Объекты

Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств. Свойство – это пара «ключ: значение», где ключ – **это строка** (также называемая «именем свойства»), а значение может быть чем угодно.

Пустой объект («пустой ящик») можно создать, используя один из двух вариантов синтаксиса:

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

Обычно используют вариант с фигурными скобками {...}. Такое объявление называют литералом объекта или литеральной нотацией. Например:

```
let user = {     // объект
  name: "John",  // под ключом "name" хранится значение "John"
  age: 30        // под ключом "age" хранится значение 30
 "likes birds": true // имя свойства из нескольких слов должно быть в кавычках
};
```

Для обращения к свойствам используется запись «через точку»:

```
alert( user.name ); // John
```

Для удаления свойства мы можем использовать **оператор delete**:

```
delete user.age;
```

Объект, объявленный через const, может быть изменён.Например:

```
const user = {
  name: "John"
};
user.name = "Pete"; // (*)
alert(user.name); // Pete
```

Для свойств, имена которых состоят из нескольких слов, доступ к значению «через точку» не работает: Для таких случаев существует альтернативный способ доступа к свойствам через квадратные скобки. Такой способ сработает с любым именем свойства:

```
user["likes birds"] = true;
```

Мы можем использовать квадратные скобки в литеральной нотации для создания вычисляемого свойства. Пример:

```
let fruit = prompt("Какой фрукт купить?", "apple");
let bag = {
  [fruit]: 5, // имя свойства будет взято из переменной fruit
};
```

В реальном коде часто нам необходимо использовать существующие переменные как значения для свойств с тем же именем. Например:

```
function makeUser(name, age) {
  return {
    name: name,
    age: age
    // ...другие свойства
  };
}
```

Такой подход настолько распространён, что существуют специальные короткие свойства для упрощения этой записи. Вместо name:name мы можем написать просто name:

```
function makeUser(name, age) {
  return {
    name, // то же самое, что и name: name
    age   // то же самое, что и age: age
  };
}
```

Мы можем использовать как обычные свойства, так и короткие в одном и том же объекте.

Имя переменной не может совпадать с зарезервированными словами, такими как «for», «let», «return» и т.д. Но для свойств объекта такого ограничения нет. Иными словами, нет никаких ограничений к именам свойств. Они могут быть в виде строк или символов (специальный тип для идентификаторов, который будет рассмотрен позже). Все другие типы данных будут автоматически преобразованы к строке. Есть небольшой подводный камень, связанный со специальным свойством **__proto__**. Мы не можем установить его в необъектное значение. Присвоение примитивного значения игнорируется.

Проверка существования свойств

Можно получить доступ к любому свойству. Даже если свойства не существует – ошибки не будет! При обращении к свойству, которого нет, возвращается undefined. Это позволяет просто проверить существование свойства:

```
let user = {};
alert( user.noSuchProperty === undefined ); // true означает "свойства нет"
```

Также существует специальный оператор "in" для проверки существования свойства в объекте.

"key" in object

Этот оператор работает правильно, даже если свойство существует, но его значение = undefined.

Опциональная цепочка '?.'

Опциональная цепочка ?. — это безопасный способ доступа к свойствам вложенных объектов, даже если какое-либо из промежуточных свойств не существует. Опциональная цепочка ?. останавливает вычисление и возвращает undefined, если часть перед ?. имеет значение undefined или null. Для краткости в этой статье мы будем говорить о значении, что оно «существует», если оно отличается от null или undefined.
Вот безопасный способ обратиться к свойству user.address.street:

```
let user = {}; // пользователь без адреса
alert( user?.address?.street ); // undefined (без ошибки)
```

Переменная перед ?. должна быть объявлена.

Синтаксис опциональной цепочки ?. имеет три формы:

- obj?.prop – возвращает obj.prop, если существует obj, и undefined в противном случае.
- obj?.[prop] – возвращает obj[prop], если существует obj, и undefined в противном случае.
- obj.method?.() – вызывает obj.method(), если существует obj.method, в противном случае возвращает undefined.

Как мы видим, все они просты и понятны в использовании. ?. проверяет левую часть выражения на равенство null/undefined, и продолжает дальнейшее вычисление, только если это не так.

