Chapter 3

CORE COMPONENTS AND STYLING (PART II)

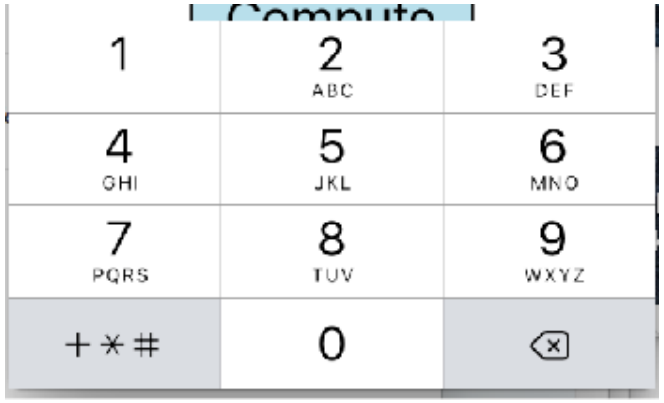
TextInput

- ทำหน้าที่รับอินพุตที่เป็นข้อความจากผู้ใช้
- Ref: https://reactnative.dev/docs/textinput
- เราสามารถปรับแต่งคุณสมบัติ (props) ของ TextInput ได้หลายแบบ
  - autoCapitalize : กำหนดรูปแบบให้แสดงอักษรตัวใหญ่แบบต่างๆ
  - placeholder : กำหนดข้อความแนะนำในช่องอินพุต
  - style : ปรับหน้าตาของอินพุต
  - editable : อนุญาตให้แก้ไขข้อความในอินพุตหรือไม่
  - keyboardType : กำหนดประเภทของคีย์บอร์ด
  - value : ข้อความที่แสดงในอินพุต ทุกครั้งที่กรอกข้อความจะอัพเดทค่า value
  - onChangeText : กำหนดฟังก์ชัน ซึ่งจะถูกเรียกใช้งานเมื่อข้อความในอินพุตมีการเปลี่ยนแปลง

