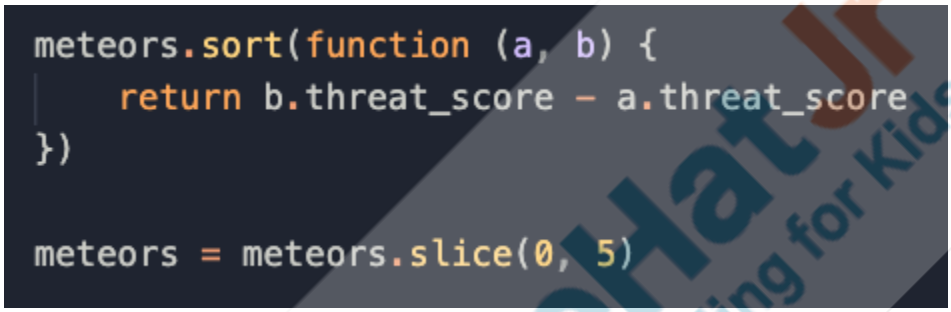
Topic	METEOR SCREEN 2	
Class Description	**Students learn to Create a carousel effect from FlatLists.**	
Class	**C80**	
Class time	**45 mins**	
Goal	• Display the data of meteors using the FlatList in carousel effect horizontally. • Advanced styling.	
Resources Required	• Teacher Resources: ○ Visual Code Studio Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen • Student Resources: ○ Visual Code Studio Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen	
Class structure	**Warm Up** **Teacher & Student Collaborative Activity** **Wrap up**	**5 mins** **30 mins** **5 min**
Credits	• **Open source API for getting updates on meteors offered by Nasa's open repository APIs.**	
WARM UP SESSION - 5 mins		
Teacher starts slideshow  **from slides 1 to 13** Refer to speaker notes and follow the instructions on each slide.		

Teacher Action	Student Action
Hey <student name>. How are you? It's great to see you! Are you excited to learn something new today? **Run the presentation from slide 1 to slide 3.** **Following are the warm up session deliverables:** ● Connecting students to the previous class.	**ESR**: Hi, thanks, yes I am excited about it! Click on the slide show tab and present the slides.
QnA Session	
Question	**Answer**
Which NASA API we are using in our app? A. APOD B. Asteroids NeoWs C. Earth D. EPIC	**B**
Why are we using the threat score of meteors in our app? A. based on this threat score, we can take the top 5 meteors and display just those B. to reduce the number of meteors we are displaying because there are many meteors which are displayed C. we are calculating a threat score for a meteor, which will tell how threatful it is to Earth D. all of the above	**D**
Continue the warm up session	
Teacher Action	**Student Action**

Run the presentation from slide 4 to slide 13 to set the problem statement. **Following are the warm up session deliverables:** • Talk about the different sizes and threats of the meteor	Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.
Teacher ends slideshow 	
TEACHER & STUDENT COLLABORATIVE ACTIVITY - 30 mins	
Teacher Initiates Screen Share	
ACTIVITY • **Get the API key by signing up on the official website.** • **Write a function to get data from an API.**	
Teacher Action	**Student Action**
<The teacher opens the code from the previous class. Refer to Teacher Activity 6.> *<Teacher should take note that this is a collaborative class. The student is expected to code with Teacher's guidance.>*	*Student refers to Student Activity 4 for previous class code.*
Now that we already have the threat score calculated for the meteors, we can observe that we still have data for a lot of meteors! It will be feasible for us to display only the top 5 meteors that are going to be most threatful to the Earth. Can you tell me how we can get the top 5 most threatful meteors?	**ESR:** We can first sort the array in descending order based on the threat score of the

	meteor objects, and then take the first 5 with the ***slice()*** method.
Student-led Activity (with Teacher's help) Excellent! Let's quickly add the code to do that -	*Teacher helps the student in writing the code.*
 ``` meteors.sort(function (a, b) { return b.threat_score - a.threat_score }) meteors = meteors.slice(0, 5) ```	
Here, we are then sorting the objects inside the array based on their threat score in decreasing order and then we are finally taking the first 5 meteor objects.	
Let's change our code in the return statement to the following -	

```
meteors.sort(function (a, b) {
    return b.threat_score - a.threat_score
})
meteors = meteors.slice(0, 5)
return (
    <View style={styles.container}>
        <SafeAreaView style={styles.droidSafeArea} />
        <FlatList
            keyExtractor={this.keyExtractor}
            data={meteors}
            renderItem={this.renderItem}
            horizontal={true}
        />
    </View>
)
```

Here, we first have a <View> with styles of container. Inside it, we have a <SafeAreaView> like in previous screens to avoid any UI issues in different operating systems.

We then finally have a <FlatList>.

We will import the <FlatList> and the <SafeAreaView> from "react-native".

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

Don't forget the styles.

```
const styles = StyleSheet.create({
    container: {
        flex: 1
    },
    droidSafeArea: {
        marginTop: Platform.OS === "android" ? StatusBar.currentHeight : 0
    }
});
```

Now, we used a <FlatList>, but what is it?	**ESR:** Varied.
Consider a situation where you have an array with 1000 elements and you want to display all the data in the elements of the array in a similar way. Will you be creating the same element 1000 times? That's where <FlatList> comes into play! It takes an array of data and renders all the data in a similar way. In our case, since we have the objects of meteor data structured in a similar way inside an array, we can use a FlatList to display the meteor data. In this FlatList, we have passed the following - 1. keyExtractor - a way for FlatList to differentiate different elements from each other, basically a unique key; 2. data - the array whose elements we want to render; 3. renderItem - a function which defines what exactly needs to be rendered for all the data; and 4. horizontal - by default, it's false but since we want to build a carousel effect (which is always horizontal), hence, we kept it as true.	ESR: Varied.

Now, for the keyExtractor, we have passed **this.keyExtractor**, so let's add it outside the render() function -

```
getMeteors = () => {
    axios
        .get("https://api.nasa.gov/neo/rest/v1/feed?api_key=nAkq24DJ2dH
        .then(response => {
            this.setState({ meteors: response.data.near_earth_objects }
        })
        .catch(error => {
            Alert.alert(error.message)
        })
}

keyExtractor = (item, index) => index.toString();
```

Next, we will build the most important part of our screen, the **renderItem** function -

```
renderItem = ({ item }) => {
  let meteor = item
  let bg_img, speed, size;
  if (meteor.threat_score <= 30) {
     bg_img = require("../assets/meteor_bg1.png")
     speed = require("../assets/meteor_speed3.gif")
     size = 100
  } else if (meteor.threat_score <= 75) {
     bg_img = require("../assets/meteor_bg2.png")
     speed = require("../assets/meteor_speed3.gif")
     size = 150
  } else {
     bg_img = require("../assets/meteor_bg3.png")
     speed = require("../assets/meteor_speed3.gif")
     size = 200
```

```
  }
  return (
    <View>
      <ImageBackground source={bg_img} style={styles.backgroundImage}>
        <View styles={styles.gifContainer}>
          <Image source={speed} style={{ width: size, height: size, alignSelf:
"center" }}></Image>
          <View>
            <Text style={[styles.cardTitle, { marginTop: 400, marginLeft: 50
}]}>{item.name}</Text>
            <Text style={[styles.cardText, { marginTop: 20, marginLeft: 50
}]}>Closest to Earth -
{item.close_approach_data[0].close_approach_date_full}</Text>
            <Text style={[styles.cardText, { marginTop: 5, marginLeft: 50
}]}>Minimum Diameter (KM) -
{item.estimated_diameter.kilometers.estimated_diameter_min}</Text>
            <Text style={[styles.cardText, { marginTop: 5, marginLeft: 50
}]}>Maximum Diameter (KM) -
{item.estimated_diameter.kilometers.estimated_diameter_max}</Text>
            <Text style={[styles.cardText, { marginTop: 5, marginLeft: 50
}]}>Velocity (KM/H) -
{item.close_approach_data[0].relative_velocity.kilometers_per_hour}</Text>
            <Text style={[styles.cardText, { marginTop: 5, marginLeft: 50
}]}>Missing Earth by (KM) -
{item.close_approach_data[0].miss_distance.kilometers}</Text>
          </View>
        </View>
      </ImageBackground>
    </View>
  );
};
```

Let's have a walkthrough of this code.

First thing we do is define 3 variables, **bg_img, speed and size**. Now since our meteors may have different **threat_scores**, some having less than 30, some less than 75 and some even beyond that, we want to differentiate how they look in our UI.

For example, a meteor with a **threat_score above 75** (most dangerous) should look different from a meteor with a **threat_score below 75 (**maybe dangerous**).** A meteor with **threat_score even below 30 (**the least dangerous**)** should look completely different.

To differentiate between different types, we can make them have different sizes, speeds and backgrounds.

Based on the **threat_score** the meteor has, we have 3 different backgrounds and meteor gifs with 3 different speeds.

Therefore, based on the threat score, we are first deciding the background image, the gif that we want to use and the size of the gif for a particular meteor.

Next, we have the **return()** function. In this function, We have **<View>** with **<ImageBackground>** for our screen.

Inside this, we have a **<View>** for our **<Image>** component that displays the meteor gif and below that, the text values.

Note that the first **<Text>** has **marginTop** set to **400**. This is to have a nice space between the gif and the text.

The styling for the components is as follows -

```
const styles = StyleSheet.create({
  container: {
    flex: 1
  },
  droidSafeArea: {
    marginTop: Platform.OS === "android" ? StatusBar.currentHeight : 0
  },
  backgroundImage: {
    flex: 1,
    resizeMode: 'cover',
    width: Dimensions.get('window').width,
    height: Dimensions.get('window').height
```

```
},
titleBar: {
    flex: 0.15,
    justifyContent: "center",
    alignItems: "center"
},
titleText: {
    fontSize: 30,
    fontWeight: "bold",
    color: "white"
},
meteorContainer: {
    flex: 0.85
},
listContainer: {
    backgroundColor: 'rgba(52, 52, 52, 0.5)',
    justifyContent: "center",
    marginLeft: 10,
    marginRight: 10,
    marginTop: 5,
    borderRadius: 10,
    padding: 10
},
cardTitle: {
    fontSize: 20,
    marginBottom: 10,
    fontWeight: "bold",
    color: "white"
},
cardText: {
    color: "white"
},
threatDetector: {
    height: 10,
    marginBottom: 10
},
gifContainer: {
    justifyContent: "center",
```

```
    alignItems: "center",
    flex: 1
  },
  meteorDataContainer: {
    justifyContent: "center",
    alignItems: "center",

  }
});
```

With this, our meteor screen is complete! Run the code and check the output -

Teacher starts slideshow :Slide 13

Run the presentation for slide 13 to set the student activity context.	
Go ahead and share your screen with me.	*The student shares his/her screen, opens the Student Activity, and adds code to it.*

Teacher Stops Screen Share

<table>
<tr><th colspan="3">WRAP UP SESSION - 5 Mins</th></tr>
<tr><td colspan="3">Teacher starts slideshow  from slide 14 to slide 23</td></tr>
<tr><th colspan="2">Activity details</th><th>Solution/Guidelines</th></tr>
<tr><td colspan="2">Run the presentation from slide 14 to slide 23

Following are the warm up session deliverables:
• Explain the facts and trivia
• Next class challenge
• Project for the day
• Additional Activity</td><td>Guide the student to develop the project and share with us.</td></tr>
<tr><td></td><td>Let's quickly wrap up today's class.
What did we learn?</td><td>ESR:
We completed the meteor screen! We displayed the meteors using a FlatList with a carousel effect.</td></tr>
<tr><th colspan="3">Quiz time - Click on in-class quiz</th></tr>
<tr><th colspan="2">Question</th><th>Answer</th></tr>
<tr><td colspan="2">What does the keyExtractor prop of the FlatList do?

A. The keyExtractor assigns index to the items.
B. The keyExtractor is an array in which data is stored.
C. The keyExtractor takes an item from data and renders it into the list.
D. The keyExtractor renders items next to each other horizontally instead of stacking them vertically.</td><td>A</td></tr>
<tr><td colspan="2">Which property can get our FlatList to scroll from left to right?</td><td>A</td></tr>
</table>

A. horizontal={true} B. vertical={true} C. leftToRight={true} D. topToBottom={false}	
What does the FlatList component do? A. it takes an array of data and renders all the data in a similar way B. it renders the data in an image format C. it displays the text repeatedly D. none of the above	**A**
End the quiz panel	
FEEDBACK • **Appreciate the student for their efforts in the class.** • **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**	
Teacher Action	**Student Action**
Did you enjoy today's class?	**ESR:** Varied.
Amazing work today! You get a "hats-off".	*Make sure you have given at least 2 Hats Off during the class for:* 

In the next class, we will start working on a new app called the Storytelling App. It would be a social media like app for story sharing. See you in the next class.	
PROJECT OVERVIEW: **Stellar Stage 4** **Goal of the Project:** In Class 79, we have designed the meteors screen to show all the meteors that are going close to the Earth. In this project, you will create a similar "Daily Pic" screen for the Stellar App. *This is a continuation project of Projects-76, 77 & 78. Make sure to complete that one before attempting this one. **Story:** Jeff is impressed with the constellation's screen output; he wants to use a similar feature to highlight daily pictures received by NASA on space happenings. He would like you to create a separate screen for that. I am very excited to see your project solution and I know you will do really well. Bye Bye!	
Teacher ends slideshow 	

Teacher Clicks	✖ End Class

ADDITIONAL ACTIVITY	
Encourage the student to write reflection notes in their reflection journal using markdown. Use these as guiding questions: ● What happened today? ○ Describe what happened. ○ The code I wrote. ● How did I feel after the class? ● What have I learned about programming and developing games? ● What aspects of the class helped me? What did I find difficult?	*The student uses the markdown editor to write their reflections in a reflection journal.*

Activity	Activity Name	Links
Teacher Activity 1	Nasa's website	https://api.nasa.gov/
Teacher Activity 2	API URL	https://api.nasa.gov/neo/rest/v1/feed?api_key=DEMO_KEY
Teacher Activity 3	JSON Prettifier	https://jsonformatter.curiousconcept.com/
Teacher Activity 4	Reference code	https://github.com/pro-whitehatjr/C80_ISSTracker_TeacherReferenceCode
Teacher Activity 5	Teacher Aid	https://drive.google.com/file/d/1WA1BQff4dmgv5BInU3f_imk4vlpvAyMa/view?usp=sharing
Teacher Activity 6	Previous class code	https://github.com/pro-whitehatjr/C79_ISSTracker_TeacherReferenceCo

		de
Student Activity 1	Nasa's website	https://api.nasa.gov/
Student Activity 2	API URL	https://api.nasa.gov/neo/rest/v1/feed?api_key=DEMO_KEY
Student Activity 3	JSON Prettifier	https://jsonformatter.curiousconcept.com/
Student Activity 4	Previous class code	https://github.com/pro-whitehatjr/C79_ISSTracker_TeacherReferenceCode
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C80-withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/d45bf816-d93e-45d7-9552-819f8f1d92b2.pdf