### Конструкторы, создание объектов через "new"

Обычный синтаксис {...} позволяет создать только один объект. Но зачастую нам нужно создать множество однотипных объектов, таких как пользователи, элементы меню и т.д.

Это можно сделать при помощи функции-конструктора и оператора "new".

### Функция-конструктор

Функции-конструкторы являются обычными функциями. Но есть два соглашения:

1. Имя функции-конструктора должно начинаться с большой буквы.
2. Функция-конструктор должна вызываться при помощи оператора "new".

Например:

function User(name) {

this.name = name;

this.isAdmin = false;

}

let user = new User("Вася");

alert(user.name); // Вася

alert(user.isAdmin); // false

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

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

Другими словами, вызов new User(...) делает примерно вот что:

function User(name) {

// this = {}; (неявно)

// добавляет свойства к this

this.name = name;

this.isAdmin = false;

// return this; (неявно)

}

То есть, результат вызова new User("Вася") – это тот же объект, что и:

let user = {

name: "Вася",

isAdmin: false

};

Теперь, когда нам необходимо будет создать других пользователей, мы можем использовать new User("Маша"), new User("Даша") и т.д. Данная конструкция гораздо удобнее и читабельнее, чем каждый раз создавать литерал объекта. Это и является основной целью конструкторов – удобное повторное создание однотипных объектов.

Ещё раз заметим: технически любая функция может быть использована как конструктор. То есть, каждая функция может быть вызвана при помощи оператора new, и выполнится алгоритм, указанный выше в примере. Заглавная буква в названии функции является всеобщим соглашением по именованию, она как бы подсказывает разработчику, что данная функция является функцией-конструктором, и её нужно вызывать через new.

new function() { … }

Если в нашем коде большое количество строк, создающих один сложный объект, мы можем обернуть их в функцию-конструктор следующим образом:

let user = new function() {

this.name = "Вася";

this.isAdmin = false;

// ...другой код для создания пользователя

// возможна любая сложная логика и выражения

// локальные переменные и т. д.

};

Такой конструктор не может быть вызван дважды, так как он нигде не сохраняется, просто создаётся и тут же вызывается. Таким образом, такой метод создания позволяет инкапсулировать код, который создаёт отдельный объект, но без возможности его повторного использования.

### Возврат значения из конструктора return

Обычно конструкторы ничего не возвращают явно. Их задача – записать все необходимое в this, который в итоге станет результатом.

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

* При вызове return с объектом, будет возвращён объект, а не this.
* При вызове return с примитивным значением, примитивное значение будет отброшено.

Другими словами, return с объектом возвращает объект, в любом другом случае конструктор вернёт this.

В примере ниже return возвращает объект вместо this:

function BigUser() {

this.name = "Вася";

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 = "Вася";

return; // <-- возвращает this

}

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

Обычно у конструкторов отсутствует return. В данном блоке мы упомянули особое поведение с возвращаемыми объектами, чтобы не оставлять пробелов в изучении языка.

Отсутствие скобок

Кстати, мы можем не ставить скобки после new, если вызов конструктора идёт без аргументов.

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

// то же, что и

let user = new User();

Пропуск скобок считается плохой практикой, но синтаксис языка такое позволяет.

### Создание методов в конструкторе

Использование конструкторов для создания объектов даёт большую гибкость. Можно передавать конструктору параметры, определяющие, как создавать объект, и что в него записывать.

В this мы можем добавлять не только свойства, но и методы.

Например, в примере ниже, new User(name) создаёт объект с данным именем name и методом sayHi:

function User(name) {

this.name = name;

this.sayHi = function() {

alert( "Меня зовут: " + this.name );

};

}

let vasya = new User("Вася");

vasya.sayHi(); // Меня зовут: Вася

/\*

vasya = {

name: "Вася",

sayHi: function() { ... }

}

\*/

Для создания сложных объектов есть и более «продвинутый» синтаксис – [классы](https://learn.javascript.ru/classes), которые мы разберём позже.

### Итого

* Функции-конструкторы или просто конструкторы являются обычными функциями, именовать которые следует с заглавной буквы.
* Конструкторы следует вызывать при помощи оператора new. Такой вызов создаёт пустой this в начале выполнения и возвращает заполненный в конце.

Мы можем использовать конструкторы для создания множества похожих объектов.

JavaScript предоставляет функции-конструкторы для множества встроенных объектов языка: например, Date, Set и других, которые нам ещё предстоит изучить.