1

I. ES6 (ECMAScript6)

Keyword `let` and `const`

- let to declare variables

```
for (let i = 0; i < 5; i++) {
    console.log(i); // 0,1,2,3,4
}
console.log(i); // undefined
```

- const define the constants

```
const PI = 3.14;
console.log(PI); // 3.14

PI = 10; // error
```


2

I. ES6 (ECMAScript6)

- Loop `for of`
  - Loop through arrays or loop through objects easily

```
let letters = ["a", "b", "c"];
for (let letter of letters) {
    console.log(letter);
}
```

- Arrow Function: The arrow syntax (=>) is a function abbreviation.

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
                => { return expression; }
```

- Parentheses are optional when there is only one parameter name:

```
(singleParam) => { statements }
singleParam => { statements }
```

- The parameter list for a function without parameters must be written with a pair of parentheses:

```
() => { statements }
```


I. ES6 (ECMAScript6)

- Arrow Function

```
// Function Expression
var sum = function(a, b) {
    return a + b;
}
console.log(sum(2, 3)); // 5
// Arrow function
var sum = (a, b) => a + b;
console.log(sum(2, 3)); // 5
```

- Default value for parameter

```
function sayHello(name = "A") {
    var name = name;
    return `Xin chào ${name}!`;
}
console.log(sayHello()); // Xin chào A!
console.log(sayHello('B')); // Xin chào B!
```

I. ES6 (ECMAScript6)

- Rest Parameters (Or Rest Operator)
  - Pass some arbitrary parameters to the function as an array.
  - Add to front operator parameters...

```
function sortNumbers(...numbers) {
    return numbers.sort();
}
console.log(sortNumbers(3, 5, 7));
console.log(sortNumbers(3, 5, 7, 1, 0));
```


5

II. UI Components

Basic components of React Native:

https://facebook.github.io/react-native/docs/components-and-apis.html

- Basic Components
- User Interface
- List View

- Most common
  - View
  - Text
  - Image
  - TextInput
  - ScrollView
  - StyleSheet
- UI
  - Button
  - Switch
  - Picker
  - Slider
- On screen list views
  - FlatList
  - SectionList

- Components shared by the community:

  http://www.awesome-react-native.com/#components

6

II. UI Components

<View>: Similar to HTML , there is no default interface.

Also often used for laying out child components.

Prop Name	Description	Example
`style`	Defines the style of the view.	`<View style={{ flex: 1 }} />`
`onLayout`	Callback when the view's layout changes.	`<View onLayout={handleLayoutChange} />`
`accessible`	Indicates whether the view is accessible.	`<View accessible={true} />`
`accessibilityLabel`	Specifies a label for accessibility services.	`<View accessibilityLabel="Label" />`
`testID`	Used for testing to identify the view.	`<View testID="test-view" />`
`hitSlop`	Extends the touchable area by adding padding.	`<View hitSlop={{ top: 10, left: 10 }} />`
`onAccessibilityTap`	Callback when tapped by an accessibility service.	`<View onAccessibilityTap={handleTap} />`

II. UI Components

<Text> - Display text to the screen. They can be nested

Prop Name	Description	Example
`style`	Defines the style of the text.	`<Text style={{ fontSize: 16 }} />`
`children`	The text content to be displayed.	`<Text>Hello, World!</Text>`
`numberOfLines`	Limits the number of lines displayed.	`<Text numberOfLines={2}>Long text...</Text>`
`ellipsizeMode`	Defines how text is truncated (if needed).	`<Text ellipsizeMode="tail">Long text...</Text>`
`adjustsFontSizeToFit`	Automatically adjusts font size.	`<Text adjustsFontSizeToFit={true}>Resizable Text</Text>`
`onPress`	Callback when the text is pressed.	`<Text onPress={handlePress}>Clickable Text</Text>`
`allowFontScaling`	Controls whether text can be scaled.	`<Text allowFontScaling={false}>Fixed Size Text</Text>`

```
<Text style={{ fontWeight: 'bold' }}>
  This is some bold text.
  <Text style={{ fontStyle: 'italic' }}>
    This is some bold and italic text.
  </Text>
</Text>
```

This is some bold text.*This is some bold and italic text.*

II. UI Components

<Image> - Display the image on the screen.

Can render images locally and network

- In React Native, you can specify separate images for each platform by specifying image.ios.png and image.android.png

Prop	Description
`source`	The source of the image (usually a URL, local file, or require).
`resizeMode`	Controls how the image is resized within the specified dimensions.
`style`	Custom styles to apply to the image component.
`blurRadius`	Applies a blur effect to the image.
`onLoad`	A callback function when the image is loaded successfully.
`onLoadEnd`	A callback function when the image loading ends, either in success or failure.
`onLoadStart`	A callback function when the image loading starts.
`onError`	A callback function when an error occurs while loading the image.
`defaultSource`	An image source to use when the main source fails to load.
`loadingIndicatorSource`	An image source to display as a loading indicator.
`fadeDuration`	The duration of the fade-in transition when the image loads.

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

```
<Image style={{ width: 150, height: 200 }}
source={{ uri: 'http://via.placeholder.com/150x200' }} />
```


II. UI Components

<TextInput> - Allow users to enter text.

Prop	Description
`value`	The value of the input field (controlled component).
`defaultValue`	The initial value of the input field (uncontrolled component).
`placeholder`	Placeholder text displayed when the input is empty.
`placeholderTextColor`	The color of the placeholder text.
`onChangeText`	A callback function called when the text input's text changes.
`secureTextEntry`	Indicates whether the text input should be a secure (password) input.
`autoCapitalize`	Controls capitalization of the text (none, words, sentences, characters).
`autoCorrect`	Controls automatic correction of the text.
`keyboardType`	The type of keyboard to display (default, numeric, email, etc.).
`multiline`	Allows multiple lines of input (boolean).
`numberOfLines`	The number of lines to display when `multiline` is `true`.
`maxLength`	Limits the maximum number of characters that can be entered.
`returnKeyType`	The type of return key to display on the keyboard (done, go, search, etc.).
`onBlur`	A callback function when the input field loses focus.
`onFocus`	A callback function when the input field gains focus.
`onSubmitEditing`	A callback function when the return key is pressed.
`editable`	Specifies whether the input can be edited (boolean).
`style`	Custom styles to apply to the text input component.
`onEndEditing`	A callback function when editing is finished.

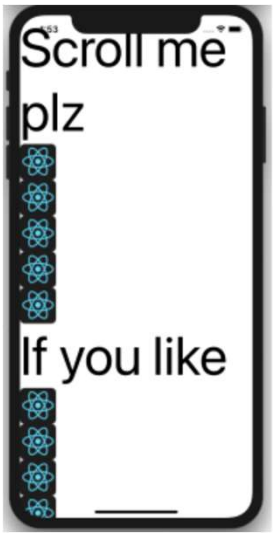
```
<View style={{ padding: 10, flex: 1, justifyContent: 'center' }}>
  <TextInput
    style={{ height: 40 }}
    placeholder="Type here!"
    onChangeText={(text) => setValue({ text })}
  />
</View>
```

II. UI Components

ScrollView is a common scrolling container that can store multiple components and views

```
const logo = {
  uri: 'https://reactnative.dev/img/tiny_logo.png',
  width: 64,
  height: 64
};

export default App = () => (
  <ScrollView>
    <Text style={{ fontSize: 96 }}>Scroll me plz</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Text style={{ fontSize: 96 }}>If you like</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Text style={{ fontSize: 96 }}>Scrolling down</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    …
  </ScrollView>
);
```



II. UI Components

ScrollView

Prop	Description
`horizontal`	Indicates whether the `ScrollView` should scroll horizontally (boolean).
`contentContainerStyle`	Style for the content container within the `ScrollView`.
`showsHorizontalScrollIndicator`	Controls the visibility of the horizontal scroll indicator (boolean).
`showsVerticalScrollIndicator`	Controls the visibility of the vertical scroll indicator (boolean).
`onScroll`	A callback function called when the `ScrollView` is scrolled.
`scrollEnabled`	Indicates whether scrolling is enabled (boolean).
`keyboardShouldPersistTaps`	Determines whether taps outside the `ScrollView` dismiss the keyboard.
`contentOffset`	The offset of the content in the `ScrollView`.
`contentInset`	The padding around the content in the `ScrollView`.
`decelerationRate`	Controls the rate of deceleration after the user stops scrolling.
`scrollEventThrottle`	The rate at which the `onScroll` event is fired.
`bounces`	Controls whether the `ScrollView` bounces when reaching the edge (boolean).
`pagingEnabled`	Enables paging behavior, where the content snaps to the center of the `ScrollView`.

II. UI Components

ListView

React Native provides components for presenting lists of data, using **FlatList** or **SectionList**

FlatList component displays scrolling list

FlatList components require two props:

- **Data**: the source of information for the list
- **renderItem**: the function receives an item from the source and returns the component that is Display format

https://reactnative.dev/docs/flatlist

II. UI Components

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

const FlatListBasics = () => { Example (II)
        return (
                <View style={styles.container}>
                <FlatList
                data={[
                        { key: 'Devin' },
                        { key: 'Dan' },
                        { key: 'Dominic' },
                        { key: 'Jackson' },
                        { key: 'James' },
                        { key: 'Joel' },
                        { key: 'John' },
                        { key: 'Jillian' },
                        { key: 'Jimmy' },
                        { key: 'Julie' },
                ]}
                renderItem={({ item }) =>
                <Text style={styles.item}>{item.key}</Text>}
                />
                </View>
        );
}
export default FlatListBasics;
```

II. UI Components

SectionList: An effective way to represent fragmented data

```
const DATA = [
  {
    title: 'Main dishes',
    data: ['Pizza', 'Burger', 'Risotto'],
  },
  {
    title: 'Sides',
    data: ['French Fries', 'Onion Rings', 'Fried Shrimps'],
  },
  {
    title: 'Drinks',
    data: ['Water', 'Coke', 'Beer'],
  },
  {
    title: 'Desserts',
    data: ['Cheese Cake', 'Ice Cream'],
  },
];
```

```
const App = () => (
  <SafeAreaView style={styles.container}>
    <SectionList
      sections={DATA}
      keyExtractor={(item, index) => item + index}
      renderItem={({item}) => (
        <View style={styles.item}>
          <Text style={styles.title}>{item}</Text>
        </View>
      )}
      renderSectionHeader={({section: {title}}) =>
        <Text style={styles.header}>{title}</Text>
      )}
    />
  </SafeAreaView>
);
```

https://reactnative.dev/docs/sectionlist

II. UI Components

Button: A basic component for displaying a button, Button has 2 important props: title and onPress

```
const App = () => (
  <SafeAreaView style={styles.container}>
    <View>
      <View style={styles.fixToText}>
        <Button
          title="Left button"
          onPress={() => Alert.alert('Left button pressed')}
        />
        <Button
          title="Right button"
          onPress={() => Alert.alert('Right button pressed')}
        />
      </View>
    </View>
  </SafeAreaView>
);
```

II. UI Components

TouchableOpacity:

- Can create buttons from Text and TouchableOpacity
- Allows changing the opacity of an element when touched

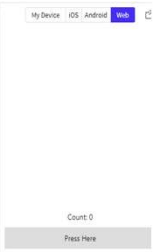
```
import React, { useState } from "react";
import { StyleSheet, Text, TouchableOpacity, View } from "react-native";

const App = () => {
  const [count, setCount] = useState(0);
  const onPress = () => setCount(prevCount => prevCount + 1);

  return (
    <View style={styles.container}>
      <View style={styles.countContainer}>
        <Text>Count: {count}</Text>
      </View>
      <TouchableOpacity
        style={styles.button}
        onPress={onPress}
      >
        <Text>Press Here</Text>
      </TouchableOpacity>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    paddingHorizontal: 10
  },
  button: {
    alignItems: "center",
    backgroundColor: "#DDDDDD",
    padding: 10
  },
  countContainer: {
    alignItems: "center",
    padding: 10
  }
});

export default App;
```



II. UI Components

- **TouchableHighlight** The background of the view will be darkened when the user clicks the button.
- **TouchableNativeFeedback** on Android to display ripples as the default effect with Material style in response to user touch.
- **TouchableOpacity** can be used to provide feedback by reducing button opacity, allowing the background to be seen through while the user presses.
- **TouchableWithoutFeedback** Don't want users to see any feedback displayed

II. UI Components

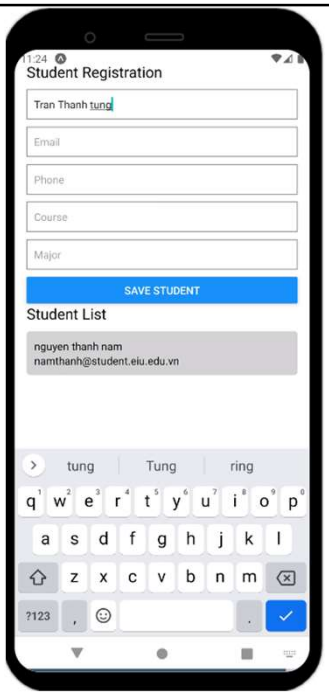
- Example: https://snack.expo.dev/@taitv/touchable

II. UI Components

- Other components
  - **Picker**
  - **SafeAreaView**
  - ActivityIndicator
  - **KeyboardAvoidingView**
  - **Modal**
  - **Pressable**
  - **RefreshControl**
  - **StatusBar**
  - **VirtualizedList**

Exercises

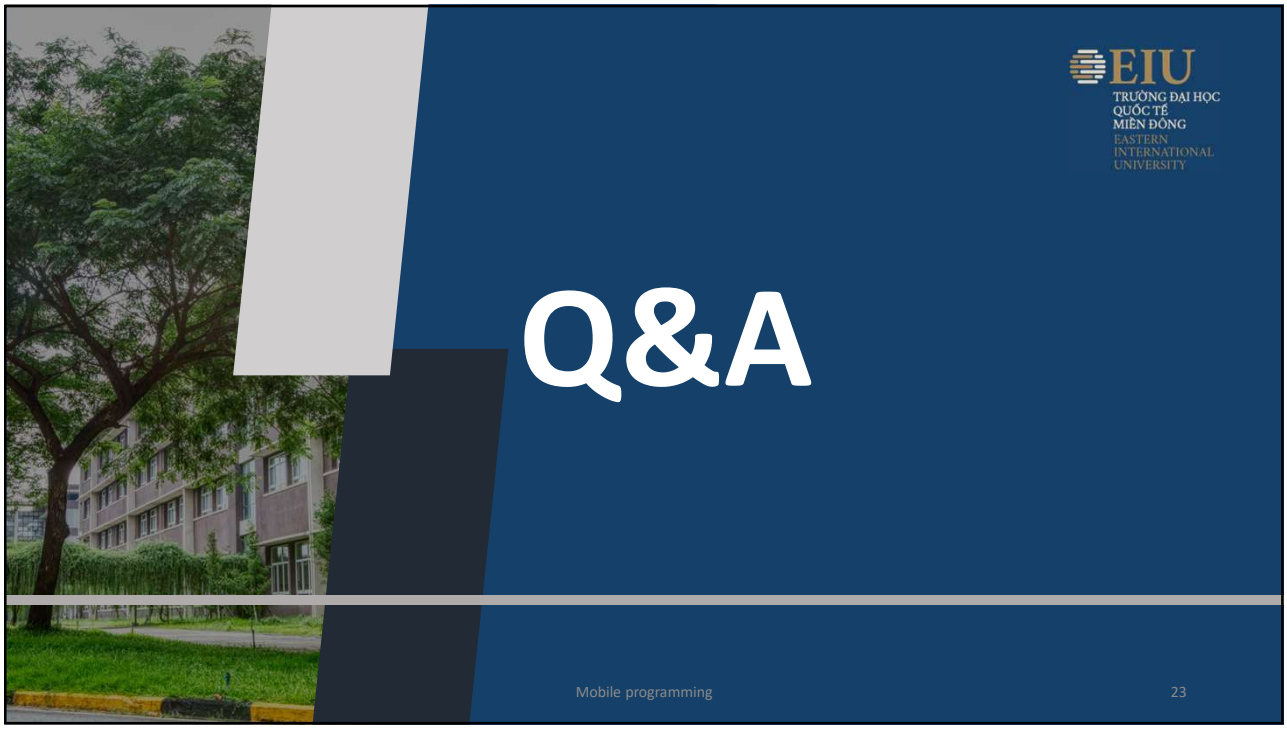
Use https://snack.expo.dev/ **to design the interface as shown**



Exercises

- https://snack.expo.dev/@taitv/studentform

https://snack.expo.dev/@taitv/exercise-1


EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY
Q&A
Mobile programming
23

23