Объекты обычно создаются, чтобы представлять сущности реального мира, будь то пользователи, заказы и так далее:

// Объект пользователя

let user = {

name: "John",

age: 30

};

И так же, как и в реальном мире, пользователь может *совершать действия*: выбирать что-то из корзины покупок, авторизовываться, выходить из системы, оплачивать и т.п.

Такие действия в JavaScript представлены функциями в свойствах.

Функцию, которая является свойством объекта, называют методом этого объекта.

let user = {

name: "John",

age: 30

};

user.sayHi = function() {

alert("Привет!");

};

user.sayHi(); // Привет!

Можно использовать заранее объявленную функцию в качестве метода.

**[Сокращённая запись метода](https://learn.javascript.ru/object-methods" \l "sokraschyonnaya-zapis-metoda)**

Существует более короткий синтаксис для методов в литерале объекта:

// эти объекты делают одно и то же

user = {

sayHi: function() {

alert("Привет");

}

};

// сокращённая запись выглядит лучше, не так ли?

user = {

sayHi() { // то же самое, что и "sayHi: function(){...}"

alert("Привет");

}

};

Можно пропустить ключевое слово "function" и двоеточие.

Нужно отметить, что эти две записи не полностью эквивалентны. Есть тонкие различия, связанные с наследованием объектов.

## [Ключевое слово «this» в методах](https://learn.javascript.ru/object-methods" \l "klyuchevoe-slovo-this-v-metodah)

**Для доступа к информации внутри объекта метод может использовать ключевое слово this.**

Значение this – это объект «перед точкой», который используется для вызова метода.

Например:

let user = {

name: "John",

age: 30,

sayHi() {

// "this" - это "текущий объект".

alert(this.name);

}

};

user.sayHi(); // John

Здесь во время выполнения кода user.sayHi() значением this будет являться user (ссылка на объект user).

Технически также возможно получить доступ к объекту без ключевого слова this, обратившись к нему через внешнюю переменную (в которой хранится ссылка на этот объект):

let user = {

name: "John",

age: 30,

sayHi() {

alert(user.name); // "user" вместо "this"

}

};

…Но такой код ненадёжен. Если мы решим скопировать ссылку на объект user в другую переменную, например, admin = user, и перезапишем переменную user чем-то другим, тогда будет осуществлён доступ к неправильному объекту при вызове метода из admin.

let user = {

name: "John",

age: 30,

sayHi() {

alert( user.name ); // приведёт к ошибке

}

};

let admin = user;

user = null; // перезапишем переменную для наглядности, теперь она не хранит ссылку на объект.

admin.sayHi(); // TypeError: Cannot read property 'name' of null\Если бы мы использовали this.name вместо user.name внутри alert, тогда этот код бы сработал.

## [«this» не является фиксированным](https://learn.javascript.ru/object-methods" \l "this-ne-yavlyaetsya-fiksirovannym), Значение this определяется во время исполнения кода.

This можно использовать в любой функции, даже если это не метод объекта.

function sayHi() {

alert( this.name );

}

Например, одну и ту же функцию можно назначить двум разным объектам и получить разное значение «this» в вызовах:

let user = { name: "John" };

let admin = { name: "Admin" };

function sayHi() {

alert( this.name );

}

// используем одну и ту же функцию в двух объектах

user.f = sayHi;

admin.f = sayHi;

// эти вызовы имеют разное значение this

// "this" внутри функции - это объект "перед точкой"

user.f(); // John (this == user)

admin.f(); // Admin (this == admin)

admin['f'](); // Admin (нет разницы между использованием точки или квадратных скобок для доступа к объекту)

**Вызов функции без объекта: this == undefined**

function sayHi() {

alert(this);

}

sayHi(); // undefined

В строгом режиме ("use strict") в таком коде значением this будет являться undefined. Если мы попытаемся получить доступ к this.name – это вызовет ошибку.

В нестрогом режиме значением this в таком случае будет глобальный объект window в браузере. Это – исторически сложившееся поведение this, которое исправляется использованием строгого режима ("use strict").

Обычно подобный вызов является ошибкой программирования. Если внутри функции используется this, тогда она ожидает, что будет вызвана в контексте какого-либо объекта.

## [У стрелочных функций нет «this»](https://learn.javascript.ru/object-methods" \l "u-strelochnyh-funktsiy-net-this)

Стрелочные функции особенные: у них нет своего «собственного» this. Если мы ссылаемся на this внутри такой функции, то оно берётся из внешней «нормальной» функции.

Например, здесь arrow() использует значение this из внешнего метода user.sayHi():

let user = {

firstName: "Ilya",

sayHi() {

let arrow = () => alert(this.firstName);

arrow();

}

};

user.sayHi(); // Ilya

Это особенность стрелочных функций. Она полезна, когда мы на самом деле не хотим иметь отдельное this, а скорее хотим взять его из внешнего контекста.