1. **Объекты.**

Как известно, в JavaScript существует семь типов данных. Шесть из них называются примитивными, так как содержат только одно значение. Объекты же используются для хранения коллекций различных значений и более сложных сущностей. В JavaScript объекты используются очень часто, это одна из основ языка. Поэтому стоит их досконально изучить.

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 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.

[**Литералы и свойства**](https://learn.javascript.ru/object#literaly-i-svoystva)

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

let user = {     // объект

  name: "John",  // под ключом "name" хранится значение "John"

  age: 30        // под ключом "age" хранится значение 30

};

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 объекте user сейчас находятся два свойства:

1. Первое свойство с именем "name" и значением "John".
2. Второе свойство с именем "age" и значением 30.

Для обращения к свойствам используется запись «через точку»:

// получаем свойства объекта:

alert( user.name ); // John

alert( user.age ); // 30

Значение может быть любого типа. Добавим свойство с логическим значением:

user.isAdmin = true;

Для удаления свойства можно использовать оператор delete:

delete user.age;

Имя свойства может состоять из нескольких слов, но тогда оно должно быть заключено в кавычки. Последнее свойство объекта может заканчиваться висячей запятой:

let user = {

  name: "John",

  age: 30,

  "likes birds": true,

};

**[Квадратные скобки](https://learn.javascript.ru/object" \l "kvadratnye-skobki)**

Для свойств, имена которых состоят из нескольких слов, доступ к значению «через точку» не работает:

// приведет к синтаксической ошибкк

user.likes birds = true

Так происходит, потому что точка требует, чтобы ключ был именован по правилам именования переменных. То есть не имел пробелов, не начинался с цифры и не содержал специальные символы, кроме $ и \_. Для таких случаев существует альтернативный способ доступа к свойствам через квадратные скобки. Такой способ сработает с любым именем свойства:

let user = {};

// присваивание значения свойству

user["likes birds"] = true;

// получение значения свойства

alert(user["likes birds"]); // true

// удаление свойства

delete user["likes birds"];

Квадратные скобки также позволяют обратиться к свойству, имя которого может быть результатом выражения. Например, имя свойства может храниться в переменной:

let key = "likes birds";

user[key] = true;

Здесь переменная key может быть вычислена во время выполнения кода или зависеть от пользовательского ввода. После этого можно использовать её для доступа к свойству. Пример:

let user = {

  name: "John",

  age: 30

};

let key = prompt("Что вы хотите узнать о пользователе?", "name");

alert( user[key] ); // John (если ввели "name")

Запись «через точку» такого не позволяет:

let user = {

  name: "John",

  age: 30

};

let key = "name";

user.key // undefined

**[Вычисляемые свойства](https://learn.javascript.ru/object" \l "vychislyaemye-svoystva)**

Можно 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

};

alert( bag.apple ); // 5, если fruit="apple"

Смысл вычисляемого свойства прост: запись [fruit] означает, что имя свойства необходимо взять из переменной fruit. И если посетитель введёт слово "apple", то в объекте bag теперь будет храниться свойство {apple: 5}.

Можно использовать и более сложные выражения в квадратных скобках:

let fruit = 'apple';

let bag = {

  [fruit + 'Computers']: 5 // bag.appleComputers = 5

};

Таким образом, когда имена свойств известны и просты, используется запись через точку. Если же нужно что-то более сложное, то следует использовать квадратные скобки. Зарезервированные слова разрешено использовать как имена свойств:

let obj = {

  for: 1,

  let: 2,

  return: 3

};

alert( obj.for + obj.let + obj.return );  // 6

**[Свойство из переменной](https://learn.javascript.ru/object" \l "svoystvo-iz-peremennoy)**

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

function makeUser(name, age) {

  return {

    name: name,

    age: age

    // ...другие свойства

  };

}

let user = makeUser("John", 30);

alert(user.name); // John

В примере выше название свойств name и age совпадают с названиями переменных, которые указываются в качестве значений этих свойств. Такой подход настолько распространен, что существуют специальные короткие свойства для упрощения этой записи. Вместо name:name можно написать просто name. Можно использовать как обычные свойства, так и короткие в одном и том же объекте:

function makeUser(name, age) {

  return {

    name,

    age

    height: 170// ...

