Cross-Platform Mobile Application Development

Styling and Layout in ReactNative

HUMBER

WE ARE HUMBER

Agenda

- What is StyleSheet?
- What is Flex?
- Why Use Flex?
- Using Flexbox to develop responsive layouts.

style attribute

- With React Native, you style your application using JavaScript.
- All of the core components accept a prop named **style**.
- The style names and values usually match how CSS works on the web, except names are written using camel casing, e.g. backgroundColor rather than background-color.
- The style prop can be a plain old **JavaScript object**.
- You can also pass an array of styles.

- https://reactnative.dev/docs/style

StyleSheet object

- A StyleSheet is an abstraction similar to CSS StyleSheets.
- **StyleSheet.create()** is an identity function for creating styles.
- The main practical benefit of creating styles inside StyleSheet.create() is static type checking against native style properties.

- https://reactnative.dev/docs/stylesheet

- StyleSheet object can be created within individual screens/components.
- You can also create global stylesheet for consistent style.

What is global StyleSheet?

A centralized file containing reusable styles for React Native components.

- Promotes consistency across screens.
- Simplifies maintenance and updates.
- Enhances scalability with themes.

What is Flex?

- Flexbox is a **layout implementation** that React-Native uses to provide an efficient way for users to create UIs and control positioning.

- Essentially, the Flexbox model is a **box model that's flexible**! You have a box/container, and you have child elements within that box. Both the container and the child elements are flexible in how they're rendered.

What is Flex?

- Flex aims to give you an easy way to decide alignment, and distribution of space among items in a layout, even when their size isn’t known or is dynamic.
- With few exceptions, React Native Flex works just the same as CSS Flex.
- Experiment : https://flexbox.buildwithreact.com/

Why Use Flex?

- Flex is useful as it is consistent across different devices and screen sizes.
- It helps design **responsive user interfaces**.
- Items can be rearranged and moved around easily based on the available space.
- You can decide which element to show or hide depending on the screen size/device type.

Experimenting with Flexbox :

https://flexbox.buildwithreact.com/

https://yogalayout.com/playground/

Setting up Flex using the **flex** property – 1/2

- Unlike the usual way of defining “display: flex” in CSS, in React Native you use the keyword **flex** in your style and assign it an **integer value**.
- The **flex** value must be a **positive integer number**

- The flex value represents the ability of an element to change its size to fill the space of the parent container.
- The parent container will then consider all the flex values of its children and distribute the space accordingly.

Setting up Flex using the **flex** property – 2/2

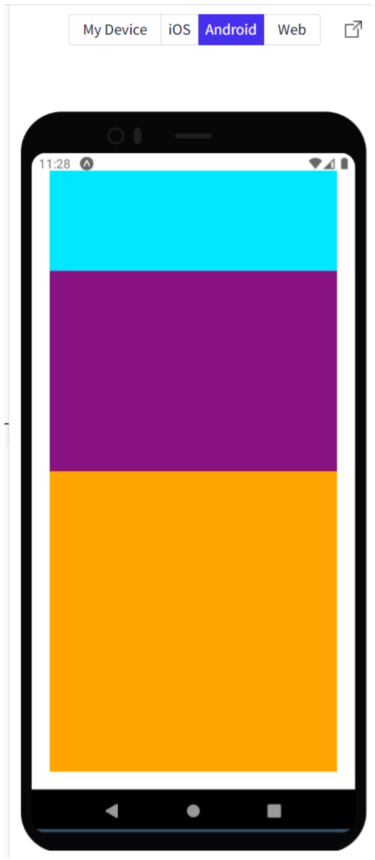
- In this example, the parent container, will sum up all the flex values of its children which is 6. Then gives Cyan 1/6, purple 2/6 an orange 3/6 of its available space.

