If you set the `max_string` argument to an integer, Rich will truncate strings over that length. Truncated string will be appended with the number of characters that have not been shown. Here's an example:

```
>>> pprint("Where there is a Will, there is a Way", max_string=21)
```

7.2 Pretty renderable

Rich offers a *Pretty* class which you can use to insert pretty printed data in to another renderable.

The following example displays pretty printed data within a simple panel:

```
from rich import print
from rich.pretty import Pretty
from rich.panel import Panel

pretty = Pretty(locals())
panel = Panel(pretty)
print(panel)
```

There are a large number of options to tweak the pretty formatting, See the *Pretty* reference for details.

7.3 Rich Repr Protocol

Rich is able to syntax highlight any output, but the formatting is restricted to built-in containers, dataclasses, and other objects Rich knows about, such as objects generated by the attrs library. To add Rich formatting capabilities to custom objects, you can implement the *rich repr protocol*.

Run the following command to see an example of what the Rich repr protocol can generate:

```
python -m rich.repr
```

First, let's look at a class that might benefit from a Rich repr:

```
class Bird:
    def __init__(self, name, eats=None, fly=True, extinct=False):
        self.name = name
        self.eats = list(eats) if eats else []
        self.fly = fly
        self.extinct = extinct

    def __repr__(self):
        return f"Bird({self.name!r}, eats={self.eats!r}, fly={self.fly!r}, extinct={self.
↪extinct!r})"

BIRDS = {
    "gull": Bird("gull", eats=["fish", "chips", "ice cream", "sausage rolls"]),
    "penguin": Bird("penguin", eats=["fish"], fly=False),
    "dodo": Bird("dodo", eats=["fruit"], fly=False, extinct=True)
}
print(BIRDS)
```

The result of this script would be: