Javascript - Day 8

Destructuring Assignment

Destructuring Assignment

When we pass objects or arrays to a function, we may not need all of it. The function might only require certain elements or properties.

Destructuring assignment is a special syntax that allows us to “unpack” arrays or objects into a bunch of variables, as sometimes that’s more convenient.

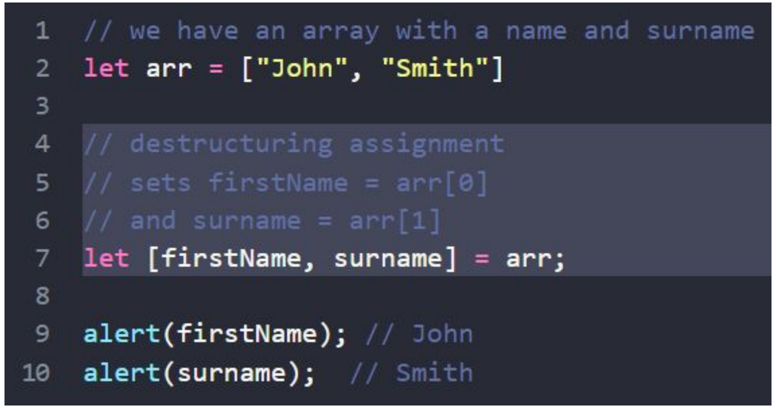
1. Array Destructuring
2. Object Destructuring
3. Nested Destructuring

Array Destructuring

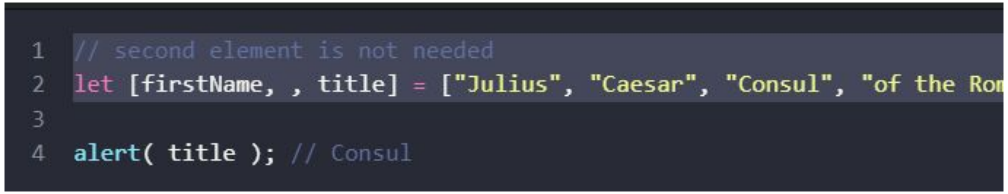
Now we can work with variables instead of array members.

We can ignore elements that we don't need using commas

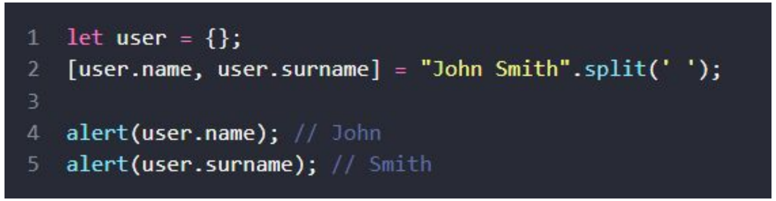
We can also use objects instead of variables for destructuring



```javascript
// we have an array with a name and surname
let arr = ["John", "Smith"]

// destructuring assignment
// sets firstName = arr[0]
// and surname = arr[1]
let [firstName, surname] = arr;

alert(firstName); // John
alert(surname);  // Smith
```



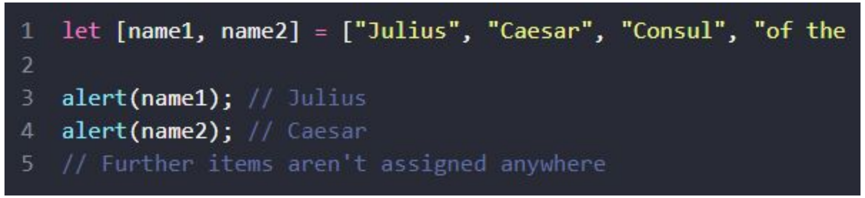
```javascript
// second element is not needed
let [firstName, , title] = ["Julius", "Caesar", "Consul", "of the Rom

alert( title ); // Consul
```



```javascript
let user = {};
[user.name, user.surname] = "John Smith".split(' ');

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

Array Destructuring - rest

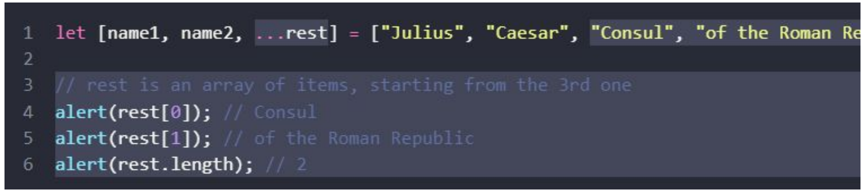
Usually, if the array is longer than the list at the left, the “extra” items are omitted.



