



**Doctor**

---

**Elliott Phillips**

 /ellsphillips/doctor

16/03/2022

## Introduction

---

Usage

Examples

Contribution

Next steps

## Introduction


---

- About
- Users
- Design



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



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

Transparent and opinionated, without the WYSIWYG faff.



Introduction

○○●○○○

Usage

○○○○○

Examples

○○

Contribution

○○

Next steps

○○

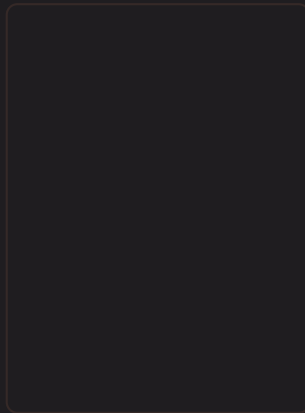
Personal side project



Business-critical application



Internal collaborative platform



Introduction

○○●○○○

Usage

○○○○○

Examples

○○

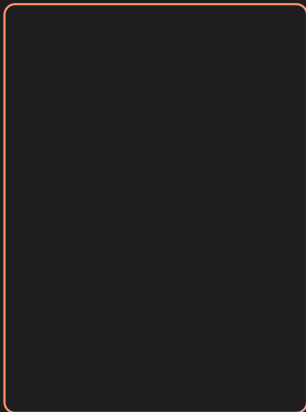
Contribution

○○

Next steps

○○

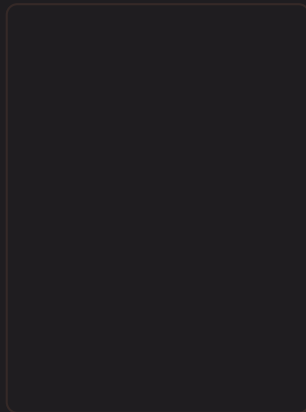
Personal side project



Business-critical application



Internal collaborative platform



Introduction

○○●○○○

Usage

○○○○○

Examples

○○

Contribution

○○

Next steps

○○

Personal side project



Business-critical application



Internal collaborative platform



$C_1$	$C_2$	$C_3$	$C_4$
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





Introduction

○○○○●○

Usage

○○○○○

Examples

○○

Contribution

○○

Next steps

○○

BRR demo





Elliott Phillips

Data Scientist  
BSc OR IIIBetter Python  
Software development & design  
introductory handbook

Last updated: January 22, 2022

## PATTERNS

## Factories

5

Design patterns were conceived in the 80s 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 decorators.

## 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" referring to 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.



Figure 4.1: Factory pattern (UML diagram) 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 its separation creation from use.

## 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.

```
1 class VideoExporter(Protocol):
2     """Base representation of video exporting codes."""
3
4     def prepare_export(self, video_data: str) -> None:
5         """Prepare video data for exporting."""
6         raise NotImplementedError
7
8     def do_export(self, video: Path) -> None:
9         """Export the video data to a video."""
10        raise NotImplementedError
```

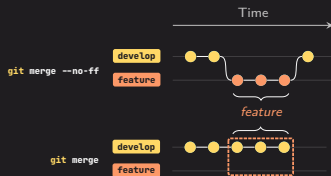
This user can choose to use the fast, high or master quality exporter to render their video and audio files, of which these are subclasses of the exporter factory. To facilitate these different factories, we create an object dictionary `FACTORIES` and define a method `read_factory()` to read the user's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

```
1 FACTORIES = {}
2
3 def read_factory() -> ExporterFactory:
4     """Reads the user's desired output quality and returns the corresponding factory to use the appropriate video and audio exporters, and then prepares and does the export.
5
6     """
7     quality = input("Enter desired output quality (F, H, M): ")
8     if quality in FACTORIES:
9         return FACTORIES[quality]
10    else:
11        raise ValueError("Invalid output quality option: {}".format(quality))
```



## Model – Feature branches

16/21

About  
○○○○Basics  
○○○○○Branches  
○○○●○○Closing  
○○Git Good  
Elliott Phillips · 10/11/2021

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



Introduction

**Usage**

Examples

Contribution

Next steps

# Usage

---

- **Installation**
- **API**

Introduction  
○○○○○○

Usage  
●○○○○

Examples  
○○

Contribution  
○○

Next steps  
○○

## Module installation



Introduction  
○○○○○○

Usage  
○●○○○

Examples  
○○

Contribution  
○○

Next steps  
○○

```
1 import doctor as dr
2
3
4 def main() → None:
5     ...
6
7
8 if __name__ == "__main__":
9     main()
```



```
1 tabular = dr.table(  
2     dr.data.table(  
3         [  
4             dr.data.numeric.rand(10),  
5             dr.data.text.lorem(10),  
6         ]  
7     ),  
8     columns=["numbers", "words"]  
9 )
```



```
1 figure = dr.plot(  
2     "line",  
3     data={  
4         "timeseries_2020": dr.data.series.brownian(),  
5         "timeseries_2021": dr.data.series.brownian(),  
6         "timeseries_2022": dr.data.series.brownian(),  
7     },  
8     options={  
9         "plot type": "ybar",  
10        "data source": "src/plots/example.dat",  
11        "caption": "Demonstration of the doctor-plot env",  
12        "label": "example-plot",  
13    },  
14 )
```