  };

}

[**Проверка существования свойства**](https://learn.javascript.ru/object#proverka-suschestvovaniya-svoystva)

Особенность объектов в том, что 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 – сравнением его с undefined:

let user = {};

alert( user.noSuchProperty === undefined ); // true означает "свойства нет"

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

"key" in object

Пример:

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

alert( "age" in user ); // true, user.age существует

alert( "blabla" in user ); // false, user.blabla не существует

Обратите внимание, что слева от оператора in должно быть имя свойства. Обычно это строка в кавычках. Если кавычки опускаются, это значит, что используется переменная, в которой находится имя свойства. Например:

let user = { age: 30 };

let key = "age";

alert( key in user ); // true

Обычно строгого сравнения "=== undefined" достаточно для проверки наличия свойства. Но есть особый случай, когда оно не подходит, и нужно использовать "in". Это когда свойство существует, но содержит значение undefined:

let obj = {

  test: undefined

};

alert( obj.test ); // выведет undefined

alert( "test" in obj ); // true, свойство существует

В примере выше свойство obj.test технически существует в объекте. Оператор in сработал правильно. Подобные ситуации случаются очень редко, так как undefined обычно явно не присваивается. Для «неизвестных» или «пустых» свойств используется значение null.

[**Цикл «for…in»**](https://learn.javascript.ru/object#tsikl-for-in)

Для перебора всех свойств объекта используется цикл for..in. Этот цикл отличается от изученного ранее цикла for(;;). Синтаксис:

for (key in object) {

  // тело цикла выполняется для каждого свойства объекта

}

Например, выведем все свойства объекта user:

let user = {

  name: "John",

  age: 30,

  isAdmin: true

};

for (let key in user) {

  alert( key );  // name, age, isAdmin

  alert( user[key] ); // John, 30, true

}

Обратите внимание, что все конструкции «for» позволяют объявлять переменную внутри цикла, как, например, let key здесь. Кроме того, можно использовать другое имя переменной. Например, часто используется вариант "for (let prop in obj)".

[**Упорядочение свойств объекта**](https://learn.javascript.ru/object#uporyadochenie-svoystv-obekta)

Свойства объекта упорядочены особым образом: свойства с целочисленными ключами сортируются по возрастанию, остальные располагаются в порядке создания. В качестве примера рассмотрим объект с телефонными кодами:

let codes = {

  "49": "Германия",

  "41": "Швейцария",

  "44": "Великобритания",

  // ..,

  "1": "США"

};

for (let code in codes) {

  alert(code); // 1, 41, 44, 49

}

Телефонные коды идут в порядке возрастания, потому что они являются целыми числами: 1, 41, 44, 49.

Термин «целочисленное свойство» означает строку, которая может быть преобразована в целое число и обратно без изменений. То есть, "49" – это целочисленное имя свойства, потому что если его преобразовать в целое число, а затем обратно в строку, то оно не изменится. А вот свойства "+49" или "1.2" таковыми не являются. Если ключи не целочисленные, то они перебираются в порядке создания, например:

let user = {

  name: "John",

  surname: "Smith"

};

user.age = 25;

for (let prop in user) {

  alert( prop ); // name, surname, age

}

Таким образом, чтобы телефонные коды выводились, в том порядке в котром записаны, надо сделать коды не целочисленными свойствами. Для этого надо добавить знак "+" перед каждым кодом. Пример:

let codes = {

  "+49": "Германия",

  "+41": "Швейцария",

  "+44": "Великобритания",

  // ..,

  "+1": "США"

};

for (let code in codes) {

  alert( +code ); // 49, 41, 44, 1

}

[**Копирование по ссылке**](https://learn.javascript.ru/object#kopirovanie-po-ssylke)

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». Например:

let message = "Hello!";

let phrase = message;

В результате имеются две независимые переменные, каждая из которых хранит строку "Hello!". Объекты ведут себя иначе. Переменная хранит не сам объект, а его «адрес в памяти», другими словами «ссылку» на него. Например:

let user = {

  name: "John"

};

