React native Component

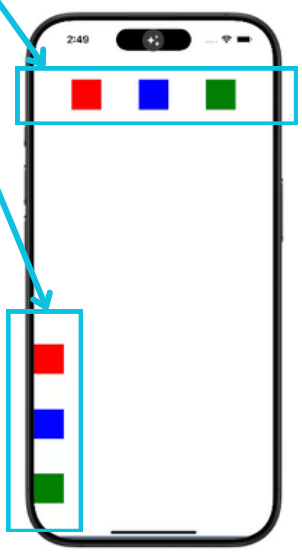
- justify content
- flexWrap
- alignItems
- alignSelf
- alignContent
- Width and Hight
- Absolute & Relatie Layout
- [Flex Basis, Grow, and Shrink]
- [RowGap, ColumnGap and Gap]

React native justify content

```jsx
import React from 'react';
import { View, StyleSheet, Text } from 'react-native';

const JustifyContentExample = () => {
  return (
    <View style={styles.cont}>
      <View style={[styles.container, {flexDirection: 'row'}]}>
        <View style={styles.box1} />
        <View style={styles.box2} />
        <View style={styles.box3} />
      </View>
      <View style={[styles.container, {flexDirection: 'column'}]}>
        <View style={styles.box1} />
        <View style={styles.box2} />
        <View style={styles.box3} />
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  cont: {
    flex: 1,
    marginTop: 100,
    flexDirection: 'column',
  },
  container: {
    flex: 1,
    justifyContent: 'space-evenly', // กำหนดการจัดเรียง
  },
  box1: {
    width: 50,
    height: 50,
    backgroundColor: 'red',
  },
  box2: {
    width: 50,
    height: 50,
    backgroundColor: 'blue',
  },
  box3: {
    width: 50,
    height: 50,
    backgroundColor: 'green',
  },
});

export default JustifyContentExample;
```

justifyContent
ใช้สำหรับจัดตำแหน่งขององค์ประกอบภายในคอนเทนเนอร์ตาม แกนหลัก (main axis) ซึ่งแกนหลักนี้จะถูกกำหนดโดยคุณสมบัติ flexDirection:

flexDirection



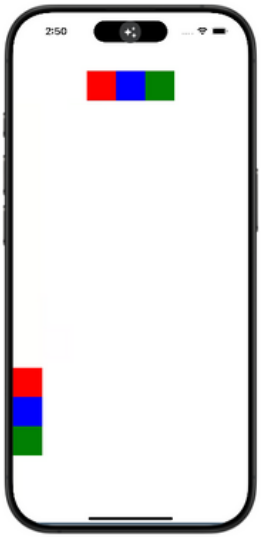
- justifyContent _____

นี่คือตัวเลือกทั่วไปสำหรับ justifyContent และผลลัพธ์:

1. **flex-start** (ค่าเริ่มต้น): จัดตำแหน่งองค์ประกอบไปที่จุดเริ่มต้นของแกนหลัก
2. **flex-end:** จัดตำแหน่งองค์ประกอบไปที่จุดสิ้นสุดของแกนหลัก
3. **center:** จัดตำแหน่งองค์ประกอบให้อยู่ตรงกลางของแกนหลัก
4. **space-between:** กระจายองค์ประกอบให้มีช่องว่างเท่ากันระหว่างแต่ละองค์ประกอบ โดยองค์ประกอบแรกจะอยู่ที่จุดเริ่มต้น และองค์ประกอบสุดท้ายจะอยู่ที่จุดสิ้นสุด
5. **space-around:** กระจายองค์ประกอบให้มีช่องว่างเท่ากันรอบ ๆ องค์ประกอบแต่ละตัว โดยช่องว่างที่ปลายทั้งสองด้านของคอนเทนเนอร์จะเป็นครึ่งหนึ่งของช่องว่างระหว่างองค์ประกอบ
6. **space-evenly:** กระจายองค์ประกอบให้มีช่องว่างเท่ากันทั้งหมด รวมถึงปลายทั้งสองด้านของคอนเทนเนอร์

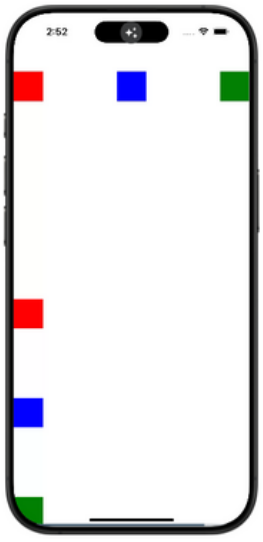
Example

flex-start
จัดองค์ประกอบไว้ที่จุดเริ่มต้นของแกนหลัก

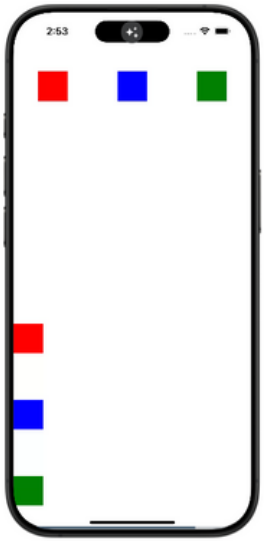


center
จัดองค์ประกอบให้อยู่กึ่งกลางแกนหลัก

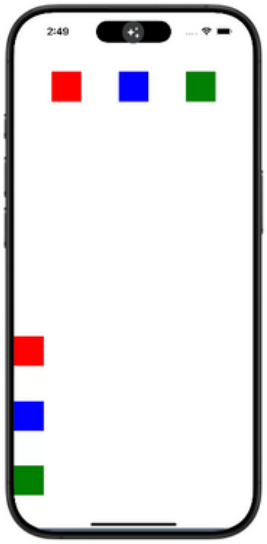
flex-end
จัดลูกองค์ประกอบไว้ที่จุดสิ้นสุดของแกนหลัก



space-between
กระจายองค์ประกอบให้มีระยะห่างเท่ากัน ขอบนอกขององค์ประกอบแรกและสุดท้ายจะชิดกับขอบของ container



