



Doctor

Elliott Phillips

 /ellsphillips/doctor

24/03/2022

Head's up! I'm a full-time senior Data Scientist,
currently on secondment and juggling projects as
I go...

But more on that later...



Introduction

Examples

Usage

Next steps

Contribution


Introduction

- About
- Users
- Design



An automated documentation assistant built in Python and \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.



Personal side project

- ▶ L^AT_EX University requirement
- ▶ Self-taught Python at ONS
- ▶ Wanted to inject flair whilst automating corporate documents

Business-critical application

- ▶ Lead project to deliver real-time financial estimates
- ▶ Complex data to visualise and communicate
- ▶ Templated cohesion between report releases

Internal collaborative platform

- ▶ SCS requested S&T to National Statistician
- ▶ Delivered demos, assembled cross-ONS development team
- ▶ Coach colleagues in software design and effective versioning



Personal side project

- ▶ L^AT_EX University requirement
- ▶ Self-taught Python at ONS
- ▶ Wanted to inject flair whilst automating corporate documents

Business-critical application

- ▶ Lead project to deliver real-time financial estimates
- ▶ Complex data to visualise and communicate
- ▶ Templated cohesion between report releases

Internal collaborative platform

- ▶ SCS requested S&T to National Statistician
- ▶ Delivered demos, assembled cross-ONS development team
- ▶ Coach colleagues in software design and effective versioning



Personal side project

- ▶ L^AT_EX University requirement
- ▶ Self-taught Python at ONS
- ▶ Wanted to inject flair whilst automating corporate documents

Business-critical application

- ▶ Lead project to deliver real-time financial estimates
- ▶ Complex data to visualise and communicate
- ▶ Templated cohesion between report releases

Internal collaborative platform

- ▶ SCS requested S&T to National Statistician
- ▶ Delivered demos, assembled cross-ONS development team
- ▶ Coach colleagues in software design and effective versioning



As technical project lead, requirement to transpose regular income data stream into a monthly insights report on businesses' resilience in response to COVID-19, delivering to CO...



As technical project lead, requirement to transpose regular income data stream into a monthly insights report on businesses' resilience in response to COVID-19, delivering to CO...

Automate the boring stuff!





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" meaning 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 (initial 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 its separate 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.

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

    def prepare_export(self, video_data: str) -> None:
        """Prepare video data for exporting."""
        raise NotImplementedError

