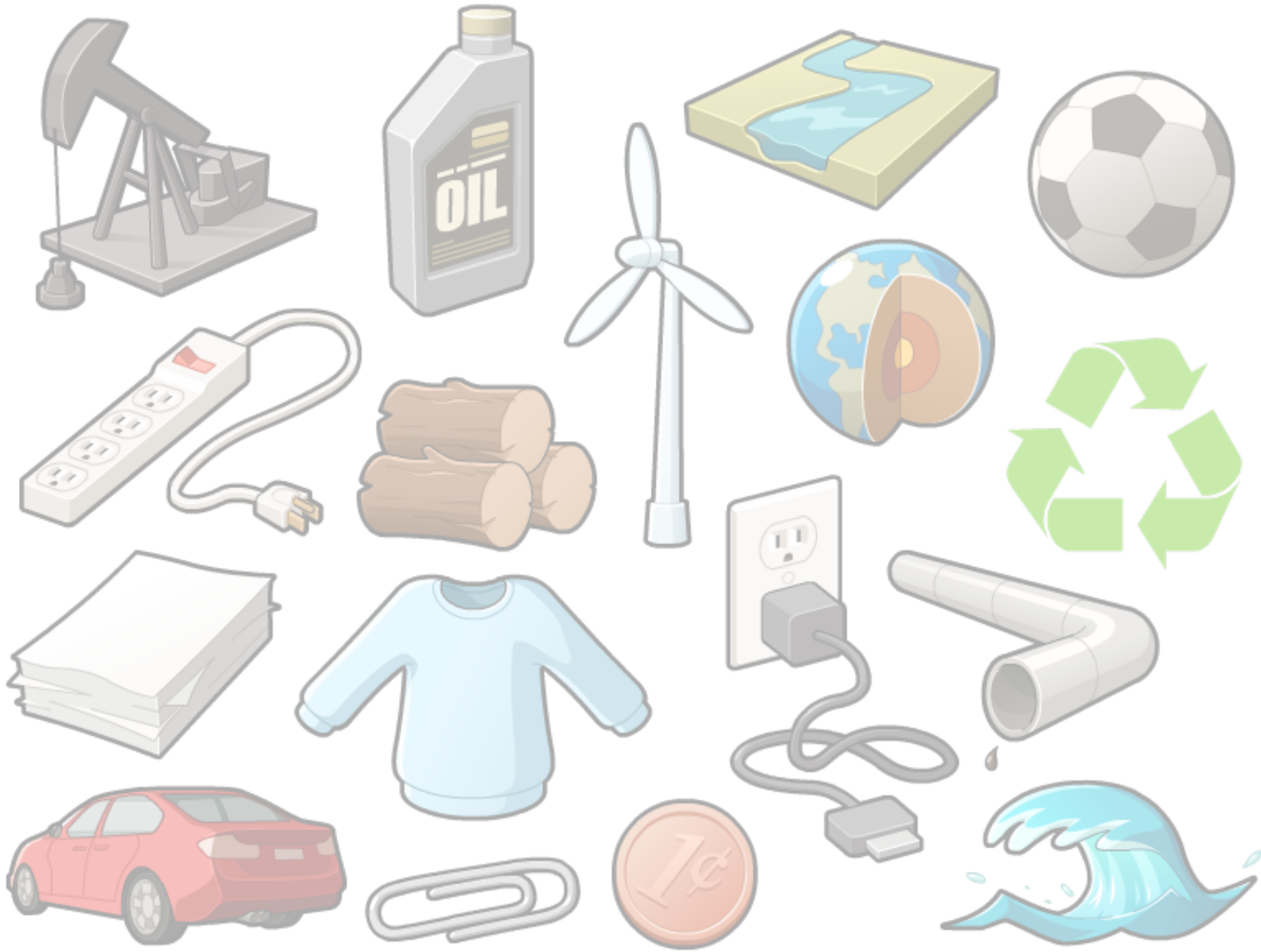

Яндекс

Яндекс

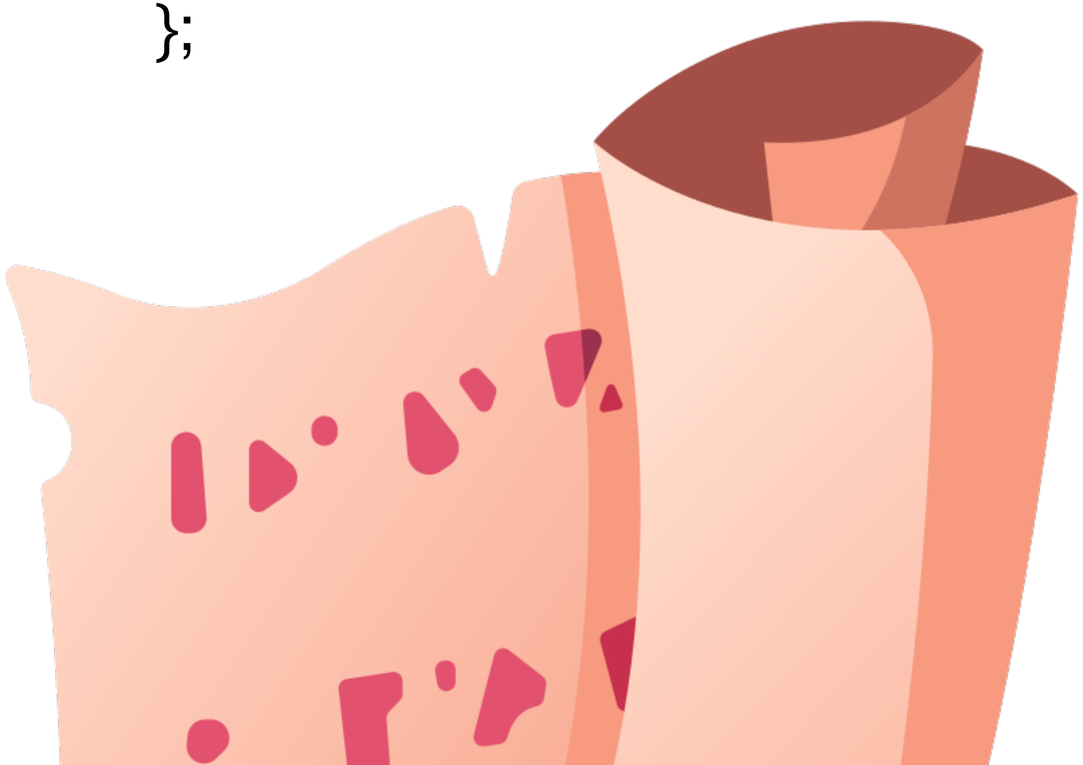
Объекты



Объявление объекта

Объектный литерал

```
var user = {
    name: 'admin'
};
```



Функция-конструктор

```
var emptyObject = new Object();
```

Очень скоро мы узнаем, как работает

В объектном литерале вокруг имени поля не ставятся кавычки, если имя поля соответствует правилам именования переменных

Именем поля может быть любая строка. Совершенно любая

Объявление объекта

Объектный литерал

```
var user = {
    '😉': 'admin'
};
```

Доступ к полю объекта

На чтение:

```
var user = {
    name: 'admin'
};
```

console.log(user['name']); // можно обращаться к полю аналогично тому, как мы это делаем с массивами

console.log(user.name); // обращение к полю через точку используется чаще

Доступ к полю объекта

На чтение:

```
var user = {
    '😉': 'admin', статус: 404
};
console.log(user['😉']); // в данном случае, нет другого способа обратиться к полю объекта

console.log(user.😉); // ошибка, недопустимый символ

console.log(user.статус); // угадаете, что будет?
```

Доступ к полю объекта

На чтение:

```
var user = { name: 'admin' };
```

```
var field = 'name';
```

```
console.log(user[field]); // с помощью этого синтаксиса можно обратиться к полю объекта, имя которого находится в переменной
```

```
console.log(user.field); // неверно, будет искать поле field, которого нет
```

Доступ к полю объекта

На запись:

```
var user = { name: 'admin' };

user.name = 'dusty';

console.log(user.name); // dusty
```

Доступ к несуществующему полю объекта

На запись:

```
var user = { name: 'admin' };
```

```
user.password = 'password';
```

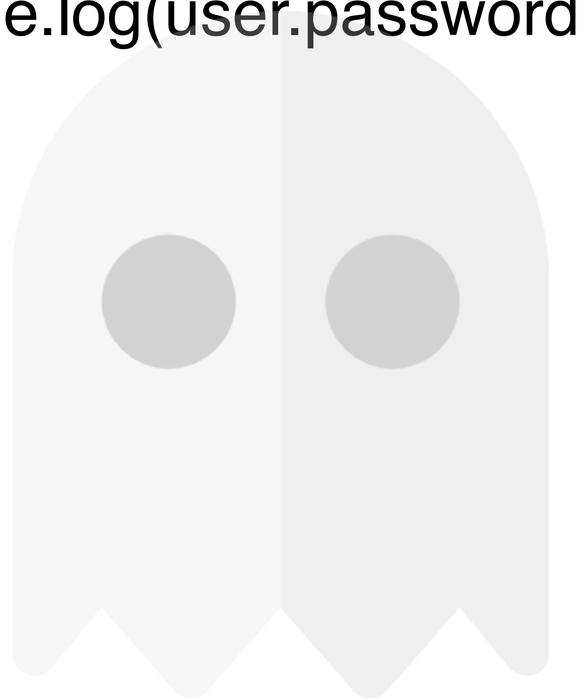
```
console.log(user.password); // поле было добавлено в объект, и значение поля - строка 'password' - сохранено
```

Доступ к несуществующему полю объекта

На чтение:

```
var user = { name: 'admin' };

console.log(user.password); // undefined
```



Полем объекта может быть любой тип данных

Древовидный объект

С учетом того, что полем объекта может быть, в свою очередь, объект, а также массив (помимо примитивных типов), мы можем использовать древовидные структуры

```
var user = {
    name: 'admin',
    access: [{ root: true }]
};


var isRoot = user.access[0].root;
```

Методы

Полем объекта может быть любой тип данных, и, в том числе, функция

В этом случае функцию называют «методом»

Метод

Учитывая все вышеизученное про доступ к полям, вызвать функцию-метод очень просто

```
var user = {
    name: 'admin',
    say: function(message) {
        console.log(message);
    }
};

user.say('hello');
```

Метод тоже может возвращать значения с помощью return

Метод

```
var user = {
    name: 'admin',
    say: function(message) {
        return message[0].toUpperCase() + message.slice(1);
    }
};

console.log(user.say(`hello')); // Hello
```

this

Использование this

Главное отличие «метода» от функции - наличие this, позволяющего использовать значения полей объекта. Это позволяет «оживить» объект, наполнить его поведением

```
var user = {
    name: 'admin',
    say: function(message) {
        return message[0].toUpperCase() + message.slice(1) +
            ', ' + this.name;
    }
};

console.log(user.say('hello')); // Hello, admin
```

Лотерейная машина

Функция-конструктор

Вставьте
изображение

Функция-конструктор

Если любую функцию вызвать с испольхованием ключевого слова new, то она вернет объект

```
function ololo() {
    return 'trololo';
};


var emptyObject = new ololo(); // {}
```

Фокус именно в использовании , если вызвать функцию обычным образом, она все еще вернет trololo

Правильное использование функции-конструктора

Правильно использование заключается в «конструировании» объекта, добавляя поля в возвращаемый объект с помощью this

```
function User() {
    this.name = 'admin';
    this.say = function() {
        return message[0].toUpperCase() + message.slice(1) +
          ', ' + this.name;
    }
};

var user = new User();
```

Правильное использование функции-конструктора

Функция-конструктор, как и обычная, может иметь аргументы, что позволяет создавать разные объекты за счет одной функции.

```
function User(name) {
    this.name = name;
    this.say = function() {
        return message[0].toUpperCase() + message.slice(1) +
          ', ' + this.name;
    }
};
var admin = new User('admin');
var guest = new User('guest');
```

ES6 синтаксис

ES6

В относительно новом ES6 появился новый, более приятный синтаксис для классов.

```
class User {

    constructor(name) {

        this.name = name;

    }

    say(message) {

        return message[0].toUpperCase() + message.slice(1) +

            ', ' + this.name;

    }

}

var admin = new User('admin');
```

Интересно, что если сделать console.log(User); для ES6 «класса» то в консоли вы увидите все ту же функцию.

Это говорит о том, что никаких «классов» на самом деле нет, а мы имеем дело с синтаксическим сахаром в данном случае.

Другими словами, на ES6 нельзя написать ничего такого, чего нельзя было бы написать на ES5.

