Destructuring Of Arrays and Objects in ES6 Model of JS

**Destructuring assignment**

The Destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

**Destructuring of Arrays**

The Destructuring of arrays can be happen in the following ways

1. Accessing the values sequential Destructuring

array = [12,3,4,56,4,3,54];

//sequential

const [a, b, c] = array

console.log(a); // 12

console.log(b); // 3

console.log(c); // 4

1. Accessing value in- between for example 1, ,3,4, 5, , 6 , ,7 and so on

array = [12,3,4,56,4,3,54];

//sequential

const [a, , c, d, , e] = array

console.log(a); // 12

console.log(c); // 4

console.log(d); // 56

console.log(e); // 4

1. Accessing rest of the values using spread operator …

array = [12,3,4,56,4,3,54];

//sequential

const [a, b, c, …rest] = array

console.log(a); // 12

console.log(b); // 3

console.log(c); // 4

console.log(rest); // 56,4,3,54

examples

array = [1,3,4,5,6,7,8,89,6];

// old way to access the specific value from array

const old\_a = array[0];

console.log(old\_a);

const old\_b = array[1];

console.log(old\_b);

// ES6 Way to access the specific value form array

// [] because we using the array to destructor

const [a,b] = array;

console.log(a);

console.log(b);

let a, b, rest;

[a, b] = [10, 20];

console.log(a);

// Expected output: 10

console.log(b);

// Expected output: 20

// … is the spread operator

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);

// Expected output: Array [30, 40, 50]

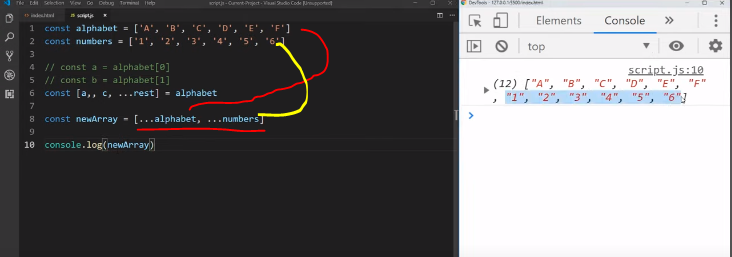
**Combining two arrays with spread operator …**

let ArrayNumbers = [11,3,43,4,3,345,5,34];

let ArrayAlphabets = [a,d,g,e,d,c,s,g,s,d,s,sd,df]l;

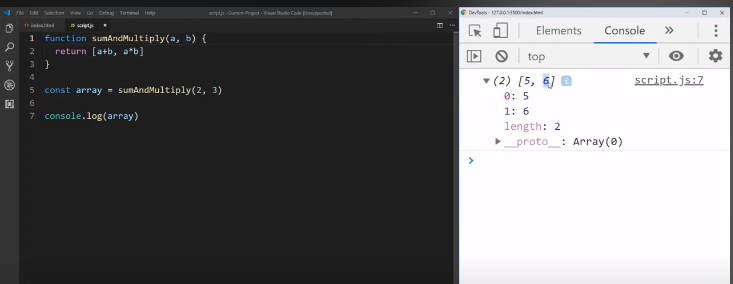
const newArray = […ArrayNumbers, … ArrayNumbers];

console.log(newArray); // [11,3,43,4,3,345,5,34 a,d,g,e,d,c,s,g,s,d,s,sd,df]

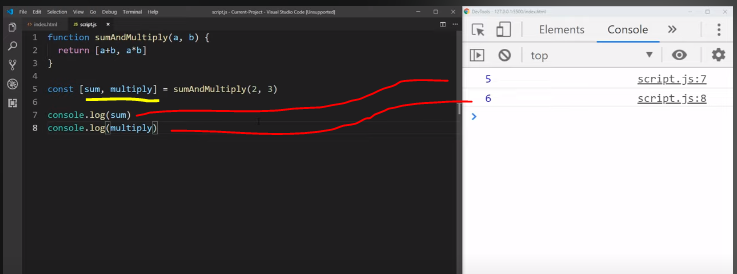


**Destructuring function that return an array**

This function returns an array of sum and multiply of numbers



Since the function returning an array we can destructor them



Another Example using Arrow Function

const func = (a,b) => [a+b, a\*b];

const array = func(3,5);

console.log(array); // [8, 15]

