CSS exercise

Table of Contents

knowledge@outsystems.com

www.outsystems.com

Outline

In this exercise we will focus on:

- Create and Edit styles using CSS and Visual Style Editor
- Use static styles
- Use dynamic styles
- How style sheet precedence and overriding styles work

Upon completion, we will have a Screen that is using multiple CSS classes we defined. We will also create a Block with it's own style class definitions. This setup will allow us to see first hand how overriding styles and style sheet precedence impact the final look and feel of our Screen.

Goal

In this exercise, we will use a Reactive Web application called **CSSFrontend** with a single Reactive Web module called **CSSDemo**. We will then add a Screen with some static content displayed and a custom Block. Our goal is to practice the use of different ways to apply styles to our Screen elements.

How-To

In this section, we'll show you how to do this exercise, with a thorough step-by-step description.

Getting Started

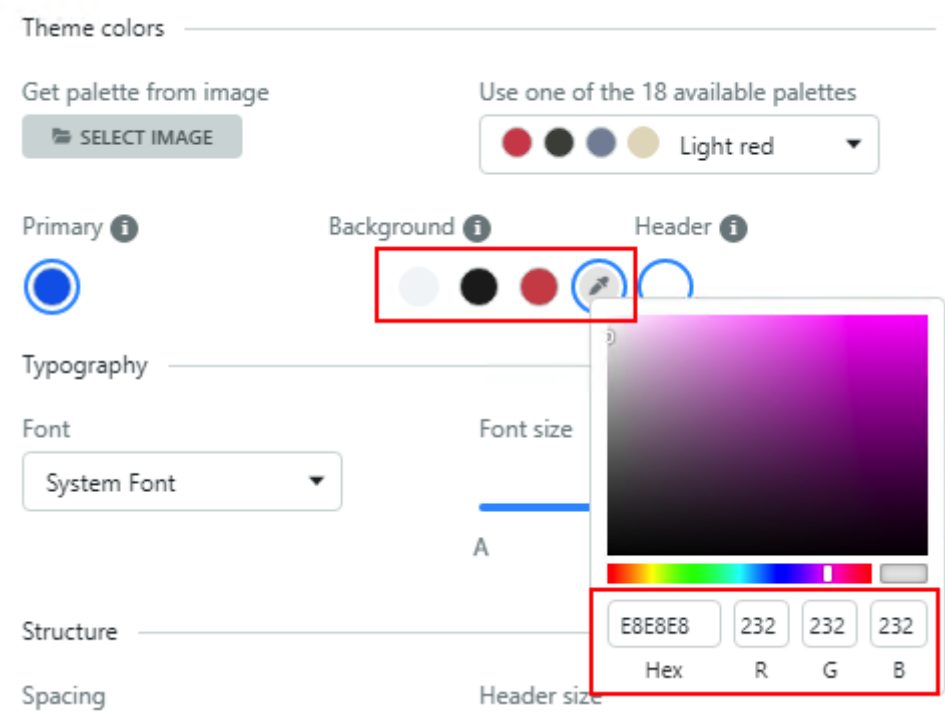
We will start this exercise, by creating a new Reactive Web Application from scratch. When creating the application set it's name to **CSSFrontend**. You can choose any color of your preference (or optionally upload a custom icon) in the app creation pop up window. Next, add a new Reactive Web App module and name it **CSSDemo**.

Theme Editor

Our goal at this stage is use the Theme Editor to modify the default background color of the canvas so we can hightlight our Screen content a bit more.

1) Open Theme Editor

2) Select the background color picker and type in to: **#E8E8E8**



3) Check the changes to your application:

   - Open the Style Sheet Editor

   - Go to the Theme Editor tab and confirm that CSS code variables are set accordingly.

     ```
     :root {

         /* App Settings */
         --color-background-body: #e8e8e8;
         --color-background-login: #e8e8e8;
     ```

   - Close the Style Sheet Editor and confirm that the canvas background color is also updated.

Styles Editor

Our main task, at this point, is to use the visual Styles Editor to create a new CSS class and apply it to a container on the screen. The final result should be similar to the one below.

Introducing CSS styles in OutSystems

Let's start by creating the Screen and its content:

1) Add a new Blank Screen with a name of your choice with **Anonymous** role permissions. As it is the only screen in your module, it will automatically be set as your module's Default Screen.

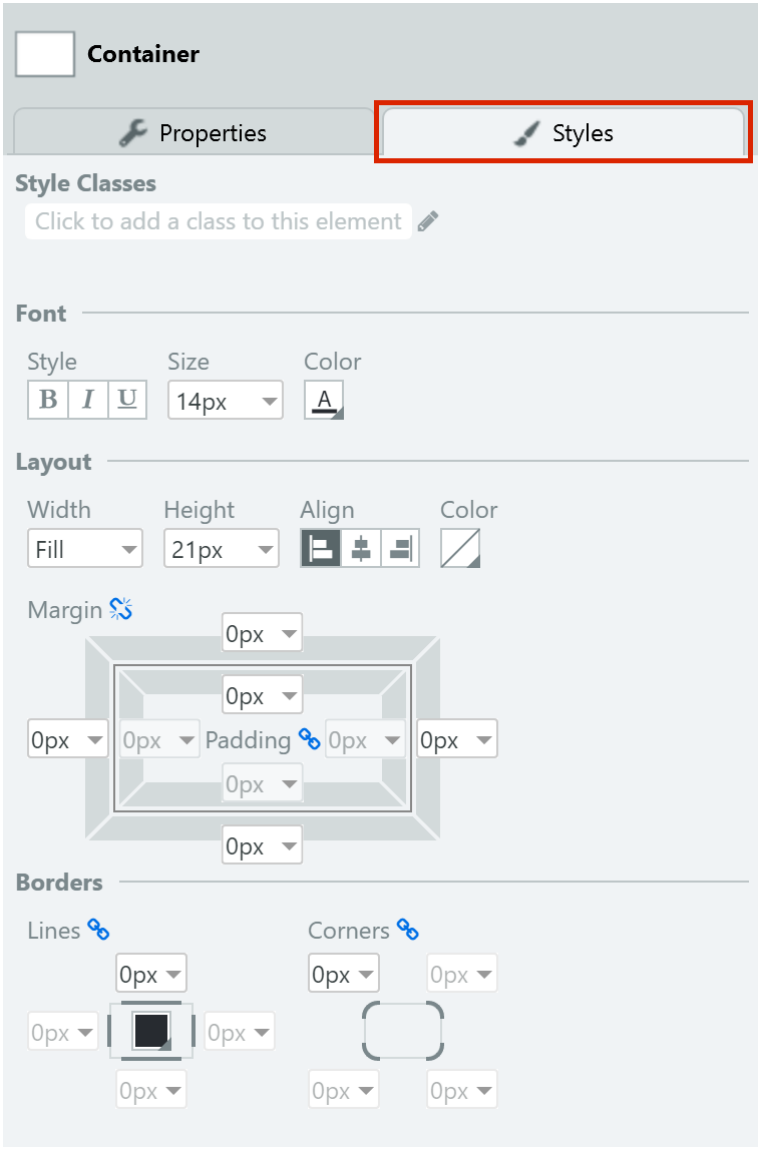
2) Insert a new Container widget into the *MainContent* placeholder of the screen

3) Place a new text widget containing the sentence: `Introducing CSS styles in OutSystems`.

Defining inline style

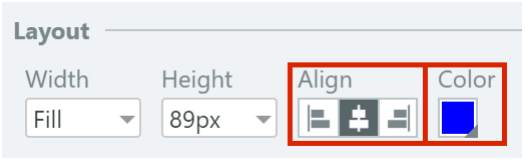
We are now ready to change the style of the new Container.

1) Select the Container and switch to the **Styles** tab to see the visual Styles Editor.

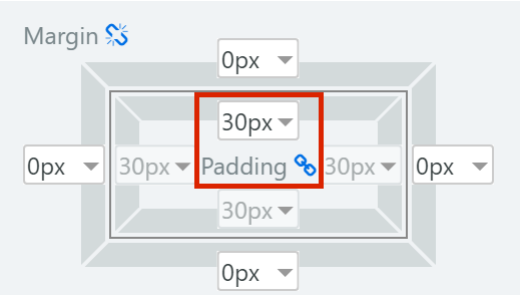


2) Set following style properties:

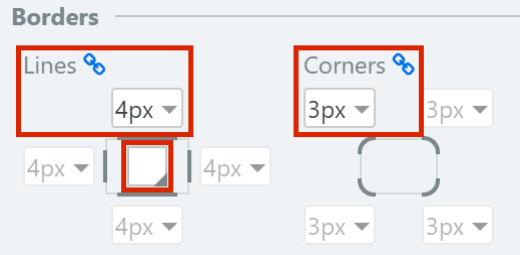
- **Center** Aligned
- Background Color: blue (RGB code: **#00F**)



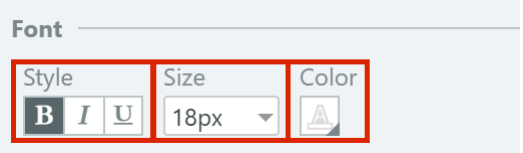
- Padding: **30px** on all sides



- Border Lines: **4px** on all sides
- Border Color: white (RGB code: **#FFF**)
- Border Corners: **3px** radius on all corners



- Text Style: **Bold**
- Text Size: **18px**
- Text Color: white (RGB code: **#FFF**)

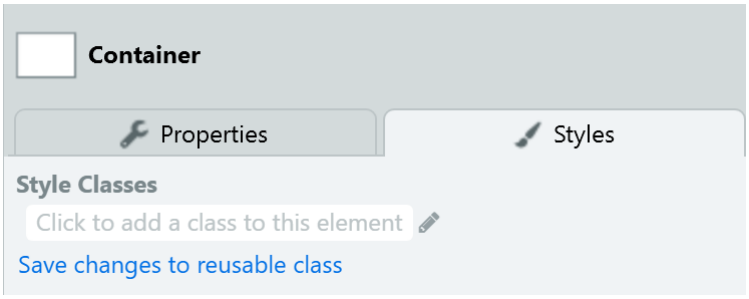


3) After you finish, check the canvas and confirm that it matches our initial goal.

Promoting inline style to class

Now that we already have our style properties defined, we will be promoting these style changes into a new class, so it can be reused on other widgets.

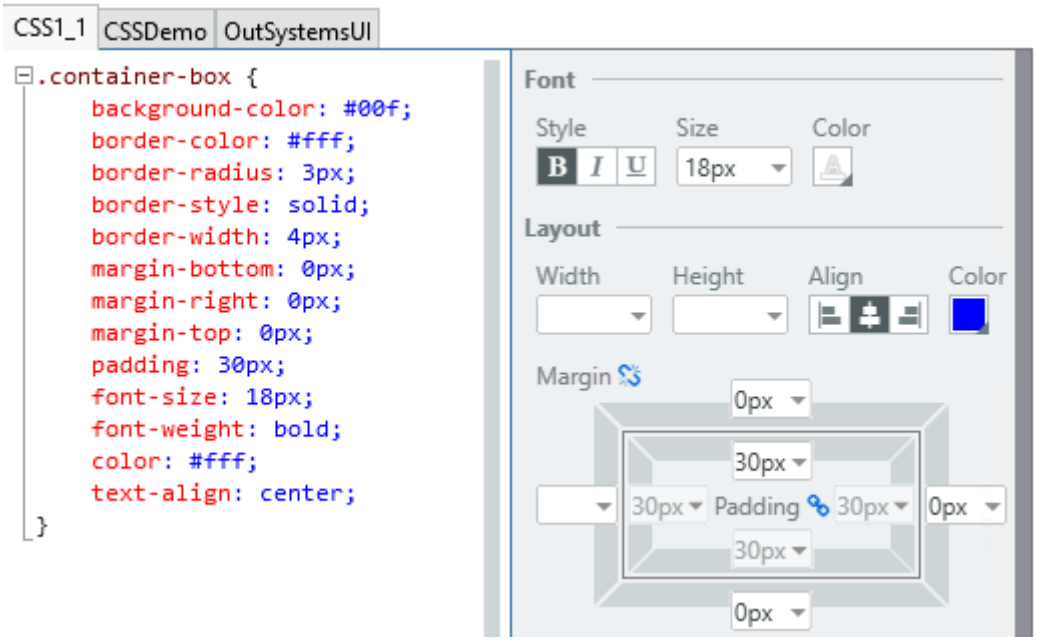
1) Still on the visual Styles Editor, save the changes to a reusable class and name it: **container-box**.



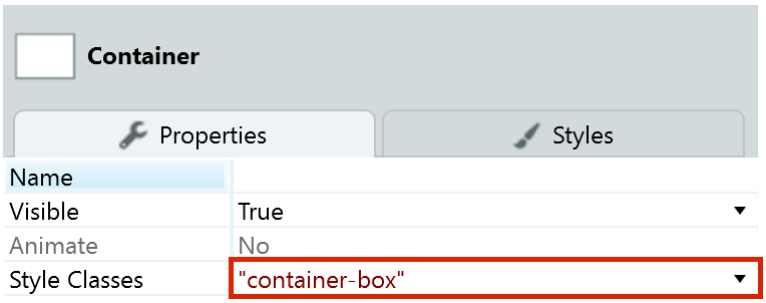
2) As the Style Sheet Editor opens, check that the following CSS code was added to module's theme tab.

```
.container-box {
background-color: #00f;
border-color: #fff;
border-radius: 3px;
border-style: solid;
border-width: 4px;
margin-bottom: 0px;
margin-right: 0px;
margin-top: 0px;
padding: 30px;
font-size: 18px;
font-weight: bold;
color: #fff;
text-align: center;
}
```

3) Move the CSS code to the empty Screen tab. (in the example our screen name is **CSS1_1**)



4) Close the Style Sheet Editor and go back to the container Properties tab and check that the **Style Classes** value is `"container-box"`



5) Publish your module and open it in the browser to check the final result.

You've just applied new style properties to a widget by using its visual Styles Editor and promoted those changes to a reusable class.

Applying multiple styles

Our goal in this section is to assign multiple CSS classes to a single widget.

Let's pick up from where we've left off in the previous section and create two new classes in the screen's style sheet. We will be refactoring some of the existing style properties of the **container-box** class into the new classes.

1) Create two new classes named **container-box-info** and **container-text**

2) Move style properties from the **"container-box"** class according to the following criteria:

   - the text-related style properties (size, weight and color) should go into the **container-text** class;
   - the background and border colors should go into the **container-box-info** class.

3) Leave the remaining properties on the **container-box** class. Your screen's style sheet should be similar to:

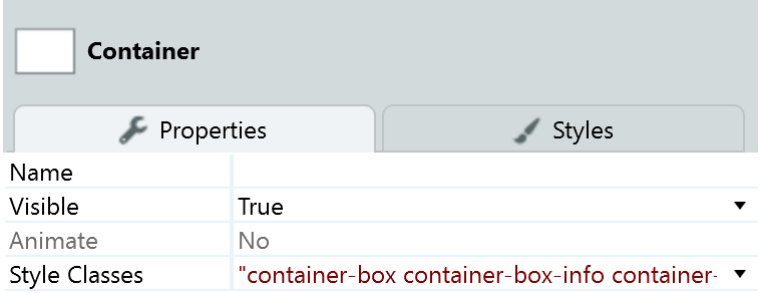
```
.container-box {
    border-radius: 3px;
    border-style: solid;
    border-width: 4px;
    padding: 30px;
    text-align: center;
}

.container-box-info {
    background-color: #00f;
    border-color: #fff;
}

.container-text {
    font-size: 18px;
    font-weight: bold;
    color: #fff;
}
```

   Go back to the canvas and notice how the Container widget lost its color and font styling. We will be recovering that look by assigning multiple classes to the widget.

4) Select the Container widget again and add the two new classes to the existing value of **Style Classes** property. Keep in mind that class names are separated by a space.

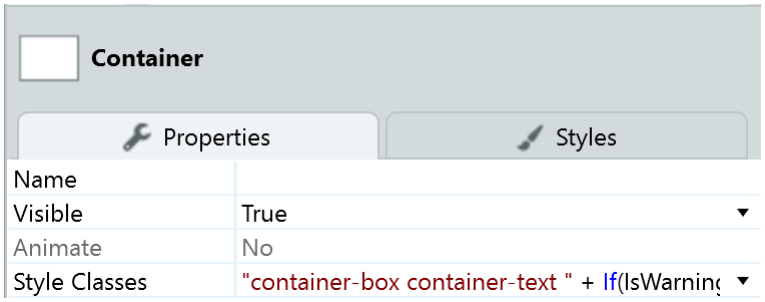


5) Check the Container widget again in canvas and confirm that style is back to what it was before we defined the two new classes. As expected all style properties are still being applied, but separately provided by three distinct classes. This allows to reuse any of the classes in other widgets on the Screen.

6) Finally, publish your changes and open the page in the browser to check the final result.

Applying styles dynamically

As we've seen most widgets allow us to assign CSS classes to them by using the **Style Classes** property. Up until now we've been using a literal text value containing the class names, but in this section we will be assigning classes dynamically , by setting an expression to the **Style Classes** property.



Let's start by creating a new class based on the **container-box-info** class and name it **container-box-warning**:

1) Open the Style Sheet Editor

2) Go to the leftmost tab (the Screen style sheet) and create a new class named **container-box-warning** with the following properties:

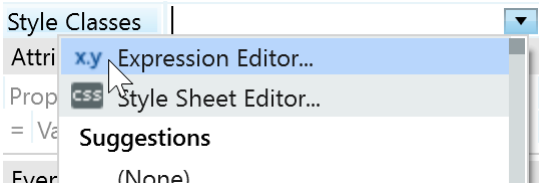
   - yellow background
   - black border color
   - black text color

   The **container-box-warning** CSS class should look like this:

```
.container-box-warning {
background-color: yellow;
border-color: #000;
color:#000;
}
```

3) Create a new Screen variable of type **Boolean** and name it **IsWarning**

4) Go to the Container **Style Classes** property and choose the **Expression Editor...** option from the dropdown



1) Type the following expression on the **Expression Editor**:

   "container-box container-text " + If(IsWarning, "container-box-warning", "container-box-info")"

2) Publish your changes and open the page in the browser.

3) Confirm that there were no changes on the page. This is because the same CSS classes are still being set to this container via **Style Classes** property, since the **IsWarning** boolean variable is by default `False`.

4) Go back to the Screen, and set the **IsWarning** variable's default value to `True`.

5) Publish again and open the page in the browser. The container should now be rendered like the image below.

Introducing CSS styles in OutSystems

Extra challenge

On our Screen create a button with the text **"Toggle Box style"** and bind it to a new Screen Action. As the button's label indicates, this screen action will toggle the current logical value of the **IsWarning** variable.

Since the variable is being used on-screen, changes made to its value will trigger the re-rendering of the affected container widget.

When you finish this challenge, publish again all the changes made, and open the page in the browser. Observe the new dynamic outcome by clicking the new button a few times to swap between styles.

CSS Hierarchy and style loading order

Our goal for this section is to have a practical example how style sheet precedence works.

The style sheet precedence can have a big impact on the expected Screen look-and-feel. Often styles classes are defined simultaneously at the Theme, Module, Screen and Block level. If so, this can lead to style properties being overriden.

Let's get started and create a new Block that uses its own CSS class.

1) Create a new Block and name it **Box**

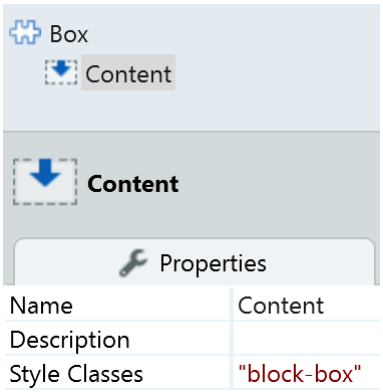
2) Edit the block's Style Sheet and create a new CSS class named **block-box** with the following parameters:

   - blue text
   - 14px font size
   - underlined and italic text

   The CSS class is as follows:

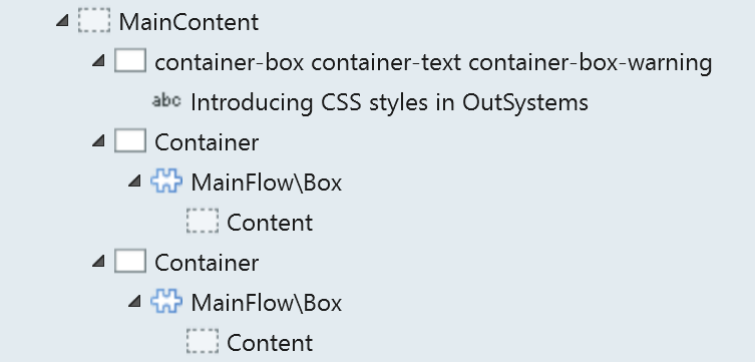
```
.block-box{
color: blue;
font-size: 14px;
font-style: italic;
text-decoration: underline;
}
```

3) Drag a new Placeholder into the block and call it **Content**. Assign the class just created to the placeholder's **Style Classes** property.

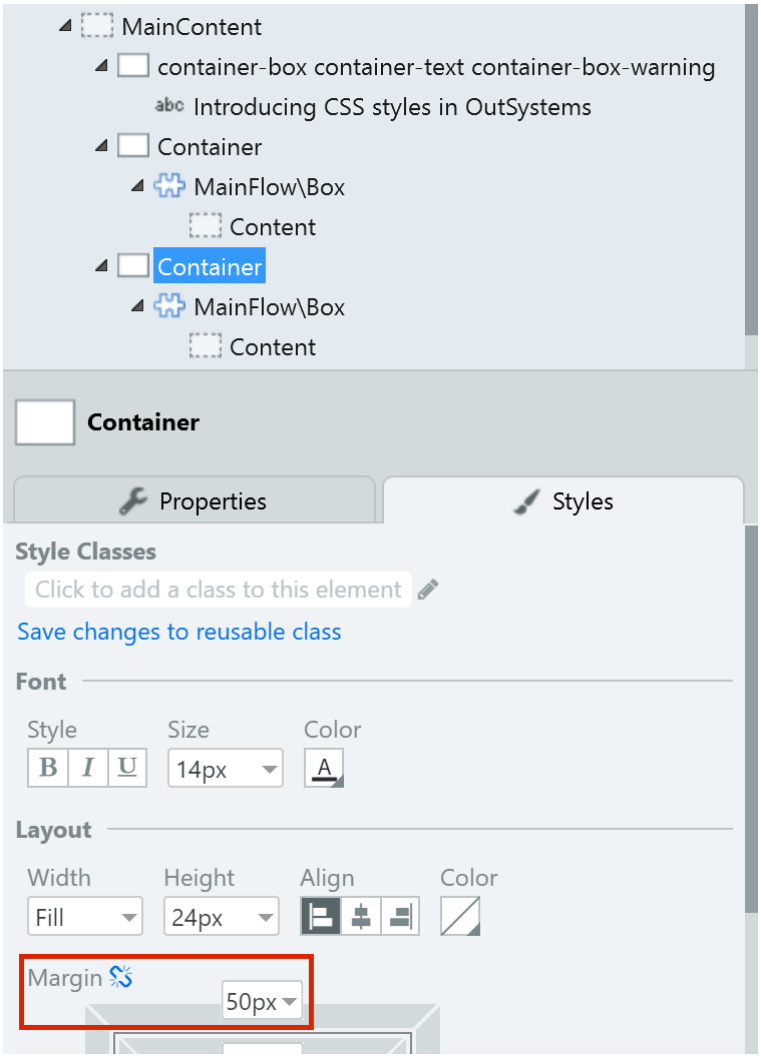


4) Go back to your screen and drag two new Containers below the one that already exists in the Screen.

5) Place an instance of the **Box** block inside each new container



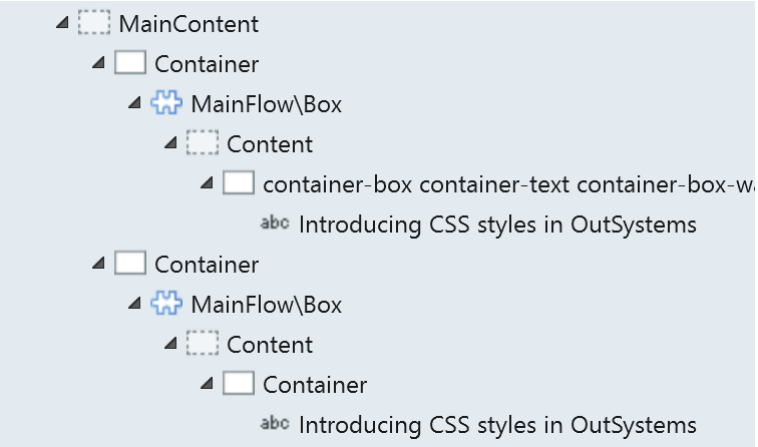
6) Go to the last container and change its top margin so there is a **50px** gap from the previous container.



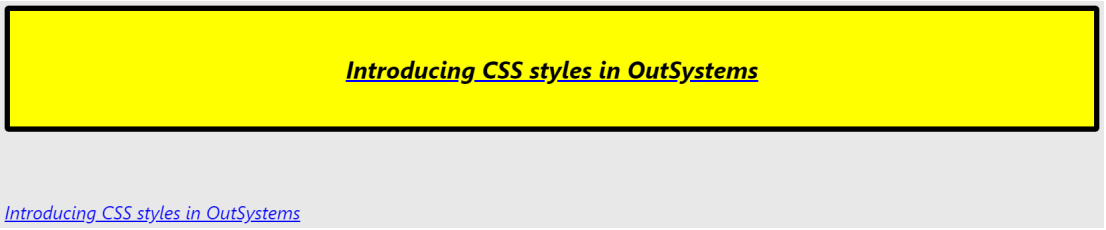
7) Now move the first container (that we created earlier) to the **Content** placeholder of the first block instance.

8) On the **Content** placeholder of the second block instance, place a new text widget with the same sentence: `Introducing CSS styles in Outsystems.`

9) Enclose the text in a Container too. The Screen widget tree should be as follows



10) Publish your changes and open the page in the browser.



Despite having the same style definition, the two blocks display their contents differently.

On the second instance of the block, its contents inherit the text style properties, namely the **size**, **style**, **decoration** and **color**.

On the first instance, the contents have their own classes that override some of the text style properties. In particular they override the text **size** and **color**, and of course add the **borders**, **background** and **padding**. But since they were not overriden, some of the text style properties are still inherited, like the **decoration** and **style**.

Thank you!