**[Next →](https://www.javatpoint.com/es6-object-destructuring)**[**← Prev**](https://www.javatpoint.com/es6-array-destructuring)

ES6 Objects

Objects are the collection of key/value pairs that can be modified throughout the lifecycle of an object as similar to hash map or dictionary. Objects allow us to define custom data types in [JavaScript](https://www.javatpoint.com/javascript-tutorial).

Unlike primitive data types, we can represent complex or multiple values using objects. The values can be the array of objects or the scalar values or can be the functions. Data inside objects are unordered, and the values can be of any type.

An object can be created by using curly braces **{...}** along with an optional properties list. The property is a **"key:value"** pair, where the key is a string or a property name, and the value can be anything.

Syntax

There are two syntaxes to create an empty object:

* By using object literal
* By using object constructor

1. var user =  {};  // 'object literal' syntax
2. var name = **new** Object();  //'object constructor' syntax

Object Literal Syntax Extensions in ES6

Similar to primitive data types, objects also have a literal syntax. Object literal is one of the widely used patterns to create objects in JavaScript.

[ES6](https://www.javatpoint.com/es6) makes the object literal more concise and robust by extending the syntax in different ways.

Let us see the shorthand for object property initializer.

Object Property Initializer

Before ES6, the object literal is a collection of name-value pairs. For example,

**In ES5**

1. function user(name, division) {
2. **return** {
3. name: name,
4. divison: division
5. };
6. }

The above function **user()** takes two arguments, which are **name** and **division,** and returns a new object literal having two properties **name** and **division**. The property **name** and **division** take the values of the function parameters.

The above syntax looks repetitive because the **name** and **division** mentioned twice in keys and values of properties.

The syntax in **ES6** eliminates the duplication when an object property is the same as the name of a local variable. Let's understand this by rewriting the above **user()** [function in ES6](https://www.javatpoint.com/es6-functions).

**In ES6**

1. function user(name, division) {
2. **return** {
3. name,
4. divison
5. };
6. }

In the above code snippet, the JavaScript engine assigns the value of **name** and **division** arguments to the **name** and **division** properties.

Similarly, we can construct an object literal by local variables, as shown in the following example:

**Example**

1. let uname = 'Anil',
2. udivision = 'First';
4. let user = {
5. uname,
6. udivision
7. };
8. console.log(user.uname);
9. console.log(user.udivision);

**Output**

Anil

First

By using this shorthand syntax, the JavaScript engine looks in the scope for a variable with the same name. If it is found, then the value of the variable is assigned to the property. But if it is not found, then a reference error will occur.

Computed property name

Before ES6, we could use the square brackets **[]** to enable the computed property names for the object properties. Square bracket notation allows us to use variables and string literals as the name of the property.

ES6 introduced a new feature **'computed property names,'** which is a part of object literal syntax, and it uses the square bracket **[]** notation. It allows us to put an expression within the square brackets **[]** that will be computed and used as the name of the property.

Let us understand computed property names by using the following example:

**In ES5**

1. var emp = {
2. id : 101,
3. name : 'Amit'
4. }
5. var department = 'dep\_name';
6. emp[department]='Sales';
7. console.log(emp);

**Output**

{ id: 101, name: 'Amit', dep\_name: 'Sales' }

**In ES6**

1. var department = 'dep\_name';
2. var emp = {
3. id : 102,
4. name : 'Anil',
5. [department]:'Production'
6. }
7. console.log(emp);

**Output**

{ id: 102, name: 'Anil', dep\_name: 'Production' }

Concise method syntax

Before ES6, defining a method for an object literal, we must have to specify the name and definition of a complete function as represented in the following example:

**Example**

1. let user = {
2. firstName : "Sunil",
3. lastName : "Kumar",
4. fullName : function(){
5. **return** **this**.firstname + " " + **this**.lastName;
6. }
7. };

ES6 uses shorthand syntax, which is also known as the **concise method syntax** for making a method of the object literal concise by removing the colon **(:)** and the **function** keyword.

Let's use this syntax on the above example by rewriting the object **user**.

1. let user = {
2. firstName : "Sunil",
3. lastName : "Kumar",
4. fullName(){
5. **return** **this**.firstname + " " + **this**.lastName;
6. }
7. };

Merge objects in ES6

It is possible to merge two JavaScript objects in ES6 by using two methods, which are listed below:

* Object.assign() method
* Object spread syntax method

Let us try to understand these two methods.

By using Object.assign() method

This method is used for copying the values and properties from one or more source object to a target object. It returns the target object, including properties and values copied from the target object.

**Syntax**

1. Object.assign(target, sources)

**Example**

1. var obj1 = {1 : "Hello", 2: "World"};
2. var obj2 = { 3 : "Welcome", 4: "to"};
3. var obj3 = { 5 : "javaTpoint"}
5. // Using Object.assign()
6. var final\_obj = Object.assign(obj1, obj2, obj3);
7. console.log(final\_obj);

**Output**

{

'1': 'Hello',

'2': 'World',

'3': 'Welcome',

'4': 'to',

'5': 'javaTpoint'

}

Object Cloning

Cloning is the process of copying an object from one variable to another variable. We can clone an object by using the **object.assign() method**.

Let us try to understand it with the following example:

**Example**

1. let obj1 = {
2. name: 'Anil',
3. age: 22
4. };
5. let cloneobj = Object.assign({}, obj1);
7. cloneobj.age = 32;
9. console.log(obj1);
10. console.log(cloneobj);

**Output**

{ name: 'Anil', age: 22 }

{ name: 'Anil', age: 32 }

By using Object spread syntax

It is widely used in a variable array where multiple values are expected. As the objects in JavaScript are key-value paired entities, we can merge them into one by using the spread operator.

**Syntax**

1. var new\_obj = [...obj1, ...obj2, ...]

**Example**

1. var obj1 = {1 : "Hello", 2: "World"};
2. var obj2 = { 3 : "Welcome", 4: "to"};
3. var obj3 = { 5 : "javaTpoint"}
5. var final\_obj = {...obj1, ...obj2, ...obj3};
6. console.log(final\_obj);

**Output**

{

'1': 'Hello',

'2': 'World',

'3': 'Welcome',

'4': 'to',

'5': 'javaTpoint'

}

Delete properties

It can be possible to remove or delete a property by using the **delete** operator. Let us understand how to remove a property by using the following example.

**Example**

1. var obj = **new** Object;
2. obj.a = 50;
3. obj.b = 200;

6. delete obj.a;
7. console.log (obj.a);

**Output**

undefined

Object Destructuring

It is similar to array destructuring except that instead of values being pulled out of an array, the properties (or keys) and their corresponding values can be pulled out from an object.

When destructuring the objects, we use keys as the name of the variable. The variable name must match the property (or keys) name of the object. If it does not match, then it receives an **undefined** value. This is how JavaScript knows which property of the object we want to assign.