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

const Flex = () => {
  return (
    <View style={styles.container}>
      <View style={{ flex: 1, backgroundColor: "cyan" }} />
      <View style={{ flex: 2, backgroundColor: "purple" }} />
      <View style={{ flex: 3, backgroundColor: "orange" }} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20
  },
});

export default Flex;
```



Flex Direction – 1/2

Flex direction is controlled using the **flexDirection** properly and it accepts four different settings :

- **row**: Aligns the children from left to right.
- **column**: Aligns children from top to bottom. This is the default value for **flexDirection**.
- **column-reverse**: Aligns children from bottom to top.
- **row-reverse:** aligns children from right to left.

Flex Direction is also referred to as the **main axis** or **primary axis**.

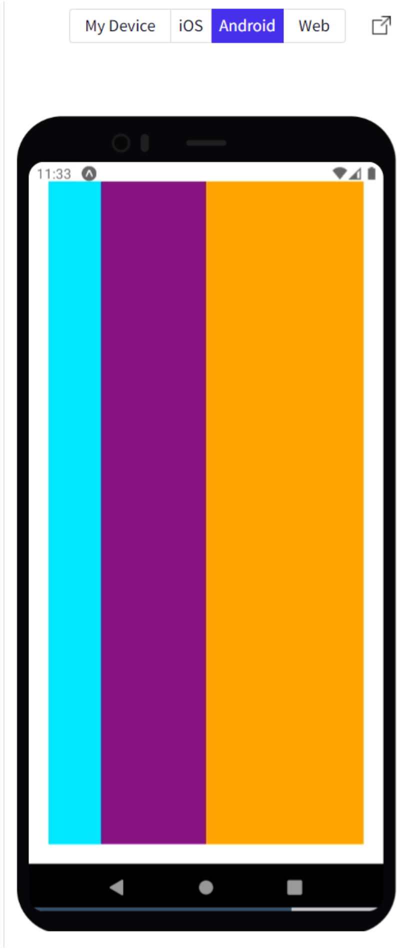
Flex Direction – 2/2

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

const Flex = () => {
  return (
    <View style={styles.container}>
      <View style={{ flex: 1, backgroundColor: "cyan" }} />
      <View style={{ flex: 2, backgroundColor: "purple" }} />
      <View style={{ flex: 3, backgroundColor: "orange" }} />
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    flexDirection:"row"
  },
});

export default Flex;
```



Layout Direction – 1/2

As the name implies, the Layout Direction refers to the direction in which the content is laid out. To manipulate the layout direction, you must use the **direction** property. There are two values for this property:

- **ltr**: This is the default value which stands for “left to right”, and it renders children and text from left to right.
- **rtl**: This value stands for “right to left” and will render the content from right to left. This is especially useful for languages that are written from right to left.

- **This is not supported by Android**.

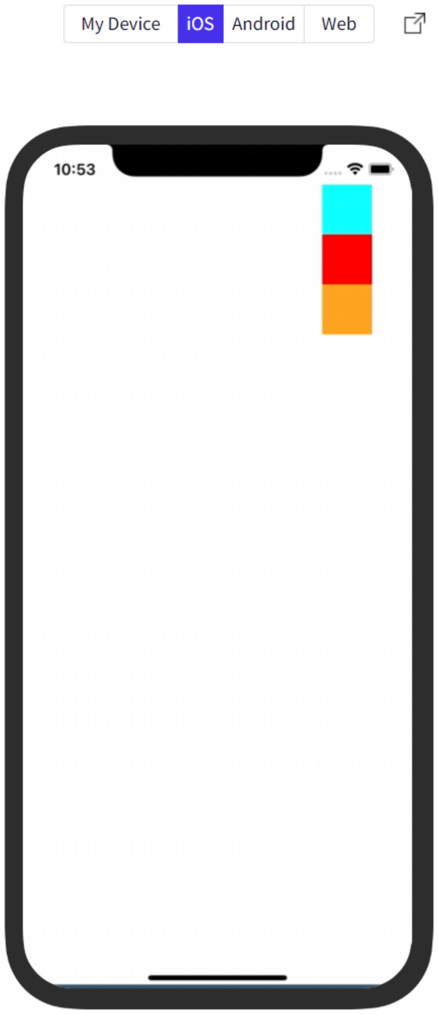
Layout Direction – 2/2

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

const Flex = () => {
 return (
  <View style={styles.container}>
   <View style={[styles.child, { backgroundColor: "cyan" }]} />
   <View style={[styles.child, { backgroundColor: "red" }]} />
   <View style={[styles.child, { backgroundColor: "orange" }]} />
  </View>
 );
};