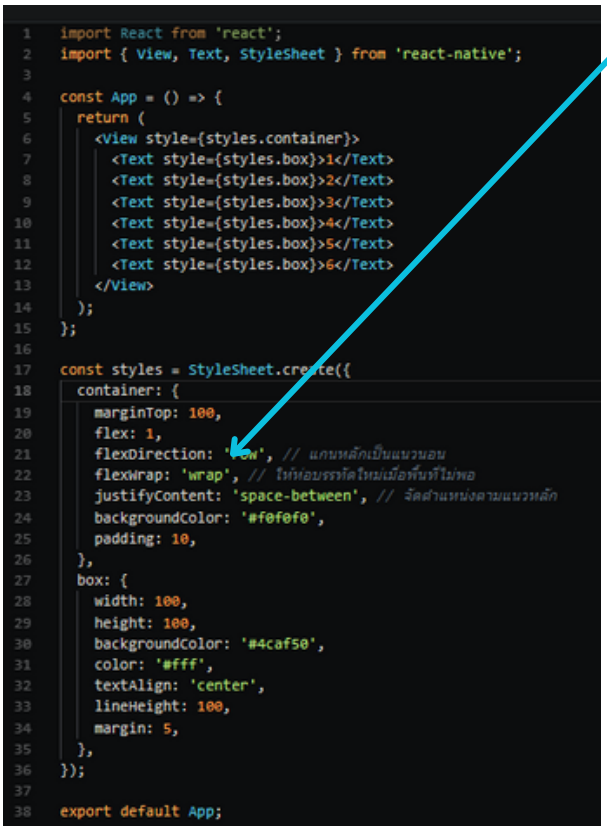
space-around
กระจายองค์ประกอบโดยมีระยะห่างรอบตัวเท่ากัน ระยะห่างระหว่างองค์ประกอบจะมากกว่าระยะห่างที่ขอบนอก



space-evenly
กระจายองค์ประกอบให้มีระยะห่างเท่ากันทุกด้าน

flexWrap

ใช้สำหรับกำหนดการจัดการพื้นที่ขององค์ประกอบลูก (child elements) ภายในคอนเทนเนอร์ที่ใช้ Flexbox เมื่อพื้นที่ในแนวหลักไม่เพียงพอ องค์ประกอบลูกจะถูกย้ายไปยังบรรทัดถัดไป (wrap)



```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.box}>1</Text>
      <Text style={styles.box}>2</Text>
      <Text style={styles.box}>3</Text>
      <Text style={styles.box}>4</Text>
      <Text style={styles.box}>5</Text>
      <Text style={styles.box}>6</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 100,
    flex: 1,
    flexDirection: 'row', // แกนหลักเป็นแนวนอน
    flexWrap: 'wrap', // ให้ห่อบรรทัดใหม่เมื่อพื้นที่ไม่พอ
    justifyContent: 'space-between', // จัดตำแหน่งตามแนวหลัก
    backgroundColor: '#f0f0f0',
    padding: 10,
  },
  box: {
    width: 100,
    height: 100,
    backgroundColor: '#4caf50',
    color: '#fff',
    textAlign: 'center',
    lineHeight: 100,
    margin: 5,
  },
});

export default App;
```

การทำงานของแต่ละค่า

nowrap

- องค์ประกอบจะเรียงต่อกันในบรรทัดเดียว แม้ว่าจะเกินพื้นที่ของคอนเทนเนอร์

wrap

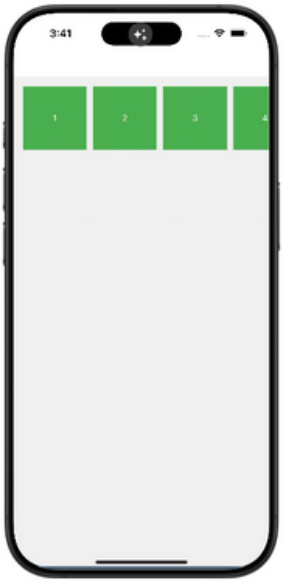
- องค์ประกอบจะย้ายไปยังบรรทัดถัดไปเมื่อพื้นที่ไม่พอ

wrap-reverse

- บรรทัดใหม่จะถูกเพิ่มด้านบนแทนที่จะเป็นด้านล่าง

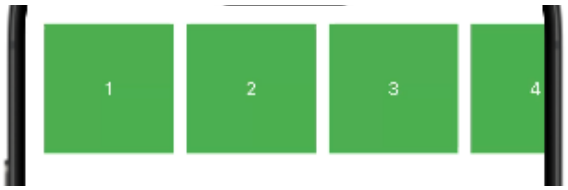
มีประโยชน์มากเมื่อทำงานกับ UI แบบ Responsive หรือจัดการองค์ประกอบที่มีขนาดแตกต่างกันในคอนเทนเนอร์เดียวกัน!

