21.2 Setting renderables

The first position argument to `Layout` can be any Rich renderable, which will be sized to fit within the layout's area. Here's how we might divide the "right" layout in to two panels:

```
from rich.panel import Panel

layout["right"].split(
    Layout(Panel("Hello")),
    Layout(Panel("World!"))
)
```

You can also call *update()* to set or replace the current renderable:

```
layout["left"].update(
    "The mystery of life isn't a problem to solve, but a reality to experience."
)
print(layout)
```

21.3 Fixed size

You can set a layout to use a fixed size by setting the `size` argument on the Layout constructor or by setting the attribute. Here's an example:

```
layout["upper"].size = 10
print(layout)
```

This will set the upper portion to be exactly 10 rows, no matter the size of the terminal. If the parent layout is horizontal rather than vertical, then the size applies to the number of characters rather that rows.

21.4 Ratio

In addition to a fixed size, you can also make a flexible layout setting the `ratio` argument on the constructor or by assigning to the attribute. The ratio defines how much of the screen the layout should occupy in relation to other layouts. For example, let's reset the size and set the ratio of the upper layout to 2:

```
layout["upper"].size = None
layout["upper"].ratio = 2
print(layout)
```

This makes the top layout take up two thirds of the space. This is because the default ratio is 1, giving the upper and lower layouts a combined total of 3. As the upper layout has a ratio of 2, it takes up two thirds of the space, leaving the remaining third for the lower layout.

A layout with a ratio set may also have a minimum size to prevent it from getting too small. For instance, here's how we could set the minimum size of the lower sub-layout so that it won't shrink beyond 10 rows:

```
layout["lower"].minimum_size = 10
```