

Продвинутый JavaScript

Лекция 6



Освежим в памяти

Переменные и типы данных

- `let`
- `var`
- `const`
- Целые числа | Дробные числа
- Логические
- Строки
- Массивы
- Объекты
- Функции

Функции

- *Function declaration*

```
function myFunction(p1, p2) {  
    return p1 * p2;  
}
```
- *Function expression*

```
myFunction = (p1, p2) => { return p1 * p2; }  
  
let x = myFunction(4, 3);
```

Ещё раз про переменные

- **const** x = 3;
- **var** y = 5.0;
- **let** z;

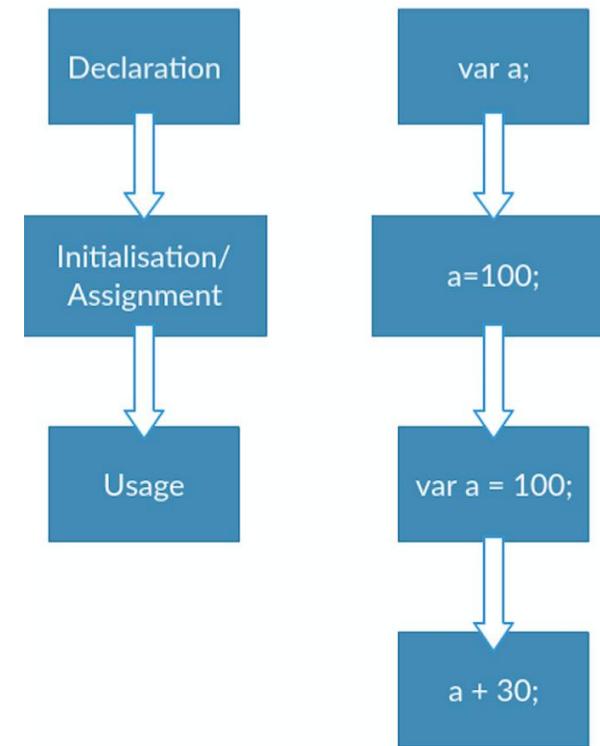
```
y = 2;  
z = x + y;  
console.log(z); // 5
```

```
z = "test" + z;  
console.log(z); // "test5"
```

- **var** – устаревший формат, обладает своими особенностями
- **let** – современный формат (рекомендованный)
- **const** – константы (но необычные)

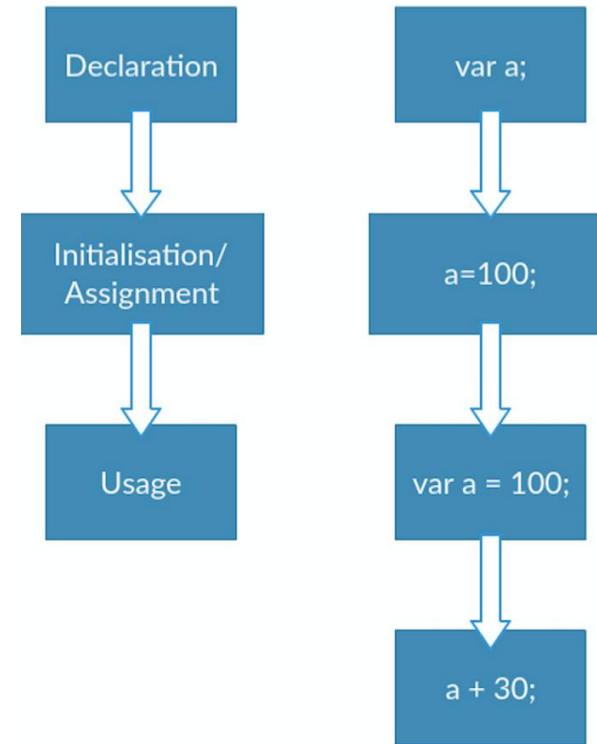
HOISTING

```
function test() {  
  a = 20;  
  var b = 100;  
}  
  
test();  
  
console.log(a); // 20  
console.log(b); // ReferenceError: b is not defined
```