```
let [name1, name2] = ["Julius", "Caesar", "Consul", "of the

alert(name1); // Julius
alert(name2); // Caesar
// Further items aren't assigned anywhere
```

For example, here only two items are taken, and the rest is just ignored:

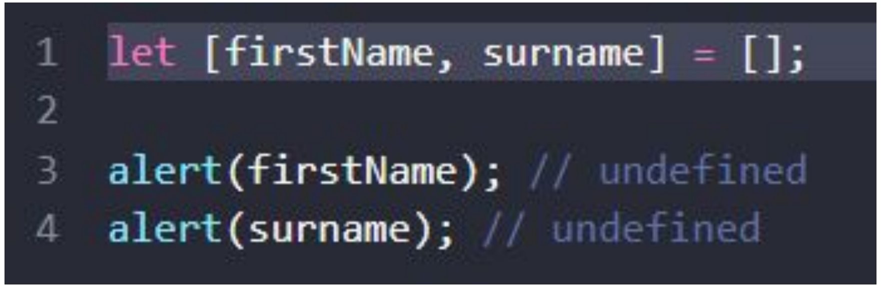
If we’d like also to gather all that follows – we can add one more parameter that gets “the rest” using three dots "...":



```
let [name1, name2, ...rest] = ["Julius", "Caesar", "Consul", "of the Roman Re

// rest is an array of items, starting from the 3rd one
alert(rest[0]); // Consul
alert(rest[1]); // of the Roman Republic
alert(rest.length); // 2
```

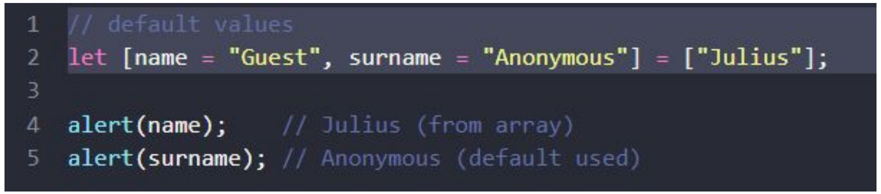
Array Destructuring - Default Values

If the array is shorter than the list of variables on the left, there will be no errors. Absent values are considered undefined:



```javascript
let [firstName, surname] = [];

alert(firstName); // undefined
alert(surname); // undefined
```

If we want a “default” value to replace the missing one, we can provide it using =:



```javascript
// default values
let [name = "Guest", surname = "Anonymous"] = ["Julius"];

alert(name);    // Julius (from array)
alert(surname); // Anonymous (default used)
```

Object Destructuring - Default Values

The destructuring assignment also works with objects.

The basic syntax is:

```
let {var1, var2} = {var1:…, var2:…}
```

Properties options.title, options.width and options.height are assigned to the corresponding variables.

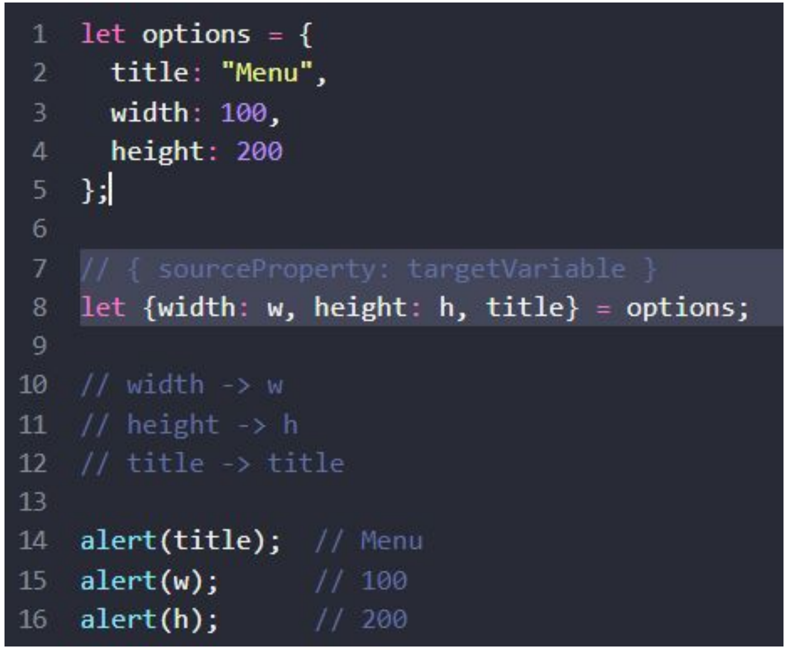
```
let options = {
  title: "Menu",
  width: 100,
  height: 200
};

let {title, width, height} = options;

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
```

The order does not matter. This works too:

```
// changed the order in let {...}
let {height, width, title} = { title: "Menu", height: 200, width: 100 }
```

Object Destructuring

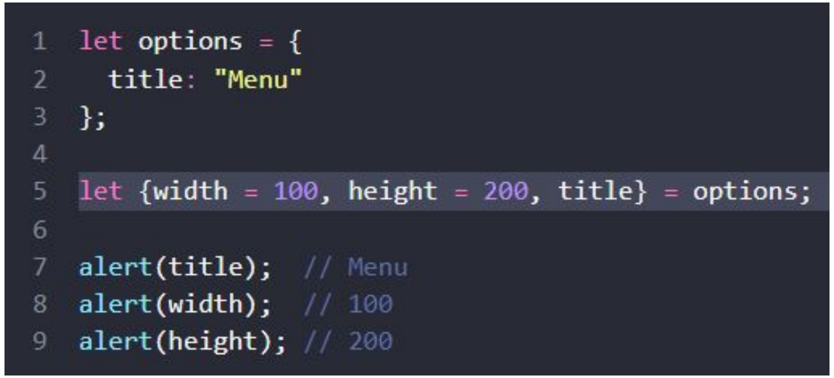
If we want to assign a property to a variable with another name, for instance, make options.width go into the variable named w, then we can set the variable name using a colon:



```
let options = {
  title: "Menu",
  width: 100,
  height: 200
};

// { sourceProperty: targetVariable }
let {width: w, height: h, title} = options;

// width -> w
// height -> h
// title -> title

alert(title);  // Menu
alert(w);      // 100
alert(h);      // 200
```

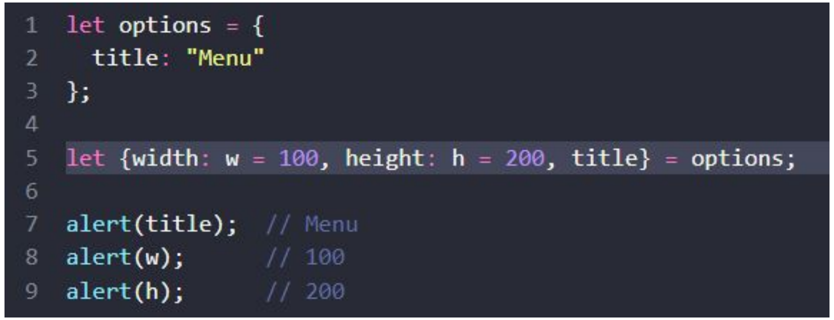
Object Destructuring - Default Values

For potentially missing properties we can set default values using "=", like this:



```javascript
let options = {
  title: "Menu"
};

let {width = 100, height = 200, title} = options;

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
```

We also can combine both the colon and equality:

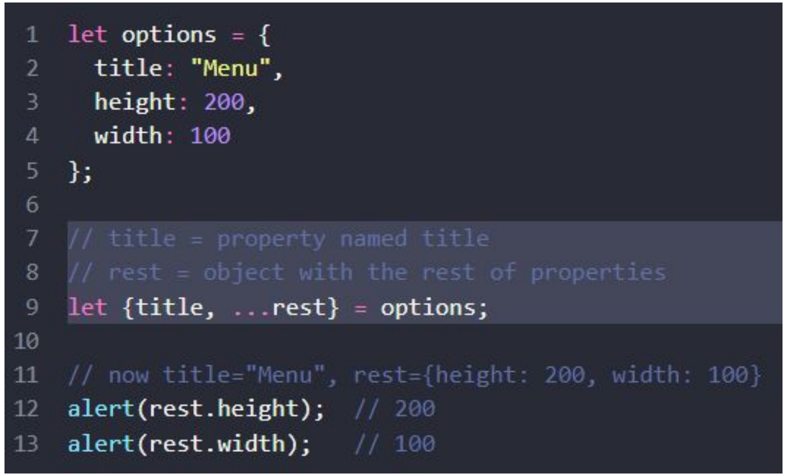


```javascript
let options = {
  title: "Menu"
};

let {width: w = 100, height: h = 200, title} = options;

alert(title);  // Menu
alert(w);      // 100
alert(h);      // 200
```

Object Destructuring - rest

What if the object has more properties than we have variables? Can we take some and then assign the “rest” somewhere?

We can use the rest pattern, just like we did with arrays. It’s not supported by some older browsers (IE, use Babel to polyfill it), but works in modern ones.

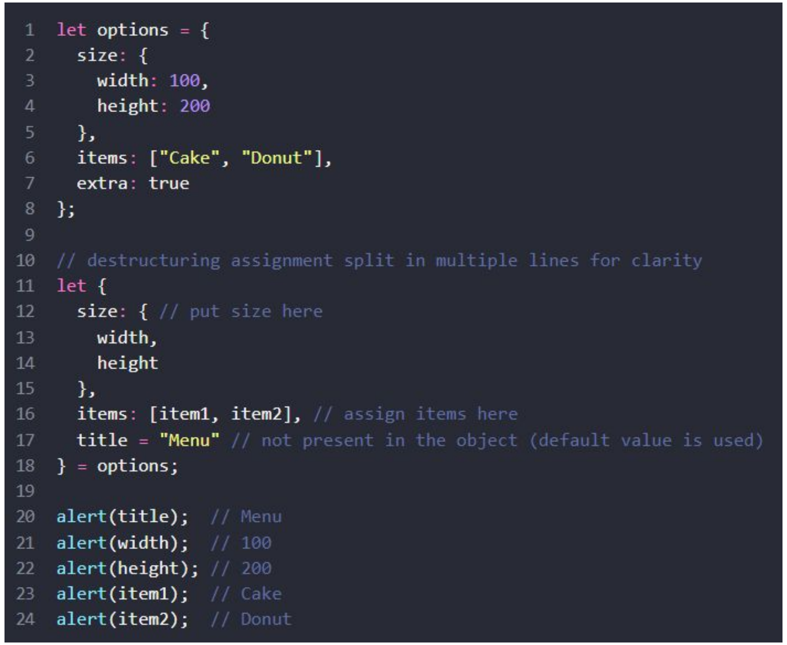


