國立高雄科技大學 智慧商務系
National Kaohsiung University of Science and Technology Department of Intelligent Commerce

7. 版面設計

廖奕雯

yiwen923@nkust.edu.tw

大綱

- 樣式 Style
- 高度和寬度 Height and width
- Flexbox

樣式 Style

- 在 React Native 中，你並不需要學習什麼特殊的語法來定義樣式。我們仍然是使用 JavaScript 來寫樣式。所有的核心組件都接受名為style的屬性。
- 這些樣式名基本上是遵循了 web 上的 CSS 的命名，只是按照 JS 的語法要求使用了駝峰命名法，例如將background-color改為backgroundColor。
- style屬性可以是一個普通的 JavaScript Object。這是最簡單的用法，因而在代碼中很常見。你還可以傳入一個陣列，在陣列中位置居後的樣式物件比居前的優先順序更高，這樣你可以間接實現樣式的繼承。
- 實際開發中元件的樣式會越來越複雜，建議使用StyleSheet.create來集中定義元件的樣式。

三種設定 Style方法

- 方法一: 使用 StyleSheet.create 來集中定義元件的樣式
- 方法二: inline style
- 方法三:銜接樣式

使用StyleSheet.create

1. import { StyleSheet } from 'react-native';
2. 建立 styles object

```
const styles = StyleSheet.create({
  container: {
    marginTop: 50,
  },
  bigBlue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
});
```

建立 styles object;
固定語法。

Style的名字

Style的內容;
把CSS 的名稱改成駝峰命名法。

3. 在 component 中使用

```
<Text style={styles.bigBlue}>just bigBlue</Text>
<Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>
```

使用{ styles.名稱} 來套用 style

使用 style

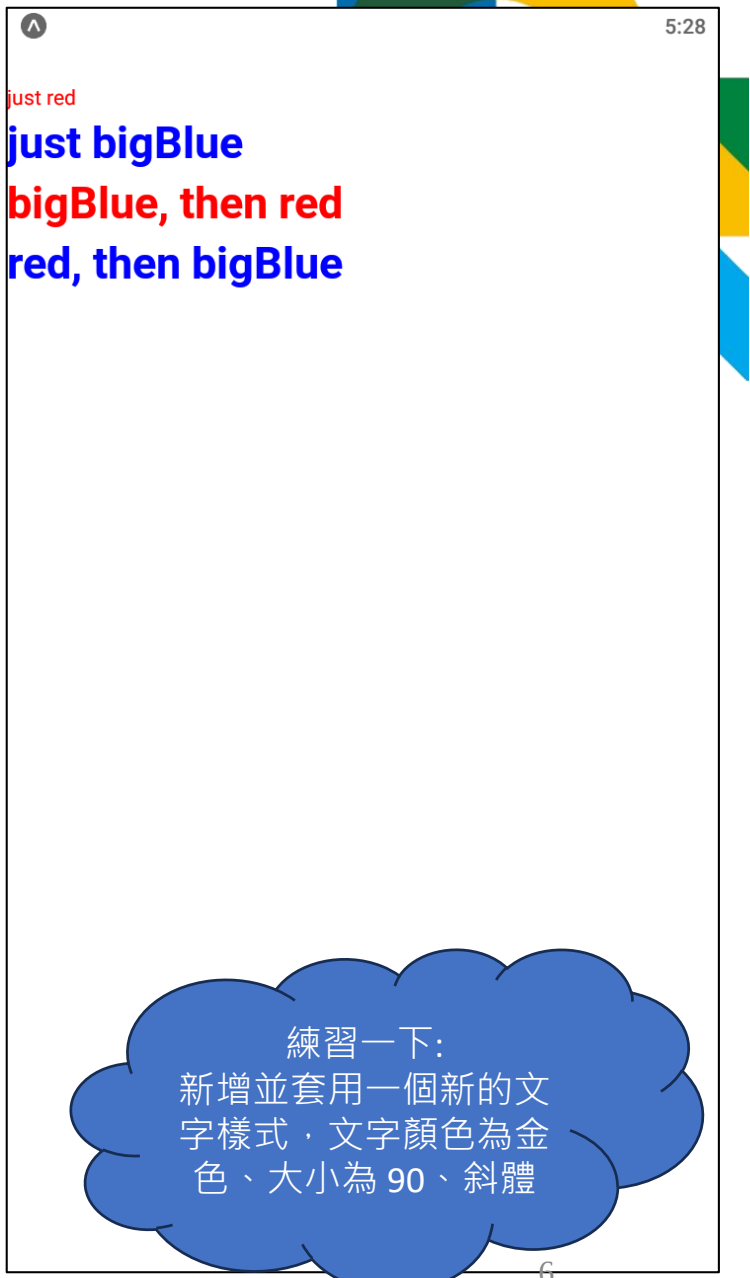
使用陣列來套用多個 styles

File: App-style.js

```
01: import React from 'react';
02: import { StyleSheet, Text, View } from 'react-native';
03:
04: export default function App() {
05:     return (
06:       <View style={styles.container}>
07:         <Text style={styles.red}>just red</Text>
08:         <Text style={styles.bigBlue}>just bigBlue</Text>
09:         <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>
10:         <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>
11:       </View>
12:     );
13: };
14:
15: const styles = StyleSheet.create({
16:   container: {
17:     marginTop: 50,
18:   },
19:   bigBlue: {
20:     color: 'blue',
21:     fontWeight: 'bold',
22:     fontSize: 30,
23:   },
24:   red: {
25:     color: 'red',
26:   },
27: });
28:
```

② 使用樣式

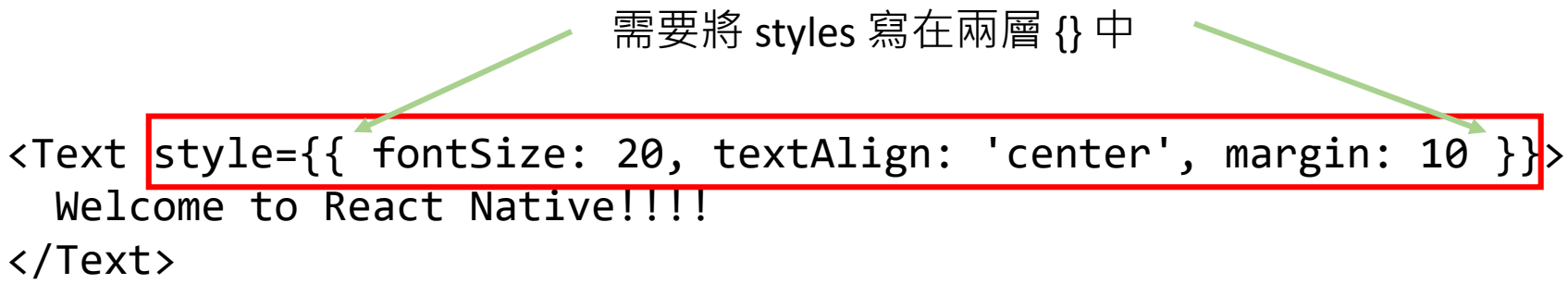
① 建立樣式



練習一下:
新增並套用一個新的文字樣式，文字顏色為金色、大小為 90、斜體

inline style

- 直接將 style 寫在 style props 裡面




```
<Text style={{ fontSize: 20, textAlign: 'center', margin: 10 }}>
  Welcome to React Native!!!!
</Text>
```

inline style

練習一下:
新增並套用一個新的文字樣式，文字顏色為金色、大小為 90、斜體

File: App-style.js

```
04: export default function App() {
05:     return (
06:       <View style={styles.container}>
07:         <Text style={styles.red}>just red</Text>
08:         <Text style={styles.bigBlue}>just bigBlue</Text>
09:         <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>
10:         <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>
11:
12:         <Text style={{ fontSize: 20, textAlign: 'center', margin: 10 }}>
13:           Welcome to React Native!!!!
14:         </Text>
15:       </View>
16:     );
17: };
```

使用 stylesheet

使用 inline style

銜接樣式

- style 除了傳入一個 object 外，也能使用陣列，後面那個 style 會覆蓋前面的。

使用 array，可以同時使用多個 styles;
可以使用 stylesheet 或 inline

```
<Text style={[ styles.welcome, { color: 'green' } ]}>
          Welcome to React Native!!!!
</Text>
```

File: App-style.js

```
04: export default function App() {
05:     return (
06:       <View style={styles.container}>
11:
12:         <Text style={{ fontSize: 20, textAlign: 'center', margin: 10 }}>
13:           Welcome to React Native!!!!
14:         </Text>
15:
16:         <Text style={[ styles.welcome, { color: 'green' } ]}>
17:           Welcome to React Native!!!!
18:         </Text>
19:       </View>
20:     );
21: };
22:
23: const styles = StyleSheet.create({
24:   container: {
25:     marginTop: 50,
26:   },
32:   welcome: {
33:     fontSize: 20,
34:     textAlign: 'center',
35:     margin: 10,
36:     color: 'blue'
37:   },
38:   red: {
39:     color: 'red',
40:   },
41: });
```

後面的 styles 會覆蓋掉前面的

練習一下:
新增並套用一個新的文字樣式，文字顏色為金色、大小為 90、斜體

高度和寬度 Height and width

- Component 的高度和寬度決定了其在螢幕上顯示的尺寸。

- React Native 使用 density-independent pixels (dp, dip)做為顯示的單位。

高度和寬度 Height and width

```
File: App.js
20: <View style={{width: 50, height: 50, backgroundColor: 'powderblue' }} />
21: <View style={{width: 100, height: 100, backgroundColor: 'skyblue' }} />
22: <View style={{width: 150, height: 150, backgroundColor: 'steelblue' }} />
```

單位不用寫，單位是 dp

px, PPI, DPI

- px: pixel 像素
- PPI: Pixels Per Inch
  - PPI是一種解析度的指標，常用來作為圖像的解析度指標。
- DPI: Dots Per Inch
  - DPI是一種解析度的指標，常用在印表機、掃描器、滑鼠等物理設備解析度指標上。

- DPI和PPI從定義上來看，一個指"點"，一個指"像素"。兩個的區別，僅僅是使用場合的不同，比如表示印表機的解析度就用DPI，表示圖像的解析度就用PPI。

density-independent pixels (dp, dip)

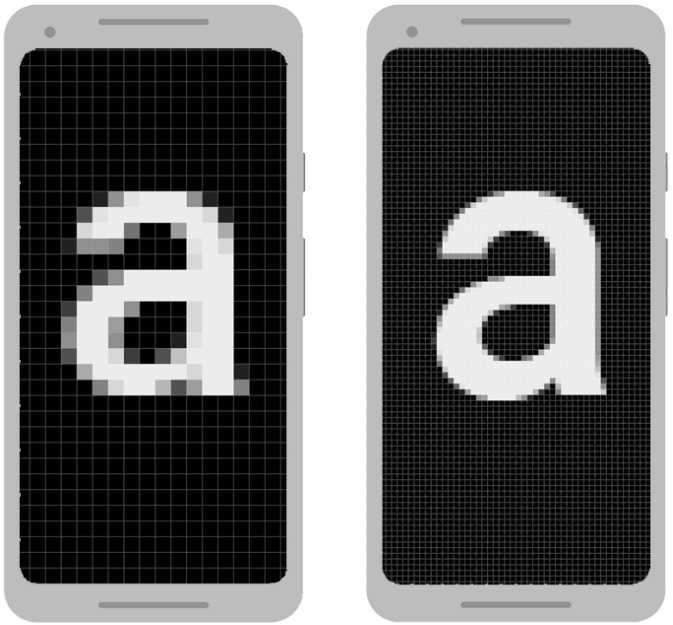
- Android 設備不僅有不同的螢幕尺寸（手機、平板電腦、電視等），而且其螢幕也有不同的像素(pixels)尺寸。
- 有可能一部設備的螢幕為每平方英寸 160像素，而另一部設備的螢幕在相同的空間內可以容納 480像素。
- 如果不考慮像素密度的這些差異，系統可能會縮放圖片導致圖片變模糊，或者圖片可能會以完全錯誤的尺寸顯示。

尺寸相同的兩個螢幕可能具有不同數量的pixels

density-independent pixels (dp, dip)

- 要在密度不同的螢幕上保留相同的UI可見範圍，您必須使用density-independent pixels (dp, dip) 作為度量單位來設計UI。
- dp 是一個虛擬pixel單位，1 dp 約等於中密度螢幕（160dpi；"基準"密度）上的 1 pixel。對於其他每個密度，Android 會將此值轉換為相應的實際pixel數。

例如，考慮圖中的兩部設備。如果將某個視圖定義為 "100px" 寬，那麼它在左側設備上看起來要大得多。因此，您必須改用 "100dp" 來確保它在兩個螢幕上看起來大小相同。



將 dp 單位轉換為pixel單位

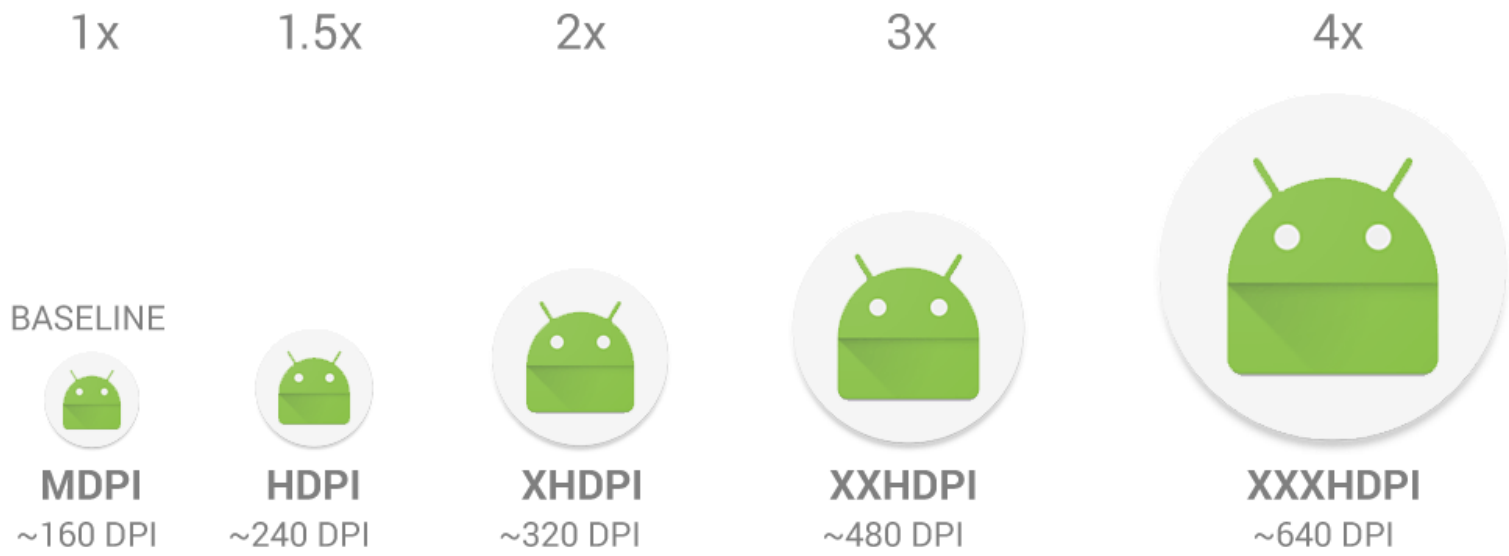
- 一般情況下，您需要以 dp 表示 UI 尺寸，並且將其轉換為pixels。
- dp 單位轉換為 pixels 公式:

$$px = dp \times \left(\frac{dpi}{160}\right)$$

- 例如: 手機 dpi = 480 ，dp=100 => px= 300

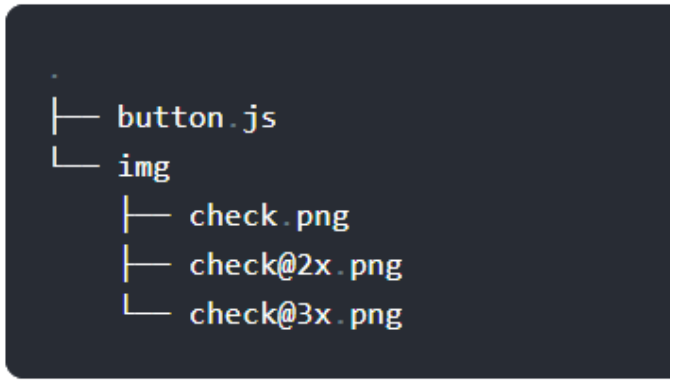
如何為不同的螢幕提供不同的圖片

- 根據 dp 和螢幕精細度，一般圖片的大小比例如圖



如何為不同的螢幕提供不同的圖片

- 假設 Button 有一張圖片為 check.png，可以將不同的圖案按照尺寸大小加上 @2 @3等後綴詞，來為不同的螢幕提供不同解析度的圖片。目錄結構如下:



- 並且button.js裡有這樣的代碼：

```
<Image source={require('./img/check.png')} />
```

- Packager 會打包所有的圖片並且依據螢幕精度提供對應的資源。
- 如果沒有圖片恰好滿足螢幕解析度，則會自動選中最接近的一個圖片。

注意

- 假設在某一應用中，用戶的手指至少移動 16 pixels之後，系統才會識別出滾動或滑動手勢。在中密度螢幕上，使用者必須移動 16 pixels / 160 dpi（等於一英寸的 1/10 或 2.5 毫米），系統才會識別該手勢。而在配備高密度顯示幕 (240dpi) 的設備上，使用者的手指必須至少移動 16 pixels / 240 dpi，相當於 1 英寸的 1/15（1.7 毫米）。此距離短得多，因此用戶會感覺應用在該設備上更靈敏。

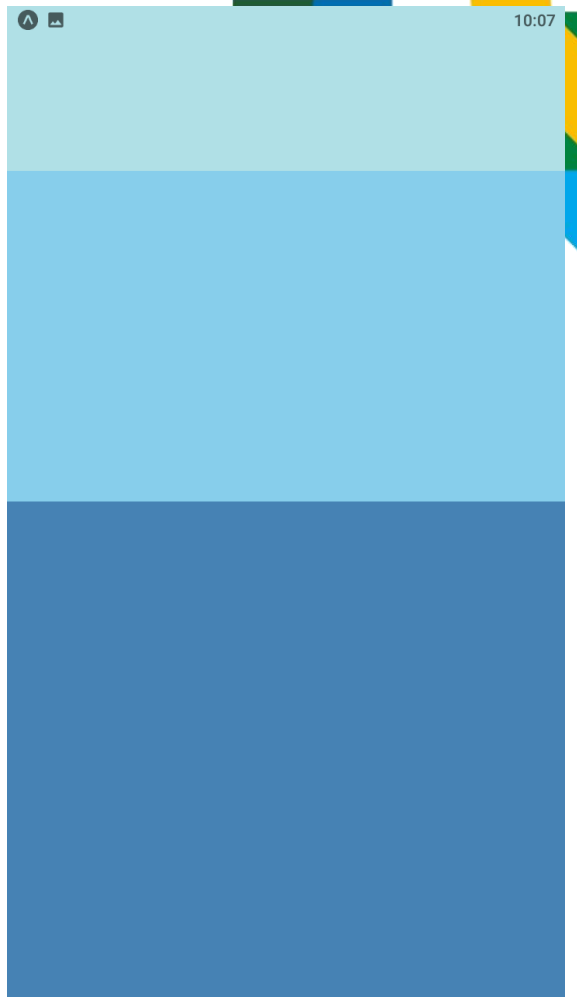
- 要解決此問題，必須在代碼中以 dp 表示手勢thread，然後再轉換為實際 pixel。

Flex Dimensions

- 在component style 中使用flex可以使其在可利用的空間中動態地擴張或收縮。
- 一般而言我們會使用 flex:1 來指定某個component擴張以填滿所有剩餘的空間。
- 如果有多個並列的子元件使用flex
  - flex:1，則這些子元件會平分父容器中剩餘的空間。
  - Flex 值不一樣，則依照比例分配。
- Component 能夠填滿剩餘空間的前提是其 parent component的尺寸不為零。如果parent component 沒有固定的width和height，也沒有設定flex，則父容器的尺寸為零。其子元件如果使用了flex，也是無法顯示的。

Flex 比例不同

```
File: App-flex-dimensions.js
01: import React from 'react';
02: import { View } from 'react-native';
03:
04: export default function App() {
05:   return (
06:     // 練習，調整 parent component size
07:     <View style={{flex: 1}}>
08:       <View style={{flex: 1, backgroundColor: 'powderblue'}} />
09:       <View style={{flex: 2, backgroundColor: 'skyblue'}} />
10:       <View style={{flex: 3, backgroundColor: 'steelblue'}} />
11:     </View>
12:   );
13: };
14:
```



練習:

1. 調整 flex 值
2. 不使用 flex ，改用 dp

Flex 方向

來源自 CSS flex-direction

- flexDirection: "row"|"row-reverse"|"column"|"column-reverse"
- 用來標註 flex 的方向
  - row 水平
  - row-reverse 水平反序
  - column 垂直
  - column-reverse 垂直反序

練習:

1. 使用四種方向 property 的效果

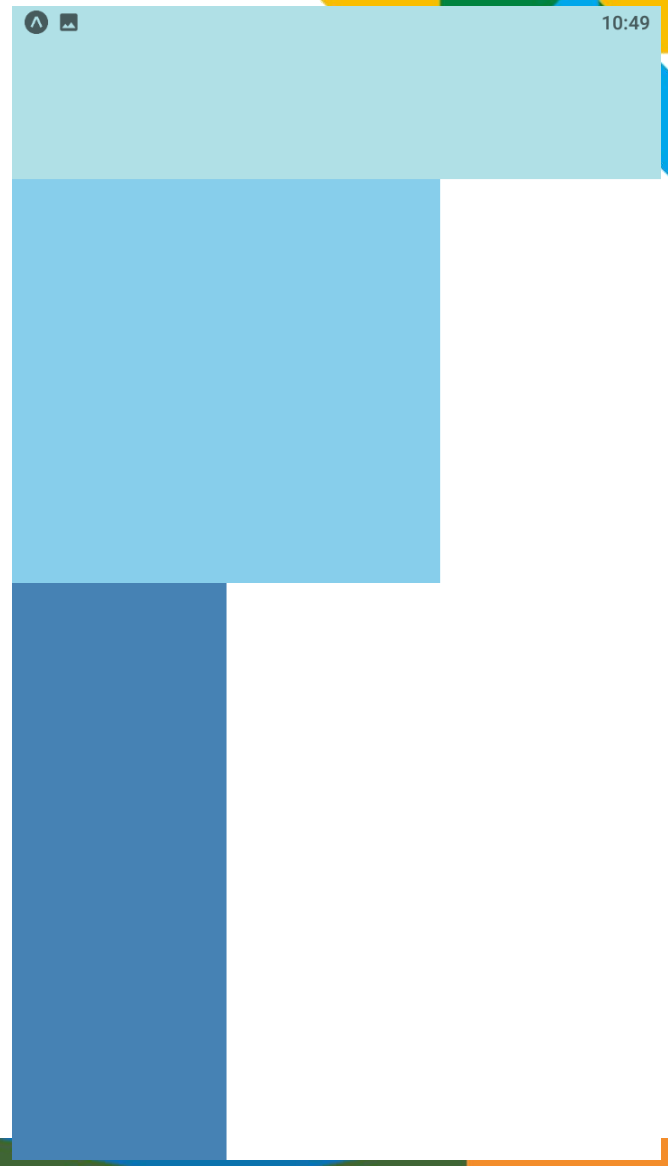
Percentage Dimensions 百分比長寬

- 除了使用 flex外，還可以使用 百分比來設定長寬比例。
- Parent component 同樣需要有設定長寬，child component 才能夠顯示出來。

Percentage Dimensions 百分比長寬

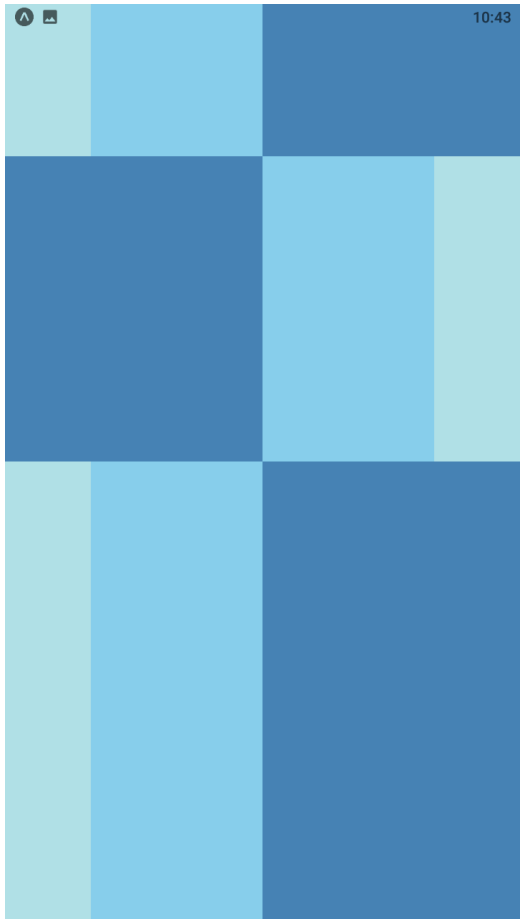
File: App.js

```
01: import React from 'react';
02: import { View } from 'react-native';
03: const PercentageDimensionsBasics = () => {
04:   // Try removing the `height: '100%'` on the parent View.
05:   // The parent will not have dimensions, so the children can't expand.
06:   return (
07:     <View style={{ height: '100%' }}>
08:       <View style={{
09:         height: '15%', backgroundColor: 'powderblue'
10:       }} />
11:       <View style={{
12:         width: '66%', height: '35%', backgroundColor: 'skyblue'
13:       }} />
14:       <View style={{
15:         width: '33%', height: '50%', backgroundColor: 'steelblue'
16:       }} />
17:     </View>
18:   );
19: };
20: export default PercentageDimensionsBasics;
```

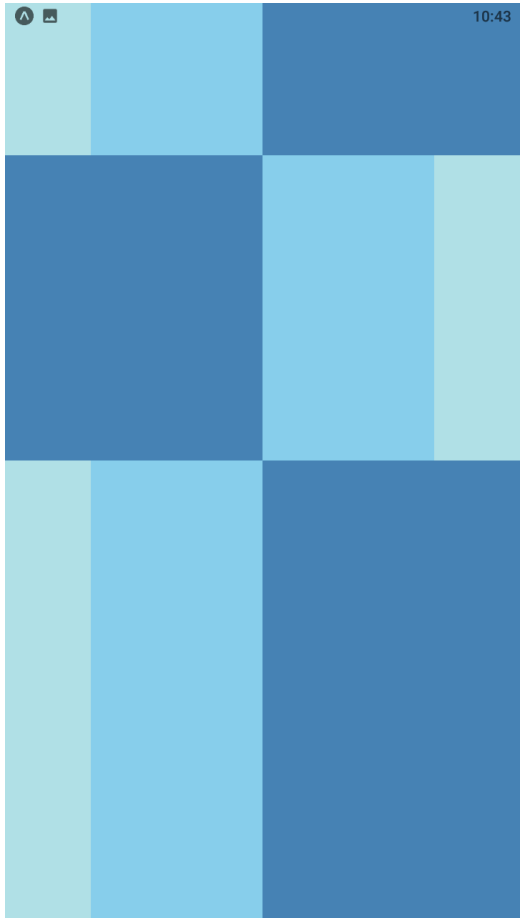


練習一下

- 試著做出類似右邊的效果



一個方法



File: App.js

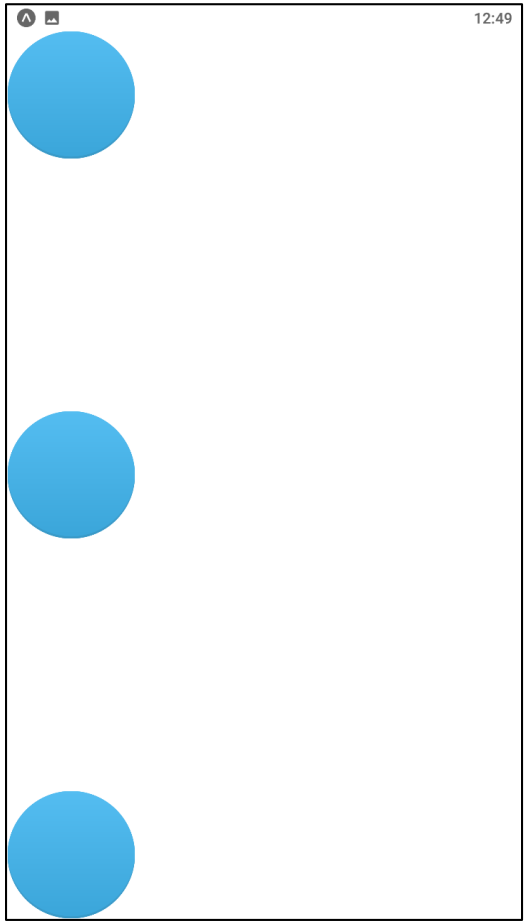
```
01: import React from 'react';
02: import { View } from 'react-native';
03:
04: export default function App() {
05:   return (
06:     // 練習，調整 parent component size
07:     <View style={{flex: 1}}>
08:       <View style={{flexDirection:"row", flex: 1}}>
09:         <View style={{flex: 1, backgroundColor: 'powderblue'}} />
10:         <View style={{flex: 2, backgroundColor: 'skyblue'}} />
11:         <View style={{flex: 3, backgroundColor: 'steelblue'}} />
12:       </View>
13:       <View style={{flexDirection:"row-reverse", flex: 2}}>
14:         <View style={{flex: 1, backgroundColor: 'powderblue'}} />
15:         <View style={{flex: 2, backgroundColor: 'skyblue'}} />
16:         <View style={{flex: 3, backgroundColor: 'steelblue'}} />
17:       </View>
18:       <View style={{flexDirection:"row", flex: 3}}>
19:         <View style={{flex: 1, backgroundColor: 'powderblue'}} />
20:         <View style={{flex: 2, backgroundColor: 'skyblue'}} />
21:         <View style={{flex: 3, backgroundColor: 'steelblue'}} />
22:       </View>
23:     </View>
24:   );
25: };
```

