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

The output is long enough to wrap on to the next line, which can make it hard to read. The repr strings are informative but a little verbose since they include default arguments. If we print this with Rich, things are improved somewhat:

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

Rich knows how to format the container dict, but the repr strings are still verbose, and there is some wrapping of the output (assumes an 80 character terminal).

We can solve both these issues by adding the following `__rich_repr__` method:

```
def __rich_repr__(self):
    yield self.name
    yield "eats", self.eats
    yield "fly", self.fly, True
    yield "extinct", self.extinct, False
```

Now if we print the same object with Rich we would see the following:

```
{
    '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)
}
```

The default arguments have been omitted, and the output has been formatted nicely. The output remains readable even if we have less room in the terminal, or our objects are part of a deeply nested data structure:

```
{
    'gull': Bird(
        'gull',
        eats=[
            'fish',
            'chips',
            'ice cream',
            'sausage rolls'
        ]
    ),
    'penguin': Bird(
        'penguin',
        eats=['fish'],
        fly=False
    ),
```

(continues on next page)