Example



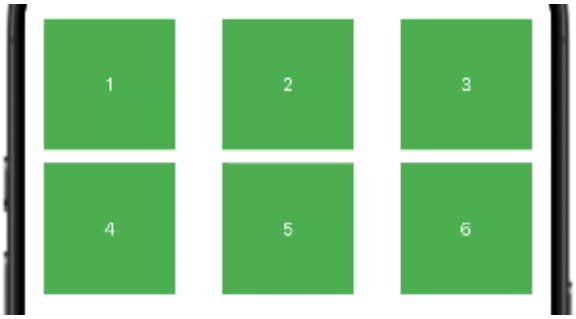
nowrap

- องค์ประกอบจะเรียงต่อกันในบรรทัดเดียว แม้ว่าจะเกินพื้นที่ของคอนเทนเนอร์



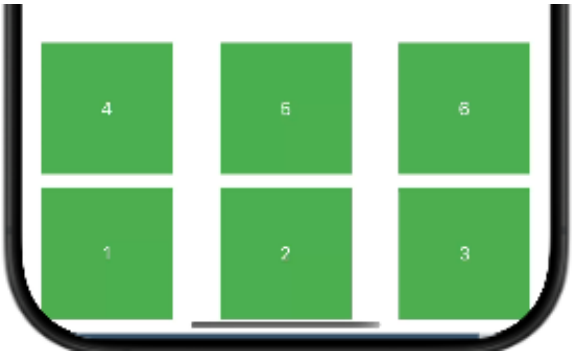
wrap

- องค์ประกอบจะย้ายไปยังบรรทัดถัดไปเมื่อพื้นที่ไม่พอ



wrap-reverse

- บรรทัดใหม่จะถูกเพิ่มด้านบนแทนที่จะเป็นด้านล่าง



alignItems

ใช้สำหรับจัดตำแหน่งขององค์ประกอบลูก (child elements) ภายในคอนเทนเนอร์ตามแกนที่กำหนดโดย flexDirection:

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.box}>กล่องที่ 1</Text>
      <Text style={styles.box}>กล่องที่ 2</Text>
      <Text style={styles.box}>กล่องที่ 3</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'row', // แกนหลักเป็นแนวนอน
    justifyContent: 'center', // จัดองค์ประกอบให้อยู่ตรงกลางของแกนหลัก
    alignItems: 'flex-end', // จัดองค์ประกอบของแกนขวาง
    backgroundColor: '#f0f0f0',
  },
  box: {
    width: 100,
    height: 50,
    backgroundColor: '#4caf50',
    color: '#fff',
    textAlign: 'center',
    lineHeight: 50,
    margin: 5,
  },
});

export default App;
```

การทำงานของแต่ละค่า

flex-start

- องค์ประกอบลูกทั้งหมดจะชิดด้านเริ่มต้นของแกนขวาง

flex-end

- องค์ประกอบลูกทั้งหมดจะชิดด้านสิ้นสุดของแกนขวาง

center

- องค์ประกอบลูกจะอยู่ตรงกลางแกนขวาง

stretch

- องค์ประกอบลูกที่ไม่ได้กำหนดขนาดจะถูกยืดให้เต็มพื้นที่ของแกนขวาง

baseline

- ใช้จัดองค์ประกอบตามเส้นฐานของข้อความ โดยเหมาะสำหรับแกนหลักที่เป็นแนวนอน

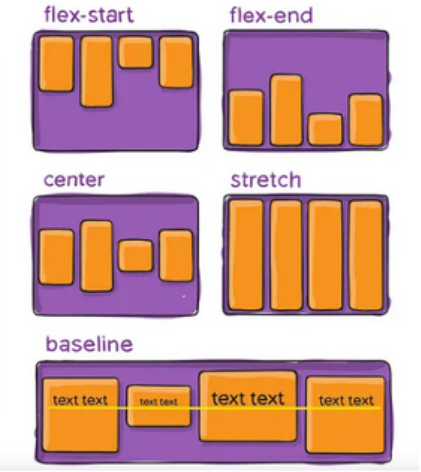
Example

flex-start

flex-end

center

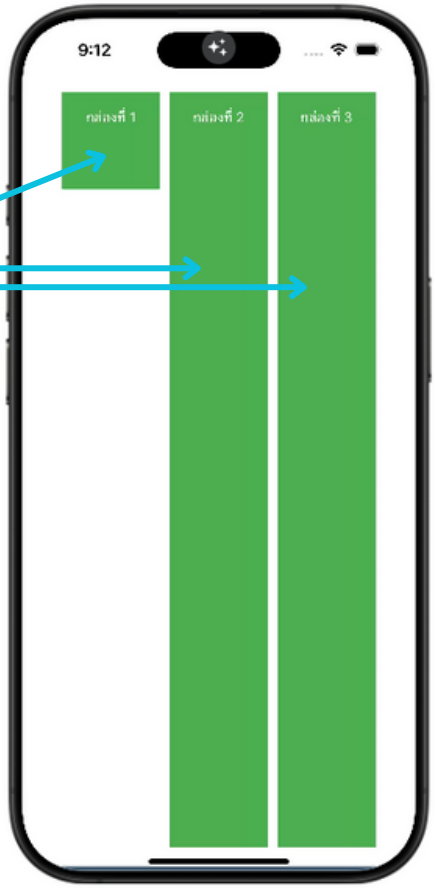
baseline



stretch

- องค์ประกอบลูกที่ไม่ได้กำหนดขนาดจะถูกยืดให้เต็มพื้นที่ของแกนขวาง

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={[styles.box,{width: 100, height: 100}]}>กล่องที่ 1</Text>
      <Text style={styles.box}>กล่องที่ 2</Text>
      <Text style={styles.box}>กล่องที่ 3</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 50,
    padding:20,
    flex: 1,
    flexDirection: 'row', // แกนหลักเป็นแนวนอน
    justifyContent: 'center', // จัดองค์ประกอบให้อยู่ตรงกลางของแกนหลัก
    alignItems: 'stretch', // จัดองค์ประกอบไปที่จุดสิ้นสุดของแกนขวาง
  },
  box: {
    width: 100,
    backgroundColor: '#4caf50',
    color: '#fff',
    textAlign: 'center',
    lineHeight: 50,
    margin: 5,
  },
});

export default App;
```



