Introduction

Like most great inventions, there’s always a simple core idea that holds everything together. In React, it’s arguably the **props** system that allows us to treat React components just like JavaScript functions.

In this chapter, we are going to be looking at the React Props, including:

What are these props?

How to create a prop?

The different types of props.

What are the children props?

How to define the default props?

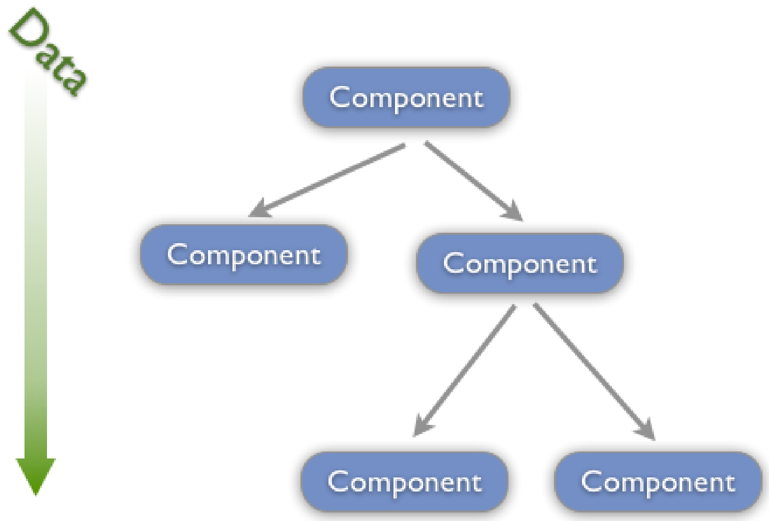
What are the prop types?

One-way data flow

Remember when we presented React's features? we spoke about the **one-way data flow**. What does it mean, you ask?

This approach means that the data can only travel, in a unidirectional fashion, from a parent to a child component (top to bottom) and not the other way around. Unlike other libraries and the frameworks, React makes sure that this feature remains in its architecture.

The reason behind React keeping this perk in its ecosystem is because the one-way data flow makes it easier to trace the data and it’s changes over the component tree, it sounds a little bit confusing but let’s view an illustration to further clarify the concept.



What are props?

To make this unidirectional data flow possible, React uses **props**.

Props or **properties** are the tools that React uses to pass data from parent to child elements.

Let’s think of it as a parameter in a JavaScript function.

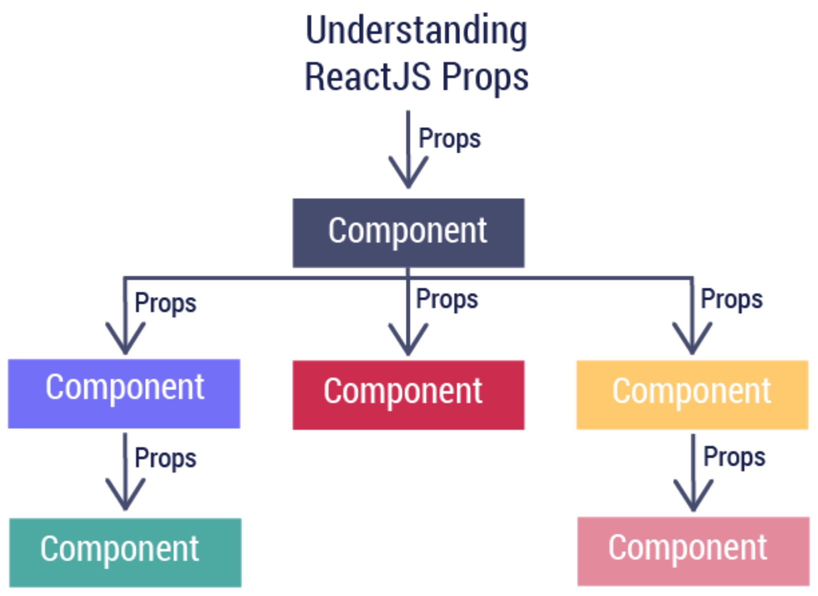
We can change the value of this parameter but the sets of instructions to be executed remain the same.

Take your facebook page for example. You will find your name on the right, next to your picture.

From a coding perspective, facebook developers can't create a customized navbarfor every profile.

This is only possible and achievable using React props.

The structure of these elements will be fixed and the content (name, image url...) will be received as a props.



JS example

Here is another example to consolidate what we were just talking about.

```
function Greeting(name) {

return "hello " + name;

}

console.log(Greeting('Jon Snow'))// it will display "hello Jon Snow"

console.log(Greeting('Ramsay Bolton')) // it will display "hello Ramzay Bolton"

console.log(Greeting('I am nobody')) // it will display "hello I am nobody"
```

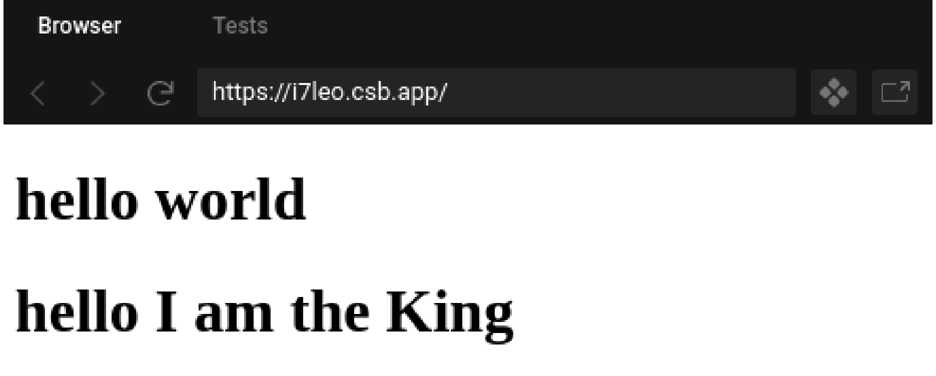
The Greeting function receives a name as an argument. However, with whichever name we set, it will still display a concatenated `hello` alongside the name.

React example

Now let’s try the same idea with a React component.

```
function Greeter(props) {

 return <h1>hello {props.name}</h1>;

}

// And invoking the <Greeter/> component…

const App = () => {

 return (

   <div>

     <Greeter name="world" /> {/* 💥 "world" is the prop value*/}

     <Greeter name="I am the King" /> {/* 💥 "I am the King" is the prop value*/}

   </div>

 );

};

export default App;
```

Output



hello world

hello I am the King

Let's recap!

React Props are like function arguments in JavaScript and attributes in HTML. To send props into a component, use the same syntax as HTML attributes:

`<Greeter name="world" />`

```
function Greeter(props) {
 return <h1>hello {props.name}</h1>;
}
```

As we can see, the component Greeter receives the name as a props and displays it in an h1 tag.

Another thing to keep in mind is that props are an object and whatever we pass as attributes in the component is called in the parent.

Notice : Props are immutable which mean that the child can only read the content of a props. It cannot change it.

‹ Previous next ›