Keyboard Types (Cross-Platform) - iOS

<TextInput keyboardType='default'>

default



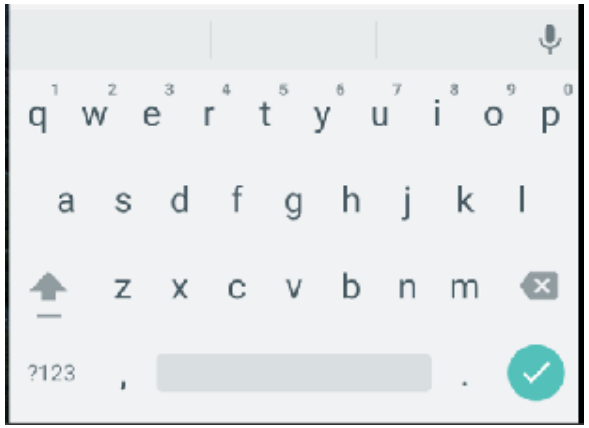
phone-pad

numeric

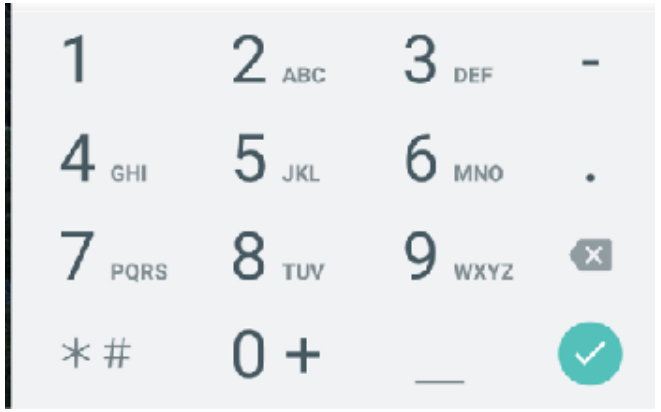
email-address

Keyboard Types (Cross-Platform) - Android

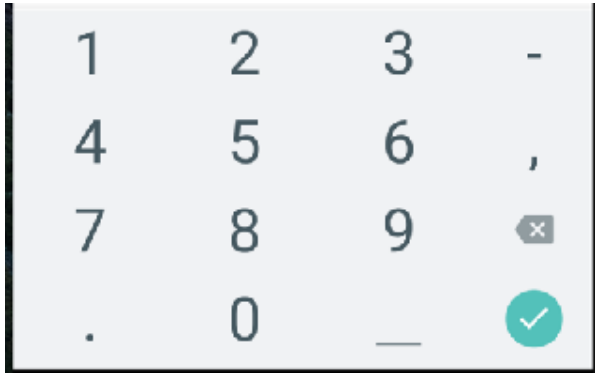
<TextInput keyboardType='default'>



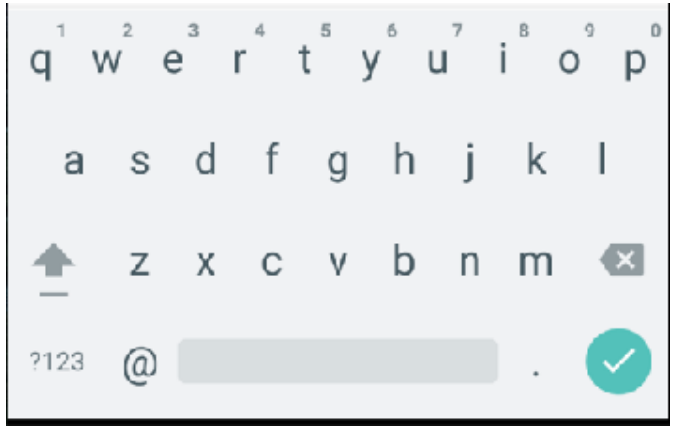
default



phone-pad



numeric



email-address

State

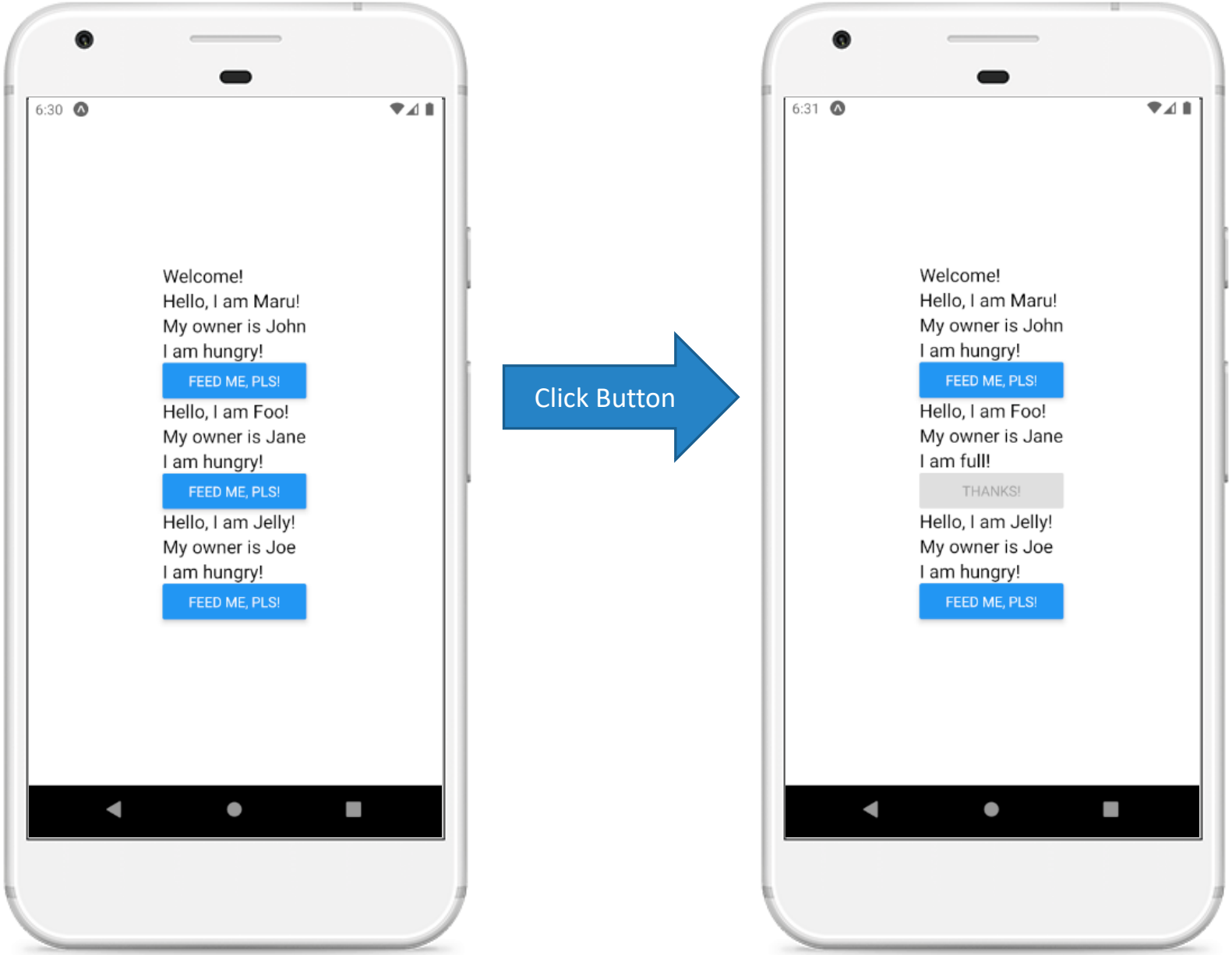
- State คือ ข้อมูลที่เก็บอยู่ในคอมโพเนนต์
- เมื่อค่าของ State มีการเปลี่ยนแปลง React Native จะทำการเรนเดอร์เนื้อหาที่อยู่ในคอมโพเนนต์นั้นใหม่ ซึ่งสามารถเปลี่ยนแปลงได้ตามค่าของ State ได้ทันที
- การกำหนด State สามารถทำได้ทั้งใน
  - Class Component
  - Function Component

State: Function Component

- ต้องทำการ import {useState} from ‘react’
- การกำหนด state ให้กับคอมโพเนนต์ -> ใช้ useState()
  - รูปแบบ : [*ตัวแปรข้อมูลใน state*, *ฟังก์ชันอัพเดทค่าข้อมูลใน state*] = useState(*ค่าเริ่มต้น*)
  - const [name, setName] = useState(‘James’);
  - const [gender, setGender] = useState(‘male’);
- การเรียกใช้ค่าใน state สามารถเรียกผ่านชื่อตัวแปรข้อมูลได้เลย
  - name หรือ gender
- การกำหนดค่าใหม่ใน state -> ใช้ฟังก์ชันที่กำหนดไว้ก่อนหน้านี้
  - setName(‘Jane’);
  - setGender(‘female’);

States: Function Component (Cat.js)

```
import React, { useState } from "react";
import { Button, Text, View } from "react-native";

const Cat = (props) => {
 const [isHungry, setIsHungry] = useState(true);

 return (
  <View>
   <Text>Hello, I am {props.name} !</Text>
   <Text>My owner is {props.owner}</Text>
   <Text>I am {isHungry ? "hungry" : "full"}!</Text>
   <Button
    onPress={() => { setIsHungry(false);}}
    disabled={!isHungry}
    title={isHungry ? “Feed me, pls!" : "Thanks!"}
   />
  </View>
 );
}
export default Cat;
```


6:30
Welcome!
Hello, I am Maru!
My owner is John
I am hungry!
FEED ME, PLS!
Hello, I am Foo!
My owner is Jane
I am hungry!
FEED ME, PLS!
Hello, I am Jelly!
My owner is Joe
I am hungry!
FEED ME, PLS!
Click Button
6:31
Welcome!
Hello, I am Maru!
My owner is John
I am hungry!
FEED ME, PLS!
Hello, I am Foo!
My owner is Jane
I am full!
THANKS!
Hello, I am Jelly!
My owner is Joe
I am hungry!
FEED ME, PLS!

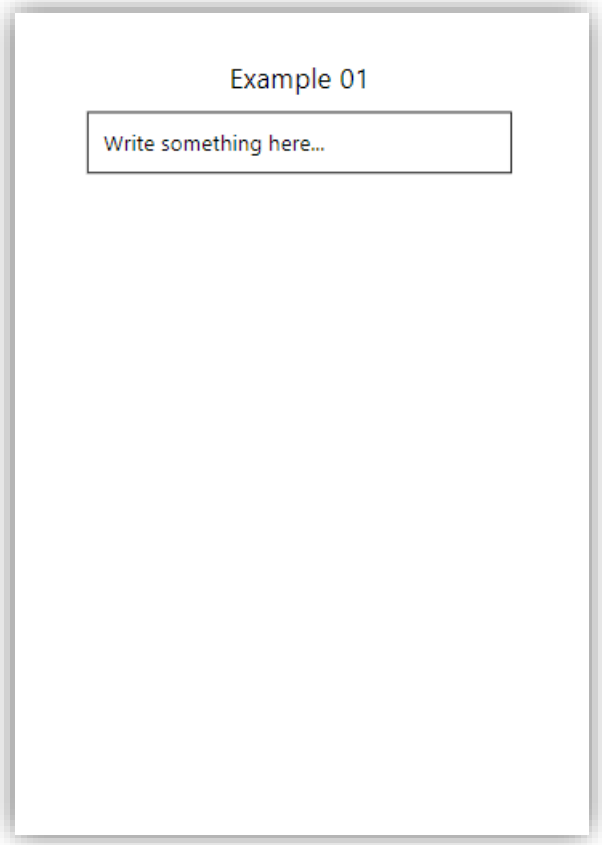
ผลลัพธ์

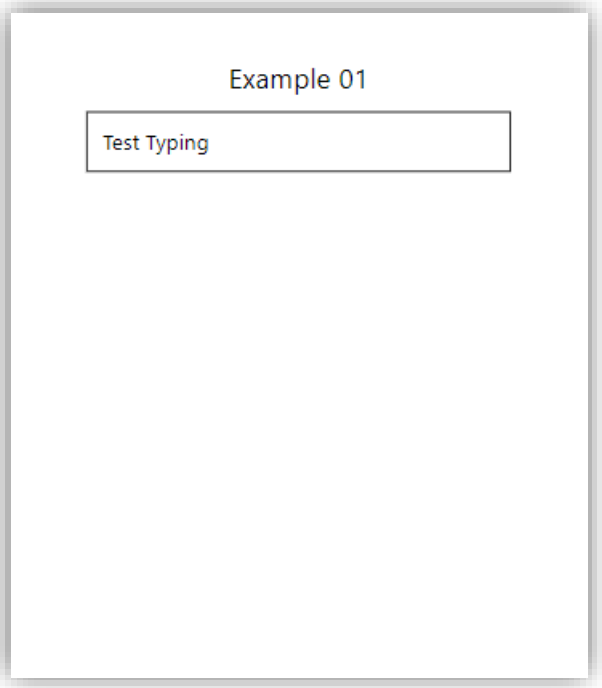
การกำหนด state เพื่อเก็บอินพุตของ TextInput

