## Creating our first prop

Similar to an HTML attribute, in order to send a props from a parent to a child component, we add the props as an attribute in the component call process. Here is an example to better understand this.

```
<Welcome name="Sara" />;
```

A component can have an unlimited number of props, meaning that we can pass as many props as we wish. Take a look at this example :

```
<Welcome name= 'Sara' lastName = 'Smith' Age='27'/>
```

Please consider that the names we give to props are personalized.

Even though we can call the props whatever we want, there are some best practices to follow:

The name should be significant and meaningful so others can understand the props' value.

Use lower camel case (e.g. isActive).

keep the name short ( < 50 characters ).

Props should be descriptive. Props describe what a component does and not why it does it. A common mistake is naming props after the current user or the current environment. For example : hasSubmitPermission describes the permission of the user and the reason for variation but not the component itself. A better name could be hasSubmitButton. i.e.`<MyForm hasSubmitButton= {user.canSubmit} />`

## Using props

It's time to use our first props after creating it!

First thing to do when using props is to pass it as a parameter to the component's function.

```
// we always use the keyword props in the function parameter
const Welcome = props =>{
 return (
   <h1>
```

```
      `Hello`
    </h1>
  )
}
```

We can use any parameter name but by convention we use the keyword **props**.

## The props object

If we perform a console.log() on the received props, we'll find the following result:

---

App.js

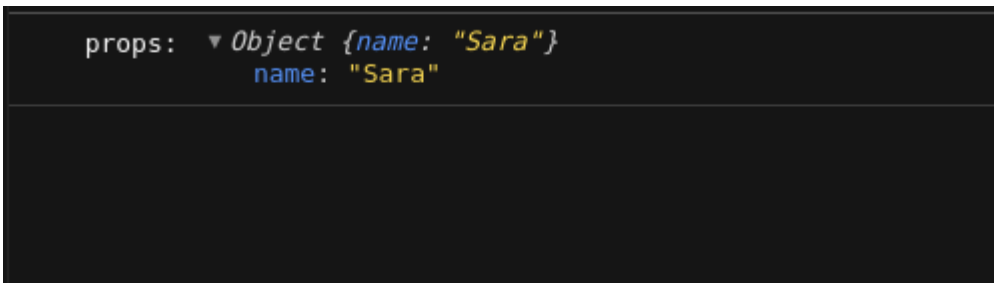
```
const App = () => (
 <div>
   <Welcome name="Sara" />
 </div>
);
```

Welcome.js

```
const Welcome = props =>{
console.log(`props:`,props)
 return (
   <h1> welcome </h1>
 )
}
```

---

Output



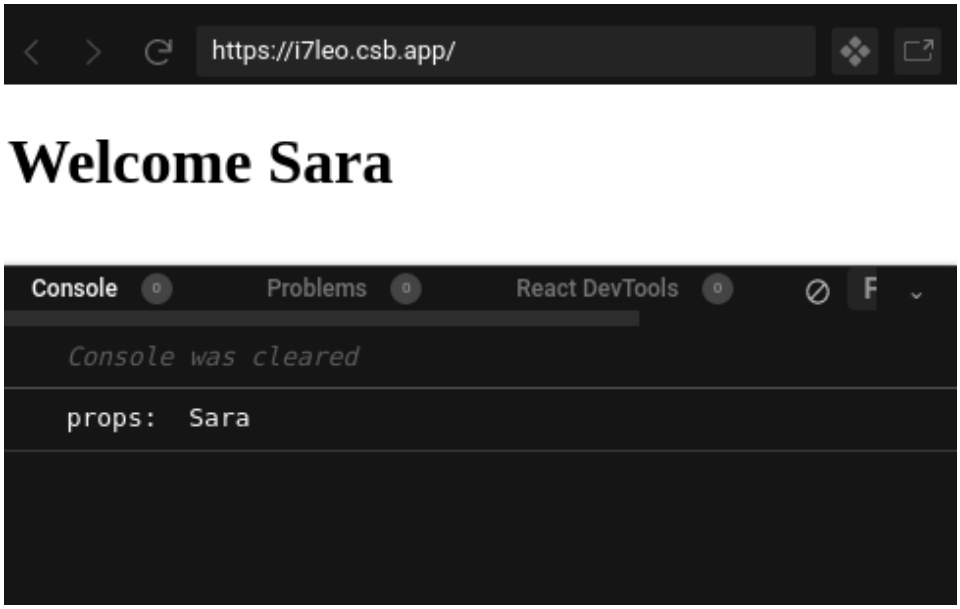
As we can see in the example above, even if the props were passed as a string, it still had arrived as an object to the child component.

## the props object

In the previous slide, we saw that props always arrive as an object to child components.

In order to use it, we need to treat it as an object. We can access the desired value by using the dot property accessor.

```
const Welcome = props =>{
console.log(`props:`,props)
 return (
   <h1> welcome {props.name}</h1>
 )
}
```



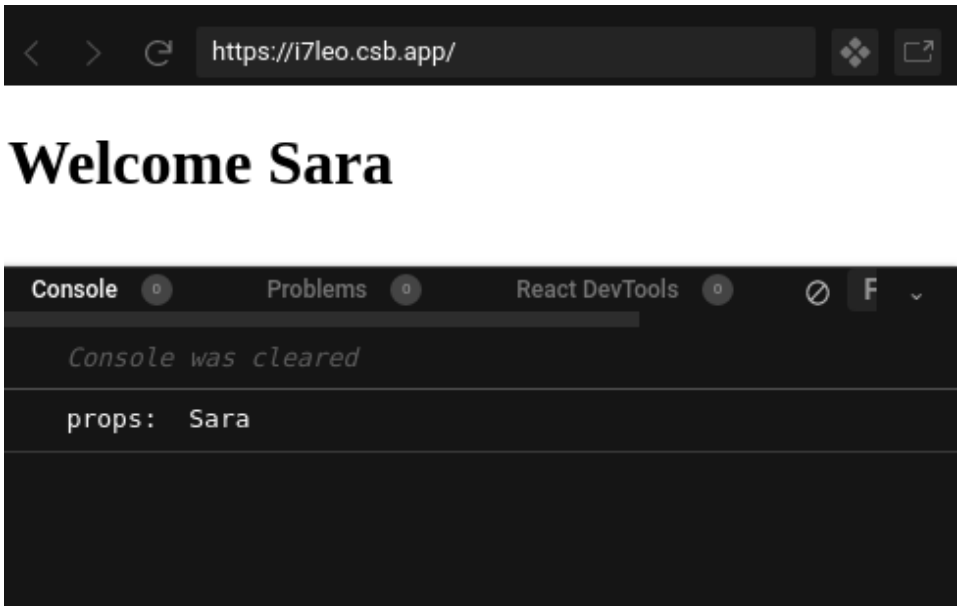

## Rules of Props

We've said earlier that props are **immutable** but what if we try to change its value?

Take a look at the example below :

```
const Welcome = props =>{
console.log(`props:`,props.name)
props.name ='John'// try to change the value into 'John'
 return (
   <h1> Welcome {props.name}</h1>
```

```
  )
}
```




We'll have the same result as before that's because the props are read-only.

## Functions as props

We know that props can hold any JavaScript data type, but what about functions?

Of course, we can pass a function as props from parent to child. Actually, this trick is used to pass data from child to parent. Don't panic, we know that React is a one way data flow, but in real life application, we are obliged some time to get data from the child to the parent.

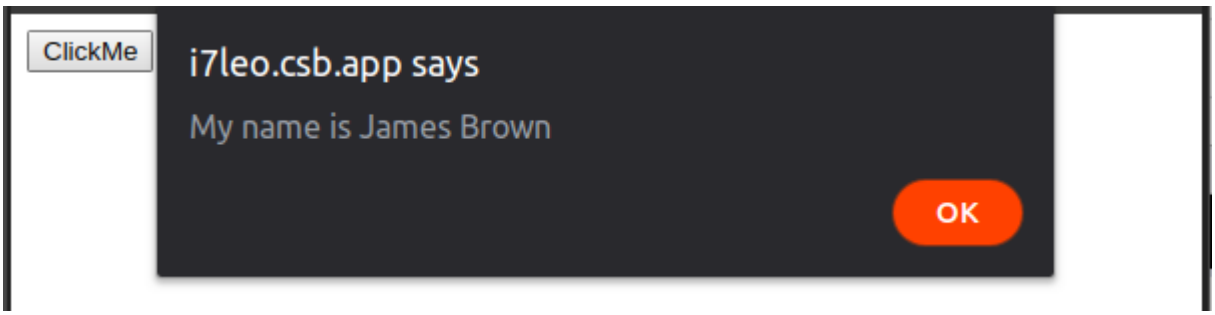
To make it possible, we take advantage of the function. Here is an example of what we're talking about.

App.js

```
const App = () => {
 const alertMyInput = name => alert(name);
 return (
   <div>
     <Welcome name="Sara" alertMyInput={alertMyInput} />
   </div>
 );
};
```

Welcome.js

```
const Welcome = props => {
 console.log(`props:`, props.name);
 return (
   <button onClick={() => props.alertMyInput(`My name is James Brown `)}>
     ClickMe
   </button>
 );
};
```




## Conclusion

We know it's a lot of material to take in, but the key points we should remember are:

- Props are arguments that are passed into React components.
- Props are used to ensure communication inside the component tree.
- Props are immutable as their value cannot be changed by the child component.
- Props are only passed from parent to child.

Previous next