const styles = StyleSheet.create({
 child: {
  width: 50, height: 50
 },
 container: {
  flex: 1,
  padding: 40,
  flexDirection: "column",
  direction: “rtl"
 },
});

export default Flex;
```



WE ARE HUMBER

Justifying Content – 1/3

- Justifying content is set using the **justifyContent** property and it defines how space is distributed between and around flex items along the **primary axis** of the container.
- Six options are available as values for justifying content:
  - **center**: centers the children within the parent container. The free space is distributed on both sides of the clustered group of children.
  - **flex-start**: It is the default value and groups the components at the beginning of the primary axis.
  - **flex-end:** it groups the items together at the end of the primary axis.

Justifying Content – 2/3

- **space-between**: evenly spaces the children along the primary axis. The <u>remaining space</u> will be distributed between the children.
- **space-around:** Works like **space-between**, with the exception that it also distributes the space to the beginning of the first child and the end of the last child.
- **space-evenly**: unlike space-around, which distributed the remaining space before the first child and after the last child, this makes sure that all spacings are equal, whether before or between the children.

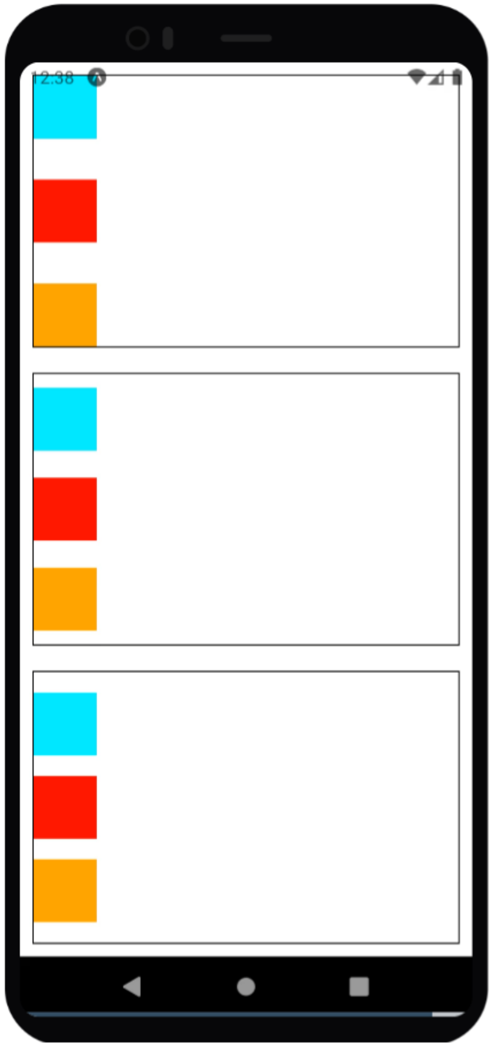
Justifying Content – 3/3

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

const Flex = () => {
  return (
    <>
     <View style={[styles.containerBase, styles.container1]}>
       <View style={[styles.child, { backgroundColor: 'cyan' }]} />
       <View style={[styles.child, { backgroundColor: 'red' }]} />
       <View style={[styles.child, { backgroundColor: 'orange' }]} />
     </View>
     <View style={[styles.containerBase, styles.container2]}>
       <View style={[styles.child, { backgroundColor: 'cyan' }]} />
       <View style={[styles.child, { backgroundColor: 'red' }]} />
       <View style={[styles.child, { backgroundColor: 'orange' }]} />
     </View>
      <View style={[styles.containerBase, styles.container3]}>
       <View style={[styles.child, { backgroundColor: 'cyan' }]} />
       <View style={[styles.child, { backgroundColor: 'red' }]} />
       <View style={[styles.child, { backgroundColor: 'orange' }]} />
     </View>
    </>
  );
};

const styles = StyleSheet.create({
  child: {width: 50,height: 50},
  containerBase: {flex: 1,borderWidth: 1,margin: 10},
  container1: {justifyContent: 'space-between'},
  container2: {justifyContent: 'space-around'},
  container3: {justifyContent: 'space-evenly'},});
```

```
export default Flex;
```



Align Items – 1/3

- Aligning items is performed using the alignItems property and which describes how children are aligned on the cross axis. It is very similar to justifyContent only on the cross axis. The Cross axis is the axis that is not primary! There are five values available for this property:
- **stretch:** Stretches the size of the children to match the height of the container on the cross axis. This is the default value. You must not that you should not assign **height** or **width** values to the children.
- **flex-start** Aligns the children to the start of the container on the cross axis.

Align Items – 2/3

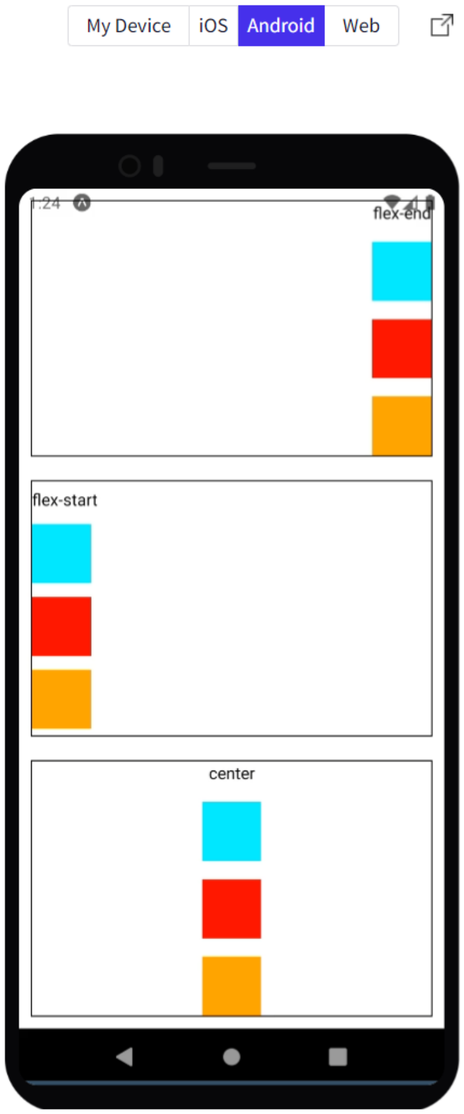
- **flex-end:** Aligns the children to the end of the container on the cross axis.
- **center:** Aligns the children in the center of the container on the cross axis.
- **baseline:** Aligns the children along a common baseline.
- Lets explore an example in the next page!

Align Items – 3/3

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

function FlexContainer  (props)  {
 return (
  <View style={[styles.containerBase, {justifyContent: props.justifyContent, alignItems: props.alignItems}]}>
  <Text>{props.alignItems}</Text>
    <View style={[styles.child, { backgroundColor: 'cyan' }]} />
    <View style={[styles.child, { backgroundColor: 'red' }]} />
    <View style={[styles.child, { backgroundColor: 'orange' }]} />
  </View>
 );
}
const Flex = () => {
 return (
  <View style={{ height: '100%' }}>
    <FlexContainer justifyContent="space-between" alignItems="flex-end" />
    <FlexContainer justifyContent="space-around" alignItems="flex-start" />
    <FlexContainer justifyContent="space-between" alignItems="center" />
  </View>
 );
};

const styles = StyleSheet.create({
 child: { width: 50, height: 50 },
 containerBase: { flex: 1, borderWidth: 1, margin: 10 },
});
export default Flex;
```



Align Self – 1/2

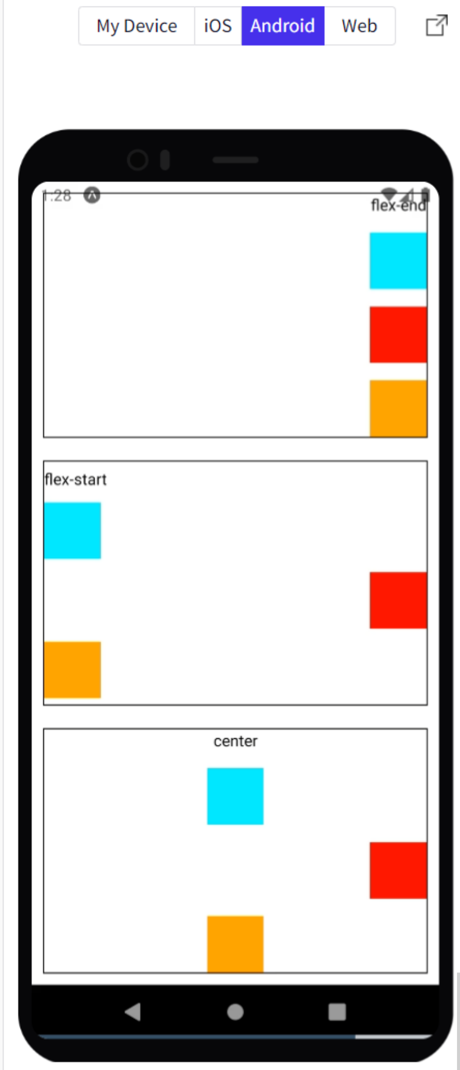
- To allow a single child of a parent container to have a different **alignItems** value than those projected by the parent, You will use the **alignSelf** property.
- The values used are the same as those that are used for the alignItems property.
- let’s review an example on the next slide!

Align Self – 2/2

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

function FlexContainer  (props) {
 return (
  <View style={[styles.containerBase, {justifyContent: props.justifyContent, alignItems: props.alignItems}]}>
  <Text>{props.alignItems}</Text>
   <View style={[styles.child, { backgroundColor: 'cyan' }]} />
   <View style={[styles.child, styles.selfAligned, { backgroundColor: 'red' }]} />
   <View style={[styles.child, { backgroundColor: 'orange' }]} />
  </View>
 );
}
const Flex = () => {
 return (
  <View style={{ height: '100%' }}>
   <FlexContainer justifyContent="space-between" alignItems="flex-end" />
   <FlexContainer justifyContent="space-around" alignItems="flex-start" />
   <FlexContainer justifyContent="space-between" alignItems="center" />
  </View>
 );
};

const styles = StyleSheet.create({
 child: { width: 50, height: 50 },
 selfAligned : {alignSelf : "flex-end"},
 containerBase: { flex: 1, borderWidth: 1, margin: 10 },
});
export default Flex;
```



Align Contents – 1/3

You can use the **alignContent** property to set the distribution of lines when **flexWrap** is set to "**wrap**". Essentially it will set where the next line of wrapped content must start on the cross axis. It has 6 different settings:

flex-start: Aligns the wrapped lines to the start of the container on the cross axis. This is the default value.
flex-end: Aligns the wrapped lines to the end of the container on the cross axis.
stretch: Stretches the wrapped lines to match the height of the container on the cross axis.

Align Contents – 2/3

center: Aligns the wrapped lines in the center of the container on the cross axis.
space-between: Spaces the wrapped lines across the container on the cross axis evenly. The remaining space will be distributed between the lines.
space-around: Same as space-between, but the remaining space will be distributed to the beginning of the first line and the end of the last line.

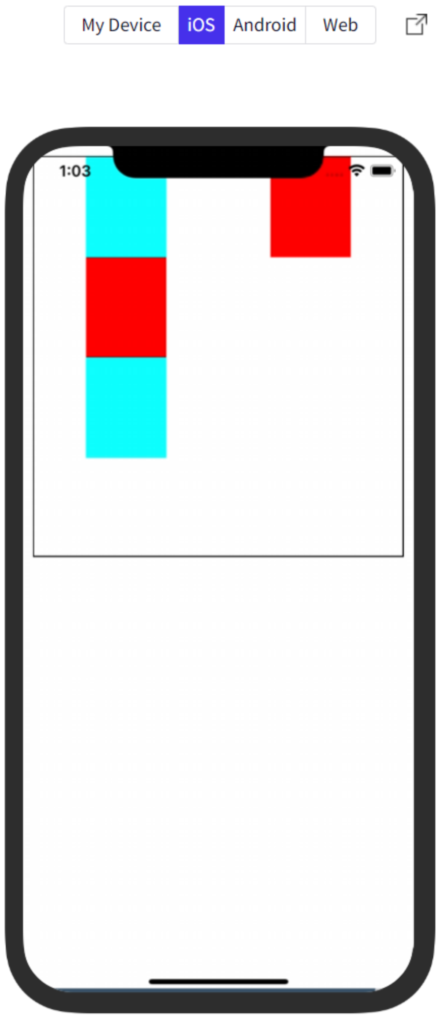
- Let’s explore the alignContent in an example on the next page!

Align Contents – 3/3

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

function FlexContainer(props) {
  return (
    <View style={[styles.containerBase, { alignContent: props.alignContent }]}>
      <View style={[styles.child, { backgroundColor: 'cyan' }]} />
      <View style={[styles.child, { backgroundColor: 'red' }]} />
      <View style={[styles.child, { backgroundColor: 'cyan' }]} />
      <View style={[styles.child, { backgroundColor: 'red' }]} />
    </View>
  );
}
const Flex = () => {
  return (
    <View style={{ height: '100%' }}>
      <FlexContainer alignContent="space-around" />
    </View>
  );
};

const styles = StyleSheet.create({
  child: { width: 80, height: 100 },
  containerBase: {flex: 1,borderWidth: 1,margin: 10,flexWrap: 'wrap',maxHeight:
400,flexDirection: 'column'},
});
export default Flex;
```



Flex Wrap – 1/2

You can use the **flexWrap** property to decide how the parent container renders its children when the children overflow over the primary axis. An example would be when you have more items you can fit on one row or column. There are two options for this property:

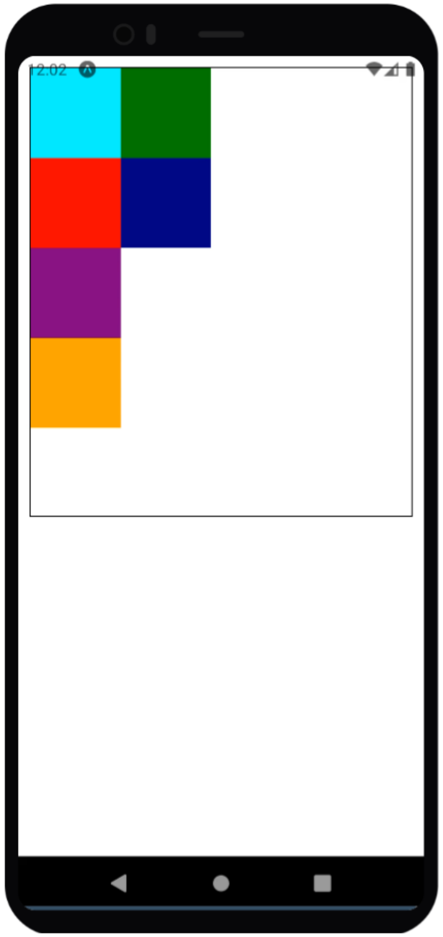
- **wrap**: allows the children to be wrapped to the next line in case they overflow
- **nowrap**: No wrapping will be allowed. Children must appear all on the same line on the primary axis.

Flex Wrap – 2/2

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

function FlexContainer() {
 return (
   <View style={[styles.containerBase]}>
     <View style={[styles.child, { backgroundColor: 'cyan' }]} />
     <View style={[styles.child, { backgroundColor: 'red' }]} />
     <View style={[styles.child, { backgroundColor: 'purple' }]} />
     <View style={[styles.child, { backgroundColor: 'orange' }]} />
      <View style={[styles.child, { backgroundColor: 'green' }]} />
      <View style={[styles.child, { backgroundColor: 'navy' }]} />
   </View>
 );
}
const Flex = () => {
 return (
   <View style={{ height: '100%' }}>
     <FlexContainer/>
   </View>
 );
};

const styles = StyleSheet.create({
 child: { width: 80, height: 80 },
 containerBase: {flex: 1,borderWidth: 1,margin: 10,flexWrap: 'wrap',maxHeight:
400,flexDirection: 'column'},
});
export default Flex;
```



Flex Basis, Flex Grow and Flex Shrink – 1/2

- You can use **flexBasis**, **flexGrow**, and **flexShrink** to decide how children are stretched or collapsed on the primary axis.
- These properties are useful when the width of children on the primary axis (flexDirection: “row”) or the height of children on a primary axis (flexDirection: “column”) overflow that primary axis and need to be resized to fit that primary axis:
    - **flexBasis** is the default size of the item on the primary axis and is subject to change depending on **flexGrow** and **flexShrink** values.
    - **flexGrow** decides how the space should be distributed among the children of that row or column on the primary axis. Accepts all floating points >= 0 and 0 is the default.
    - **flexShrink** decides how to distribute the space in case children’s size exceeds the primary axis. Accepts all floating points >= 0 and 0 is the default.

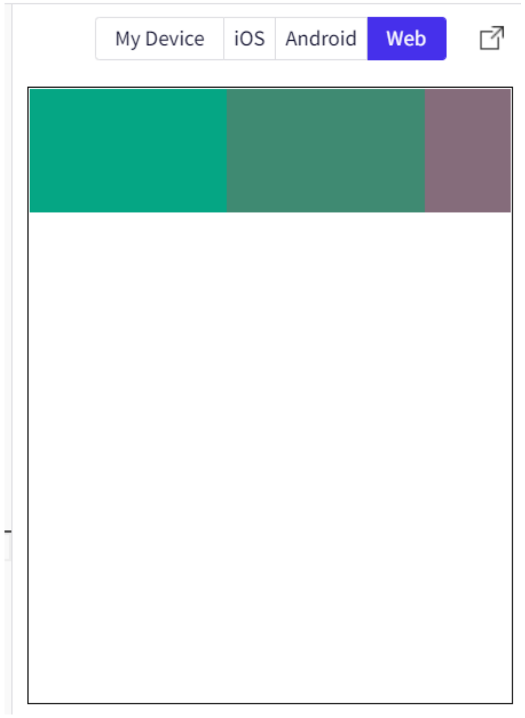
Flex Basis, Flex Grow and Flex Shrink – 1/2

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

const randomNumber = () => {return Math.floor(Math.random() * 256)}
const randomColor = () => { return 'rgb(' + randomNumber() + ',' + randomNumber() + ',' + ran
domNumber() + ')'; }

function Child(props) {
 return (
   <View style={[styles.child, { backgroundColor: randomColor(),
   flexBasis : props.flexBasis, flexShrink: props.flexShrink, flexGrow:props.flexGrow }]} />
 );
}
function FlexContainer() {
 return (
  <View style={[styles.containerBase]}>
   <Child flexBasis={200} flexShrink={1} flexGrow={1} />
   <Child flexBasis={200} flexShrink={1} flexGrow={1} />
   <Child flexBasis={200} flexShrink={2} flexGrow={1} />
  </View>
 );
}
const Flex = () => {
 return (
  <View style={{ height: '100%' }}>
   <FlexContainer/>
  </View>
 );
};

const styles = StyleSheet.create({
 child: { height: 80 },
 containerBase: {flex: 1,borderWidth: 1,margin: 10,flexWrap: 'nowrap',maxHeight: 400,flexDir
ection: 'row'},
});
export default Flex;
```



Try to modify the variables and see what happens!

Setting the Size using Width and Height – 1/2

- Like HTML and CSS, you can use the **width** and **height** properties to modify the width and height of elements, respectively. They accept three different types of values:
  - **pixels (“<value>px”)**: You can use define the size of the elements in pixels. For example, “200px” sets the element size to 200 pixels.
  - **auto**: This option allows react-native to decide the width or height depending on the content size.
  - **percentage (“<value>%”)**: This allows the child to occupy a percentage of its parent.

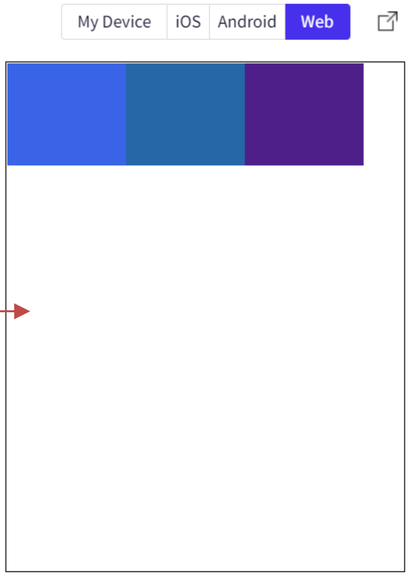
Setting the Size using Width and Height – 2/2

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

const randomNumber = () => {return Math.floor(Math.random() * 256)}
const randomColor = () => { return 'rgb(' + randomNumber() + ',' + randomNumber() + ',' + randomNumber() + ')'; }

function Child(props) {
 return (
    <View style={[styles.child, { backgroundColor: randomColor()}]} />
 );
}
function FlexContainer() {
 return (
   <View style={[styles.containerBase]}>
    <Child />
    <Child />
    <Child />
   </View>
 );
}
const Flex = () => {
 return (
   <View style={{ height: '100%' }}><FlexContainer/></View>
 );
};

const styles = StyleSheet.create({
 child: { height: "80",width:"30%" },
 containerBase: {flex: 1,borderWidth: 1,margin: 10,flexWrap: 'nowrap',maxHeight: 400,flexDirection: 'row'},
});
export default Flex;
```



WE ARE HUMBER

Absolute and Relative Positioning – 1/2

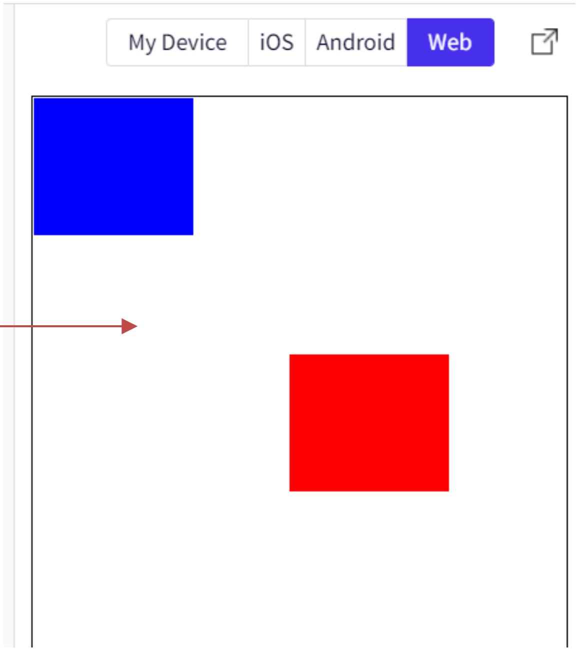
- In React-Native you can choose how a component is positioned using the **position** property. You have two options for it:
  - **relative**: This is the default value for positioning an element. The element will follow the normal layout and you can use **top**, **right**, **bottom** and **left** properties to add offsets. This offsetting does not affect other children.
  - **absolute**: The element does not follow the normal layout and is positioned independently based on the **top**, **right**, **bottom** and **left** properties.

Absolute and Relative Positioning – 2/2

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

function FlexContainer() {
  return (
    <View style={[styles.containerBase]}>
      <View style={[styles.child, { backgroundColor:"red",position:"absolute", to
p:150,left:150}]} />
      <View style={[styles.child, { backgroundColor:"blue",position:"relative"}]}
/>
    </View>
  );
}
const Flex = () => {
  return (
    <View style={{ height: '100%' }}><FlexContainer/></View>
  );
};

const styles = StyleSheet.create({
  child: { height: 80,width:"30%" },
  containerBase: {flex: 1,borderWidth: 1,margin: 10},
});
export default Flex;
```



References

Boduch, A., & Derks, R. (2020). *React and react native - third edition*. Packt Publishing.

Ward, D. (2019). *React native cookbook - second edition*. Packt Publishing.

Dabit, N. (2019). *React native in action: Developing IOS and android apps with JavaScript*. Manning Publications Co. LLC.

THANK YOU.

HUMBER

WE ARE HUMBER