alignSelf

ใช้สำหรับจัดตำแหน่งของ องค์ประกอบแต่ละตัว (child element) ตามแกนขวาง (cross axis) โดยเฉพาะ ซึ่งจะมีผลเฉพาะกับองค์ประกอบลูกนั้นๆ และจะ มีลำดับความสำคัญสูงกว่า การตั้งค่า alignItems ของคอนเทนเนอร์แม่ (parent container)

วิธีทำงานของ alignSelf

ถ้าคอนเทนเนอร์กำหนด alignItems ไว้ เช่น alignItems: 'center' แต่คุณต้องการให้บางองค์ประกอบไม่ได้อยู่กึ่งกลาง คุณสามารถใช้ alignSelf เพื่อกำหนดตำแหน่งเฉพาะขององค์ประกอบนั้น

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={[styles.box, styles.box1]}>กล่องที่ 1</Text>
      <Text style={[styles.box, styles.box2]}>กล่องที่ 2</Text>
      <Text style={[styles.box, styles.box3]}>กล่องที่ 3</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop: 100,
    flex: 1,
    flexDirection: 'column',
    alignItems: 'center', // กำหนดค่าเริ่มต้นให้ทุกกล่องอยู่กึ่งกลาง
  },
  box: {
    width: 100,
    height: 50,
    backgroundColor: '#4caf50',
    color: '#fff',
    textAlign: 'center',
    lineHeight: 50,
    margin: 5,
  },
  box1: {
    alignSelf: 'flex-start', // กล่องนี้จะชิดด้านบนของคอนเทนเนอร์
  },
  box2: {
    alignSelf: 'center', // กล่องนี้จะอยู่ตรงกลางตามแกนขวาง
  },
  box3: {
    alignSelf: 'flex-end', // กล่องนี้จะชิดด้านล่างของคอนเทนเนอร์
  },
});

export default App;
```

ผลลัพธ์

- กล่องที่ 1 จะจัดตำแหน่งชิดด้านซ้ายบน
- กล่องที่ 2 จะอยู่ตรงกลางตามแกนขวาง
- กล่องที่ 3 จะจัดตำแหน่งชิดด้านขวาล่าง

alignSelf ใช้สำหรับการจัดตำแหน่งเฉพาะตัวขององค์ประกอบลูก โดยไม่กระทบกับการตั้งค่า alignItems ของคอนเทนเนอร์แม่ เหมาะสำหรับกรณีที่ต้องการให้บางองค์ประกอบมีตำแหน่งที่แตกต่างจากองค์ประกอบอื่นๆ ในคอนเทนเนอร์เดียวกัน!

alignContent

ใช้สำหรับจัดตำแหน่งของ แถว (rows) หรือ บรรทัด (lines) ขององค์ประกอบลูก (child elements) ภายในคอนเทนเนอร์ เมื่อมีหลายบรรทัด โดยจะทำงานได้ก็ต่อเมื่อ:

- มีการตั้งค่า flexWrap: 'wrap' ให้กับคอนเทนเนอร์
- องค์ประกอบลูกมีขนาดรวมที่ทำให้ต้องห่อบรรทัดใหม่

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Text style={styles.box}>1</Text>
      <Text style={styles.box}>2</Text>
      <Text style={styles.box}>3</Text>
      <Text style={styles.box}>4</Text>
      <Text style={styles.box}>5</Text>
      <Text style={styles.box}>6</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop:50,
    flex: 1,
    flexDirection: 'row', // แกนหลักเป็นแนวนอน
    flexWrap: 'wrap', // อนุญาตให้ห่อบรรทัดใหม่
    alignContent: 'space-between', // กระจายแถวให้มีระยะห่างเท่ากัน
    justifyContent: 'center', // จัดตำแหน่งตามแกนหลัก
    padding: 10,
  },
  box: {
    width: 100,
    height: 100,
    backgroundColor: '#4caf50',
    color: '#fff',
    textAlign: 'center',
    lineHeight: 100,
    margin: 5,
  },
});

export default App;
```

alignContent

flex-start

- จัดแถวทั้งหมดไปที่จุดเริ่มต้นของแกนขวาง (cross axis)

flex-end

- จัดแถวทั้งหมดไปที่จุดสิ้นสุดของแกนขวาง

center

- จัดแถวให้อยู่ตรงกลางของแกนขวาง

stretch (ค่าเริ่มต้น)

- ยืดแถวทั้งหมดให้ครอบคลุมพื้นที่ของคอนเทนเนอร์

space-between

- กระจายแถวให้มีระยะห่างเท่ากัน โดยแถวแรกอยู่ที่จุดเริ่มต้นและแถวสุดท้ายอยู่ที่จุดสิ้นสุด

space-around

- กระจายแถวให้มีระยะห่างเท่ากันรอบ ๆ แต่ละแถว (ช่องว่างที่ปลายจะเป็นครึ่งหนึ่งของช่องว่างระหว่างแถว)

space-evenly

- กระจายแถวให้มีระยะห่างเท่ากันทั้งหมด รวมถึงระยะห่างจากขอบของคอนเทนเนอร์

Example

alignItems และ alignContent

- alignItems: ใช้จัด องค์ประกอบลูก ในแกนขวาง
- alignContent: ใช้จัด แถวขององค์ประกอบลูก ในแกนขวาง
- alignContent จะทำงานเฉพาะเมื่อ flexWrap เป็น 'wrap' และมีหลายแถวเท่านั้น!