Знакомьтесь - CanIUse

Особенности объектов

Вставьте изображение

Объекты передаются по ссылке

«Скопируем» объект и у копии изменим кое-что

```
var admin = { name: 'admin', password: 'password' };

var  guest = admin;

guest.name = 'guest'; // меняем поле с помощью переменной guest

console.log(admin.name); // интрига…
```

Перебор полей в объекте

С помощью метода Object.keys() вы можете получить массив полей объекта и обойти их примерно так:

```
var checkPassword = prompt('пароль?');
var passwords = { admin: 'pass', dusty: 'dusty1' };
var users = Object.keys(passwords);
var user;
for (var i = 0; i < users.length; i++) {
    user = users[i];
    if(checkPassword === passwords[user]) {
        console.log(‘Привет, ' + user);
        break;
    }
}
```

Удаление полей объекта

Есть два способа удалить поле у объекта

```
var user = { name: 'admin', password: 'password' };
```

```
user.password = undefined; // поле останется в объекте и будет попадать в массив, генерируемый Object.keys
```

```
delete user.password; // поле будет полностью удалено
```

По-честному удалять сильно медленнее

Объекты-обертки

Вставьте
изображение

Мы уже неоднократно сталкивались с тем, что к различным необъектам применяется точка (например .map, .upperCase)

Точка - однозначно и определенно используется только для того, чтобы обратиться к полю объекта.

Что это? Противоречие? Магия?

Срываем покровы:

Объекты обертки

Когда интерпретатор видит, что точка применяется к необъекту, тот «заворачивается» во временный объект путем вызова функции конструктора

```
var hello = 'hello';
```

```
hello.toUpperCase(); // код, который видим мы
```

```
new String(hello).toUpperCase(); // код, который видим мы
```

Теперь точка становится легальной. С объектом «приезжают» всяческие встроенные в язык методы и свойства.

Какую именно функцию вызвать интерпретатор решает, глядя на тип данных, стоящих слева от точки

Полученный объект - временный, он не будет записан в переменную и вообще исчезнет, в переменной сохранится прежнее значение

Загадка

```
var hello = 'hello';

hello.test = 'test';

console.log(hello.test); // ???
```

«Стыковка» методов

Некоторые методы объектов-оберток

toFixed

Получение «классического» округленного представления числа

```
var number = 36.6;

number.toFixed(); // '37' - округление, как на уроках математики

number.toFixed(2); // '36.60' - точность до сотых
```

Обратите внимание на то, что результатом является строка

toPrecision

Получение округленного представления числа, с точностью до переданного числа значащих цифр

```
var number = 36.6;

number.toPrecision(2); // '37' - оставили две значащих цифры

number.toPrecision(1); // '4e+1' - если округлять до десятков, метод ведет себя странно
```

slice

Самый часто используемый метод для вырезания подстроки из строки

```
var hello = 'Alice please turn on the music';

hello.slice(6, 12); // 'please' - начиная от символа с номером 6 до символа с номером 11

hello.slice(-5); // 'music' - начиная от пятого символа с конца и до конца строки, т.к. второй параметр не указан
```

indexOf / includes

Методы для поиска подстроки в строке

```
var hello = 'Alice please turn on the music';

hello.indexOf('turn'); // 13

hello.indexOf('ololo'); // -1

hello.includes('turn'); // true
```

.includes - доступен начиная с ES6, это можно увидеть в MDN

replace

Методы для поиска подстроки в строке

- var hello = 'Alice please turn on the music';
- hello.replace('on', 'off'); // заменяет первое вхождение первого аргумента на второй
- hello.replace(/ /g, '_'); // если надо заменить все вхождения, то используем несложную регулярку

toUpperCase / toLowerCase

Методы для преобразования строк к верхнему/нижнему регистру

```
var hello = 'Alice please turn on the music';

hello.toUpperCase(); // 'ALICE PLEASE TURN ON THE MUSIC'

hello.toLowerCase(); // 'alice please turn on the music'

'Привет'.toUpperCase(); // 'ПРИВЕТ' - кириллицу тоже понимает
```

split / join

Методы для разбиения строки на массив и обратно

- var hello = 'Alice please turn on the music';
- hello.split(' '); // ['Alice', 'please', 'turn', 'on the music']
- hello.split(''); // ['A', 'l', 'i', 'c', 'e', ' ', 'p' … 'c'] - разбиение посимвольно
- ['Alice', 'please', 'turn', 'on the music’].join(' '); // 'Alice please turn on the music'

Яндекс

Спасибо

Шлейко Александр

Разработчик интерфейсов


dusty@yandex-team.ru


@dustyo_O