Destructuring the function array

const func = (a,b) => [a+b, a\*b];

const [sum, mul] = func(3,5);

console.log(sum); // 8

console.log(mul); // 15

**Sending default value when the value is not provided in array Destructuring**

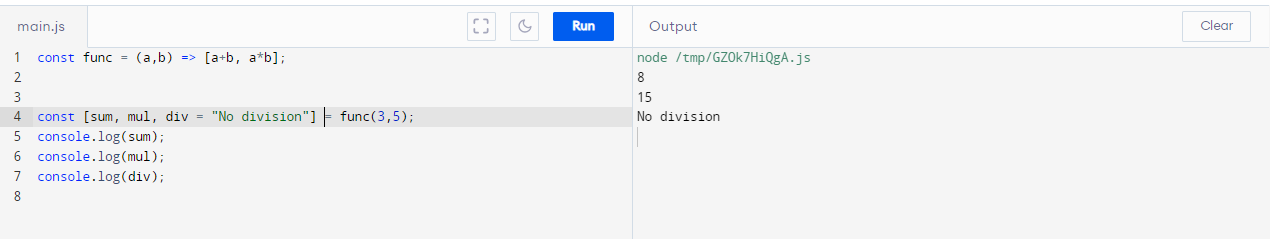
const func = (a,b) => [a+b, a\*b];

const [sum, mul, div = "No division"] = func(3,5);

console.log(sum); // 8

console.log(mul); // 15

console.log(div); // no division





But if the value is passes then

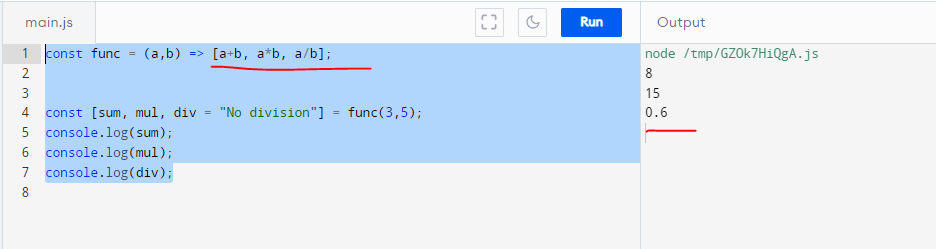
const func = (a,b) => [a+b, a\*b, a/b];

const [sum, mul, div = "No division"] = func(3,5);

console.log(sum); // 8

console.log(mul); // 15

console.log(div); //0.6



**Object Destructuring:**

let person = {

firstName: 'John',

lastName: 'Doe'

};

Prior to ES6, when you want to assign properties of the person object to variables, you typically do it like this:

let firstName = person.firstName;

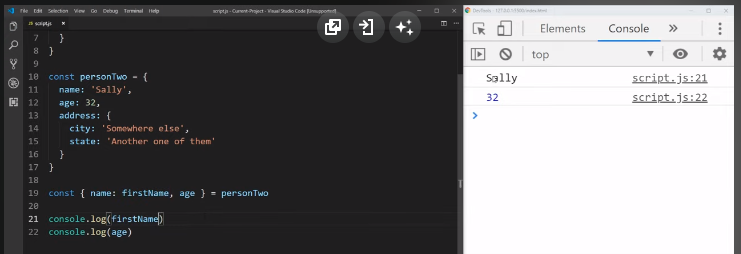
let lastName = person.lastName;

ES6 introduces the object destructuring syntax that provides an alternative way to assign properties of an object to variables:

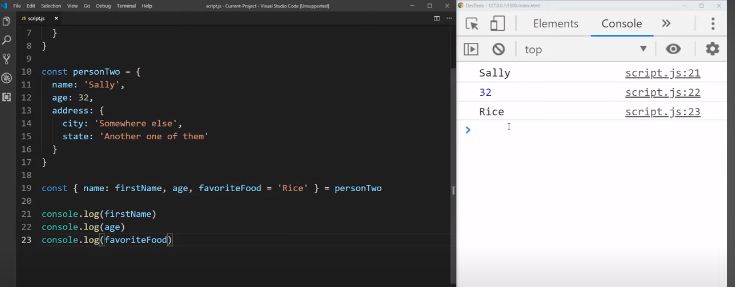
let { firstName: fname, lastName: lname } = person;