цикл for..in.

Для перебора всех свойств объекта используется цикл for..in.

```
for (let key in object) {
  // тело цикла выполняется для каждого свойства объекта
}
```

Упорядочение свойств объекта

Свойства упорядочены особым образом: свойства с целочисленными ключами сортируются по возрастанию, остальные располагаются в порядке создания. Термин «целочисленное свойство» означает строку, которая может быть преобразована в целое число **и обратно** без изменений.
"+49" – можно сделать так, чтобы было по порядку создания.

Копирование объектов и ссылки

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

```
let user = { name: 'Иван' };
let admin = user;
admin.name = 'Петя'; // изменено по ссылке из переменной "admin"
alert(user.name); // 'Петя', изменения видны по ссылке из переменной "user"
```

Операторы равенства == и строгого равенства === для объектов работают одинаково. Два объекта равны только в том случае, если это **один и тот же объект**.

Но что, если нам всё же нужно дублировать объект? Создать независимую копию, клон? Нужно создать новый объект и скопировать свойства в него. Кроме того, для этих целей мы можем использовать метод Object.assign.

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

- Первый аргумент dest — целевой объект.
- Остальные аргументы src1, ..., srcN (может быть столько, сколько нужно)) являются исходными объектами
- Метод копирует свойства всех исходных объектов src1, ..., srcN в целевой объект dest. То есть, свойства всех перечисленных объектов, начиная со второго, копируются в первый объект.
- Возвращает объект dest.

Например, объединим несколько объектов в один:let user = { name: "Иван" };

```
let permissions1 = { canView: true };
let permissions2 = { canEdit: true };
// копируем все свойства из permissions1 и permissions2 в user
Object.assign(user, permissions1, permissions2);
// теперь user = { name: "Иван", canView: true, canEdit: true }
```

Если принимающий объект (user) уже имеет свойство с таким именем, оно будет перезаписано.
Простое клонирование:

```
let clone = Object.assign({}, user);
```

Также с помощью Object.assign можно добавлять новые свойства в объект:

```
let userInfo = {
name: “Вася”,
age: 30,
}
Object.assign(userInfo, {city: “Moscow”, [‘likes JS’: true]});
```

Для вложенных объектов используется глубокое клонирование с помощью метода из сторонней JS-библиотеки _.cloneDeep(obj) (https://lodash.com/docs#cloneDeep).

Методы объекта, "this"

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

```
// эти объекты делают одно и то же (одинаковые методы)
user = {
  sayHi: function() {
    alert("Привет");
  }
};
// сокращённая запись выглядит лучше, не так ли?
user = {
  sayHi() { // то же самое, что и "sayHi: function()"
    alert("Привет");
  }
};
```

Для доступа к информации внутри объекта метод может использовать ключевое слово this. Значение this – это объект «перед точкой», который использовался для вызова метода. В JavaScript this является «свободным», его значение вычисляется в момент вызова метода и не зависит от того, где этот метод был объявлен, а зависит от того, какой объект вызывает метод (какой объект стоит «перед точкой»). Вызов без объекта: this == undefined.

Стрелочные функции особенные: у них нет своего «собственного» this. Если мы используем this внутри стрелочной функции, то его значение берётся из внешней «нормальной» функции.

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".** (см. Функции)

Object.keys, values, entries

Для простых объектов доступны следующие методы:

- Object.keys(obj) – возвращает **массив** ключей.
- Object.values(obj) – возвращает **массив** значений.
- Object.entries(obj) – возвращает **массив** пар [ключ, значение].

Object.fromEntries(array) - чтобы преобразовать массив (Object.entries(obj)) обратно в объект.

```
let prices = {
  banana: 1,
  orange: 2,
  meat: 4,
};
let doublePrices = Object.fromEntries(
  // преобразовать в массив, затем map, затем fromEntries обратно объект
  Object.entries(prices).map(([key, value]) => [key, value * 2])
);
alert(doublePrices.meat); //
```

Так же, как и цикл for..in, эти методы игнорируют свойства, использующие **Symbol**(...) в качестве ключей.