Width and Hight

ขนาด ขององค์ประกอบได้โดยใช้คุณสมบัติ width และ height ภายใน StyleSheet เพื่อกำหนดความกว้างและความสูงแบบคงที่ หรือแบบยืดหยุ่นสำหรับองค์ประกอบนั้น

การกำหนด width และ height

ค่าคงที่ (Fixed Dimensions)

- ระบุค่าความกว้าง (width) และความสูง (height) เป็นตัวเลขที่แน่นอน (หน่วยเป็นพิกเซล)

ค่าร้อยละ (Percentage Dimensions)

- ใช้สำหรับขนาดที่อ้างอิงจากขนาดของคอนเทนเนอร์พาเรนต์ (หน่วย %)

ค่าอัตโนมัติ (Auto Dimensions)

- ไม่กำหนดค่า width และ height เพื่อให้ React Native คำนวณจากเนื้อหาภายในขององค์ประกอบเอง

ค่าความยืดหยุ่น (Flex Dimensions)

- ใช้ flex เพื่อกำหนดขนาดตามสัดส่วนของพื้นที่ในคอนเทนเนอร์

การใช้ % ใน width และ height ขึ้นอยู่กับขนาดของ คอนเทนเนอร์พาเรนต์
หากไม่ได้กำหนดค่า width หรือ height จะใช้ค่าที่ React Native คำนวณจากเนื้อหา

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      {/* Fixed Dimensions */}
      <View style={[styles.box, styles.fixedBox]}>
        <Text style={styles.text}>Fixed</Text>
      </View>
      {/* Percentage Dimensions */}
      <View style={[styles.box, styles.percentageBox]}>
        <Text style={styles.text}>Percentage</Text>
      </View>
      {/* Flex Dimensions */}
      <View style={[styles.box, styles.flexBox]}>
        <Text style={styles.text}>Flex</Text>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop:70,
    flex: 1,
    justifyContent: 'space-around',
    alignItems: 'center',
  },
  box: {
    justifyContent: 'center',
    alignItems: 'center',
    borderWidth: 2,
    borderColor: '#333',
    borderRadius: 8,
  },
  fixedBox: {
    width: 100, // กำหนดขนาดคงที่
    height: 100,
    backgroundColor: '#4caf50',
  },
  percentageBox: {
    width: '80%', // กำหนดความกว้างเป็นเปอร์เซ็นต์
    height: '20%',
    backgroundColor: '#2196f3',
  },
  flexBox: {
    flex: 1, // ใช้พื้นที่ที่เหลือของคอนเทนเนอร์
    width: '90%', // กำหนดความกว้างเป็นเปอร์เซ็นต์
    backgroundColor: '#ff9800',
  },
  text: {
    color: '#fff',
    fontWeight: 'bold',
  },
});

export default App;
```

Example

Fixed Dimensions

- กล่องสีเขียว มีขนาด 100x100 พิกเซลแบบคงที่

Percentage Dimensions

- กล่องสีน้ำเงิน มีความกว้าง 80% และความสูง 20% ของคอนเทนเนอร์

Flex Dimensions

- กล่องสีส้ม ใช้พื้นที่ที่เหลือของคอนเทนเนอร์ โดยกระจายขนาดตามสัดส่วน flex ที่กำหนด

Absolute & Relatie Layout

การกำหนดตำแหน่งองค์ประกอบ (Layout) สามารถทำได้ 2 วิธีหลัก คือ Absolute Layout และ Relative Layout โดยทั้งสองวิธีมีการใช้งานและความเหมาะสมในสถานการณ์ที่ต่างกัน

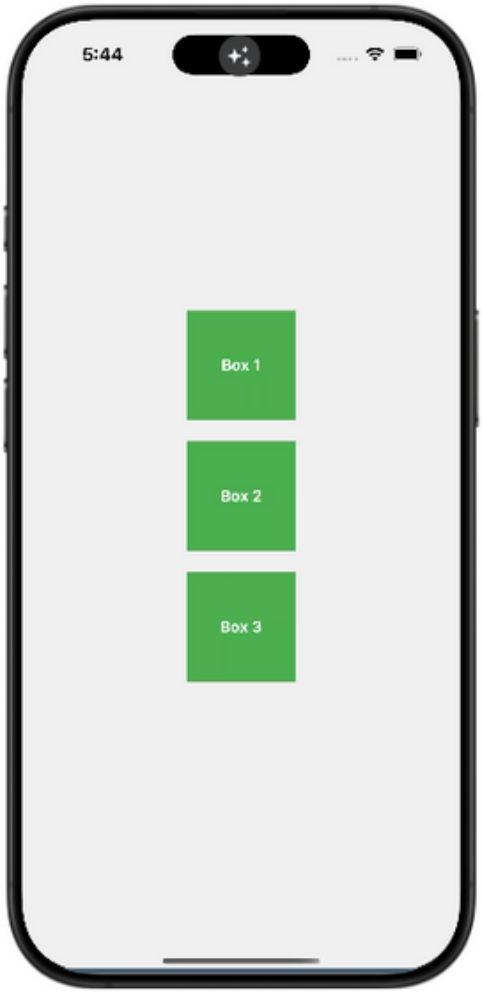
1. Relative Layout (ตำแหน่งสัมพันธ์)

- ค่าเริ่มต้น ใน React Native
- องค์ประกอบจะถูกจัดวางต่อเนื่องกันโดยอิงตามลำดับใน Flexbox (เช่น flexDirection, justifyContent, และ alignItems)
- เหมาะสำหรับการสร้าง UI ที่ยืดหยุ่น และปรับตามขนาดหน้าจอ

  กล่องแต่ละกล่อง (Box) จะถูกจัดวางตาม Flexbox โดยมีระยะห่าง (margin) ระหว่างกัน เหมาะสำหรับการจัดวางองค์ประกอบที่ต้องการปรับตัวตามขนาดหน้าจอ

Example

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const RelativeLayoutExample = () => {
  return (
    <View style={styles.container}>
      <View style={styles.box}>
        <Text style={styles.text}>Box 1</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 2</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 3</Text>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#f0f0f0',
  },
  box: {
    width: 100,
    height: 100,
    backgroundColor: '#4caf50',
    justifyContent: 'center',
    alignItems: 'center',
    margin: 10,
  },
  text: {
    color: '#fff',
    fontWeight: 'bold',
  },
});

export default RelativeLayoutExample;
```

