Advanced Widget Styling with Layout

This notebook expands on the **Widget Styling** lecture by describing the various HTML and CSS adjustments that can be made through the `layout` attribute.

The `layout` attribute

Jupyter interactive widgets have a `layout` attribute exposing a number of CSS properties that impact how widgets are laid out.

Exposed CSS properties

> The following 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`

Shorthand CSS properties

You may have noticed that certain CSS properties such as `margin-[top/right/bottom/left]` seem to be missing. The same holds for `padding-[top/right/bottom/left]` etc.

In fact, you can atomically specify `[top/right/bottom/left]` margins via the `margin` attribute alone by passing the string `'100px 150px 100px 80px'` for a respectively `top`, `right`, `bottom` and `left` margins of `100`, `150`, `100` and `80` pixels.

Similarly, the `flex` attribute can hold values for `flex-grow`, `flex-shrink` and `flex-basis`. The `border` attribute is a shorthand property for `border-width`, `border-style (required)`, and `border-color`.

In [19]:
```
import ipywidgets as widgets
from IPython.display import display
```

Conclusion

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