- ระหว่างที่ผู้ใช้กรอกข้อความลงในอินพุต ผู้ใช้จะต้องเห็นข้อมูลที่กรอกในอินพุตด้วย
- เก็บข้อมูลที่กรอกลงใน state
- เราสามารถใช้ข้อมูลที่เก็บใน state ไปใช้ต่อได้ เช่น เอาไปประมวลผลต่อ หรือ แสดงในคอมโพเนนต์อื่นๆ
- ทำการเก็บ state เมื่อข้อความมีการเปลี่ยนแปลง
  - value : ข้อความที่แสดงในอินพุต ทุกครั้งที่กรอกข้อความจะอัพเดทค่า value
  - onChangeText : กำหนดฟังก์ชัน ซึ่งจะถูกเรียกใช้งานเมื่อข้อความในอินพุตมีการเปลี่ยนแปลง

ตัวอย่างโปรแกรม
การกำหนด state เพื่อเก็บอินพุตของ TextInput

```
export default function Example01() {
  const [text, setText] = useState("");
  return (
    <View style={styles.container}>
      <Text style={{ fontSize: 18 }}>Example 01</Text>
      <TextInput
          placeholder="Write something here..."
          style={styles.input}
          value={text}
          onChangeText={(input) => { setText(input); }}
          />
    </View>
  );
}
```





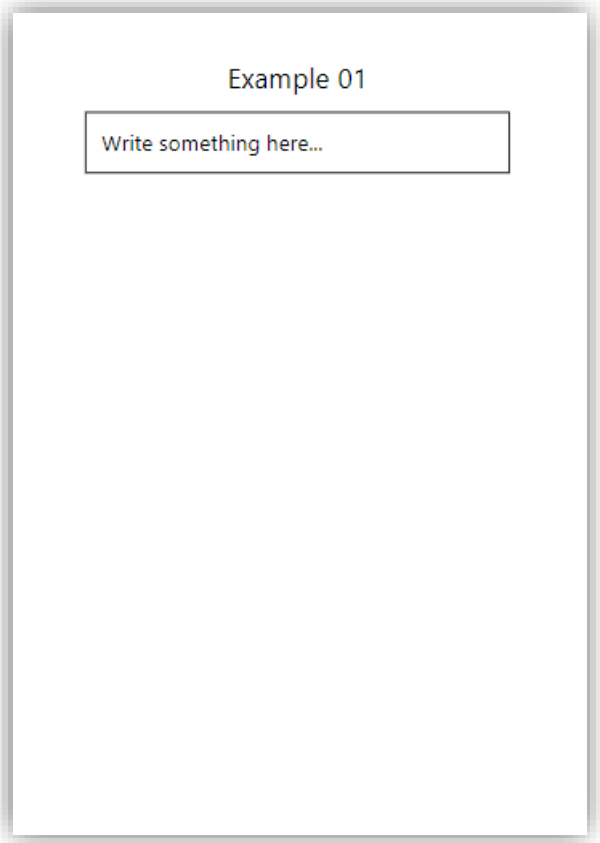
onChangeText

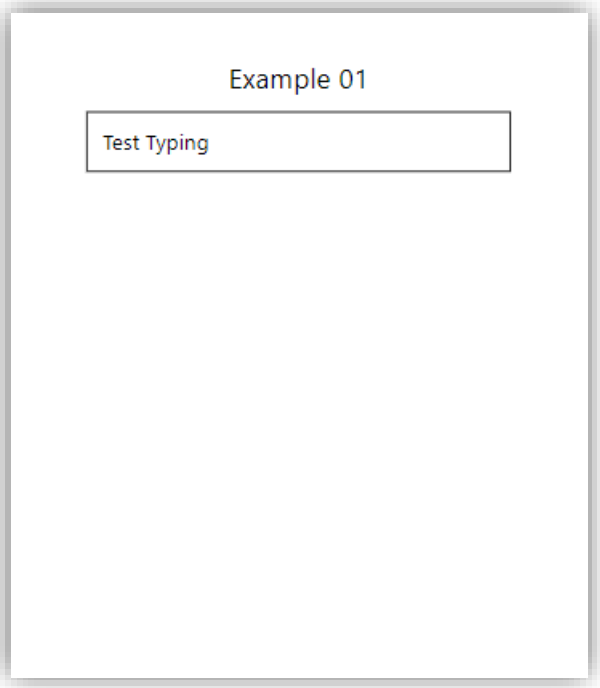
- เมื่อข้อความใน TextInput เปลี่ยนแปลง โปรแกรมกำหนดให้เก็บข้อความที่กรอก (input) ในสเตท text