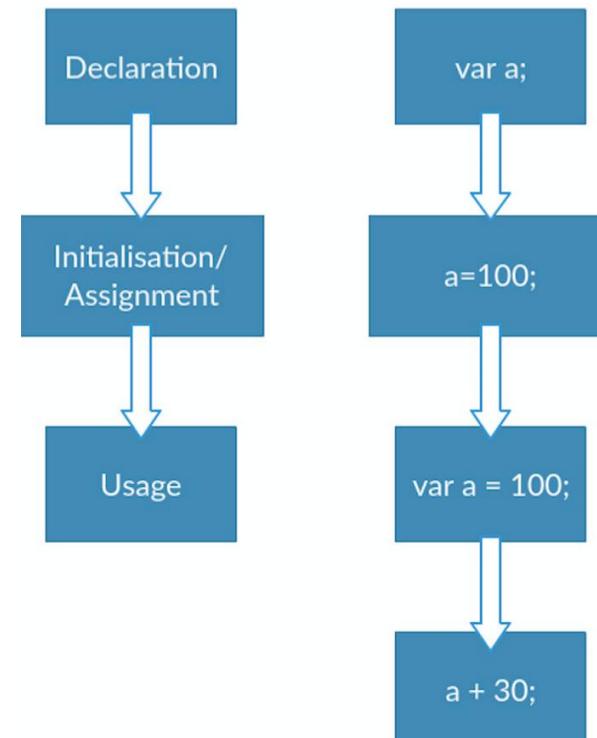
HOISTING и блоки

```
var hoist;  
console.log(hoist); // undefined  
hoist = 'The variable has been hoisted.';  
  
function hoist() {  
  console.log(message);  
  var message='Hoisting is all the rage!'  
}  
  
hoist();
```



HOISTING и функции

```
hoisted(); // "This function has been hoisted."  
  
function hoisted() {  
  console.log('This function has been hoisted.');//  
}  
  
expression(); // "TypeError: expression is not a  
function  
  
var expression = function() {  
  console.log('Will this work?');//  
};
```



use strict

```
'use strict';

let mistypeVariable;
mistypeVariable = 17;

var undefined = 5;
var Infinity = 5;
const x = 5;
x = 3;
```

Конструкция `use strict` включает проверку кода по современному стандарту, без совместимости со старыми браузерами.

Модули - всегда в **strict mode**.

Что такое модуль?

- **Объемный код** → хочется разбить на части;
- **Части взаимосвязаны**, иногда сложно;
- **Надо навести порядок в зависимостях**, учесть асинхронность загрузки, цикличность;

Раньше - require.js, CommonJS, теперь в ES6

Модули

```
// sayHi.js  
  
export function sayHi(user) {  
  console.log(`Hello, ${user}!`);  
}
```

```
// main.js  
  
import {sayHi} from './sayHi.js';  
  
console.log(typeof sayHi); // function  
sayHi('John'); // Hello, John!
```

Модули

Модули также можно встроить и в HTML

```
<script type="module">
  import {sayHi} from './say.js';
  document.body.innerHTML = sayHi('John');
</script>
```

Экспорт из модуля / Импорт в модуль

```
// myModule.js

// экспорт массива
export let months = [
  'Jan', 'Feb', 'Mar',
  'Apr', 'Aug', 'Sep',
  'Oct', 'Nov', 'Dec'
];

// экспорт константы
export const MODULES_BECAME_STANDARD_YEAR = 2015;

// экспорт класса
export class User {
  constructor(name) {
    this.name = name;
  }
}
```

```
// main.js

import {
  months,
  MODULES_BECAME_STANDARD_YEAR,
  User
} from './myModule.js';
```

Экспорт из модуля / Импорт в модуль

Именованный экспорт

```
function sayHi(user) {  
  alert(`Hello, ${user}!`);  
}  
  
function sayBye(user) {  
  alert(`Bye, ${user}!`);  
}  
  
export { sayHi as hi };  
export { sayBye };
```

export default

Только один на модуль! При этом именованных может быть сколько угодно.

```
export default class ClassName {  
  constructor(name) {  
    this.name = name;  
  }  
}
```

Экспорт из модуля / Импорт в модуль

Именованный импорт

```
import { sayHi } from './say.js';
sayHi('John');
```

```
import { sayHi as hi, sayBye as bye } from './say.js';
hi('John');
```

default

```
import * as say from './say.js';
say.sayHi('John');
```

```
import User from './user.js';
new User('John');
```

