Destructuring is a way of extracting values into variables from data stored in objects and arrays.

OBJECT DESTRUCTURING

Let’s imagine we have an object like so:

```
const obj = {first: 'Asim', last: 'Hussain', age: 39 };
```

We want to extract the `first` and `last` properties into local variables, prior to ES6 we would have to write something like this:

```
const f = obj.first;
const l = obj.last;
console.log(f); // Asim
console.log(l); // Hussain
```

With destructing we can do so in one line, like so:

```
const {first: f, last: l} = obj;
console.log(f); // Asim
console.log(l); // Hussain
```

`{first: f, last: l}` describes a PATTERN, a set of rules for HOW we want to destructure an object.

Tip

```
const {first: f} = obj;
```

translates to extract the property first and store in a constant called f.

If we wanted to extract the properties into variables with the same name we would write it like so:

```
const {first: first, last: last} = obj;
console.log(first); // Asim
console.log(last); // Hussain
```

The above is quite a common use case for destructuring so it has a shortcut, like so

```
// {prop} is short for {prop: prop}
const {first, last} = obj;
console.log(first); // Asim
console.log(last); // Hussain
```

ARRAY DESTRUCTURING

Array destructuring works in a similar way except it extracts based of the index in the array, like so:

```
const arr = ['a', 'b'];
const [x, y] = arr;
console.log(x); // a
console.log(y); // b
```

FUNCTION PARAMETER DESTRUCTURING

One useful use case for destructuring is in function parameters.

Typically, if we want to pass multiple params to a function, with maybe some optional parameters, we would pass it in as an object like so:

```
function f(options) {
  console.log(options.x);
}
f({x:1}); // 1
```

Now we can define the function parameter list as an object destructure pattern, like so:

```
function f({x}) {
  console.log(x); // Refer to x directly
}
f({x:1});
```

Notice that in the function body above we can refer to `x` directly, we don't have to refer to it through an object property like `options.x`.

In addition to that when using destructured function parameters we can also provide default values, like so:

```
function f({x=0}) {
  console.log(x);
}
f({}); // 0
```

In the above example `x` now has a default value of 0 even if it's not passed into the function when called.

SUMMARY

Destructuring is a useful feature of ES6, with it we can extract values from objects and arrays with ease.

Through function parameter destructing we now have a built in syntax for providing optional parameters to functions, including giving them default values if none are provided