- onChangeText={(input) => { setText(input); }}

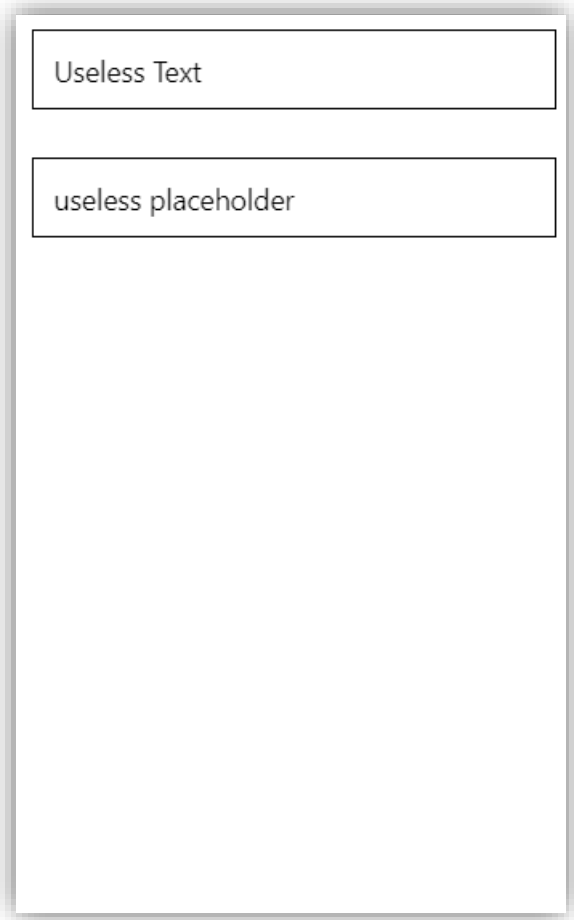
หรือ

- onChangeText={setText}





TextInput (Example)

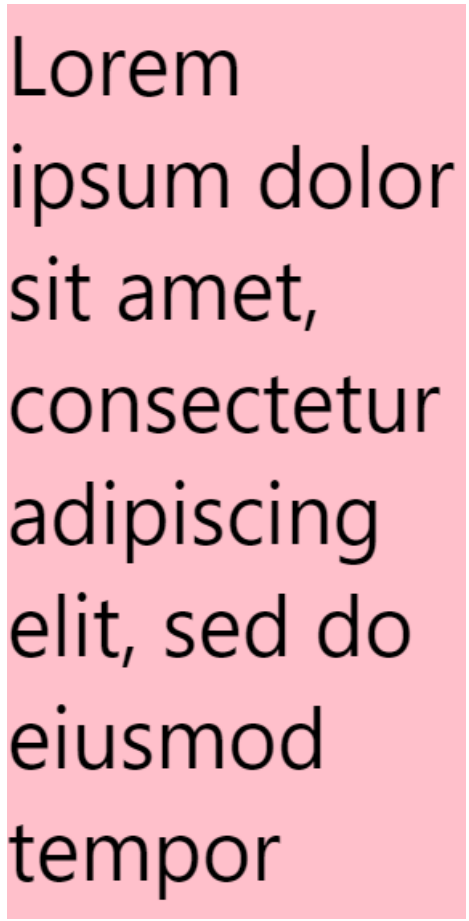



```jsx
import React from 'react';
import {SafeAreaView, StyleSheet, TextInput} from 'react-native';

const TextInputExample = () => {
  const [text, onChangeText] = React.useState('Useless Text');
  const [number, onChangeNumber] = React.useState('');

  return (
    <SafeAreaView>
      <TextInput
        style={styles.input}
        onChangeText={onChangeText}
        value={text}
      />
      <TextInput
        style={styles.input}
        onChangeText={onChangeNumber}
        value={number}
        placeholder="useless placeholder"
        keyboardType="numeric"
      />
    </SafeAreaView>
  );
};

const styles = StyleSheet.create({
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
  },
});
export default TextInputExample;
```

ScrollView

- ใช้แสดงเนื้อหาที่มีมากเกินกว่าขนาดหน้าจอ ทำให้ไม่สามารถแสดงข้อมูลได้หมดในหน้าจอเดียว
  - แสดงรายการสินค้า
  - แสดงรายชื่อนักศึกษา
- ทำการโหลดข้อมูลทั้งหมดมาเก็บในหน่วยความจำไว้ก่อน (ทั้งส่วนที่แสดงให้เห็นในหน้าจอ รวมถึงข้อมูลที่เหลือที่ยังไม่ได้แสดงในหน้าจอ)
  - หากรายการที่จะแสดงมีจำนวนมาก จะทำให้แอปพลิเคชันทำงานช้า
- Ref: https://reactnative.dev/docs/scrollview

```
import React from 'react';
import { StyleSheet, Text, SafeAreaView, ScrollView } from 'react-native';

const App = () => {
  return (
    <SafeAreaView style={styles.container}>
      <ScrollView style={styles.scrollView}>
        <Text style={styles.text}>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
          eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
          minim veniam, quis nostrud exercitation ullamco laboris nisi ut
          aliquip ex ea commodo consequat. Duis aute irure dolor in
          reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
          pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
          culpa qui officia deserunt mollit anim id est laborum.
        </Text>
      </ScrollView>
    </SafeAreaView>
  ); }
```

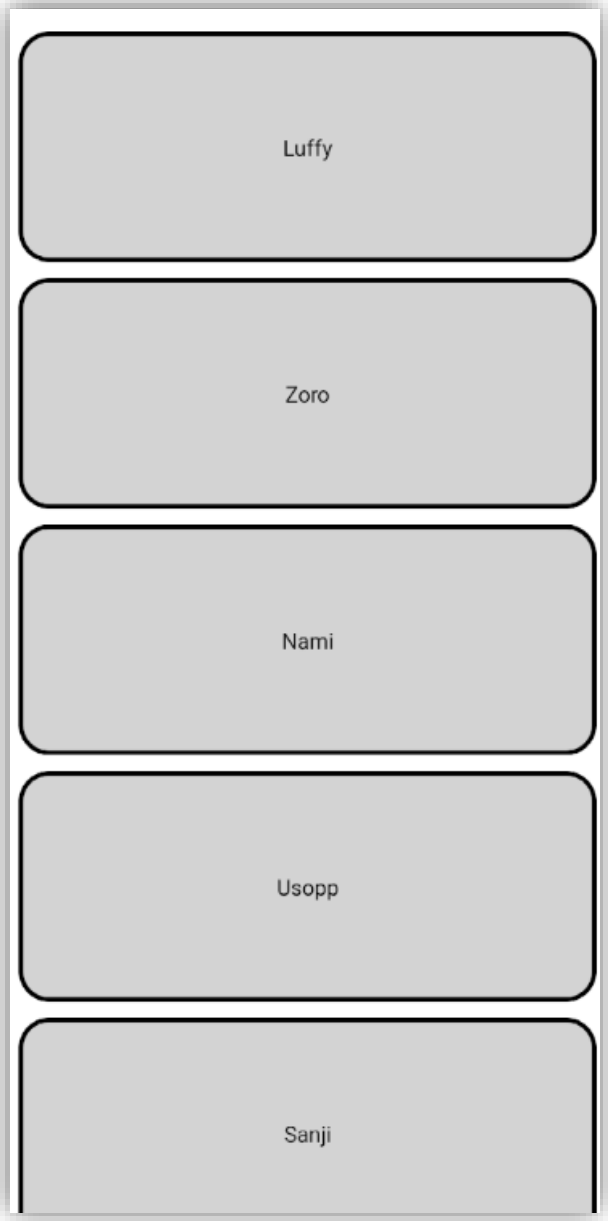


การนำข้อมูลจากอะเรย์มาแสดงใน ScrollView

- การแสดงข้อมูลจำนวนมากใน ScrollView เราอาจจะเก็บข้อมูลรายการที่จะแสดงอยู่ในอะเรย์
- การประกาศตัวแปรอะเรย์
  - const myArray = [‘Jisoo’, ‘Jenny’, ‘Rose’, ‘Lisa’]
- การนำข้อมูลในอะเรย์ไปแสดงในคอมโพเนนต์ จะต้องแปลงข้อมูลในอะเรย์ ให้อยู่ในรูปแบบ JSX ด้วยเมธอด map() ก่อน
- const list = myArray.map((item) => { // สมาชิกแต่ละตัวของอะเรย์ (item) จะถูกแปลงเป็น JSX แล้วเก็บใน list

```
        return (
            //JSX
        )
})
```

```
export default function Example02(){
    const students = [
        {id: 1, name: 'Luffy’}, {id: 2, name: 'Zoro'}, {id: 3, name: 'Nami'},
        {id: 4, name: 'Usopp'}, {id: 5, name: 'Sanji'}, {id: 6, name: 'Chopper'},
        {id: 7, name: 'Nico Robin'}, {id: 8, name: 'Franky'}, {id: 9, name: 'Brook'}];
    return (
        <SafeAreaView style={styles.container}>
            <ScrollView contentContainerStyle={styles.scrollview}>
                {students.map((item) => {
                    return(
                        <View key={item.id} style={styles.child}>
                            <Text>{item.name}</Text>
                        </View>
                    );
                })}
            </ScrollView>
        </SafeAreaView>
    );
}
```