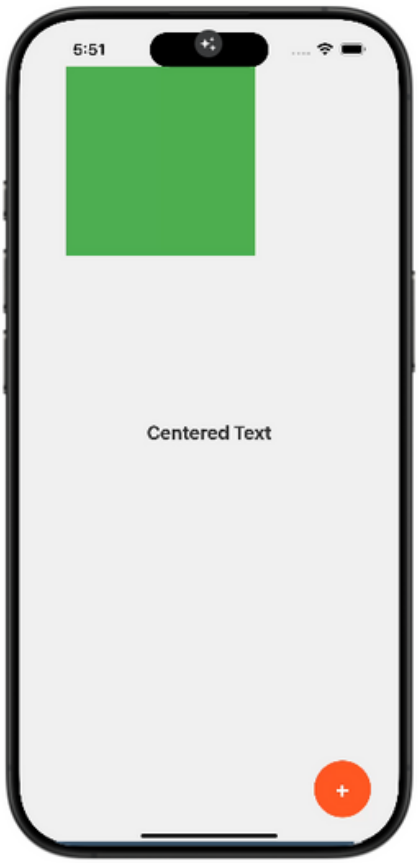


2. Absolute Layout (ตำแหน่งแน่นอน)

- ใช้ position: 'absolute' เพื่อกำหนดตำแหน่งขององค์ประกอบในหน้าจอ
- ใช้คุณสมบัติ top, right, bottom, และ left เพื่อระบุตำแหน่งแน่นอน
- เหมาะสำหรับการวางองค์ประกอบคงที่ เช่น ปุ่มลอย, แถบแจ้งเตือน, หรือ พื้นหลังที่ซ้อนกัน

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const AbsoluteLayoutExample = () => {
  return (
    <View style={styles.container}>
      <View style={styles.backgroundBox} />
      <Text style={styles.centerText}>Centered Text</Text>
      <View style={styles.floatingButton}>
        <Text style={styles.buttonText}>+</Text>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#f0f0f0',
  },
  backgroundBox: {
    position: 'absolute',
    top: 50,
    left: 50,
    width: 200,
    height: 200,
    backgroundColor: '#4caf50',
  },
  centerText: {
    fontSize: 20,
    fontWeight: 'bold',
    color: '#333',
  },
  floatingButton: {
    position: 'absolute',
    bottom: 30,
    right: 30,
    width: 60,
    height: 60,
    borderRadius: 30,
    backgroundColor: '#ff5722',
    justifyContent: 'center',
    alignItems: 'center',
  },
  buttonText: {
    fontSize: 24,
    color: '#fff',
    fontWeight: 'bold',
  },
});