React Native flexbox 元素

- flexDirection: 'column' || 'row'
  - 排列方向
- justifyContent: 'flex-start' || 'flex-end' || 'center' || 'space-around' || 'space-between'
  - Component 是否對齊主軸(main axis)對齊
- alignItems: 'stretch' || 'flex-start' || 'flex-end' || 'center'
  - Component 是否對齊交叉軸(cross axis)
- flexWrap: 'nowrap' || 'wrap' || 'wrap-reverse'
  - Component 是否換行

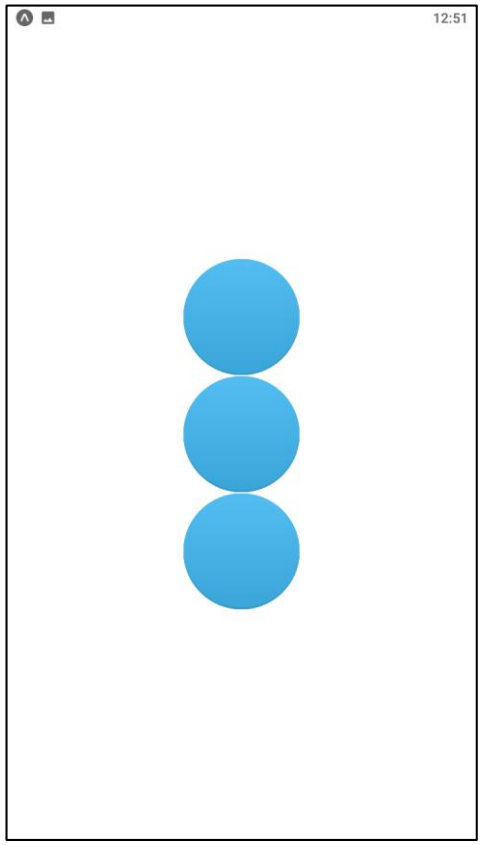
justifyContent:
'flex-start' || 'flex-end' || 'center' || 'space-around' || 'space-between'

```
File: App.js
09: export default function App() {
10:     return (
11:       <View style={styles.container}>
12:         <Image style={styles.image} source={require('./img/point.png')}/>
13:         <Image style={styles.image} source={require('./img/point.png')}/>
14:         <Image style={styles.image} source={require('./img/point.png')}/>
15:       </View>
16:     );
17: }
18:
19: const styles = StyleSheet.create({
20:     container: {
21:         paddingTop:24,
22:         flex:1,
23:         justifyContent:'space-between'
24:         // flexWrap: 'wrap',
25:     },
26:     image: {
27:         width: 120,
28:         height: 120,
29:         padding: 20,
30:     }
31: });
```



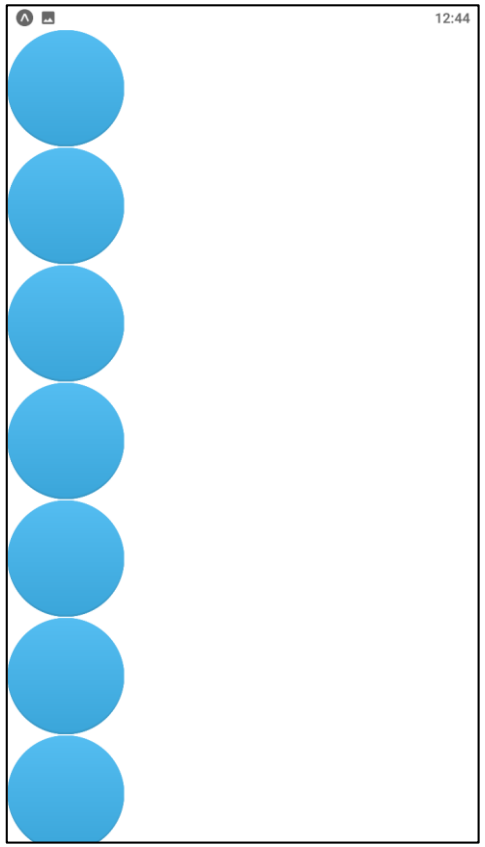
alignItems:
'stretch' || 'flex-start' || 'flex-end' || 'center'

