Можно использовать функции-конструкторы для создания множества похожих объектов.

**[Функция-конструктор](https://learn.javascript.ru/constructor-new" \l "funktsiya-konstruktor)**

Функции-конструкторы технически являются обычными функциями. технически любая функция (кроме стрелочных функций, поскольку у них нет this) может использоваться в качестве конструктора. Но есть два соглашения:

1. Имя функции-конструктора должно начинаться с большой буквы, , чтобы было ясно, что функция должна вызываться с помощью «new»
2. Функция-конструктор должна выполняться только с помощью оператора "new".

function User(name) {

this.name = name;

this.isAdmin = false;

}

let user = new User("Jack");

alert(user.name); // Jack

alert(user.isAdmin); // false

Когда функция вызывается как new User(...), происходит следующее:

1. Создаётся новый пустой объект, и он присваивается this.
2. Выполняется тело функции. Обычно оно модифицирует this, добавляя туда новые свойства.
3. Возвращается значение this.

Другими словами, new User(...) делает что-то вроде:

function User(name) {

// this = {}; (неявно)

// добавляет свойства к this

this.name = name;

this.isAdmin = false;

// return this; (неявно)

}

Реализовать код для многократного создания однотипных объектов - является основной целью конструкторов. Данная конструкция гораздо удобнее и читабельнее, чем многократное создание литерала объекта.

**new function() { … }**

Если в коде присутствует большое количество строк, создающих один сложный объект, то можно обернуть их в функцию-конструктор, которая будет немедленно вызвана, вот так:

// создаём функцию и сразу же вызываем её с помощью new

let user = new function() {

this.name = "John";

this.isAdmin = false;

// ...другой код для создания пользователя

// возможна любая сложная логика и инструкции

// локальные переменные и так далее

};

Такой конструктор не может быть вызван снова, так как он нигде не сохраняется, просто создаётся и тут же вызывается. Таким образом, этот трюк направлен на инкапсуляцию кода, который создаёт отдельный объект, без возможности повторного использования в будущем.

## [Возврат значения из конструктора, return](https://learn.javascript.ru/constructor-new" \l "vozvrat-znacheniya-iz-konstruktora-return)

Обычно конструкторы не имеют оператора return. Их задача – записать все необходимое в this, и это автоматически становится результатом.

Но если return всё же есть, то применяется простое правило:

* При вызове return с объектом, вместо this вернётся объект.
* При вызове return с примитивным значением, оно проигнорируется.

Другими словами, return с объектом возвращает этот объект, во всех остальных случаях возвращается this.

К примеру, здесь return замещает this, возвращая объект:

function BigUser() {

this.name = "John";

return { name: "Godzilla" }; // <-- возвращает этот объект

}

alert( new BigUser().name ); // Godzilla, получили этот объект

А вот пример с пустым return (или мы могли бы поставить примитив после return, неважно):

function SmallUser() {

this.name = "John";

return; // <-- возвращает this

}

alert( new SmallUser().name ); // John

Обычно у конструкторов отсутствует return. Здесь мы упомянули особое поведение с возвращаемыми объектами в основном для полноты картины.

**Пропуск скобок**

Кстати, мы можем не ставить круглые скобки после new:

let user = new User; // <-- без скобок

// то же, что и

let user = new User();

Пропуск скобок считается плохой практикой, но просто чтобы вы знали, такой синтаксис разрешён спецификацией.

## [Создание методов в конструкторе](https://learn.javascript.ru/constructor-new" \l "sozdanie-metodov-v-konstruktore)

function User(name) {

this.name = name;

this.sayHi = function() {

alert( "Меня зовут: " + this.name );

};

}

let john = new User("John");

john.sayHi(); // Меня зовут: John

/\*

john = {

name: "John",

sayHi: function() { ... }

}

\*/

Для создания сложных объектов есть и более продвинутый синтаксис – [классы](https://learn.javascript.ru/classes).