ARRAY AND OBJECT DESTRUCTURING IN JAVASCRIPT

The Destructuring assignment is a cool feature that came along with ES6. Destructuring is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables. That is, we can extract data from arrays and objects and assign them to variables

Extracting data from an array

```
let introduction = ["Hello", "I" , "am", "Sarah"];
let greeting = introduction[0];
let name = introduction[3];
console.log(greeting);//"Hello"
console.log(name);//"Sarah"
```

The ES6 destucturing assignment makes it easier to extract this data. How is this so? First, we will discuss the destructuring assignment with arrays. Then we will move on to object destructuring.

Basic Array Destructuring

If we want to extract data from arrays, it's quite simple using the destructuring assignment

Example

```
let introduction = ["Hello", "I" , "am", "Sarah"];
let [greeting, pronoun] = introduction;

console.log(greeting);//"Hello"
console.log(pronoun);//"I"
```

We can also do this with the same result

```
let [greeting, pronoun] = ["Hello", "I" , "am", "Sarah"];

console.log(greeting);//"Hello"
console.log(pronoun);//"I"
```

Declaring Variables before Assignment

Variables can be declared before being assigned like this:

```
let greeting, pronoun;
[greeting, pronoun] = ["Hello", "I" , "am", "Sarah"];
```

```
console.log(greeting);//"Hello"
console.log(pronoun);//"I"
```
Notice that the variables are set from left to right. So the first variable gets the first item in the array, the second variable gets the second variable in the array, and so on.

Skipping Items in an Array

What if we want to get the first and last item on our array instead of the first and second item, and we want to assign only two variables?this can be done:

```
let [greeting,,,name] = ["Hello", "I" , "am", "Sarah"];
console.log(greeting);//"Hello"
console.log(name);//"Sarah"
```
What just happened?

Look at the array on the left side of the variable assignment. Notice that instead of having just one comma, we have three. The comma separator is used to skip values in an array. So if you want to skip an item in an array, just use a comma.

Skipping the first and third item on the list

```
let [,pronoun,,name] = ["Hello", "I" , "am", "Sarah"];

console.log(pronoun);//"I"
console.log(name);//"Sarah"
```
Skipping all items

```
let [,,,,] = ["Hello", "I" , "am", "Sarah"];
```

Assigning the rest of an array

What if we want to assign some of the array to variables and the rest of the items in an array to a particular variable? In that case, we would do this:

```
let [greeting,...intro] = ["Hello", "I" , "am", "Sarah"];

console.log(greeting);//"Hello"
console.log(intro);//["I", "am", "Sarah"]
```
Using this pattern, you can unpack and assign the remaining part of an array to a variable.

Destructuring Assignment with Functions

We can also extract data from an array returned from a function. Let's say we have a function that returns an array like the example below:

```
function getArray() {
   return ["Hello", "I" , "am", "Sarah"];
}
let [greeting,pronoun] = getArray();

console.log(greeting);//"Hello"
console.log(pronoun);//"I"
```

Using Default Values

Default values can be assigned to the variables just in case the value extracted from the array is undefined:

```
let [greeting = "hi",name = "Sarah"] = ["hello"];

console.log(greeting);//"Hello"
console.log(name);//"Sarah"
```

So name falls back to "Sarah" because it is not defined in the array.

Swapping Values using the Destructuring Assignment

One more thing. We can use the destructuring assignment to swap the values of variables:

```
let a = 3;
let b = 6;

[a,b] = [b,a];

console.log(a);//6
console.log(b);//3
```

Object Destructuring

First, let's see why there is a need for object destructuring.

Say we want to extract data from an object and assign to new variables. Prior to ES6, how would this be done?

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};
```

```
let name = person.name;
let country = person.country;
let job = person.job;

console.log(name);//"Sarah"
console.log(country);//"Nigeria"
console.log(job);//Developer"
```

Basic Object Destructuring

Let's repeat the above example with ES6. Instead of assigning values one by one, we can use the object on the left to extract the data:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};

let {name, country, job} = person;

console.log(name);//"Sarah"
console.log(country);//"Nigeria"
console.log(job);//Developer"
```

You'll get the same results. It is also valid to assign variables to an object that haven't been declared:

```
let {name, country, job} = {name: "Sarah", country: "Nigeria", job: "Developer"};

console.log(name);//"Sarah"
console.log(country);//"Nigeria"
console.log(job);//Developer"
```

Variables declared before being assigned

Variables in objects can be declared before being assigned with destructuring. Let's try that:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};
let name, country, job;

{name, country, job} = person;
```

```
console.log(name);// Error : "Unexpected token ="
```

Wait, what just happened?! Oh, we forgot to add () before the curly brackets.
The ( ) around the assignment statement is required syntax when using the object literal destructuring assignment without a declaration. This is because the {} on the left hand side is considered a block and not an object literal. So here's how to do this the right way:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};
let name, country, job;

({name, country, job} = person);

console.log(name);//"Sarah"
console.log(job);//"Developer"
```

It is also important to note that when using this syntax, the () should be preceded by a semicolon. Otherwise it might be used to execute a function from the previous line.
Note that the variables in the object on the left hand side should have the same name as a property key in the object person. If the names are different, we'll get undefined:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};

let {name, friends, job} = person;

console.log(name);//"Sarah"
console.log(friends);//undefined
```

But if we want to use a new variable name, well, we can.

Using a new Variable Name

If we want to assign values of an object to a new variable instead of using the name of the property, we can do this:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};

let {name: foo, job: bar} = person;

console.log(foo);//"Sarah"
console.log(bar);//"Developer"
```

So the values extracted are passed to the new variables foo and bar.

Using Default Values

Default values can also be used in object destructuring, just in case a variable is undefined in an object it wants to extract data from:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};

let {name = "myName", friend = "Annie"} = person;

console.log(name);//"Sarah"
console.log(friend);//"Annie"
```

So if the value is not undefined, the variable stores the value extracted from the object as in the case of name. Otherwise, it used the default value as it did for friend.
We can also set default values when we assign values to a new variable:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer"};

let {name:foo = "myName", friend: bar = "Annie"} = person;

console.log(foo);//"Sarah"
console.log(bar);//"Annie"
```

So name was extracted from person and assigned to a different variable. friend, on the other hand, was undefined in person, so the new variable bar was assigned the default value.

Computed Property Name

Computed property name is another object literal feature that also works for destructuring. You can specify the name of a property via an expression if you put it in square brackets:

```
let prop = "name";

let {[prop] : foo} = {name: "Sarah", country: "Nigeria", job: "Developer"};

console.log(foo);//"Sarah"
```

Combining Arrays with Objects

Arrays can also be used with objects in object destructuring:

```
let person = {name: "Sarah", country: "Nigeria", friends: ["Annie", "Becky"]};

let {name:foo, friends: bar} = person;
```

```
console.log(foo);//"Sarah"
console.log(bar);//["Annie", "Becky"]
```

Nesting in Object Destructuring

Objects can also be nested when destructuring:

```
let person = {
   name: "Sarah",
   place: {
      country: "Nigeria",
      city: "Lagos" },
   friends : ["Annie", "Becky"]
};

let {name:foo,
    place: {
       country : bar,
       city : x}
   } = person;

console.log(foo);//"Sarah"
console.log(bar);//"Nigeria"
```

Rest in Object Destructuring

The rest syntax can also be used to pick up property keys that are not already picked up by the destructuring pattern. Those keys and their values are copied into a new object:

```
let person = {name: "Sarah", country: "Nigeria", job: "Developer" friends: ["Annie", "Becky"]};

let {name, friends, ...others} = person;

console.log(name);//"Sarah"
console.log(friends);//["Annie", "Becky"]
console.log(others);// {country: "Nigeria", job: "Developer"}
```

Here, the remaining properties whose keys where not part of the variable names listed were assigned to the variable others. The rest syntax here is ...others. others can be renamed to whatever variable you want.

One last thing – let's see how Object Destructing can be used in functions.

Object Destructuring and Functions

Object Destructuring can be used to assign parameters to functions:

```
function person({name: x, job: y} = {}) {
   console.log(x);
}

person({name: "Michelle"});//"Michelle"
person();//undefined
person(friend);//Error : friend is not defined
```

Notice the {} on the right hand side of the parameters object. It makes it possible for us to call the function without passing any arguments. That is why we got undefined. If we remove it, we'll get an error message.
We can also assign default values to the parameters:

```
function person({name: x = "Sarah", job: y = "Developer"} = {}) {
   console.log(x);
}

person({name});//"Sarah"
```