**ОБЪЕКТЫ**

Определение и использование

Объекты – это составной тип данных в JavaScript, который содержит коллекцию свойств, каждое из которых имеет пару ключ-значение.  
Каждое свойство состоит из пары ключ-значение. Ключ – это имя свойства, по которому можно получить его значение (каждое свойство должно иметь уникальное имя, которое может быть строкой или числом).  
Значениями могут быть числа, строки, булевые значения, функции или же другие объекты.  
Свойства, у которых в значениях указаны функции называются методами.

Создание объектов

Объекты можно создавать с помощью литералов объектов, ключевого слова new или функции Object.create().

let object = {};

let newObject = new Object();

let createObject = Object.create();

Создание объекта с помощью литерала (конструкция {}) или с помощью ключевого слова new создаст одинаковые пустые объекты. Создание с помощью Object.create() имеет свои особенности и как правило не используется для создания простых, пустых объектов, этот способ создания будет рассмотрен в последующих темах. Таким образом, чаще всего, для создания объектов используется литеральная форма.

Определение свойств объекта

Определить свойство объекта можно сразу при его инициализации. После каждой пары ключ-значение обязательно ставится запятая.

let user = {

firstName: 'John',

secondName: 'Connor'

};

Так же можно добавить свойство в уже объявленный объект при помощи точки (.) или квадратных скобок ([]). При присвоении значения через квадратные скобки, ключ необходимо указывать как строку.

let user = {};

user.firstName = 'John';

user['secondName'] = 'Connor';

Получение и изменение свойств

Для получения или изменения свойств объекта используется тот же синтаксис что и при добавлении новых свойств с помощью точки (.) или квадратных скобок ([]).

let user = {

firstName: 'John',

secondName: 'Connor',

login: 'T2',

password: 'doomsday',

age: 25

};

// Получение значения

let name = user.firstName; // 'John'

let surname = user['secondName']; // 'Connor'

// Изменение значения

user.password; // 'doomsday'

user.password = 'judgement\_day';

user.password; // 'judgement\_day'

При попытке обратиться к значению, которое отсутствует, получим undefined.

let user = {

firstName: 'John',

secondName: 'Connor'

};

user.age; // undefined

Удаление свойств

с помощью оператора delete можно удалить свойство из объекта.

let user = {

firstName: 'John',

secondName: 'Connor',

login: 'T2',

password: 'doomsday',

age: 25

};

user.age; // 25

delete user.age;

user.age; // undefined

user['secondName']; // 'Connor'

delete user['secondName'];

user['secondName']; // undefined

Как видно, оба синтаксиса обращения к свойству объекта (точка (.) или квадратные скобки ([])) можно использовать во всех случаях. На практике же чаще используется синтаксис с точкой.  
Синтаксис с квадратными скобками используется в тех случаях, когда заранее не известно ключ свойства или происходит динамическое вычисление имени ключа. Например, при переборе объекта в цикле.

let user = {

firstName: 'John',

secondName: 'Connor',

login: 'T2',

password: 'doomsday',

age: 25

};

let key;

for (key in user) {

console.log(user[key]);

}

// John

// Connor

// T2

// doomsday

// 25

В примере выше, в цикле, мы указываем, что «искать» ключ (key) в объекте (здесь это объект user). Перебирая все свойства объекта, при каждой новой итерации, в переменную key по очереди подставляется каждый ключ. То есть, при каждой новой итерации, динамически вычисляется значение ключа в объекте. Интерпретатор наперед не знает какие ключи содержит в себе объект.  
Таким образом, выводя в консоль, в примере выше, user[key], мы буквально пишем user[‘firstName’], user[‘secondName’] и т.д. На каждой итерации, по очереди, подставляются все ключи.  
Если же указать в цикле user.key, то интерпретатор поймет это буквально как обращение к свойству с ключом key, а так как свойства с таким ключом несуществует, то мы получим undefined. При чем такой результат будет выведен столько раз, сколько свойств содержит объект, ведь при каждой итерации будет происходить попытка обратиться к свойству с ключом key.

let user = {

firstName: 'John',

secondName: 'Connor',

login: 'T2',

password: 'doomsday',

age: 25

};

let key;

for (key in user) {

console.log(user.key);

