Version: 3.x

Your First Animation

In this section, we'll guide you through the basic concepts of Reanimated. If you're new to Reanimated, you're in the right hands! We're going to start by building a simple animation which will help you develop a basic understanding of the library. Then, in the following sections, we're going to build on top of this knowledge and further expand your skills. Let's go!

Using an Animated component

Let's start by having something that we could see on the screen. First, to create an animatable component you need to import an `Animated` object:

```
import Animated from 'react-native-reanimated';
```

This `Animated` object wraps React Native built-ins such as `View`, `ScrollView` or `FlatList`.

You use these components as any other JSX components:

```
import Animated from 'react-native-reanimated';

export default function App() {
  return (
    <Animated.View
      style={{
        width: 100,
        height: 100,
        backgroundColor: 'violet',
      }}
    />
  );
}
```

INFO

> You can create your own custom Animated components with `createAnimatedComponent`.

Defining a shared value

A **shared value** is a driving factor of all your animations. You can think of it as a React state which is automagically kept in sync between the "JavaScript" and the "native" side of your app (hence the name). You create shared values using a `useSharedValue` hook:

```
import { useSharedValue } from 'react-native-reanimated';
```

As with any other **React hook**, you need to define it in your component's body. In a shared value, you can store any JS value like `number`, `string` or `boolean` but also data structures such as `array` and `object`.

For now, let's use `100` as the default value of the `useSharedValue` hook and pass the returned value as an inline style of the `Animated.View`:

```
import Animated, { useSharedValue } from 'react-native-reanimated';

export default function App() {
  const width = useSharedValue(100);

  return (
    <Animated.View
      style={{
        width,
        height: 100,
        backgroundColor: 'violet',
      }}
    />
  );
}
```

Using a shared value

Let's create a very simple animation that will animate a `width` of an element. We'll make it expand by `50px` on each button press. We can do this by modifying a shared value connected to the `width` property of an `Animated.View` component. I know it might sound complicated, but it's actually quite simple.

Values stored in shared values are accessed and modified by their `.value` property.

There's no setter or anything - you simply mutate the `.value` property like there's no tomorrow.

Let's define a `handlePress` function inside of which we'll modify the shared value:

```
import { Button, View } from 'react-native';
import Animated, { useSharedValue } from 'react-native-reanimated';

export default function App() {
  const width = useSharedValue(100);

  const handlePress = () => {
    width.value = width.value + 50;
  };

  return (
    <View style={{ flex: 1, alignItems: 'center' }}>
      <Animated.View
        style={{
          width,
          height: 100,
          backgroundColor: 'violet',
        }}
      />
      <Button onPress={handlePress} title="Click me" />
    </View>
  );
}
```

Please hold on a second before you shorten `width.value = width.value + 50` to `width.value += 50`. We're preparing this code for the final step which will finally bring our animation to life!

> ⓘ CAUTION

> It's a common mistake to modify a shared value directly like this: ~~`sv = sv + 100;`~~. Always remember to access the shared value by using the `.value` property instead. Here, the correct usage would be `sv.value = sv.value + 100;`.

Using an animation function

Finally, import `withSpring` function and wrap around `width.value + 50` in the `handlePress` function so that the value which `withSpring` returns modifies the shared value. This will create a bouncy spring animation that transitions the width of the element from its current value (here `width.value`) to the new one (here `width.value + 50`).

```
import { Button, View } from 'react-native';
import Animated, { useSharedValue, withSpring } from 'react-native-reanimated';

export default function App() {
  const width = useSharedValue(100);

  const handlePress = () => {
    width.value = withSpring(width.value + 50);
  };

  return (
    <View style={{ flex: 1, alignItems: 'center' }}>
      <Animated.View
        style={{
          width,
          height: 100,
          backgroundColor: 'violet',
        }}
      />
      <Button onPress={handlePress} title="Click me" />
    </View>
  );
}
```

And voilà, we've made our first animation using Reanimated! You can see how it works in its full glory in a preview below:

Preview Code

Summary

In this section, we gained a firm grasp on the Reanimated fundamentals. We learned about `Animated` components, shared values and how to use them to create a simple animation. To sum up:

- `Animated` components are used to define animatable elements.
- Shared values are a driving factor of all animations and we define them using a `useSharedValue` hook.
- Shared values are always accessed and modified by their `.value` property (eg. `sv.value = 100;`).
- To create smooth animations modify shared values using animation functions like `withTiming`

What's next?

In the next section, we will learn more about how to animate styles and props using `useAnimatedStyle` and `useAnimatedProps` hooks.

Edit this page