

Объекты.

JavaScript спроектирован на основе простой парадигмы. В основе концепции лежат простые объекты. Объект — это набор свойств, и каждое свойство состоит из имени и значения, ассоциированного с этим именем. Значением свойства может быть функция, которую можно назвать методом объекта.

В дополнение к встроенным в браузер объектам, вы можете определить свои собственные объекты.

Объекты в JavaScript, как и во многих других языках программирования, похожи на объекты реальной жизни. Концепцию объектов JavaScript легче понять, проводя параллели с реально существующими в жизни объектами.

В JavaScript объект — это самостоятельная единица, имеющая свойства и определенный тип. Сравним, например, с чашкой. У чашки есть цвет, форма, вес, материал, из которого она сделана, и т.д. Точно так же, объекты JavaScript имеют свойства, которые определяют их характеристики.

2 способа создать пустой объект:

```
let user = new Object(); // синтаксис "конструктор объекта"
```

```
let user = {}; // синтаксис "литерал объекта"
```

При использовании литерального синтаксиса {...} мы сразу можем поместить в объект несколько свойств в виде пар «ключ: значение»

```
let person = {  
  name : "Nicholas",  
  age : 29  
};
```

Свойства объекта также иногда называют полями объекта.

У каждого свойства есть ключ (также называемый «имя» или «идентификатор»). После имени свойства следует двоеточие ":", и затем указывается значение свойства. Если в объекте несколько свойств, то они перечисляются через запятую.

В литералах объектов имена свойств могут быть строками и числами, например:

```
let person = {  
  "name" : "Nicholas",  
  "age" : 29,  
  S: true  
};
```

Для доступа к свойствам объектов обычно применяется точечная нотация (dot notation), также использовать скобочную нотацию (bracket notation). В этом случае имя свойства указывается как строка в квадратных скобках, например:

```
alert(person["name"]); // "Nicholas"  
alert(person.name); // "Nicholas"
```

Главное преимущество скобочной нотации в том, что она позволяет использовать переменные для доступа к свойствам:

```
let propertyName = "name";  
alert(person[propertyName]); // "Nicholas"
```

Скобочная нотация также полезна, если имя свойства содержит ключевое или зарезервированное слово либо знак, вызывающий синтаксическую ошибку:

Удаление свойств:

```
delete user.name;
```


Методы(функции) в объектах

```
let user = {  
  name: "Sam",  
  age: 29  
};
```

```
user.sayHi = function() {  
  alert("Hi!");  
};
```

```
user.sayHi();
```

Сокращённая запись метода

```
user = {  
  sayHi: function() {  
    alert("Hi");  
  }  
};
```

```
user = {  
  sayHi() {  
    alert("Hi");  
  }  
};
```

Ключевое слово «this» в методах

Как правило, методу объекта необходим доступ к информации, которая хранится в объекте, чтобы выполнить с ней какие-либо действия (в соответствии с назначением метода).

Значение `this` вычисляется во время выполнения кода и зависит от контекста.

Стрелочные функции не имеют своего `this`, они получают `this` из внешней функции.

Задание

Создать объект человека со свойствами: имя, возраст и методом `sayIntro()`, в котором Вы скажете (с помощью `alert()`) как Вас зовут и сколько Вам лет.

Каждый экземпляр Object имеет свойства и методы из приведенного списка.

- **constructor** - функция, которая была использована для создания объекта. Это функция Object().

- **hasOwnProperty (имясвойства)** - указывает, есть ли у объекта (не у прототипа) данное свойство. Имя свойства должно быть указано как строка (например, `o.hasOwnProperty(" name")`).

- **isPrototypeOf(объект)** - определяет, является ли объект прототипом другого объекта

- **propertyIsEnumerable(имяСвойства)** - указывает, можно ли перебирать данное свойство в инструкции for-in Как и в случае метода `hasOwnProperty()`, имя свойства должно быть строкой.

- `toLocaleString()` - возвращает строковое представление объекта в соответствии с региональными настройками среды выполнения.
- `toString()` - возвращает строковое представление объекта.
- `valueOf()` - возвращает строковый, численный или логический эквивалент объекта, часто совпадающий с результатом вызова `toString()`.

Существует специальный оператор "in" для проверки существования свойства в объекте.

"key" in object

Пример:

```
let user = { name: "John", age: 30 };
```

```
alert( "age" in user ); // true  
alert( "blabla" in user ); // false
```

Перечисление всех свойств объекта

Начиная с ECMAScript 5, есть три способа перечислить все свойства объекта (получить их список):

- **циклы `for...in`**

Этот метод перебирает все перечисляемые свойства объекта

- **`Object.keys(o)`**

Этот метод возвращает массив со всеми собственными именами перечисляемых свойств объекта `o`.

- **`Object.getOwnPropertyNames(o)`**

Этот метод возвращает массив содержащий все имена своих свойств (перечисляемых и неперечисляемых) объекта `o`.

Пример:

```
let user = {  
  name: "John",  
  age: 30,  
  isAdmin: true  
};
```

```
for (let key in user) {  
  // ключи  
  alert( key ); // name, age, isAdmin  
  // значения ключей  
  alert( user[key] ); // John, 30, true  
}
```

Копирование по ссылке

Одним из фундаментальных отличий объектов от примитивных типов данных является то, что они хранятся и копируются «по ссылке».

Примитивные типы: строки, числа, логические значения – присваиваются и копируются «по значению».

Переменная хранит не сам объект, а его «адрес в памяти», другими словами «ссылку» на него.

Когда переменная объекта копируется – копируется ссылка, сам же объект не дублируется.

Сравнение объектов

Операторы равенства `==` и строгого равенства `===` для объектов работают одинаково.

Два объекта равны только в том случае, если это один и тот же объект.

Например, две переменные ссылаются на один и тот же объект, они равны

Клонирование и объединение объектов, Object.assign

Если нам всё же нужно создать независимую копию?

В JavaScript нет встроенного метода для этого. На самом деле, такая нужда возникает редко. В большинстве случаев нам достаточно копирования по ссылке.

Но если мы действительно этого хотим, то нам нужно создавать новый объект и повторять структуру дублируемого объекта, перебирая его свойства и копируя их, например через `for...in`

Клонирование и объединение объектов, Object.assign

Кроме того, для этих целей мы можем использовать метод `Object.assign`.

`Object.assign(dest, [src1, src2, src3...])`

- Аргументы `dest`, и `src1, ..., srcN` (может быть столько, сколько нужно) являются объектами.
- Метод копирует свойства всех объектов `src1, ..., srcN` в объект `dest`. То есть, свойства всех перечисленных объектов, начиная со второго, копируются в первый объект. После копирования метод возвращает объект `dest`.

Сборка мусора

JavaScript - это язык со сборкой мусора, то есть за управление памятью при работе сценариев отвечает среда выполнения.

JavaScript освобождает разработчиков от забот по управлению памятью, автоматически выделяя сценариям нужную память и возвращая в среду память, которая больше не используется.

Идея сборки мусора проста: нужно выяснить, какие переменные больше не потребуются, и освободить связанную с ними память. Сборщик мусора запускается периодически с заданной частотой или в predetermined моменты выполнения кода.

Сборка мусора

При нормальном жизненном цикле локальной переменной она создается в ходе выполнения функции. В этот момент для нее выделяется память в стеке. Далее переменная используется в функции, и рано или поздно функция завершается. После этого переменная больше не нужна, поэтому ее память может быть освобождена для повторного использования.

Сборка мусора

Пример:

```
let user = {  
  name: "John"  
};
```

```
user = null;
```

объект John становится недостижимым. К нему нет доступа, на него нет ссылок. Сборщик мусора удалит эти данные и освободит память.

```
let user = {  
  name: "John"  
};
```

```
let admin = user;
```

```
user = null;
```

объект John всё ещё достижим через глобальную переменную `admin`, поэтому он находится в памяти. Если бы мы также перезаписали `admin`, то John был бы удалён.

Сборка мусора

Объем памяти, доступной в веб-браузерах, гораздо меньше, чем в обычных приложениях. Он ограничивается в основном ради безопасности, чтобы сценарии JavaScript в веб-страницах не могли вызвать сбой операционной системы, израсходовав всю системную память.

Ограничения памяти влияют не только на выделение памяти для переменных, но и на стек вызовов и количество инструкций, выполняемых в одном потоке.

Сборка мусора

Экономия памяти в JavaScript-сценариях ускоряет обработку страниц, а наилучший способ оптимизировать использование памяти - это хранить в коде только те данные, которые требуются для его выполнения. Если данные больше не нужны, лучше всего присвоить соответствующей переменной значение `null`, чтобы разорвать ее связь с данными.

Этот совет относится преимущественно к глобальным переменным и свойствам глобальных объектов. В случае локальных переменных ссылки на данные удаляются автоматически, когда переменные покидают контекст

Тип данных Symbol

В качестве ключей для свойств объекта могут использоваться только строки или символы. «Символ» представляет собой уникальный идентификатор.

Создаются новые символы с помощью функции `Symbol()`

```
let id = Symbol();
```

Тип данных Symbol

Символы позволяют создавать «скрытые» свойства объектов, к которым нельзя нечаянно обратиться и перезаписать их из других частей программы.

Тип данных Symbol

Если мы хотим использовать символ при literalном объявлении объекта {...}, его необходимо заключить в квадратные скобки.

```
let id = Symbol("id");
```

```
let user = {  
  name: "John",  
  [id]: 123  
};
```

Тип данных Symbol

Символы игнорируются циклом `for...in` и методом `Object.keys`

А вот `Object.assign` копирует и строковые, и символьные свойства

Задание 1

Создать объект `car` с набором свойств по желанию. Вывести в цикле все ключи и значения объекта.

Задание 2

Создать объект `user` с свойствами `name`, `email`, `phone`, `id`. Скопировать этот объект со всеми свойствами в новую переменную `newUser`.

Задание 3

Создать объект `circle` со свойствами: `radius`, `color`.

Также создать в объекте метод `calculateCircumference()`, при вызове которого, в консоль выводится длина окружности ($2 * \text{число пи} * \text{радиус}$).

Задание 4

Создать объект `message` с полем `text` и методами: `getMessage(string)`, `showMessage()`. При вызове метода `getMessage` в поле `text` записывается переданная в качестве аргумента строка. При вызове метода `showMessage()` вызывается `alert()` с содержимым поля `text`

Задание 5

Создать объект Прямоугольник (rectangle) с параметрами высота (height) и ширина (width) и методом `getArea()`, который возвращает площадь

```
console.log(rectangle.width);    // => 10
```

```
console.log(rectangle.height);   // => 20
```

```
console.log(rectangle.getArea()); // => 200
```

Задание 6

Напишите функцию `isEmpty(obj)`, которая возвращает `true`, если у объекта нет свойств, иначе `false`.

Задание 7*

Создать объект `selfGeneratedUser` с методами:

- 1) `getInfo()`, при вызове которого мы через `prompt()` поочередно получим данные об имени (`name`), емайле (`email`) и телефоне (`phone`) пользователя и запишем их в соответствующие свойства объекта.
- 2) `introduce()`, при вызове которого мы поочередно выводим с помощью `alert` `name`, `email` и `phone`. Если поля нет - не выводим его, переходим к другому полю.