}

// undefined

// undefined

// undefined

// undefined

// undefined

Выводя в консоль просто переменную key, можно получить имена всех ключей объекта.

let user = {

firstName: 'John',

secondName: 'Connor',

login: 'T2',

password: 'doomsday',

age: 25

};

let key;

for (key in user) {

console.log(key);

}

// firstName

// secondName

// login

// password

// age

Свойства и методы

Если значение свойства объекта является функцией, то такое свойство носит название «метод», во всех остальных случаях, оно так и называется «свойство». Обратиться к методу можно с помощью того же синтаксиса что и к свойству.

let user = {

age: 25,

login: 'T2',

authenticated: true,

greating: function(name) {

return console.log('Hello ' + name);

}

}

user.greating('John'); // 'Hello John'

Как видно, методы вызываются как функции, с помощью круглых скобок. В них, как и в функции, при необходимости, можно передавать параметры. Таким образом, в примере выше, у нас есть объект с тремя свойствами и одним методом.

user.age; // свойство

user.login; // свойство

user.authenticated; // свойство

user.greating(); // метод

Объект может иметь более сложную структуру, содержать в себе еще один объект как значение свойства.

let user = {

firstName: 'John',

age: 25,

info: {

birthDay: '12/12/1985',

language: 'English'

},

showMessage: function (msg) {

console.log(msg);

}

}

user.info.language; // 'English'

Чтобы получить значение из вложенного объекта, мы сначала обращаемся к свойству, которое содержит объект как значение и далее, знакомым уже синтаксисом (с помощью точки или квадратных скобок) «движемся вглубь» вложенного объекта, указывая уже имена ключей его свойств.

Имена свойств могут состоять и из чисел. Но в этом случае к свойству можно обратиться только с помощью квадратных скобок, в другом случае будет ошибка.

let age = {

18: 'Teenager'

}

age.18; // Uncaught SyntaxError: Unexpected number

age['18']; // 'Teenager'

Концепция передачи по ссылке и по значению

Если вы хотите скопировать одну переменную в другую (если значение переменной примитив (число, строка, булевое значение, null или undefined)), то при копировании создастся новая копия переменной. При этом, меняя значение одной переменной, значение второй останется неизменным.

let jackPot = 777;

let bingo = jackPot;

jackPot; // 777

bingo; // 777

jackPot = null;

jackPot; // null

bingo; // 777

Создав переменную с присвоенным ей значением (jackPot) и передав ее как значение в другую переменную (bingo), во вторую переменную (bingo) попадает значение из первой (то есть 777). Значение из одной переменной (777 из jackPot) скопировалось в другую (bingo). При таком копировании создаются независимые переменные. По этой причине, дальнейшие изменения в какой-либо из переменных никак не влияют на другую.  
Такой механизм называется передача по значению.  
Если же в переменной хранится объект, то в ней хранится не сам объект (как в случае с примитивными значениями), а всего лишь ссылка на него, другими словами адрес на место в памяти где он хранится. Другими словами, объект содержится где-то в памяти программы, а переменная содержит всего лишь ссылку на его размещение, чтобы иметь возможность получать значения свойств этого объекта.  
Таким образом, если мы захотим скопировать переменную с объектом в другую переменную, то во второй переменной мы получим ту же ссылку на объект, что была в первой переменной.

var original = {

number: '777'