Сам объект хранится где-то в памяти. А в переменной user лежит «ссылка» на эту область памяти. Когда переменная объекта копируется – копируется ссылка, сам же объект не дублируется:

let user = { name: "John" };

let admin = user; // копируется ссылка

В результате есть две переменные, каждая из которых содержит ссылку на один и тот же объект. Можно использовать любую из переменных для доступа к объекту и изменения его содержимого:

let user = { name: 'John' };

let admin = user;

admin.name = 'Pete'; // изменено по ссылке из переменной "admin"

alert(user.name); // 'Pete', изменения видны по ссылке из переменной "user"

[**Сравнение объектов**](https://learn.javascript.ru/object#sravnenie-obektov)

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:

let a = {};

let b = a; // копирование по ссылке

alert( a == b ); // true

alert( a === b ); // true

В примере ниже два разных объекта не равны, хотя и оба пусты:

let a = {};

let b = {}; // два независимых объекта

alert( a == b ); // false

Для сравнений типа obj1 > obj2 или для сравнения с примитивом obj == 5 объекты преобразуются в примитивы. Такое сравнение используется очень редко и не рекомендуется.

[**Объекты-константы**](https://learn.javascript.ru/object#obekty-konstanty)

Объект, объявленный через const, может быть изменен:

const user = {

  name: "John"

};

user.age = 25; // (\*)

alert(user.age); // 25

Объявление const защищает от изменений только само значение user. В примере значение user – это ссылка на объект, и это значение не меняется. В строке (\*) вносятся изменения внутри объекта, а значение user не изменяется. Если же попытаться присвоить user другое значение, то const выдаст ошибку:

const user = {

  name: "John"

};

// Ошибка (нельзя переопределять константу user)

user = {

  name: "Pete"

};

Cделать константами свойства объекта тоже возможно с помощью флагов и дескрипторов свойств, кторые будут рассматриваться позже.

[**Клонирование и объединение объектов, Object.assign**](https://learn.javascript.ru/object#klonirovanie-i-obedinenie-obektov-object-assign)

При копировании переменной объекта создаётся ещё одна ссылка на тот же самый объект. Если надо создать независимую копию (клон), то необходимо создать новый объект и повторять структуру дублируемого объекта, перебирая его свойства и копируя их. Например так:

let user = {

  name: "John",

  age: 30

};

let clone = {};

for (let key in user) {

  clone[key] = user[key];

}

clone.name = "Pete";

alert( user.name );

Кроме того, для этих целей можно использовать метод [Object.assign](https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Object/assign). Синтаксис:

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.

Например, объединим несколько объектов в один:

let user = { name: "John" };

let permissions1 = { canView: true };

let permissions2 = { canEdit: true };

// user = { name: "John", canView: true, canEdit: true }

Object.assign(user, permissions1, permissions2);

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

let user = { name: "John" };

// user = { name: "Pete", isAdmin: true }

Object.assign(user, { name: "Pete", isAdmin: true });

Также можно использовать Object.assign для простого клонирования:

let user = {

  name: "John",

  age: 30

};

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

Все свойства объекта user будут скопированы в пустой объект, и ссылка на этот объект будет в переменной clone. Такое клонирование работает так же, как и через цикл, но короче.

Если свойство не примитивно, а явлется ссылкой на другой объект, то при клонировании недостаточно просто скопировать clone.sizes = user.sizes, поскольку user.sizes – это объект, он будет скопирован по ссылке. А значит объекты clone и user в своих свойствах sizes будут ссылаться на один и тот же объект:

let user = {

  name: "John",

  sizes: {

    height: 182,

    width: 50

  }

};

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

alert( user.sizes === clone.sizes );

user.sizes.width++;

alert(clone.sizes.width); // 51

Чтобы исправить это, необходимо в цикле клонирования делать проверку, не является ли значение user[key] объектом, и, если это так, – копировать и его структуру тоже. Это называется «глубокое клонирование». Существует стандартный алгоритм глубокого клонирования, [Structured cloning algorithm](http://w3c.github.io/html/infrastructure.html" \l "safe-passing-of-structured-data). Он решает описанную выше задачу, а также более сложные задачи.