Specifying a default for a function component prop

We can provide default values for props where appropriate to simplify the consumption of a component. In this lesson, we'll learn how to do this.

In the CodeSandbox project we were working on in the last lesson, we are going to implement a default value of *"How are you?"* for the `message` prop. We are going to implement this in 2 different ways.

defaultProps approach

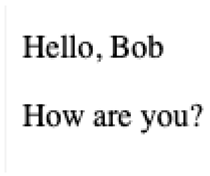
The first way is to use to use a static property called `defaultProps` on the function component. This is set to an object literal that contains the values to set the props to:

```
Component.defaultProps = {
  prop1: value1,
  prop2: value2,
  ...
}
```

In our CodeSandbox project, use the *defaultProps* approach to set the `message` prop to *"How are you?"* by default.

Show Answer

We'll now see *How are you?* rendered beneath the hello message:


Hello, Bob

How are you?

Nice!

Let’s look at the static `defaultProps` property a little closer. Let’s add a prop that doesn’t exist:

```
Hello.defaultProps = {
  message: "How are you?",
  someOtherProp: "no type error"
};
```

We’d hope a type error would be generated on `someOtherProp`, but there isn’t. This is a benefit of the `FC` type, it will raise a type error in this case.

Default function parameters approach

There is another way to provide default prop values to a function component that is arguably more readable. This approach is to use the standard syntax for default function parameters:

```
const Component = ({
  prop1 = value1,
  prop2 = value2,
  ...
}: Props) => { ... }
```

In our CodeSandbox project, refactor the `Hello` component to use the *default function parameters* approach to set the `message` prop to *“How are you?”* by default.

Show Answer

Wrap up

Well done! We can now create strongly typed props for function components

Well done! We can now create strongly-typed props for function components and specify default values where appropriate.

The *default function parameter* approach is arguably more readable than the *defaultProps* approach for defining default props. This is because the default value is closer to where the prop is defined. So, our eyes don’t need to scan right to the bottom of the component and back up again to fully understand a component’s props. It is also worth noting that `defaultProps` may be deprecated in React function components in a future release.

In the next lesson, we’ll learn how to create types for props that are objects. Keep your CodeSandbox project safe because we’ll continue to use it in the next lesson.