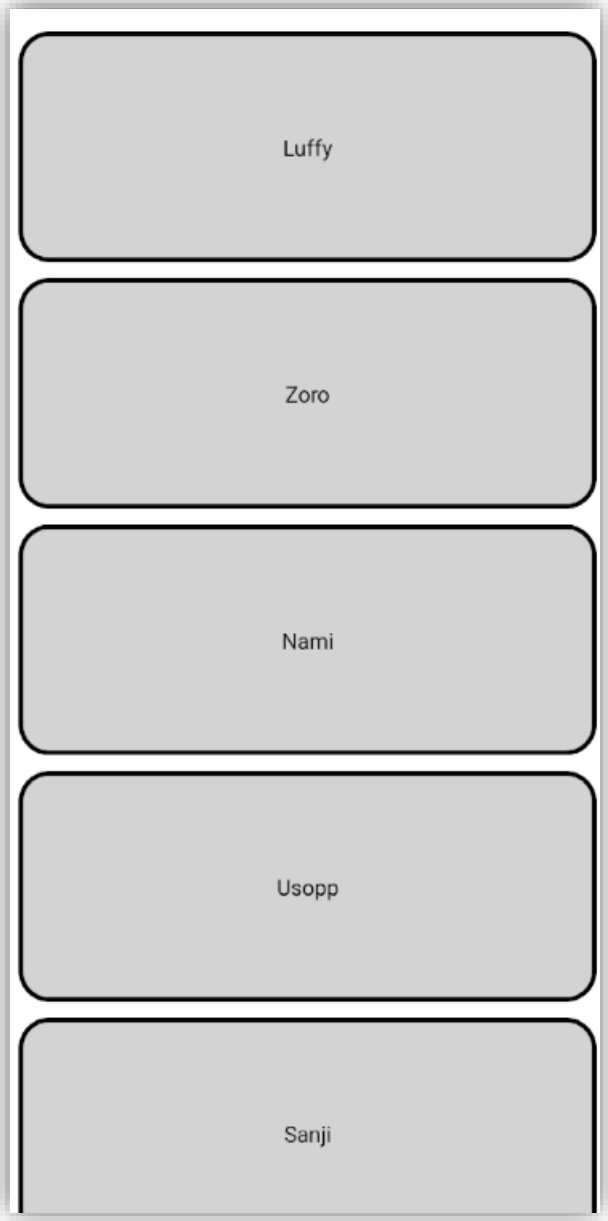
FlatList

- เช่นเดียวกับ ScrollView คือ FlatList ใช้แสดงเนื้อหาที่มีมากเกินกว่าขนาดหน้าจอ ทำให้ไม่สามารถแสดงข้อมูลได้หมดในหน้าจอเดียว
- มีประสิทธิภาพมากกว่า ScrollView
  - จะเรนเดอร์ข้อมูลเฉพาะที่จะนำมาแสดงบนหน้าจอเท่านั้น ส่วนที่นอกเหนือจากนั้น จะไม่ถูกเรนเดอร์
  - ทำให้แอปพลิเคชันทำงานได้เร็วกว่า
- Ref: https://reactnative.dev/docs/flatlist

FlatList

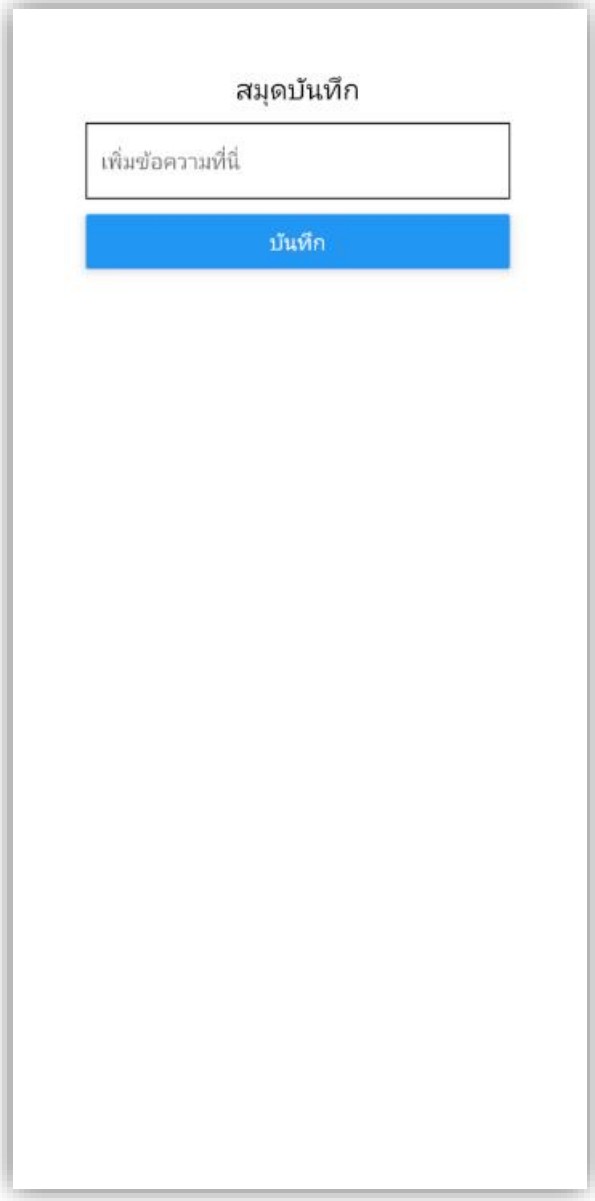
```
<FlatList
        data={list}                          // ข้อมูลอะเรย์ ที่ต้องการแสดงผล
        keyExtractor={                       // กำหนดฟังก์ชัน โดยคืนค่ามาเป็นคีย์ที่ไม่ซ้ำกัน (เป็นข้อความ)
          (item) => item.key
        }
        renderItem={ ( {item} ) => {         // กำหนดฟังก์ชัน โดยคืนค่าเป็นคอมโพเนนต์
          return <Text>{item}</Text>         // สำหรับแสดงข้อมูลแต่ละรายการใน FlatList
          }
        }
/>
```

```
export default function Example03(){
    const students = [
        {id: 1, name: 'Luffy’}, {id: 2, name: 'Zoro'}, {id: 3, name: 'Nami'},
        {id: 4, name: 'Usopp'}, {id: 5, name: 'Sanji'}, {id: 6, name: 'Chopper'},
        {id: 7, name: 'Nico Robin'}, {id: 8, name: 'Franky'}, {id: 9, name: 'Brook'}];
    return (
        <SafeAreaView style={styles.container}>
            <FlatList
                data={students}
                keyExtractor={(item)=> item.id}
                renderItem={({item}) => {
                    return (
                        <View style={styles.child}>
                            <Text>{item.name}</Text>
                        </View> )
                }}
            />
        </SafeAreaView>
    ); }
```