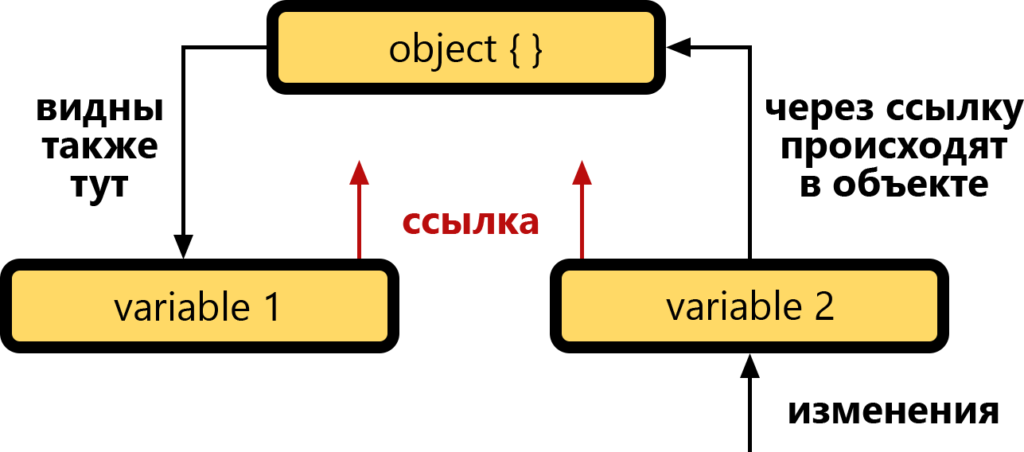
};

var linked = original;

original.number = '888';

console.log(linked.number); // '888'

В данном случае следует запомнить: изменив значение в объекте через одну из переменных, вы увидите эти изменения и в другой переменной, так как по факту вы изменили значение свойства объекта по ссылке к нему, а обе переменные имеют одинаковые ссылки на тот же объект. Такой механизм называется передача по ссылке.



Копирование объектов

При сравнении примитивов, сравниваются их значения, когда же сравниваются объекты, то сравниваются ссылки на них. Проверяется, имеют ли эти объекты ссылки на одно и тоже место в памяти.

let userA = {

age: 25,

sex: 'male'

}

let userB = {

age: 25,

sex: 'male'

}

console.log(userA == userB); // false

console.log(userA === userB); // false

В примере выше, хотя объекты имеют одинаковые свойства, они не равны. Как было указано выше, все потому, что объекты сравниваются не по значению, а по ссылке.

let userA = {

age: 25,

sex: 'male'

}

let userB = {

age: 25,

sex: 'male'

}

let userC = userA;

console.log(userA === userB); // false

console.log(userA === userC); // true

Хотя по своему наполнению объекты userA и userB одинаковы, они не равны друг другу, но объекты userA и userC при сравнении будут равны, так как имеют одинаковую ссылку на объект.  
Таким образом, невозможно скопировать один объект в другой с помощью присваивания его в другую переменную, так как в результате получим лишь копию ссылки на объект и любые изменения в одном из объектов повлекут изменения в другом. В ряде случаев возникает необходимость в создании независимой копии, клона объекта.  
Одним из способов является копирование свойств объекта с помощью цикла.

let products = {

fruit: 'banana',

vegetable: 'carrot',

juice: 'orange'

};

let key;

let newProducts = {}

for (key in products) {

newProducts[key] = products[key];

}

newProducts.fruit = 'apple';

products.fruit; // banana

newProducts.fruit; // 'apple'

products.fruit; // banana

После такого копирования объект newProducts является независимой копией объекта products и изменения в одном из них никак не отобразятся в другом.  
Стандарт ECMAScript 2015 ввел в спецификацию новый способ копирования, метод Object.assign().  
Object.assign() – копирует переданные объекты в указанный целевой объект и возвращает новую копию объекта со всеми свойствами. Первым параметром в метод передается целевой объект, объект в который будут скопированы все значения. Второй и последующие параметры – это объекты, которые необходимо скопировать.

let products = {

fruit: 'banana',

vegetable: 'carrot'

};

let newProduct = {

juice: 'orange'

}

let allProducts = Object.assign({}, products, newProduct);

allProducts; // {fruit: "banana", vegetable: "carrot", juice: "orange"}

В примере выше, как целевой объект передается пустой объект, в который и будут скопированы значения. Если же в качестве целевого объекта указать уже существующий объект, то мы изменим его, скопировав в него значения из других объектов.

let products = {

fruit: 'banana',

vegetable: 'carrot'

};

let newProduct = {

juice: 'orange'

}

let allProducts = Object.assign(products, newProduct);

products; // {fruit: "banana", vegetable: "carrot", juice: "orange"}

allProducts; // {fruit: "banana", vegetable: "carrot", juice: "orange"}

Если в объектах попадаются свойства с одинаковыми именами, то они перезаписываются в порядке слева на право.