```javascript
let options = {
  title: "Menu",
  height: 200,
  width: 100
};

// title = property named title
// rest = object with the rest of properties
let {title, ...rest} = options;

// now title="Menu", rest={height: 200, width: 100}
alert(rest.height);  // 200
alert(rest.width);   // 100
```

Nested Destructuring

If an object or an array contains other nested objects and arrays, we can use more complex left-side patterns to extract deeper portions.

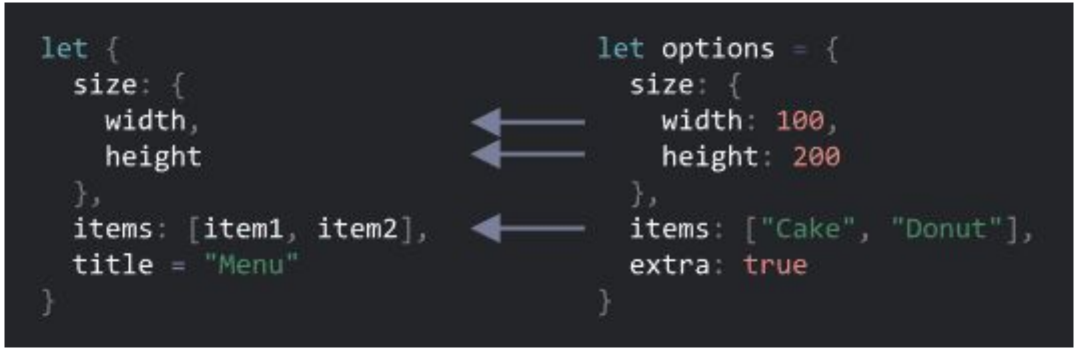
In the code below options has another object in the property size and an array in the property items. The pattern on the left side of the assignment has the same structure to extract values from them:



```
let options = {
  size: {
    width: 100,
    height: 200
  },
  items: ["Cake", "Donut"],
  extra: true
};

// destructuring assignment split in multiple lines for clarity
let {
  size: { // put size here
    width,
    height
  },
  items: [item1, item2], // assign items here
  title = "Menu" // not present in the object (default value is used)
} = options;

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
alert(item1);  // Cake
alert(item2);  // Donut
```

Nested Destructuring

All properties of options object except extra that is absent in the left part, are assigned to corresponding variables:



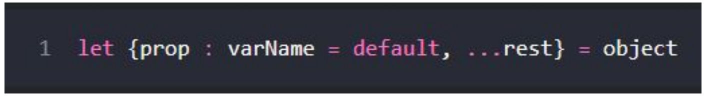
```
let {                                    let options = {
  size: {                                  size: {
    width,               <——                 width: 100,
    height               <——                 height: 200
  },                                       },
  items: [item1, item2], <——               items: ["Cake", "Donut"],
  title = "Menu"                           extra: true
}                                        }
```

Finally, we have width, height, item1, item2 and title from the default value.

Note that there are no variables for size and items, as we take their content instead.

Summary

- Destructuring assignment allows for instantly mapping an object or array into many variables.
- The full object syntax:

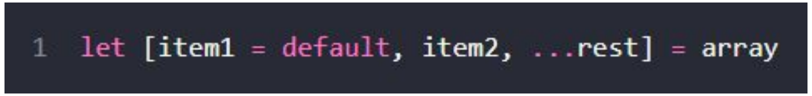


```
let {prop : varName = default, ...rest} = object
```

This means that property prop should go into the variable varName and, if no such property exists, then the default value should be used.

Object properties that have no mapping are copied to the rest object.

- The full array syntax:



```
let [item1 = default, item2, ...rest] = array
```

The first item goes to item1; the second goes into item2, all the rest makes the array rest.

- It’s possible to extract data from nested arrays/objects, for that the left side must have the same structure as the right one.