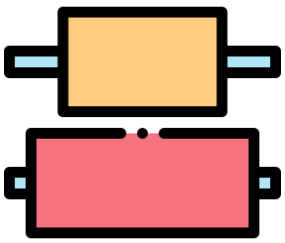
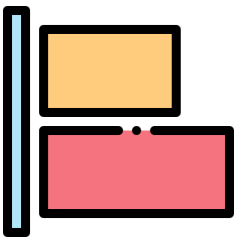
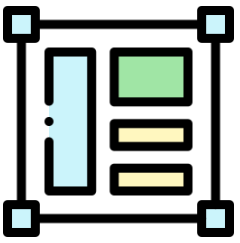
Descriptions

Here are explanations about the features used in the examples. Let's see how properties and functions are used.

Layouts allow for dynamically sizing and aligning components. It is one of the cornerstones of responsive designs.

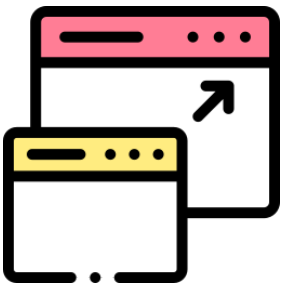
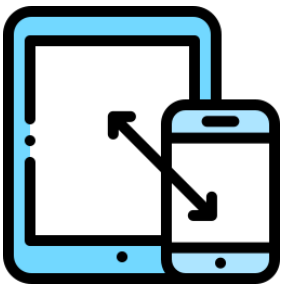
Alignment: This property determines how the component you are looking for is aligned in the Layout. The following three figures are examples of alignment. Some features you can use for alignment;

- Qt.AlignLeft
- Qt.AlignRight
- Qt.AlignBottom
- Qt.AlignTop

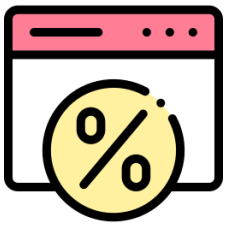
Fill: The width and height of the components change as the size of the window changes, taking into account the assigned height and width values.

- Layout.fillHeight
- Layout.fillWidth

Prefered: It determines the preferred width and height of the component in the layout. In this way, you can decide which component will occupy what percentage of space!

- Layout.preferedWidth
- Layout.preferedHeight



You can make changes to the codes as you wish. In this way, you can observe how the properties you change affect.

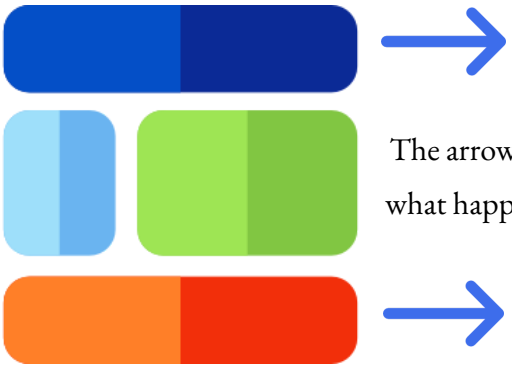
```
Rectangle {
                color: "#D65DB1"
                Layout.alignment: Qt.AlignRight
                Layout.fillHeight: true
                Layout.preferredWidth: 40
                Layout.preferredHeight: 70
                Text {
                    anchors.centerIn: parent
                    font.pixelSize: 14
                    text: index
                }
                Component.onCompleted: {
                 console.log(Layout.row +","+ Layout.column)
                }
            }
```

Here you can try to align one of the squares to the left and the other to the right! Or you might want to see what magic it does by typing **Layout.fillWidth: true** .

📌 **Note:** The Repeater creates instances using a "model". In our case we wanted it to create only 5 rectangles. However, the repeater's capabilities are not limited to this. I got how the Repeater works wonders while trying to understand the **Model-View-Delegate** structure.

Span: Allows us to specify how many columns the item will occupy.

- Layout.columnSpan
- Layout.rowSpan



The arrows shown take up 2 columns. **In Example 2**, we can see what happens by changing the Layout.columnSpan property of the bottom Rectangle!

Also in **Example 3** we can assign span property to CustomButtons. I even wrote one of them as a comment line. Find it and try it!

Where did it come from in CustomButton? Yes we can! We can edit our own objects in QML. In this way, we can create designs as we want in the endless universe of our imagination.

```
function boxButtonInit(item,color,text,icon,url) {
     item.Layout.fillWidth = true
     item.Layout.fillHeight = true
     item.backColor = color
     item.text = text
     item.iconUrl = icon
     item.openUrl = url
 }
```

This function gets 5 parameters. We can access these properties of the button in main.qml with the **property var**s we have defined in CustomButton.

```
    property var backColor: "#7BCBEB"
    property var iconUrl: ""
    property var textSize: 12
    property var openUrl: ""
```

We write our buttons inside the GridLayout in **main.qml** and tell the **Component.onCompleted** signal to trigger the boxButtonInit function.

```
Component.onCompleted: boxButtonInit(this,
                                     "color",
                                     "text",
                                     "image",
                                     "url");
```

Note that the changes you make in the CustomButton can directly affect the interface. A new button can be created by changing the properties of **"label", "background" and "Image"** in the contentItem.

 Here's how we can use **Qt.openUrlExternally()** to redirect us to the website when we click the button.

```
onClicked: {
        if (openUrl.length > 0) {
            Qt.openUrlExternally(openUrl);
        }
    }
```