  let data = await response.json();
8
9  console.log(data);
10
11
12
```

Идея заложенная в **AJAX** – не перезагружая страницу, запросить (или передать) у сервера новые данные и использовать их в документе. Функция **fetch()** позволяет выполнить запрос по адресу переданному в качестве параметра.

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>

# Задачи для практики



**Задача:** Тарифы банка за перевод средств с карты на карту: 1% за счёт личных средств и 4% в счёт кредитного лимита. Скрипт должен рассчитывать сумму комиссии за перевод (который хочет выполнить пользователь), и определять возможно ли выполнить перевод.

**Задача:** написать скрипт  
определяющая по номеру  
билета его «счастливость», т.е.  
если **сумма** первых 3  
десятичных цифр равна **сумме** 3  
последних десятичных цифр.



**Задача:** Вводится дата в формате '**YYYY-MM-DD**' (например '**2019-05-20**') необходимо преобразовать её в формат '**20 травня 2019 р.**'

**«Источник знаний»**

O'REILLY®

7-е  
издание

# JavaScript Полное руководство

Справочник по самому популярному  
языку программирования



Дэвид Флэнаган

Дэвид Флэнаган

**JavaScript:** Полное  
руководство, 7-е издание

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

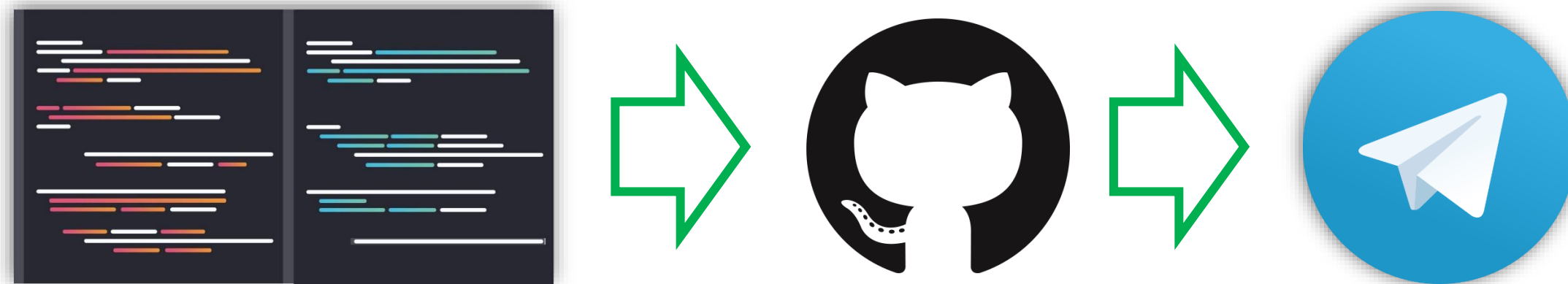
# К следующему занятию...

1. Функции в JavaScript, стрелочные (лямбда) функции;
2. Функции **setTimeout()**, **setInterval()**;
3. **Promise**, операторы **Async/Await**.
4. Узнать о методах массивов (Array) **.map()**, **.filter()**, **.reduce()**, **.sort()**, **.some()**, **.every()**;
5. Посмотрите этот ролик:  
<https://www.youtube.com/watch?v=8aGhZQkoFbQ> (да ведущий там странный) включите субтитры на русском языке.

**Домашнее задание**  
**/сделать**



Каждое домашнее задание оформляйте в виде отдельного репозитория на GitHub, в названии которого **укажите код задания** (например: **A1 Federal Tax**)



Если есть проблемы, вопросы, трудности, делаем тоже самое – код с проблемой заливаем на **GitHub** и ссылку на него, с описанием вопроса в **группу**.

# Домашнее задание #А.1



Есть в США такой вид налога как **Federal Income Tax**, ваша задача написать налоговый калькулятор, который будет рассчитывать сумму налогов в зависимости от годового дохода человека. За основу взять ставки налога для доходов полученных за **2021** г. (с оплатой в **2022** году), и для простоты - расчёт выполнять **только** для лиц не состоящих в браке: **single**.

<https://www.forbes.com/advisor/taxes/taxes-federal-income-tax-bracket/>  
<https://taxfoundation.org/publications/federal-tax-rates-and-tax-brackets/>

В репозитории занятия **есть тестер**: `./src/homework-tester` для сверки. Расхождением в **~1-2 доллара** можно пренебречь.

*О прогрессивном налогообложении в целом, с примерами:*

<http://allfi.biz/glossary/eng/P/progressive-taxation.php>