Преимущества модульности

- Своя область видимости переменных;
- Один скрипт – один модуль;
- Всегда в **strict mode**;
- Код в модуле выполняется только один раз при импорте;
- В модуле «**this**» не определён;
- Модули, даже если загрузились быстро, ожидают полной загрузки HTML документа, и только затем выполняются;

Promise

```
var promise = new Promise(function(resolve, reject) {  
    // В ней можно делать любые асинхронные  
    // операции,  
    // А когда они завершатся — нужно вызвать  
    // одно из:  
    // resolve(результат) при успешном выполнении  
    // reject(ошибка) при ошибке  
})  
  
promise.then(onFulfilled, onRejected)
```

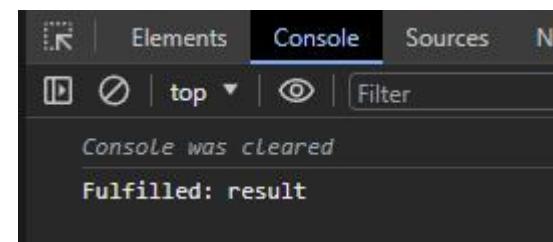
Promise

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("result");
  }, 1000);
});

promise.then(
  result => {
    console.log("Fulfilled: " + result); // result - аргумент
                                         resolve
  },
  error => {
    console.log("Rejected: " + error); // error - аргумент
                                         reject
  }
);
```

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("result");
  }, 1000);
});

promise.then(
  (result) => {
    console.log("Fulfilled: " + result); // result - аргумент resolve
  },
  (error) => {
    console.log("Rejected: " + error); // error - аргумент reject
  }
);
```



async / await

```
async function myFunction() {  
    return "Hello";  
}
```

```
function myFunction() {  
    return Promise.resolve("Hello");  
}
```

```
myFunction().then(  
    (x) => { console.log(x); }  
);
```

async/await - это лишь “синтаксический сахар”

fetch

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then((r) => r.json())
  .then((data) => console.log(data));
```

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then((r) => r.json())
  .then((data) => console.log(data));
```

```
▼ Array(100) []
  ▶ 0: {userId: 1, id: 1, title: 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit', body: 'quia et  
  ▶ 1: {userId: 1, id: 2, title: 'qui est esse', body: 'est rerum tempore vitae\\nsequi sint nihil reprehend.aperiam non debiti  
  ▶ 2: {userId: 1, id: 3, title: 'ea molestias quasi exercitationem repellat qui ipsa sit aut', body: 'et iusto sed quo iure\\n  
  ▶ 3: {userId: 1, id: 4, title: 'eum et est occaecati', body: 'ulam et saepe reiciendis voluptatem adipisci\\nsit - ipsam iur  
  ▶ 4: {userId: 1, id: 5, title: 'nesciunt quas odio', body: 'repudiandae veniam querat sunt sed\\nalias aut fugi..sse voluptat  
  ▶ 5: {userId: 1, id: 6, title: 'dolorem eum magni eos aperiam quia', body: 'ut aspernatur corporis harum nihil quis providen  
  ▶ 6: {userId: 1, id: 7, title: 'magnam facilis autem', body: 'dolore placeat quibusdam ea quo vitae\\nmagni quis e..t exceptur  
  ▶ 7: {userId: 1, id: 8, title: 'dolorem dolore est ipsam', body: 'dignissimos aperiam dolorem qui eum\\nfacilis quibus..\\nipsa  
  ▶ 8: {userId: 1, id: 9, title: 'nesciunt iure omnis dolorem tempora et accusantium', body: 'consectetur animi nesciunt iure  
  ▶ 9: {userId: 1, id: 10, title: 'optio molestias id quia eum', body: 'quo et expedita modi cum officia vel magni\\ndolorib.it  
  ▶ 10: {userId: 2, id: 11, title: 'et ea vero quia laudantium autem', body: 'delectus reiciendis molestiae occaecati non mini  
  ▶ 11: {userId: 2, id: 12, title: 'in quibusdam tempore odit est dolorem', body: 'itaque id aut magnam\\npraesentium quia et e  
  ▶ 12: {userId: 2, id: 13, title: 'dolorum ut in voluptas mollitia et saepe quo animi', body: 'aut dicta possimus sint mollit  
  ▶ 13: {userId: 2, id: 14, title: 'voluptatum eligendi optio', body: 'fuga et accusamus dolorum perferendis illo volupta..mole  
  ▶ 14: {userId: 2, id: 15, title: 'eveniet quod temporibus', body: 'reprehenderit quos placeat\\nvelit minima officia do..usand  
  ▶ 15: {userId: 2, id: 16, title: 'sint suscipit perspiciatis velit dolorum rerum ipsa laboriosam odio', body: 'suscipit nam  
  ▶ 16: {userId: 2, id: 17, title: 'fugit voluptas sed molestias voluptatem provident', body: 'eos voluptas et aut odit natus'}
```

Что ещё может fetch

```
fetch(url, {  
  method: 'POST', // *GET, POST, PUT, DELETE, etc.  
  mode: 'cors', // no-cors, *cors, same-origin  
  cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached  
  credentials: 'same-origin', // include, *same-origin, omit  
  headers: {  
    'Content-Type': 'application/json'  
  },  
  body: JSON.stringify(data)  
}).then( (response) => { ... });
```

Чтобы получить JSON

```
// *много кода до этого и обязательно fetch(url)*  
.then( response => response.json() )
```

```
let json = JSON.parse( text );  
let text = JSON.stringify( json );
```

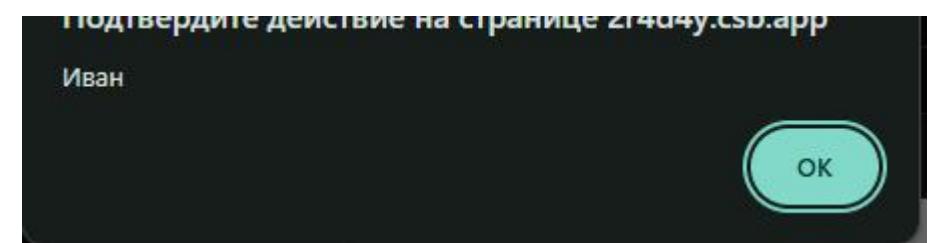
**Чтобы
получить XML**

```
fetch(url)
  .then( response => response.text() )
  .then( str => {
    let parser = new window.DOMParser();
    return parser.parseFromString(str, "text/xml")
  })
  .then(data => console.log(data));
```

Классы в JS

```
class User {  
  
    constructor(name) {  
        this.name = name;  
    }  
  
    sayHi() {  
        alert(this.name);  
    }  
  
}  
  
let user = new User("Иван");  
user.sayHi();
```

```
class User {  
    constructor(name) {  
        this.name = name;  
    }  
  
    sayHi() {  
        alert(this.name);  
    }  
}  
  
let user = new User("Иван");  
user.sayHi();
```



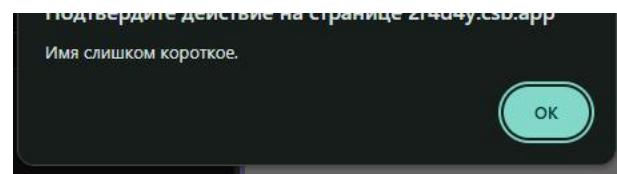
Что такое класс в JS?

- Класс – это разновидность функции
`alert(typeof User); // function`
- `class User {...}` создаёт функцию с именем **User**, код берётся из метода **constructor**
- В нее неявно дописывается `this = {};` и `return this;`
- Все методы и свойства сохраняются в `User.prototype`

Геттеры и сеттеры

```
class User {  
  constructor(name) {  
    this.name = name;  
  }  
  
  get name() {  
    return this._name;  
  }  
  
  set name(value) {  
    if (value.length < 4) { alert("Имя слишком короткое."); return; }  
    this._name = value;  
  }  
  
}  
  
let user = new User("Иван");  
alert(user.name); // Иван  
user = new User(""); // Имя слишком короткое.
```

```
class User {  
  constructor(name) {  
    this.name = name;  
  }  
  
  get name() {  
    return this._name;  
  }  
  
  set name(value) {  
    if (value.length < 4) {  
      alert("Имя слишком короткое.");  
      return;  
    }  
    this._name = value;  
  }  
  
  let user = new User("Иван");  
  alert(user.name);  
  user = new User("");
```



Дополнительная литература



1. [What is Frontend Development?](#)



1. [JavaScript Roadmap](#)