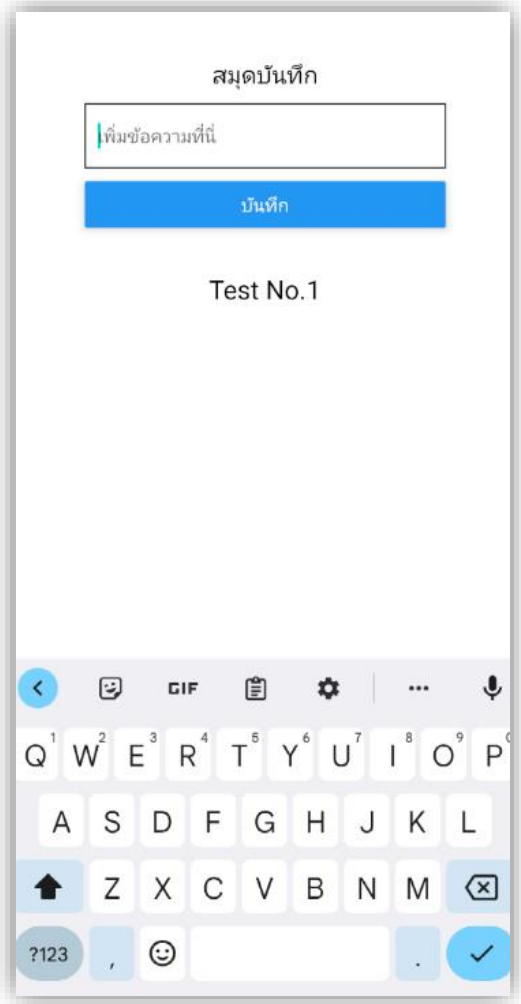
Lab 3.1

- ให้นศ.ทดลองสร้างคอมโพเนนต์ที่แสดงหน้าจอดังรูป
- เมื่อพิมพ์ข้อความ จะแสดงในส่วน TextInput
- เมื่อกด ‘บันทึก’ ข้อความที่พิมพ์ไปข้างต้น จะถูกนำมาแสดงในส่วนแสดงผลด้านล่าง และข้อความที่แสดงในส่วน TextInput จะหายไป
- นศ.สามารถทดลองเพิ่มข้อมูลไปเรื่อยๆ ได้
- เมื่อข้อมูลมีปริมาณมากกว่าแสดงได้ในหน้าจอ นศ.สามารถ scroll เพื่อดูข้อความด้านล่างได้
- ดัดแปลงจากตัวอย่างหน้า 10
  - นศ.อาจสร้าง state มาเพื่อเก็บข้อความที่จะแสดงผลด้านล่างเพิ่มเติม
  - นศ.อาจทำการกำหนดค่า state นั้น เมื่อมีการกดปุ่ม (ตัวอย่างหน้า 7)



Lab 3.1 (ต่อ)


สมุดบันทึก
Test No.1
บันทึก


สมุดบันทึก
พิมพ์ข้อความที่นี่
บันทึก
Test No.1


สมุดบันทึก
Test No.2
บันทึก
Test No.1


สมุดบันทึก
พิมพ์ข้อความที่นี่
บันทึก
Test No.1
Test No.2

Lab 3.2

- ให้นศ.ทดลองดัดแปลงโปรแกรม Lab 2.2 ให้ใช้ FlatList ในการแสดงผล
- กำหนดให้ เก็บข้อมูลเกี่ยวกับหลักสูตรในรูปแบบตัวแปรอะเรย์
- สามารถเลื่อนหน้าจอเพื่อแสดงข้อมูลส่วนที่เหลือได้

เมื่อเลื่อนหน้าจอ