Doctor

Elliott Phillips

/ellsphillips/doctor

16/03/2022

Introduction

- About
- Users
- Design

About – What is Doctor?

An automated documentation assistant built in Python and $\TeX$ for procedural, data-driven reporting.

About – Mission statement

Doctor provides services to simplify the reporting of data-oriented, beautiful, lightweight documents.

Transparent and opinionated, without the WYSIWYG faff.

About – Journey

Personal side project

Business-critical application

Internal collaborative platform

About – Journey

Personal side project

Business-critical application

Internal collaborative platform

About – Journey

Personal side project

Business-critical application

Internal collaborative platform

About – Styled components

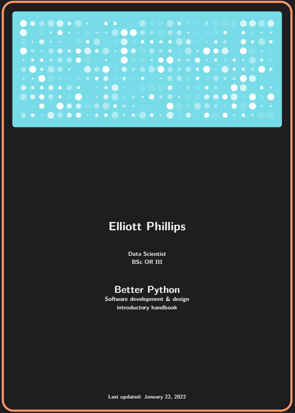
C_1	C_2	C_3	C_4
a	b	c	d
e	f	g	h
i	j	k	l

C_1	C_2	C_3	C_4
a	b	c	d
e	f	g	h
i	j	k	l

Users – Corporate reports

BRR demo

Users – Learning & Development

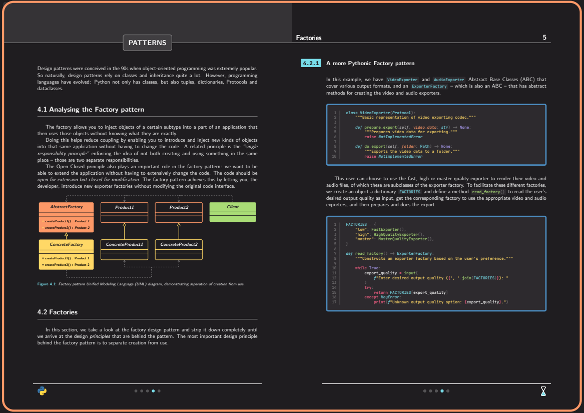


Elliott Phillips

Data Scientist
BSc OR III

Better Python
Software development & design
introductory handbook

Last updated: January 22, 2022



PATTERNS

Design patterns were conceived in the 90s when object-oriented programming was extremely popular. So naturally, design patterns rely on classes and inheritance quite a lot. However, programming languages have evolved: Python not only has classes, but also tuples, dictionaries, Protocols and dataclasses.

4.1 Analysing the Factory pattern

The factory allows you to inject objects of a certain subtype into a part of an application that then uses those objects without knowing what they are exactly.

Doing this helps reduce coupling by enabling you to introduce and inject new kinds of objects into that same application without having to change the code. A related principle is the *"single responsibility principle"* enforcing the idea of not both creating and using something in the same place – those are two separate responsibilities.

The Open Closed principle also plays an important role in the factory pattern: we want to be able to extend the application without having to extensively change the code. The code should be *open for extension but closed for modification*. The factory pattern achieves this by letting you, the developer, introduce new exporter factories without modifying the original code interface.

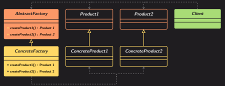

AbstractFactory
createProduct1() : Product 1
createProduct2() : Product 2
Product1
Product2
Client
ConcreteFactory
+ createProduct1() : Product 1
+ createProduct2() : Product 2
ConcreteProduct1
ConcreteProduct2

Figure 4.1: *Factory pattern Unified Modeling Language (UML) diagram, demonstrating separation of creation from use.*

4.2 Factories

In this section, we take a look at the factory design pattern and strip it down completely until we arrive at the design *principles* that are behind the pattern. The most important design principle behind the factory pattern is to separate creation from use.

Factories 5

4.2.1 A more Pythonic Factory pattern

In this example, we have `VideoExporter` and `AudioExporter` Abstract Base Classes (ABC) that cover various output formats, and an `ExporterFactory` – which is also an ABC – that has abstract methods for creating the video and audio exporters.

```
class VideoExporter(Protocol):
    """Basic representation of video exporting codec."""

    def prepare_export(self, video_data: str) → None:
        """Prepares video data for exporting."""
        raise NotImplementedError

    def do_export(self, folder: Path) → None:
        """Exports the video data to a folder."""
        raise NotImplementedError
```

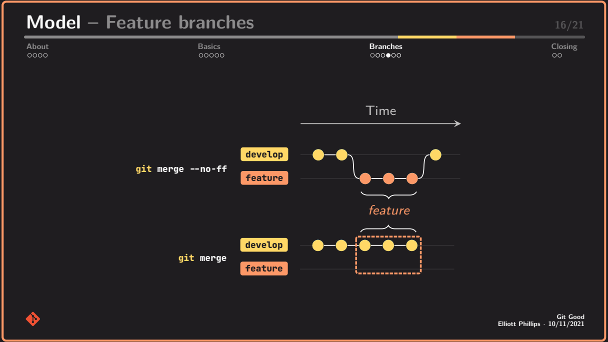
This user can choose to use the fast, high or master quality exporter to render their video and audio files, of which these are subclasses of the exporter factory. To facilitate these different factories, we create an object a dictionary `FACTORIES` and define a method `read_factory()` to read the user's desired output quality as input, get the corresponding factory to use the appropriate video and audio exporters, and then prepares and does the export.

```
FACTORIES = {
    "low": FastExporter(),
    "high": HighQualityExporter(),
    "master": MasterQualityExporter(),
}

def read_factory() → ExporterFactory:
    """Constructs an exporter factory based on the user's preference."""

    while True:
        export_quality = input(
            f"Enter desired output quality ({', '.join(FACTORIES)}): "
        )
        try:
            return FACTORIES[export_quality]
        except KeyError:
            print(f"Unknown output quality option: {export_quality}.")
```

