In ES6, you can extract data from arrays and objects into distinct variables using *destructuring*.

This probably sounds like something you’ve done before, for example, look at the two code snippets below that extract data using pre-ES6 techniques:

**const** point = [10, 25, -34];

**const** x = point[0];

**const** y = point[1];

**const** z = point[2];

console.log(x, y, z);

***Prints:****10 25 -34*

The example above shows extracting values from an array.

**const** gemstone = {

type: 'quartz',

color: 'rose',

carat: 21.29

};

**const** type = gemstone.type;

**const** color = gemstone.color;

**const** carat = gemstone.carat;

console.log(type, color, carat);

***Prints:****quartz rose 21.29*

And this example shows extracting values from an object.

Both are pretty straightforward, however, neither of these examples are actually using destructuring.

So what exactly is *destructuring*?

**Destructuring**

**Destructuring** borrows inspiration from languages like [Perl](https://en.wikipedia.org/wiki/Perl) and [Python](https://en.wikipedia.org/wiki/Python_%28programming_language%29) by allowing you to specify the elements you want to extract from an array or object *on the left side of an assignment*. It sounds a little weird, but you can actually achieve the same result as before, but with much less code; and it's still easy to understand.

Let’s take a look at both examples rewritten using destructuring.

**Destructuring values from an array**

**const** point = [10, 25, -34];

**const** [x, y, z] = point;

console.log(x, y, z);

***Prints:****10 25 -34*

In this example, the brackets [ ] represent the array being destructured and x, y, and z represent the variables where you want to store the values from the array. Notice how you don’t have to specify the indexes for where to extract the values from because the indexes are implied.

***TIP:****You can also ignore values when destructuring arrays. For example, const [x, , z] = point; ignores the y coordinate and discards it.*

**QUESTION 1 OF 2**

What do you expect to be the value of second after running the following code?

**let** positions = ['Gabrielle', 'Jarrod', 'Kate', 'Fernando', 'Mike', 'Walter'];

**let** [first, second, third] = positions;

* 

Kate

* 

Gabrielle

* Jarrod
* 

Walter

SUBMIT

**Destructuring values from an object**

**const** gemstone = {

type: 'quartz',

color: 'rose',

carat: 21.29

};

**const** {type, color, carat} = gemstone;

console.log(type, color, carat);

***Prints:****quartz rose 21.29*

In this example, the curly braces { } represent the object being destructured and type, color, and carat represent the variables where you want to store the properties from the object. Notice how you don’t have to specify the property from where to extract the values. Because gemstone has a property named type, the value is automatically stored in the type variable. Similarly, gemstone has a color property, so the value of color automatically gets stored in the color variable. And it's the same with carat.

***TIP:****You can also specify the values you want to select when destructuring an object. For example, let {color} = gemstone; will only select the color property from the gemstone object.*

**QUESTION 2 OF 2**

What do you expect to be returned from calling getArea()?

**const** circle = {

radius: 10,

color: 'orange',

getArea: **function**() {

**return** Math.PI \* **this**.radius \* **this**.radius;

},

getCircumference: **function**() {

**return** 2 \* Math.PI \* **this**.radius;

}

};

**let** {radius, getArea, getCircumference} = circle;

* 

314.1592653589793

* NaN