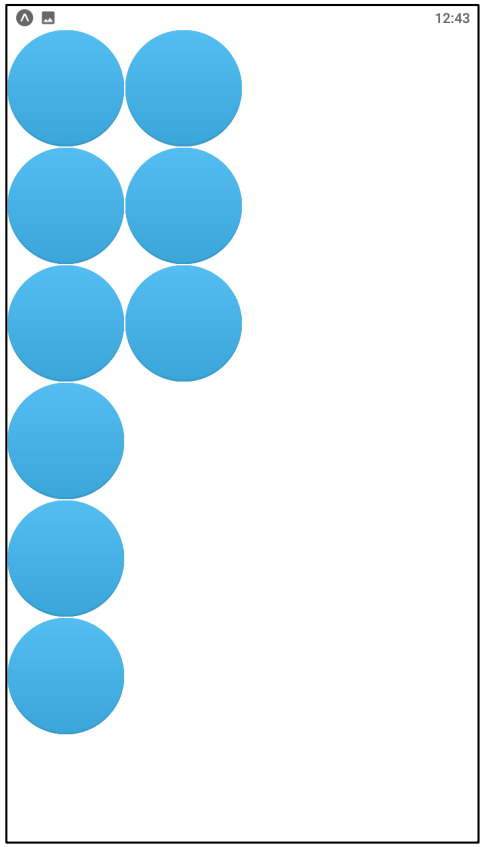
```
File: App.js
09: export default function App() {
10:     return (
11:       <View style={styles.container}>
12:         <Image style={styles.image} source={require('./img/point.png')}/>
13:         <Image style={styles.image} source={require('./img/point.png')}/>
14:         <Image style={styles.image} source={require('./img/point.png')}/>
15:       </View>
16:     );
17: }
18:
19: const styles = StyleSheet.create({
20:     container: {
21:         paddingTop:24,
22:         flex:1,
23:         justifyContent:'center',
24:         alignItems:'center'
25:         // flexWrap: 'wrap',
26:     },
27:     image: {
28:         width: 120,
29:         height: 120,
30:         padding: 20,
31:     }
32: });
```



flexWrap: 'nowrap' || 'wrap' || 'wrap-reverse'

- Component 太多時，是否換行

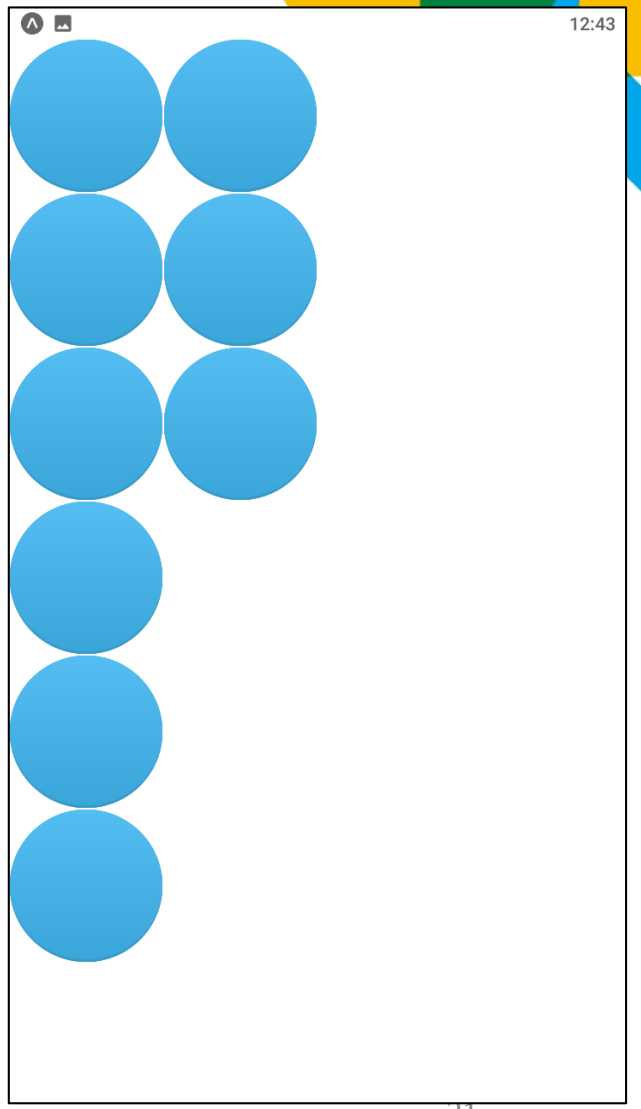




flexWrap: 'nowrap' || 'wrap' || 'wrap-reverse'

File: App.js

```
09: export default function App() {
10:     return (
11:       <View style={styles.container}>
12:         <Image style={styles.image} source={require('./img/point.png')}/>
13:         <Image style={styles.image} source={require('./img/point.png')}/>
14:         <Image style={styles.image} source={require('./img/point.png')}/>
15:         <Image style={styles.image} source={require('./img/point.png')}/>
16:         <Image style={styles.image} source={require('./img/point.png')}/>
17:         <Image style={styles.image} source={require('./img/point.png')}/>
18:         <Image style={styles.image} source={require('./img/point.png')}/>
19:         <Image style={styles.image} source={require('./img/point.png')}/>
20:         <Image style={styles.image} source={require('./img/point.png')}/>
21:       </View>
22:     );
23: }
24:
25: const styles = StyleSheet.create({
26:     container: {
27:         paddingTop:24,
28:         flex:1,
29:         flexWrap: 'wrap',
30:     },
31:     image: {
32:         width: 120,
33:         height: 120,
34:         padding: 20,
35:     }
36: });
```



練習一下

- 使用不同的 flexbox 元素，看看效果怎樣?

Q&A