The Major Problem with Using Objects Over Arrays

We have chosen to work with objects, but aren't arrays easier to iterate over? The Lodash library will solve this problem for us.

I love the idea of using objects over arrays - for most use cases. There are some caveats to be aware of though.

Caveat #1 : It's a lot easier to iterate over Arrays in your view logic.

A common situation you'll find yourself in is the need to render a list of components.

For example, to render a list of users given a users prop, your logic would look something like this:

```
const users = this.props.users;
users.map(user => {
  return <User />
})
```

Solution #1a: Use **Lodash** for Iterating over Objects

For the uninitiated, Lodash is a robust Javascript utility library. Even for iterating over arrays, many would argue that you still use Lodash as it helps deal with false values.

The syntax for using Lodash for iterating over objects isn't hard to grasp. It looks like this:

```
//import the library
import _ from "lodash"
//use it
_.map(users, (user) => {
    return <User />
})
```

You call the **map** method on the Lodash object, **_.map()**. You pass in the object to be iterated over, and then pass in a callback function like you would with the default javascript map function.

Solution #1b: Consider the usual way you'd map over an array to create a rendered list of users:

```
const users = this.props.users;
users.map(user => {
  return <User />
})
```

Now, assume that users was an object. This means we can't map over it. What if we could easily convert users to an array without much hassles?

Lodash to the rescue again. Here's what that would look like:

```
const users = this.props.users; //this is an object.
_.values(users).map(user => {
  return <User />
})
```

You see that?

_.values() will convert the object to an array. This makes map possible!

Here's how that works.

If you had a users object like this:

```
{
    user_id_1: {user_1_object},
    user_id_2 {user_2_object},
    user_id_3: {user_3_object},
    user_id_4: {user_4_object},
}
```

_.values(users) will convert that to this:

```
[
    {user_1_object},
    {user_2_object},
    {user_3_object},
    {user_4_object},
]
```

Yes! An array with the object values. Exactly what you need to iterate over

Yes! An array with the object values. Exactly what you need to iterate over. Problem solved.

There’s one more caveat. It’s perhaps a bigger one.