Tip 30: Simplify Key-Value Assignment

In this tip, you'll learn how to make objects quickly with shortened key-value assignment.

We'll cover the following

- Building objects
  - Adding properties in objects
  - Pulling out information
  - Recombining the object

You just learned how to pull apart objects in a clear and clean way. Now that you have all those pieces laying out on your proverbial work bench, you need to put them back together. It wouldn't be any good if the writers of the spec gave you a clean interface to take objects apart while leaving you no way to put them back together.

Building objects

Well, you're in luck. The same technique you'd use to take objects apart works in reverse. It's time to build new objects using similar syntax that will leave your code clear and predictable.

Start with a similar object of photo information:

```
const landscape = {
    title: 'Landscape',
    photographer: 'Nathan',
    location: [32.7122222, -103.1405556],
};
```

In this case, you have the `location` information in `latitude` and `longitude`, but what you need is the *city* and *state* names.

Elsewhere in the code, you have a helper function that looks up regional information (city, state, county) from the geographical coordinates. The implementation details aren't important. What matters here is that you get back another object of information.

```
const region = {
    city: 'Hobbs',
    county: 'Lea',
    state: {
        name: 'New Mexico',
        abbreviation: 'NM',
    },
};
```

Now you just need to take the `city` and `state` from the *return* object and assign it to the *new* object. Fortunately, adding information into objects is very simple.

Adding properties in objects

If you want to add a *key-value pair* to an object where the key is the *same* name as the variable, simply put in the variable. That's it. You don't need any extra colons.

You can also mix it up—have some key-value pairs defined with a variable and some defined the traditional way.

```
const landscape = {
    title: 'Landscape',
    photographer: 'Nathan',
    location: [32.7122222, -103.1405556],
};

function determineCityAndState([latitude, longitude]) {
    // Geo lookup would happen here
    const region = {
        city: 'Hobbs',
        county: 'Lea',
        state: {
            name: 'New Mexico',
            abbreviation: 'NM',
        },
    };
    return region;
}

function getCityAndState({ location }) {
    const { city, state } = determineCityAndState(location);
    return {
        city,
        state: state.abbreviation,
    };
}

console.log(getCityAndState(landscape));
```

In this case, you’re adding a key of `city` with *destructuring* assignment and a key of `state` with *normal* key-value assignment.

Pulling out information

What if you just want to sub out one piece of information in an object but keep everything else? For example, say you want to use `getCityAndState()` to translate the coordinates into *strings*, but you want to keep everything else from the original object.

You can combine the object spread operator with the regular key-value assignment to swap out one piece of information while retaining everything else.

```
const landscape = {
    title: 'Landscape',
    photographer: 'Nathan',
    location: [32.7122222, -103.1405556],
};

function determineCityAndState([latitude, longitude]) {
    // Geo lookup would happen here
    const region = {
        city: 'Hobbs',
        county: 'Lea',
        state: {
            name: 'New Mexico',
            abbreviation: 'NM',
        },
    };
    return region;
}

function setRegion({ location, ...details }) {
    const { city, state } = determineCityAndState(location);
    return {
        city,
        state: state.abbreviation,
        ...details,
    };
}

console.log(setRegion(landscape));
```

Don’t gloss over this code too quickly. There’s actually something interesting happening. When you use destructuring to pull out the `location` key-value pair, you’re also assigning everything else except `location` to the variable name `details`. You’re essentially *copying* the object and then running `delete photo.location`.

Recombining the object

When you recombine the object by spreading out details along with new key value pairs, you’re doing some subtle but powerful manipulation of objects to get exactly the information you want.

The result will have no `location`, but it will include all the original information along with the `city` and `state`.

```
{
    title: 'Landscape',
    photographer: 'Nathan',
    city: 'Hobbs',
    state: 'NM',
};
```

As you know, the spread operator is my favorite ES6 feature. But I know several developers who say destructuring is their favorite feature and that it’s changed the way they work with objects and functions. It will change the way you work, too.

Now that you have the tools you need to pull objects apart and put them back together, be sure to think twice before you create objects by assigning each key-value explicitly. If you’re going to assign a value to a variable, you might as well use the key name. Before long, destructuring will become second nature, and you’ll love how it transforms your code.

In the next tip, you’ll learn how to have a variable number of parameters with rest parameters using your favorite three-dot syntax.