let products = {

fruit: 'banana',

vegetable: 'carrot'

};

let newProduct = {

fruit: 'orange'

}

let allProducts = Object.assign({}, products, newProduct);

allProducts; // {fruit: 'orange', vegetable: "carrot"}

Итераторы объектов

for … in – проходится по каждому отдельному элементу (свойству) объекта.

let products = {

juice: 'orange',

fruit: 'banana',

vegetable: 'carrot'

};

for (let item in products) {

console.log('Key: ' + item + ', value: ' + products[item]);

}

// Key: juice, value: orange

// Key: fruit, value: banana

// Key: vegetable, value: carrot

Переменная item может иметь любое имя. Как уже говорилось ранее, в переменную item, попадает, при каждой итерации, имя свойства объекта.

Object.keys() – метод, который возвращает массив имен свойств объекта. Такой массив будет содержать имена свойств объект в виде строк.

let products = {

juice: 'orange',

fruit: 'banana',

vegetable: 'carrot'

};

let productsName = Object.keys(products);

console.log(productsName); // ["juice", "fruit", "vegetable"]

Spread оператор и деструктуризация объекта

Spread оператор (оператор разворота) – позволяет разворачивать один объект в другой. Синтаксис spread оператора – три точки (…). Другими словами, spread оператор, как и Object.assign(), копирует все свойства из объектов в новый объект.

const user = { name: 'David', age: 30 };

const tools = { system: 'Windows', editor: 'VS Code' };

const object = {...user, ...tools};

// { name: "David", age: 30, system: "Windows", editor: "VS Code" }

Результат от использования spread оператора и Object.assign() одинаковый. В случае с использованием spread оператора нет необходимости указывать промежуточный объект, куда все будет копироваться, он копирует все новые свойства непосредственно в определенную переменную.

В случае совпадения имен свойств, сработает тоже правило, что и при использовании Object.assign(), последнее полученное значение перетрет существующее.

const programmer = { name: 'David', age: 30, editor: 'VS Code' };

const user = { editor: 'Sublime text' };

const newUser = {...programmer, ...user};

// {name: "David", age: 30, editor: "Sublime text"}

Деструктуризация – присвоение свойств объекта сразу нескольким переменным.  
В этом нам помогает spread оператор. Без него, при необходимости присвоить свойства объекта в переменные, каждое свойство нужно отдельно записывать с одну переменную.

let products = {

juice: 'orange',

fruit: 'banana',

vegetable: 'carrot'

};

const juice = products.juice;

const fruit = products.fruit;

С использованием spread оператора все становится намного проще.

let products = {

juice: 'orange',

fruit: 'banana',

vegetable: 'carrot'

};

const {juice, fruit, vegetable} = products;

juice; // 'orange'

fruit; // 'banana'

vegetable; // 'carrot'

С лева перечисляются переменные, в которые необходимо присвоить значения, а с права объект, который необходимо «развернуть» в переменные.

Если есть необходимость присвоить значения в переменные с именами, которые отличаются от имен свойств, то в левой стороне необходимо после имени переменной, совпадающим с именем свойства, через двоеточие указать новое имя.

let products = {

juice: 'orange',

fruit: 'banana',

vegetable: 'carrot'

};

const {juice: j, fruit: f, vegetable: v} = products;

j; // 'orange'

f; // 'banana'

v; // 'carrot'

Если какого-то свойства не окажется в объекте, переменной можно задать значение по умолчанию.

let products = {

juice: 'orange',

fruit: 'banana'

};

const {juice: j, fruit: f, vegetable = 'carrot' } = products;

В случае, если свойств в объекте больше чем переменных, то можно указать специальную переменную с помощью rest оператора (оператор остатка), синтаксис которого совпадает с spread оператором.

let products = {

juice: 'orange',

fruit: 'banana',

vegetable: 'carrot'

};

const {juice, ...restProducts } = products;

juice; // 'orange'

restProducts; // {fruit: "banana", vegetable: "carrot"}

Spread оператор необходимо использовать с осторожностью, поскольку не все его возможности полностью поддерживаются браузерами.