Introduction

○○○○○

Usage

○○○○●

Examples

○○

Contribution

○○

Next steps

○○

Document class API



Introduction

Usage

**Examples**

Contribution

Next steps

## Examples

---

- Table
- Plot

Introduction

○○○○○○

Usage

○○○○○

Examples

●○

Contribution

○○

Next steps

○○



Introduction

○○○○○○

Usage

○○○○○

Examples

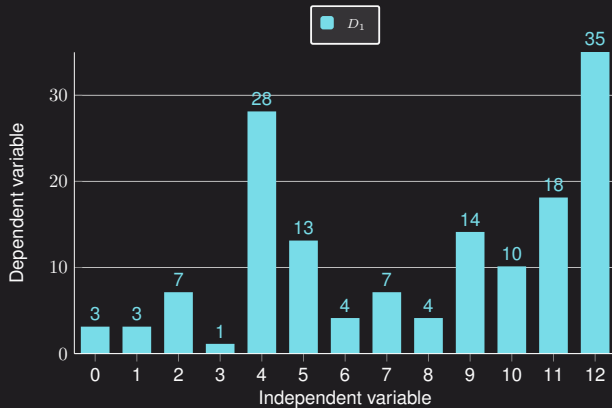
○○●

Contribution

○○

Next steps

○○



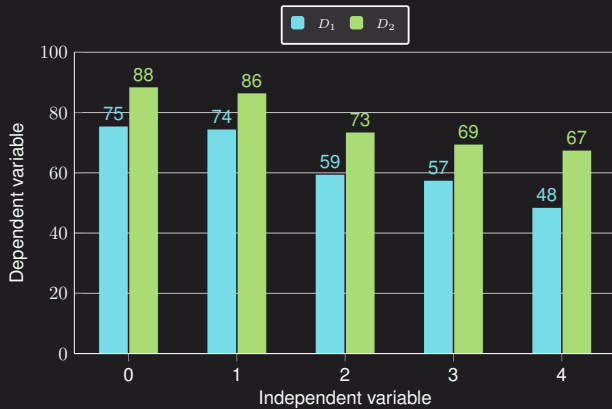
Introduction  
○○○○○○

Usage  
○○○○○

Examples  
○○●

Contribution  
○○

Next steps  
○○



Introduction

○○○○○○

Usage

○○○○○

Examples

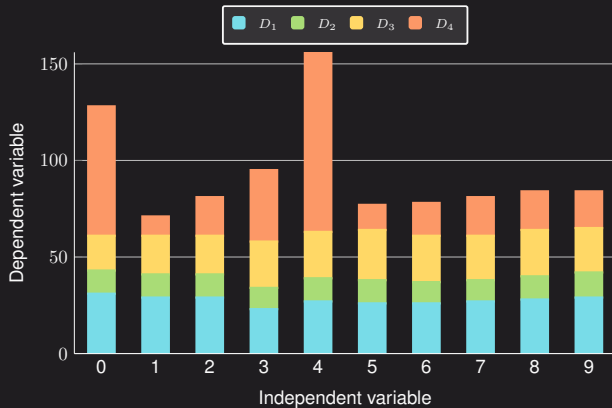
○○●

Contribution

○○

Next steps

○○



Introduction

Usage

Examples

**Contribution**

---

Next steps

# Contribution

---

– GitHub


Introduction  
○○○○○○


Usage  
○○○○○

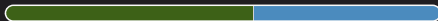
Examples  
○○

Contribution  
●○

Next steps  
○○

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

 /ellsphillips/doctor



● TeX   ● Python





Introduction  
○○○○○○

Usage  
○○○○○

Examples  
○○

Contribution  
○●

Next steps  
○○

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!



Introduction

Usage

Examples

Contribution

**Next steps**

---

## Next steps

---

– GitHub

Introduction  
○○○○○○

Usage  
○○○○○

Examples  
○○

Contribution  
○○

Next steps  
●○

Open source



Introduction  
○○○○○○

Usage  
○○○○○

Examples  
○○

Contribution  
○○

Next steps  
○●

Test & deploy





# Doctor

---

Elliott Phillips

 /ellsphillips/doctor

16/03/2022