**Создание объектов через "new"**

Обычный синтаксис {...} позволяет создать один объект. Но зачастую нужно создать много однотипных объектов.

Для этого используют «функции-конструкторы», запуская их при помощи специального оператора new.

**[Конструктор](http://learn.javascript.ru/constructor-new" \l "konstruktor)**

Конструктором становится любая функция, вызванная через new.

Например:

function Animal(name) {

this.name = name;

this.canWalk = true;

}

var animal = new Animal("ёжик");

Заметим, что, технически, любая функция может быть использована как конструктор. То есть, любую функцию можно вызвать при помощи new. Как-то особым образом указывать, что она – конструктор – не надо.

Но, чтобы выделить функции, задуманные как конструкторы, их называют с большой буквы: Animal, а не animal.

Детальнее – функция, запущенная через new, делает следующее:

1. Создаётся новый пустой объект.
2. Ключевое слово this получает ссылку на этот объект.
3. Функция выполняется. Как правило, она модифицирует this (т.е. этот новый объект), добавляет методы, свойства.
4. Возвращается this.

В результате вызова new Animal("ёжик"); получаем такой объект:

animal = {

name: "ёжик",

canWalk: true

}

Иными словами, при вызове new Animal происходит что-то в таком духе (первая и последняя строка – это то, что делает интерпретатор):

function Animal(name) {

// this = {};

// в this пишем свойства, методы

this.name = name;

this.canWalk = true;

// return this;

}

Теперь многократными вызовами new Animal с разными параметрами мы можем создать столько объектов, сколько нужно. Поэтому такую функцию и называют *конструктором* – она предназначена для «конструирования» объектов.

**new function() { … }**

Иногда функцию-конструктор объявляют и тут же используют, вот так:

var animal = new function() {

this.name = "Васька";

this.canWalk = true;

};

Так делают, когда хотят создать единственный объект данного типа. Пример выше с тем же успехом можно было бы переписать как:

var animal = {

name: "Васька",

canWalk: true

}

…Но обычный синтаксис {...} не подходит, когда при создании свойств объекта нужны более сложные вычисления. Их можно проделать в функции-конструкторе и записать результат в this.

**[Правила обработки return](http://learn.javascript.ru/constructor-new" \l "pravila-obrabotki-return)**

Как правило, конструкторы ничего не возвращают. Их задача – записать всё, что нужно, в this, который автоматически станет результатом.

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

* При вызове return с объектом, будет возвращён он, а не this.
* При вызове return с примитивным значением, оно будет отброшено.

Иными словами, вызов return с объектом вернёт объект, а с чем угодно, кроме объекта – возвратит, как обычно, this.

Например, возврат объекта:

function BigAnimal() {

this.name = "Мышь";

return { name: "Годзилла" }; // <-- возвратим объект

}

alert( new BigAnimal().name ); // Годзилла, получили объект вместо this

А вот пример с возвратом строки:

function BigAnimal() {

this.name = "Мышь";

return "Годзилла"; // <-- возвратим примитив

}

alert( new BigAnimal().name ); // Мышь, получили this (а Годзилла пропал)

Эта особенность работы new прописана в стандарте, но используется она весьма редко.

**Можно без скобок**

Кстати, при вызове new без аргументов скобки можно не ставить:

var animal = new BigAnimal; // <-- без скобок

// то же самое что

var animal = new BigAnimal();

Не сказать, что выбрасывание скобок – «хороший стиль», но такой синтаксис допустим стандартом.

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

Использование функций для создания объекта дает большую гибкость. Можно передавать конструктору параметры, определяющие как его создавать, и он будет «клепать» объекты заданным образом.

Добавим в создаваемый объект ещё и метод.

Например, new User(name) создает объект с заданным значением свойства name и методом sayHi:

function User(name) {

this.name = name;

this.sayHi = function() {

alert( "Моё имя: " + this.name );

};

}

var ivan = new User("Иван");

ivan.sayHi(); // Моё имя: Иван

/\*

ivan = {

name: "Иван",

sayHi: функция

}

\*/

**[Локальные переменные](http://learn.javascript.ru/constructor-new" \l "lokalnye-peremennye)**

В функции-конструкторе бывает удобно объявить вспомогательные локальные переменные и вложенные функции, которые будут видны только внутри:

function User(firstName, lastName) {

// вспомогательная переменная

var phrase = "Привет";

// вспомогательная вложенная функция

function getFullName() {

return firstName + " " + lastName;

}

this.sayHi = function() {

alert( phrase + ", " + getFullName() ); // использование

};

}

var vasya = new User("Вася", "Петров");

vasya.sayHi(); // Привет, Вася Петров

Мы уже говорили об этом подходе ранее, в главе [Локальные переменные для объекта](http://learn.javascript.ru/closures-usage).

Те функции и данные, которые должны быть доступны для внешнего кода, мы пишем в this – и к ним можно будет обращаться, как например vasya.sayHi(), а вспомогательные, которые нужны только внутри самого объекта, сохраняем в локальной области видимости.

**[Итого](http://learn.javascript.ru/constructor-new" \l "itogo)**

Объекты могут быть созданы при помощи функций-конструкторов:

* Любая функция может быть вызвана с new, при этом она получает новый пустой объект в качестве this, в который она добавляет свойства. Если функция не решит возвратить свой объект, то её результатом будет this.
* Функции, которые предназначены для создания объектов, называются *конструкторами*. Их названия пишут с большой буквы, чтобы отличать от обычных.