In this syntax:

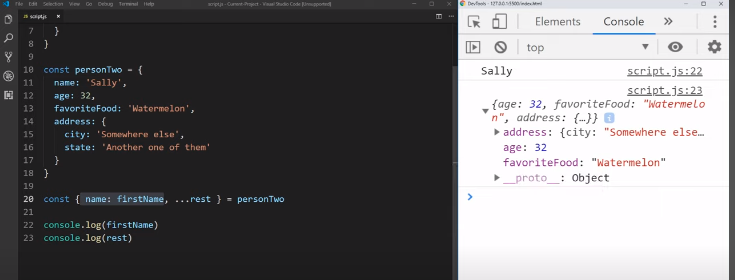
let { property1: variable1, property2: variable2 } = object;



Default value



Using Spread Operator in object Destructuring



If the variables have the same names as the properties of the object, you can make the code more concise as follows:

let { firstName, lastName } = person;

console.log(firstName); // 'John'

console.log(lastName); // 'Doe'

**Setting Default values:**

You can assign a default value to the variable when the property of an object doesn’t exist. For example:

let person = {

firstName: 'John',

lastName: 'Doe',

currentAge: 28

};

let { firstName, lastName, middleName = '', currentAge: age = 18 } = person;

console.log(middleName); // ''

console.log(age); // 28

In this example, we assign an empty string to the middleName variable when the person object doesn’t have the middleName property.

Also, we assign the currentAge property to the age variable with the default value of 18.

However, when the person object does have the middleName property, the assignment works as usual:

let person = {

firstName: 'John',

lastName: 'Doe',

middleName: 'C.',

currentAge: 28

};

let { firstName, lastName, middleName = '', currentAge: age = 18 } = person;

console.log(middleName); // 'C.'

console.log(age); // 28

**Destructuring a null object**

A function may return an object or null in some situations. For example:

function getPerson() {

return null;

}

And you use the object destructuring assignment:

let { firstName, lastName } = getPerson();

console.log(firstName, lastName);

The code will throw a TypeError:

TypeError: Cannot destructure property 'firstName' of 'getPerson(...)' as it is null.

To avoid this, you can use the OR operator (||) to fallback the null object to an empty object:

let { firstName, lastName } = getPerson() || {};

Now, no error will occur. And the firstName and lastName will be undefined.

**Nested object Destructuring**

Assuming that you have an employee object which has a name object as the property:

let employee = {

id: 1001,

name: {

firstName: 'John',

lastName: 'Doe'

}

};

The following statement destructures the properties of the nested name object into individual variables:

let {

name: {

firstName,

lastName

}

} = employee;

console.log(firstName); // John

console.log(lastName); // Doe

It’s possible to do multiple assignement of a property to multiple variables:

let employee = {

id: 1001,

name: {

firstName: 'John',

lastName: 'Doe'

}

};

let {

name: {

firstName,

lastName

},

name

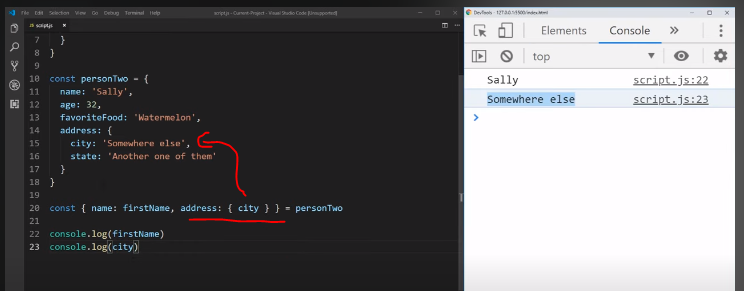
} = employee;

console.log(firstName); // John

console.log(lastName); // Doe

console.log(name); // { firstName: 'John', lastName: 'Doe' }

Destructuring Nested Objects:



**Destructuring function arguments**

Suppose you have a function that displays the person object:

let display = (person) => console.log(`${person.firstName} ${person.lastName}`);

let person = {

firstName: 'John',

lastName: 'Doe'

};

display(person);

It’s possible to destructure the object argument passed into the function like this:

let display = ({firstName, lastName}) => console.log(`${firstName} ${lastName}`);

let person = {

firstName: 'John',

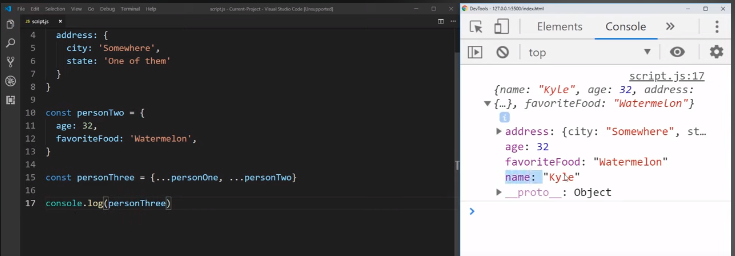
lastName: 'Doe'

};

display(person);

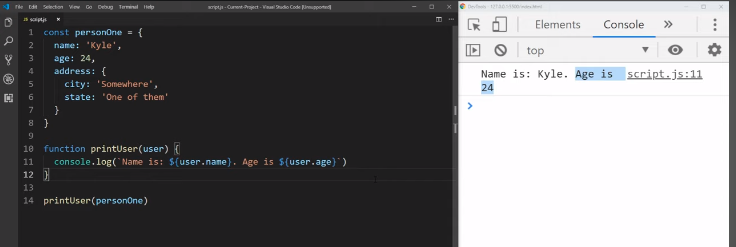
It looks less verbose especially when you use many properties of the argument object. This technique is often used in React.

**Adding Two Objects in one Object using Spread Operator …:**

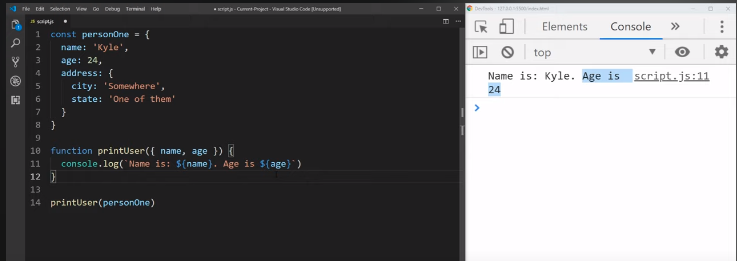


**Destructing Function object (Most Important)**

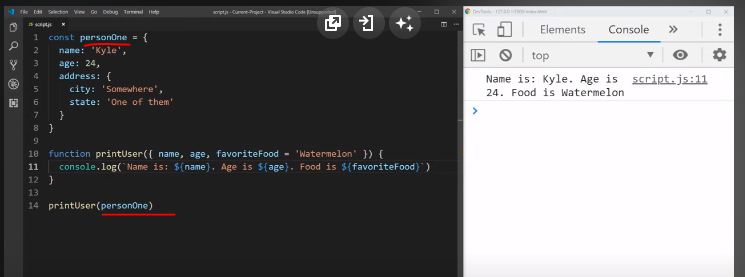
Old way using object variable

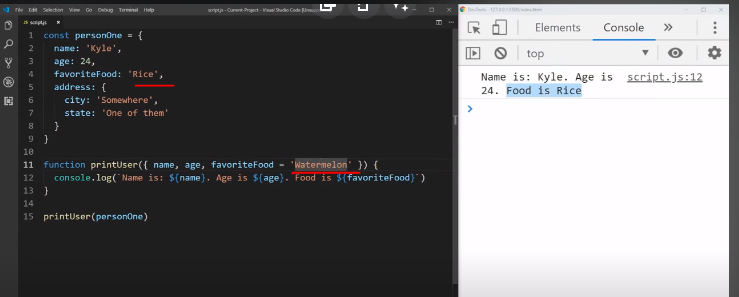


ES6 Destructuring the object and using its variables



Default value in function object Destructuring





For in-depth knowledge link MDN Page: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment>