Widget Styling

In this lecture we will learn about the various ways to style widgets!

`style` vs. `layout`

There are two ways to change the appearance of widgets in the browser. The first is through the `layout` attribute which exposes layout-related CSS properties for the top-level DOM element of widgets, such as margins and positioning. The second is through the `style` attribute which exposes non-layout related attributes like button color and font weight. While `layout` is general to all widgets and containers of widgets, `style` offers tools specific to each type of widget.

Thorough understanding of all that `layout` has to offer requires knowledge of front-end web development, including HTML and CSS. This section provides a brief overview of things that can be adjusted using `layout`. However, the full set of tools are provided in the separate notebook **Advanced Widget Styling with Layout**.

To learn more about web development, including HTML and CSS, check out the course Python and Django Full Stack Web Developer Bootcamp (https://www.udemy.com/python-and-django-full-stack-web-developer-bootcamp/)

Basic styling is more intuitive as it relates directly to each type of widget. Here we provide a set of helpful examples of the `style` attribute.

The `layout` attribute

Jupyter interactive widgets have a `layout` attribute exposing a number of CSS properties that impact how widgets are laid out. These properties map to the values of the CSS properties of the same name (underscores being replaced with dashes), applied to the top DOM elements of the corresponding widget.

Sizes

- `height`
- `width`
- `max_height`
- `max_width`
- `min_height`
- `min_width`

Display

- `visibility`
- `display`
- `overflow`
- `overflow_x`
- `overflow_y`

Box model

- `border`
- `margin`
- `padding`

Positioning

- `top`
- `left`
- `bottom`
- `right`

Flexbox

- `order`
- `flex_flow`
- `align_items`
- `flex`
- `align_self`
- `align_content`
- `justify_content`

A quick example of `layout`

We've already seen what a slider looks like without any layout adjustments:

In [ ]:

```
import ipywidgets as widgets
from IPython.display import display

w = widgets.IntSlider()
display(w)
```

Let's say we wanted to change two of the properties of this widget: `margin` and `height`. We want to center the slider in the output area and increase its height. This can be done by adding `layout` attributes to **w**

In [ ]:

```
w.layout.margin = 'auto'
w.layout.height = '75px'
```

Notice that the slider changed positions on the page immediately!

Layout settings can be passed from one widget to another widget of the same type. Let's first create a new IntSlider:

In [ ]:

```
x = widgets.IntSlider(value=15,description='New slider')
display(x)
```

Now assign ***w's layout settings to* *x**:

In [ ]:

```
x.layout = w.layout
```

That's it! For a complete set of instructions on using `layout`, visit the **Advanced Widget Styling - Layout** notebook.

Predefined styles

Before we investigate the `style` attribute, it should be noted that many widgets offer a list of pre-defined styles that can be passed as arguments during creation.

For example, the `Button` widget has a `button_style` attribute that may take 5 different values:

- `'primary'`
- `'success'`
- `'info'`
- `'warning'`
- `'danger'`

besides the default empty string `''`.

In [ ]:

```
import ipywidgets as widgets

widgets.Button(description='Ordinary Button', button_style='')
```

In [ ]:

```
widgets.Button(description='Danger Button', button_style='danger')
```

The `style` attribute

While the `layout` attribute only exposes layout-related CSS properties for the top-level DOM element of widgets, the `style` attribute is used to expose non-layout related styling attributes of widgets.

However, the properties of the `style` atribute are specific to each widget type.

In [ ]:

```
b1 = widgets.Button(description='Custom color')
b1.style.button_color = 'lightgreen'
b1
```

You can get a list of the style attributes for a widget with the `keys` property.

In [ ]:

```
b1.style.keys
```

Note that `widgets.Button().style.keys` also works.

Just like the `layout` attribute, widget styles can be assigned to other widgets.

In [ ]:

```
b2 = widgets.Button()
b2.style = b1.style
b2
```

Note that only the style was picked up by **b2**, not any other parameters like `description`.

Widget styling attributes are specific to each widget type.

In [ ]:

```
s1 = widgets.IntSlider(description='Blue handle')
s1.style.handle_color = 'lightblue'
s1
```

Widget style traits

These are traits that belong to some of the more common widgets:

Button

- `button_color`
- `font_weight`

IntSlider, FloatSlider, IntRangeSlider, FloatRangeSlider

- `description_width`
- `handle_color`

IntProgress, FloatProgress

- `bar_color`
- `description_width`

Most others such as `ToggleButton`, `Checkbox`, `Dropdown`, `RadioButtons`, `Select` and `Text` only have `description_width` as an adjustable trait.

Conclusion

You should now have an understanding of how to style widgets!