Users – Learning & Development



Model – Feature branches

```
$ git clone https://github.com/ellsphillips/git-book git-good
```

Usage

- Installation
- API

Installation – Module

Module installation

Installation – Project

```python
import doctor as dr


def main() -> None:
    ...


if __name__ == "__main__":
    main()
```

API – Tables

```python
tabular = dr.table(
    dr.data.table(
        [
            dr.data.numeric.rand(10),
            dr.data.text.lorem(10),
        ]
    ),
    columns=["numbers", "words"]
)
```

API – Plot

```python
figure = dr.plot(
    "line",
    data={
        "timeseries_2020": dr.data.series.brownian(),
        "timeseries_2021": dr.data.series.brownian(),
        "timeseries_2022": dr.data.series.brownian(),
    },
    options={
        "plot type": "ybar",
        "data source": "src/plots/example.dat",
        "caption": "Demonstration of the doctor-plot env",
        "label": "example-plot",
    },
)
```

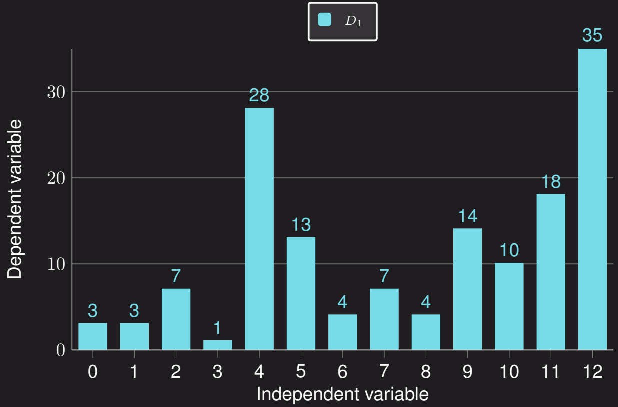
API – Document

Document class API

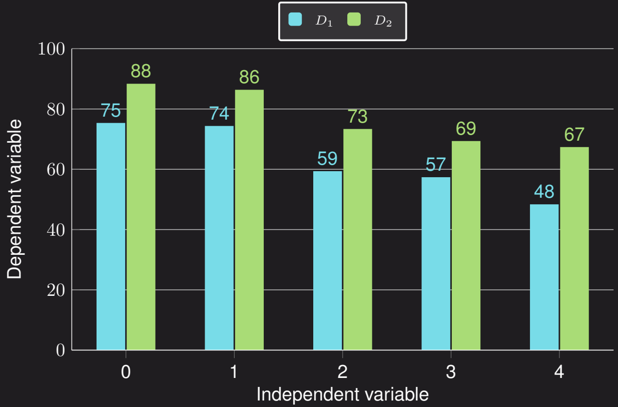
Examples

- Table
- Plot

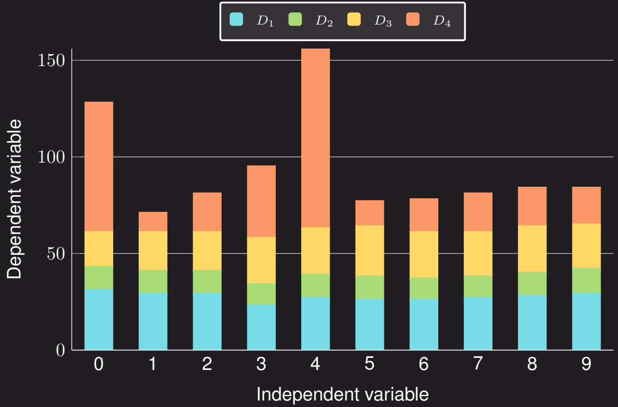
Plot – bar


D1
Dependent variable
Independent variable
0
10
20
30
0 1 2 3 4 5 6 7 8 9 10 11 12
3
3
7
1
28
13
4
7
4
14
10
18
35

Plot – bar


D_1
D_2
Dependent variable
Independent variable
0
20
40
60
80
100
0
1
2
3
4
75
88
74
86
59
73
57
69
48
67

Plot – bar


D_1
D_2
D_3
D_4
150
100
50
0
0
1
2
3
4
5
6
7
8
9
Dependent variable
Independent variable

Contribution

- GitHub

GitHub – Public repo

Interested in contributing? Doctor is developed open source! Get it touch via email or create a pull request

/ellsphillips/doctor

GitHub – Sharing is caring

You're welcome to retain a copy and share this material with anyone who may benefit.

Please ⭐ this repository if you have found this material useful and to follow its development!

Next steps

- GitHub

GitHub – Open source

Open source

GitHub – Test & deploy

Test & deploy

Doctor

Elliott Phillips

/ellsphillips/doctor

16/03/2022