Faculty of Science and Engineering
Department of Computing

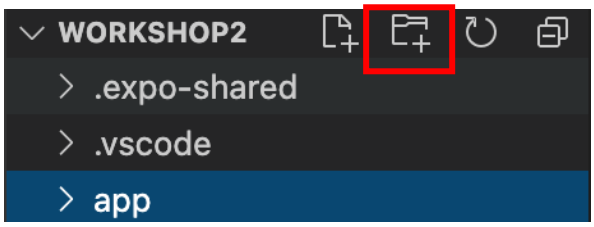
COMP3130 Mobile Application Development

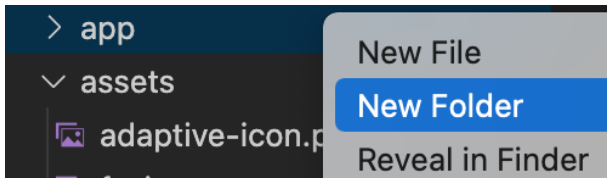
Workshop Week 3
Understanding core components, layout and styles

By the end of this workshop, you should be able to do the following

1. Change background colour of the screen
2. Change the style of the text (different colour, bold, italic style, size: 18) https://www.w3schools.com/colors/colors_picker.asp
3. onClick Event (Write an inline function and replace that with a regular function)
4. Embed images (local)
5. Embed images (network)
6. Work on interaction with images. When the image is clicked
    a. It should "blink" once
    b. it should display "Image clicked" on to the console
7. Create a button when clicked displays a message in the alert box
8. Create a custom Alert box and handle clicks
9. Understand flex-box and alignment

Suggestion:
Create a new project called Workshop2 and use that to work on the questions. Use VS Code Editor to create a new folder called app which will hold all our files.
Right click on app folder and create a new folder and call it screens. This is where we will be placing all the files for this workshop. Since this is your first workshop, keep editing the App.js screen and store the solution of each of the screens in this screen folder. Next week, you will learn how to organise your content





Screen-1:

Change the background colour of the screen to a colour of your choice

https://www.w3schools.com/colors/colors_picker.asp

```
App.js > styles > container
1   import { StatusBar } from 'expo-status-bar';
2   import React from 'react';
3   import { StyleSheet, Text, View } from 'react-native';
4
5   export default function App() {
6
7     return (
8       <View style={styles.container}>
9         <Text>Open up App.js to start working on your app!</Text>
10        <StatusBar style="auto" />
11      </View>
12    );
13  }
14
15  const styles = StyleSheet.create({
16    container: {
17      flex: 1,
18      backgroundColor: '#fff',
19      alignItems: 'center',
20      justifyContent: 'center',
21    },
22  });
23
```

As you can see, the <View> tag is styled using the container element in the styles.

```
<View style={styles.container}>
```

Any style that the View has is defined here such as the flexible view and the background colour. At the moment, the background colour is white. Change it to a colour of your choice

New code

```
backgroundColor: '#000',
```

Save and re-load the screen and see the changes

To define the style for the text, create an style object in the text tag and linking it to the Stylesheet

```
<Text style={styles.para1}>
```

In the Stylesheet, create an object called para1 and define the props and values

```
para1: {
  color:'#ff0000',
  fontStyle:'italic',
  fontSize:40,
  fontWeight:'bold',
  textTransform:'uppercase',
}
```

To enable URL linking, we need to import a module/component called Linking

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

We need an onPress event to handle the function call.

```
<Text style={styles.para1} onPress={() =>Linking.openURL('http://netflix.com.au') } > Netflix
</Text>
```

onPress = { } --- you need to write the body of the function here

onPress has no parameters – hence we have an empty arguments brackets

Linking.openURL('link') is used to redirect to another webpage

This is an inline function declaration

To remove this and write it in a separate function and use a function call

```
const handleClick = () => Linking.openURL('http://netflix.com.au');
```

```
<Text style={styles.para1} onPress={handleClick}>Netflix</Text>
```

Copy the codes and add it to a file in app/screens folder

Screen-2:

For using Images in the app, we need to import Images module/component

Local Images:

For local images, it is a common practice to store them in the assets folder. Download an image of your choice (Netflix logo has been provided for you on iLearn; Download and add this to your assets folder in MyFirstProject)

After the <Text> tag, add this

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

<Image> is your tag to embed an image

We use source property to call a function called require and pass the location of the image

To change the dimensions, you can use the style

```
<Image style={styles.logo} source={require('./assets/netflix_logo.png')}/>
```

In the StyleSheet, add

```
logo:{
  marginTop:50,
  width:100,
  height:100,
}
```

Copy the codes and add it to a file in app/screens folder

Screen-3:

Network Images:
For network images, we need to know the link of the image.
Netflix logo can be found here: https://logodix.com/logo/786654.jpg

Require is a function used for local images. For network images, we need an property called uri and the value would be the link

```
<View style={styles.container}>
  <Image style={styles.logo} source={{uri: "https://logodix.com/logo/786654.jpg"}}/>
</View>
```

If you do not set the height and width for network images, they won't appear on the screen. This is really important.
Therefore in the styles logo, you can set up the property and value

```
logo:{
  width:200,
  height:100,
}
```

This should provide an output like the desired one. Clean your code to delete all the unused modules

For image to "blink" when clicked,

Import "TouchableOpacity" component and nest the image tag within this component

```
import { StyleSheet, View, Image,TouchableOpacity} from 'react-native';
```

```
<TouchableOpacity>

  <Image
    style={styles.logo}
    source={{uri: "https://logodix.com/logo/786654.jpg"}}/>

</TouchableOpacity>
```

Now, save and try clicking on the image. It should display the background and come back (appears to blink once when clicked)

Image tag does not have an onPress event handler. Import a component called "TouchableWithoutFeedback" and nest the image tag within this component

```
import { StyleSheet, View, Image,TouchableWithoutFeedback} from 'react-native';
```

```
<TouchableWithoutFeedback>

  <Image
    style={styles.logo}
    source={{uri: "https://logodix.com/logo/786654.jpg"}}/>

</TouchableWithoutFeedback>
```

Write an inline function that prints to the console

```
<TouchableWithoutFeedback onPress={()=>console.log('Image Clicked')}>
```

You should see the message in the terminal

```
Finished building JavaScript bundle in 172ms.
Running application on Android SDK built for x86.
Image Clicked
```

Copy the codes and add it to a file in app/screens folder

Screen-4:

To add a button on the screen, import Button and add the tag into the <View> after your <TouchableOpacity> nested tags

```
import { StyleSheet, View, Image,TouchableOpacity, Button} from 'react-native';
```

```
<Button
        color="#ee3523"
        title="ENTER SITE"
        onPress={()=>alert('Site under Construction')}
      />
```

Create a button with a title (the message that would get displayed on the screen) and a colour. When you click the button, onPress event has an inline function written that calls the alert function and displays the parameter in the message, when the app is run. Save and check your code. Use marginBottom to provide some space between the image and the button.

```
 logo:{
   width:200,
   height:100,
   marginBottom:100,
 }
```

Copy the codes and add it to a file in app/screens folder

Screen-5:

To create a custom alert message, import Alert api (Not a component, but an object instead)

```
import { StyleSheet, View, Image,TouchableOpacity, Button, Alert} from 'react-native';
```

In the button OnPress event handler, let us write the function, but instead of using alert function like the example from Screen-4, let us use Alert.alert which should display the parameters Alert.alert("Title","Message",[{text:"Display Text for Button1"}, {text:"Display Text for Button2"}]

In our example, you will have something like
Alert.alert ("G'day", "Do you love Netflix?", [{text:"Yes"}, {text:"No"}])

```
<Button
        color="#ee3523"
        title="ENTER SITE"
        onPress={()=>Alert.alert("G'day","Do you love Netflix?",
          [{text:"Yes"},
          {text:"No"}])}
      />
```

This will create a custom Alert box with two buttons "Yes", "No". However, nothing will happen when you click on them since the onPress for these values have not been written. Define onPress property values after the text values

```
    <Button
        color="#ee3523"
        title="ENTER SITE"
        onPress={()=>Alert.alert("G'day","Do you love Netflix?",
          [{text:"Yes", onPress: ()=> alert("Yay!!!!")},
          {text:"No", onPress: ()=> alert("Oh No!")}])}
      />
```

Copy the codes and add it to a file in app/screens folder

Flexbox Screen-1

Delete all codes and just have view and stylesheet in import statement

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

Now within the container view, create 3 empty views with respective styles for each of them

```
  <View style={styles.container}>
    <View style={styles.box1}/>
    <View style={styles.box2}/>
    <View style={styles.box3}/>
  </View>
```

Your styles object must have the following properties

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
  },
  box1: {
    backgroundColor: '#221154',
    width:100,
    height:100,
  },
  box2: {
    backgroundColor: '#abdbe3',
    width:100,
    height:100,
  },
  box3: {
    backgroundColor: '#eab676',
    width:100,
    height:100,
  }
});
```

Copy the codes and add it to a file in app/screens folder

Flexbox Screen-2

For the second screen, the only difference is how the views are arranged. By default, the default/the main axis is your "Column axis". To change the axis to row, you will have to add a styling property to your container

```
  container: {
    flex: 1,
    flexDirection: 'row', // by default, it is column
    backgroundColor: '#fff',
  },
```

If you change the flex direction to row, it will change the axis of arrangement.
Change flex direction to other values such as row-reverse, column-reverse and see what happens

Flexbox Screen-3

For the third screen, the contents are arranged in the center. The styling property for this is 'justfyContent' which arranges content based on the flex Direction axis specified in flex Direction. By default the value of justifyContent is 'flex-start' which arranges the content from the left most position on the screen

```
container: {
  flex: 1,
  flexDirection: 'row',
  justifyContent: 'center',
  backgroundColor: '#fff',
},
```

Copy the codes and add it to a file in app/screens folder

Flexbox Screen-4

For the fourth screen, the contents are arranged in with spaces between them. The styling property for 'justfyContent' is 'space-between' which arranges the content with maximum spacing

```
container: {
  flex: 1,
  flexDirection: 'column',
  justifyContent: 'space-between',
  backgroundColor: '#fff',
},
```

Copy the codes and add it to a file in app/screens folder

Flexbox Screen-5

For the fifth screen, the content are arranged in column which means that's the Flex Direction. Also, it is in center in both main and cross axis which means justifyContent and alignItems properties must be set to center. This can be written using

```
container: {
  flex: 1,
  flexDirection: 'column',
  justifyContent: 'center',
  alignItems: 'center',
  backgroundColor: '#fff',
},
```

Copy the codes and add it to a file in app/screens folder

Flexbox Screen-6

For the sixth screen, the content are arranged in column which means that's the Flex Direction. Also, box1 and box3 are stretched to the end. To make this happen, remove the width/comment the width property in both box1 and box3

```
box1: {
  backgroundColor: '#221154',
  height:100,
},
box2: {
  backgroundColor: '#abdbe3',
  width:100,
  height:100,
},
box3: {
  backgroundColor: '#eab676',
  height:100,
}
```

In the container style, the contents start from the left-most point. This means that the justifyContent must be set to 'flex-start' and the alignItems must be 'center'. Box1 and 3 are stretched to the end. To do this, alignItems must be set to 'stretch'.

```
container: {
  flex: 1,
  flexDirection: 'column',
  justifyContent: 'flex-start',
  alignItems: 'stretch',
  backgroundColor: '#fff',
},
```

Copy the codes and add it to a file in app/screens folder

Flexbox Screen-7

For the seventh screen, there are 3 different height for the boxes.

```
box1: {
  backgroundColor: '#221154',
  width:100,
  height:100,
},
box2: {
  backgroundColor: '#abdbe3',
  width:100,
  height:200,
},
box3: {
  backgroundColor: '#eab676',
  width:100,
  height:300,
}
```

Contents are arranged in row which means that's the value for Flex Direction. Also, all the boxes are in the center of the screen which means both justifyContent and alignItems values are center

```
container: {
  flex: 1,
  flexDirection: 'row',
  justifyContent: 'center',
  alignItems: 'center',
  backgroundColor: '#fff',
},
```

Copy the codes and add it to a file in app/screens folder

Flexbox Screen-8

For the 8th screen, all the boxes are of the same height and width; you can use the align property to help with the alignment is something called alignSelf and this would help in arranging items. The container code for this would be

```
container: {
  flex: 1,
  flexDirection: 'row',
  justifyContent: 'center',
  alignItems: 'center',
  backgroundColor: '#fff',
},
```

```
 box1: {
  backgroundColor: '#221154',
  width:100,
  height:100,
  alignSelf:"flex-end",
},
box2: {
  backgroundColor: '#abdbe3',
  width:100,
  height:100,
},
box3: {
  backgroundColor: '#eab676',
  width:100,
  height:100,
  alignSelf:"flex-start",
}
```

Copy the codes and add it to a file in app/screens folder