Flutter to React Native: Quick Comparison Guide

Language Comparison

Feature	Flutter	React Native
Primary Language	Dart	JavaScript / TypeScript
Typing System	Strongly typed	Dynamically typed (TypeScript adds types)
Main Paradigm	Object-oriented	Functional programming

Basic Components

Flutter	React Native
`Container()`	`<View>`
`Text('Hello')`	`<Text>Hello</Text>`
`Row()`	`<View style={{flexDirection: 'row'}}>`
`Column()`	`<View style={{flexDirection: 'column'}}>`
`Image.network()`	`<Image source={{uri: 'url'}} />`
`TextField()`	`<TextInput />`
`ElevatedButton()`	`<TouchableOpacity>` or `<Pressable>`
`ListView.builder()`	`<FlatList>`
`SingleChildScrollView()`	`<ScrollView>`

State Management

Flutter (StatefulWidget)

```dart
class Counter extends StatefulWidget {
  @override
  _CounterState createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int count = 0;

  void increment() {
    setState(() {count++;});
  }

  @override
  Widget build(BuildContext context) {
    return Text('$count');
  }
}
```

React Native (useState hook)

```
function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => setCount(count + 1);

  return <Text>{count}</Text>;
}
```

Props / Parameters

Flutter

```dart
class Greeting extends StatelessWidget {
  final String name;
  final int age;

  const Greeting({
    required this.name,
    required this.age,
  });

  @override
  Widget build(BuildContext context) {
    return Text('Hello $name, age $age');
  }
}

// Usage
Greeting(name: 'Alice', age: 25)
```

React Native

```
function Greeting({ name, age }) {
  return <Text>Hello {name}, age {age}</Text>;
}

// Usage
<Greeting name="Alice" age={25} />
```

Styling

Flutter

```
Container(
  padding: EdgeInsets.all(20),
  margin: EdgeInsets.only(top: 10),
  decoration: BoxDecoration(
    color: Colors.blue,
    borderRadius: BorderRadius.circular(10),
  ),
  child: Text(
    'Hello',
    style: TextStyle(
      fontSize: 20,
      fontWeight: FontWeight.bold,
      color: Colors.white,
    ),
  ),
)
```

React Native

```
const styles = StyleSheet.create({
  container: {
    padding: 20,
    marginTop: 10,
    backgroundColor: 'blue',
    borderRadius: 10,
  },
  text: {
    fontSize: 20,
    fontWeight: 'bold',
    color: 'white',
  }
});

<View style={styles.container}>
  <Text style={styles.text}>Hello</Text>
</View>
```

Lifecycle Methods

Flutter

```dart
class MyWidget extends StatefulWidget {
  @override
  _MyWidgetState createState() => _MyWidgetState();
}

class _MyWidgetState extends State<MyWidget> {
  @override
  void initState() {
    super.initState();
    // Component mounted
  }

  @override
  void dispose() {
    // Component will unmount
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

React Native

```
function MyComponent() {
  useEffect(() => {
    // Component mounted (1)
    return () => {
      // Component will unmount (2)
    };
  }, []);

  return <View />;
}
```

Lists

Flutter

```
ListView.builder(
  itemCount: items.length,
  itemBuilder: (context, index) {
    final item = items[index];
    return ListTile(
      title: Text(item.name),
    );
  },
)
```

React Native

```
<FlatList
  data={items}
  keyExtractor={(item) => item.id}
  renderItem={({ item }) => (
    <Text>{item.name}</Text>
  )}
/>
```

Conditional Rendering

Flutter

```
Widget build(BuildContext context) {
  return Column(
    children: [
      if (isLoading)
        CircularProgressIndicator()
      else
        Text('Loaded!'),
    ],
  );
}
```

React Native

```
function MyComponent() {
  return (
    <View>
      {isLoading ? (
        <ActivityIndicator />
      ) : (
        <Text>Loaded!</Text>
      )}
    </View>
  );
}
```

Navigation

Flutter

```dart
// Navigate to new screen
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => DetailScreen()),
);

// Go back
Navigator.pop(context);
```

React Native (React Navigation)

```
// Navigate to new screen
navigation.navigate('Details');

// Go back
navigation.goBack();
```

Event Handling

Flutter

```dart
ElevatedButton(
  onPressed: () {
    print('Button pressed');
  },
  child: Text('Press me'),
)

GestureDetector(
  onTap: () {
    print('Tapped');
  },
  child: Container(),
)
```

React Native

```
<TouchableOpacity
  onPress={() => {
    console.log('Button pressed');
  }}
>
  <Text>Press me</Text>
</TouchableOpacity>

<Pressable onPress={() => console.log('Tapped')}>
  <View />
</Pressable>
```

Layout: Flexbox

Flutter

```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Text('Item 1'),
    Text('Item 2'),
  ],
)
```

React Native

```
<View style={{
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
}}>
  <Text>Item 1</Text>
  <Text>Item 2</Text>
</View>
```

Async Operations

Flutter

```
Future<void> fetchData() async {
  final response = await http.get(Uri.parse(url));
  final data = jsonDecode(response.body);
  setState(() {
    this.data = data;
  });
}

@override
void initState() {
  super.initState();
  fetchData();
}
```

React Native

```
const fetchData = async () => {
  const response = await fetch(url);
  const data = await response.json();
  setData(data);
};

useEffect(() => {
  fetchData();
}, []);
```

Key Differences

1. **All text must be in `<Text>` components** in React Native
   - Flutter: Can put strings anywhere
   - React Native: Must wrap in `<Text>`

```
// ❌ Error!
<View>Hello World</View>

// ✅ Use this
<View>
  <Text>Hello World</Text>
</View>
```

This is one of the most mistakes frequent Flutter programmers make when they use React Natvie.

2. **Styling approach**
    - Flutter: Widget properties (type-safe)
    - React Native: JavaScript objects (more flexible)

```
Container(
  padding: EdgeInsets.all(20),           // Type: EdgeInsets
  margin: EdgeInsets.only(top: 10),      // Type: EdgeInsets
  decoration: BoxDecoration(              // Type: BoxDecoration
    color: Colors.blue,                   // Type: Color
    borderRadius: BorderRadius.circular(10),  // Type: BorderRadius
  ),
```

```
const styles = StyleSheet.create({
  container: {
    padding: 20,                 // Just a number
    marginTop: 10,               // Just a number
    backgroundColor: 'blue',      // String (any CSS color)
    borderRadius: 10,            // Just a number
  },
```

3. **Component syntax**
    - Flutter: Class-based with build method
    - React Native: Function-based with JSX

```
class Greeting extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text('Hello World');
  }
}
Greeting()
```

```
function Greeting() {
  return <Text>Hello World</Text>;
}
<Greeting/>
```

4. **State management**
   - Flutter: `setState()` in StatefulWidget
   - React Native: `useState()` hook
5. **Imports**
   - Flutter: `import 'package:flutter/material.dart';`
   - React Native: `import { View, Text } from 'react-native';`

6. **Platform differences**
    - Both support platform-specific code
    - Flutter uses conditional imports
    - React Native uses `Platform.OS`
7. **Hot reload**
    - Both support hot reload/fast refresh
    - Similar developer experience

Mental Model Shift

When moving from Flutter to React Native:

1. Think **functions** instead of classes
2. Think **JavaScript objects** instead of constructor parameters
3. Think **hooks** (useState, useEffect) instead of lifecycle methods
4. Remember: `<View>` for layout, `<Text>` for text
5. Styling is **separate** from components (usually)
6. **Flexbox by default** - similar to Column/Row