export default AbsoluteLayoutExample;
```

Example



backgroundBox: ถูกจัดวางโดยใช้ position: 'absolute' ที่ระยะ top: 50 และ left: 50
centerText: ใช้ตำแหน่ง Relative (ค่าเริ่มต้น) และถูกจัดให้อยู่ตรงกลางตาม Flexbox
floatingButton: ปุ่มลอยอยู่ที่มุมล่างขวาของหน้าจอ (bottom: 30, right: 30)

- ใช้ Relative Layout เป็นหลักใน UI ทั่วไป
- ใช้ Absolute Layout สำหรับองค์ประกอบเฉพาะที่ต้องการตำแหน่งคงที่ (เช่น ปุ่มลอย, ภาพพื้นหลัง)

การจัดวาง Absolute Layout เหมาะสำหรับ:

- ปุ่มลอย (Floating Action Button - FAB)
- พื้นหลังหรือ Layer ซ้อน (Background overlays)
- ตำแหน่งเฉพาะเจาะจง เช่น ป้ายแจ้งเตือน

React Native [Flex Basis, Grow, and Shrink]

การจัดการกับขนาดและการขยายหรือหดตัวขององค์ประกอบภายใน Flexbox สามารถทำได้ด้วย flexBasis, flexGrow, และ flexShrink. ทั้งสามช่วยให้สามารถควบคุมขนาดและการจัดสรรพื้นที่ขององค์ประกอบในคอนเทนเนอร์ที่มีหลายองค์ประกอบ

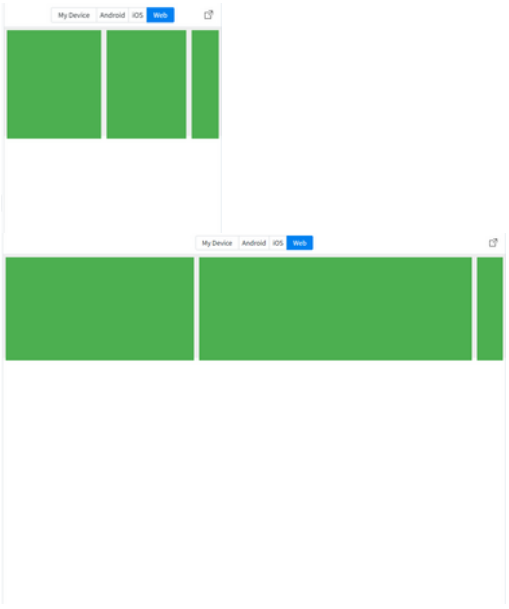
- flex-grow: กำหนดว่า item ควรจะขยายตัวเพิ่มขึ้นหรือไม่ เมื่อมีพื้นที่ว่าง.
- flex-shrink: กำหนดว่า item ควรจะลดขนาดลงหรือไม่ เมื่อพื้นที่ไม่เพียงพอ.
- flex-basis: กำหนดขนาดพื้นฐานของ item.

1. flexBasis
   - flexBasis กำหนดขนาดเริ่มต้นขององค์ประกอบในแกนหลัก (main axis) ก่อนที่จะมีการปรับขนาดตาม flexGrow หรือ flexShrink.
   - ใช้เพื่อกำหนดขนาดเริ่มต้นขององค์ประกอบก่อนที่การกระจายพื้นที่จะเกิดขึ้น
   - ค่าของ flexBasis อาจเป็น ค่าในหน่วยพิกเซล (px) หรือ เปอร์เซ็นต์ (%)
2. flexGrow
   - flexGrow ควบคุมการขยายขององค์ประกอบให้ใช้พื้นที่ที่เหลืออยู่ในคอนเทนเนอร์
   - ค่าเริ่มต้นคือ 0, ซึ่งหมายความว่าองค์ประกอบจะไม่ขยาย
   - หากมีการตั้งค่า flexGrow มากกว่า 0, องค์ประกอบนั้นจะขยายเพื่อใช้พื้นที่ที่เหลือ
   - เมื่อหลายองค์ประกอบมีค่า flexGrow ต่างกัน, พื้นที่ที่เหลือจะถูกแบ่งตามอัตราส่วนของค่า flexGrow
3. flexShrink
   - flexShrink ควบคุมการหดตัวขององค์ประกอบเมื่อพื้นที่ในคอนเทนเนอร์ไม่เพียงพอ
   - ค่าเริ่มต้นคือ 1, ซึ่งหมายความว่าองค์ประกอบจะหดตัวตามความจำเป็น
   - หากตั้งค่าเป็น 0, องค์ประกอบจะไม่หดตัวเลยเมื่อพื้นที่ไม่เพียงพอ

Example

ตัวอย่างเมื่อมีการเปลี่ยนแปลงขนาน

```jsx
import React from 'react';
import { View, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      {/* Box 1: ขนาดเริ่มต้น 150px, ขยายได้ */}
      <View style={[styles.box, { flexGrow: 1, flexBasis: 150 }]} />
      {/* Box 2: ขนาดเริ่มต้น 100px, ขยายได้ */}
      <View style={[styles.box, { flexGrow: 2, flexBasis: 100 }]} />
      {/* Box 3: ขนาดเริ่มต้น 50px, หดตัวได้ */}
      <View style={[styles.box, { flexShrink: 1, flexBasis: 50 }]} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flexDirection: 'row',
    height: 200,
    backgroundColor: '#f0f0f0',
  },
  box: {
    height: '100%',
    backgroundColor: '#4caf50',
    margin: 5,
  },
});

export default App;
```



React Native [RowGap, ColumnGap and Gap]

Gap สำหรับการเว้นระยะระหว่างองค์ประกอบในแนวนอน (Row) หรือแนวตั้ง (Column) ทำได้ผ่าน gap, rowGap, และ columnGap แต่จะใช้ได้ใน Flexbox ผ่านการตั้งค่า gap หรือใช้ margin โดยการใช้ gap ยังไม่ได้รับการสนับสนุนโดยตรงใน React Native เวอร์ชันที่ต่ำกว่า แต่ React Native รุ่นใหม่ ๆ ได้เริ่มรองรับการใช้ gap แล้ว

การใช้ gap, rowGap, columnGap

- **gap**: ใช้กำหนดระยะห่างระหว่างองค์ประกอบทั้งในแนวนอนและแนวตั้ง
- **rowGap**: ใช้กำหนดระยะห่างระหว่างองค์ประกอบในแนวนอน (row)
- **columnGap**: ใช้กำหนดระยะห่างระหว่างองค์ประกอบในแนวตั้ง (column)

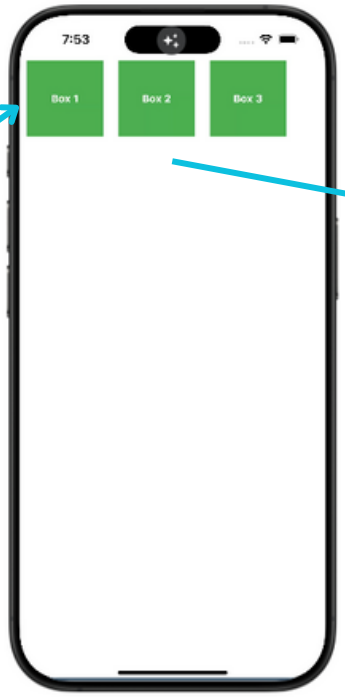
1. การใช้ gap

- ใช้เมื่อคุณต้องการเว้นระยะระหว่างองค์ประกอบทั้งในแนวนอนและแนวตั้ง

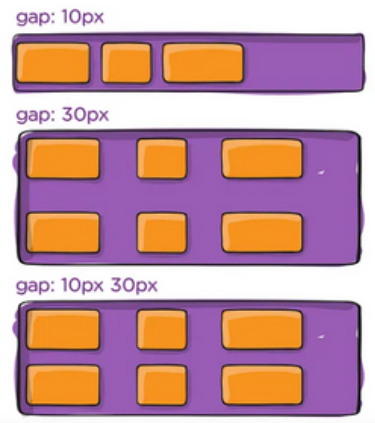
ในตัวอย่างข้างต้น เราได้กำหนด gap เป็น 10 หน่วย ซึ่งจะสร้างช่องว่างระหว่างองค์ประกอบย่อยภายในคอนเทนเนอร์

Example

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const GapExample = () => {
  return (
    <View style={styles.container}>
      <View style={styles.box}>
        <Text style={styles.text}>Box 1</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 2</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 3</Text>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    gap: 20, // กำหนดระยะห่างระหว่างองค์ประกอบทั้งแนวนอนและแนวตั้ง
    padding: 10,
  },
  box: {
    width: 100,
    height: 100,
    backgroundColor: '#4caf50',
    justifyContent: 'center',
    alignItems: 'center',
  },
  text: {
    color: '#fff',
    fontWeight: 'bold',
  },
});

export default GapExample;
```



เพิ่ม gap: 40



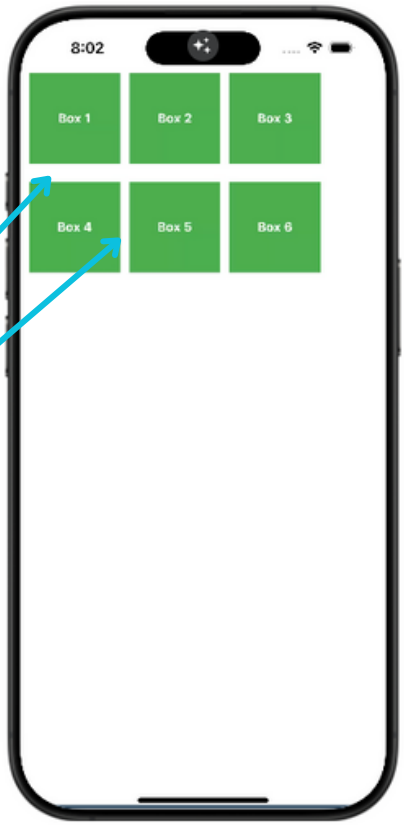
2. การใช้ rowGap และ columnGap

ใช้สำหรับการควบคุมระยะห่างระหว่างองค์ประกอบในแต่ละแกน (ทั้งแนวนอนและแนวตั้ง) โดยแยกการควบคุมทั้งสอง

- rowGap: กำหนดระยะห่างระหว่าง แถว (row) ขององค์ประกอบย่อยในคอนเทนเนอร์
- columnGap: กำหนดระยะห่างระหว่าง คอลัมน์ (column) ขององค์ประกอบย่อยในคอนเทนเนอร์

Example

```jsx
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const RowColumnGapExample = () => {
  return (
    <View style={styles.container}>
      <View style={styles.box}>
        <Text style={styles.text}>Box 1</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 2</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 3</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 4</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 5</Text>
      </View>
      <View style={styles.box}>
        <Text style={styles.text}>Box 6</Text>
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    marginTop:50,
    flexDirection: 'row',
    flexWrap: 'wrap',
    rowGap: 20, // ระยะห่างระหว่างแถว
    columnGap: 10, // ระยะห่างระหว่างคอลัมน์
    padding: 10,
  },
  box: {
    width: 100,
    height: 100,
    backgroundColor: '#4caf50',
    justifyContent: 'center',
    alignItems: 'center',
  },
  text: {
    color: '#fff',
    fontWeight: 'bold',
  },
});

export default RowColumnGapExample;
```



gap, rowGap, columnGap

- gap: ใช้เพื่อกำหนดระยะห่างระหว่างองค์ประกอบทั้งในแนวนอนและแนวตั้ง โดยไม่ต้องใช้ margin
- rowGap: ใช้เพื่อกำหนดระยะห่างระหว่างองค์ประกอบในแถวเดียวกัน (row)
- columnGap: ใช้เพื่อกำหนดระยะห่างระหว่างองค์ประกอบในคอลัมน์เดียวกัน (column)

หากต้องการเว้นระยะระหว่างองค์ประกอบใน React Native, คุณสามารถใช้ gap, rowGap, และ columnGap ได้ในเวอร์ชันใหม่ ๆ (React Native 0.71+)
หากเวอร์ชันที่ใช้ไม่รองรับ gap, ใช้ margin เพื่อควบคุมการเว้นระยะระหว่างองค์ประกอบ