    def do_export(self, video_data: str) -> None:
        """Export the video data to a video."""
        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 prepare and do the export.

```
FACTORIES = {}

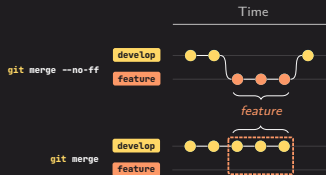
def read_factory() -> ExporterFactory:
    """Reads the user's desired output quality and returns the corresponding factory to use the appropriate video and audio exporters, and then prepare and do the export."""

    while True:
        export_quality = input()
        if export_quality in FACTORIES:
            return FACTORIES[export_quality]
        else:
            print("Invalid output quality option. (Export 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

Examples

Usage

Next steps

Contribution

Examples

- Table
- Plot

Table – Container style options

9/24

Introduction
○○○○○

Examples
●○○○

Usage
○○○○○

Next steps
○○

Contribution
○○

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



Table – Flexibility

10/24

Introduction

○○○○○

Examples

○●○○

Usage

○○○○○

Next steps

○○

Contribution

○○

C_1	C_2	C_3	C_4
1.00	2.00	3.00	4.00
1.01	2.01	3.01	4.01
1.02	2.02	3.02	4.02
1.03	2.03	3.03	4.03
1.04	2.04	3.04	4.04
⋮	⋮	⋮	⋮

<<page break>>

<<continue table>>

⋮	⋮	⋮	⋮
1.97	2.97	3.97	4.97
1.98	2.98	3.98	4.98
1.99	2.99	3.99	4.99



Plot – line

11/24

Introduction

○○○○○

Examples

○○●○

Usage

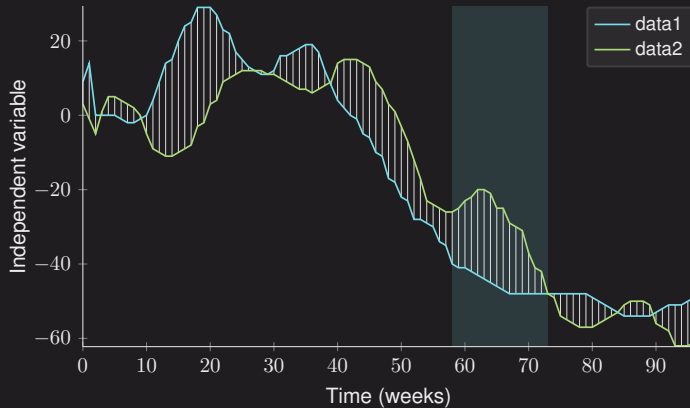
○○○○○

Next steps

○○

Contribution

○○



Introduction

○○○○○

Examples

○○●○

Usage

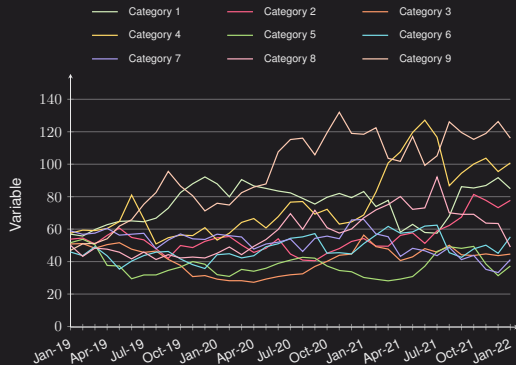
○○○○○

Next steps

○○

Contribution

○○



Source: company



Introduction

○○○○○

Examples

○○○●

Usage

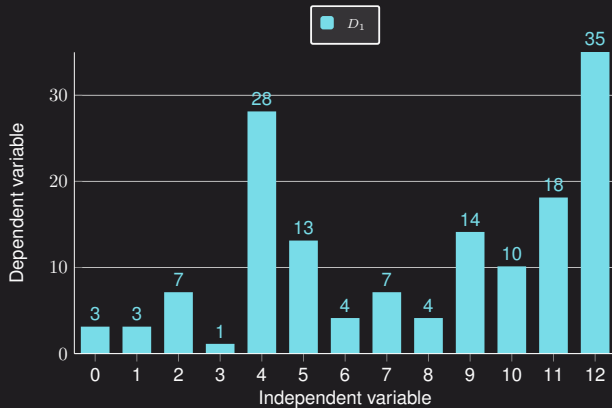
○○○○○

Next steps

○○

Contribution

○○



Introduction

○○○○○

Examples

○○○●

Usage

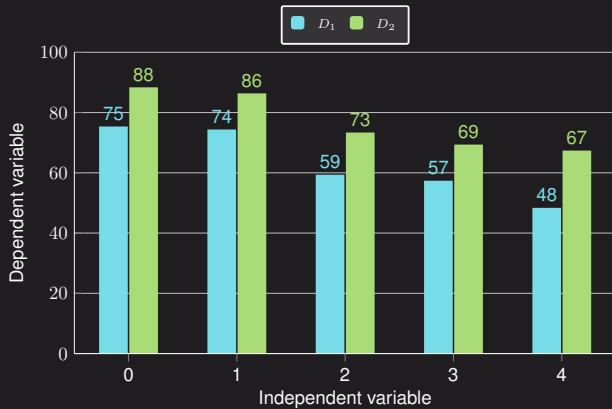
○○○○○

Next steps

○○

Contribution

○○



Introduction

○○○○○

Examples

○○○●

Usage

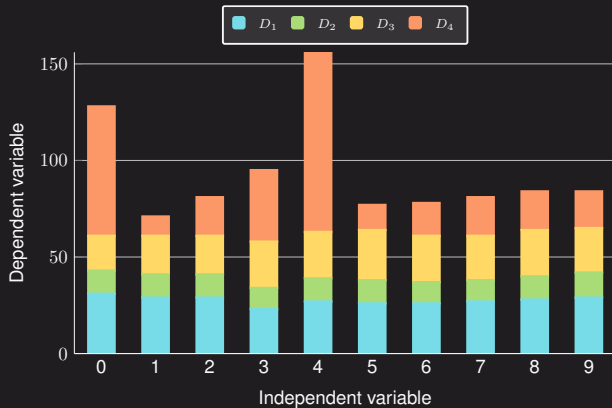
○○○○○

Next steps

○○

Contribution

○○



Introduction

Examples

Usage

Next steps

Contribution

Usage

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

Introduction
○○○○○

Examples
○○○○

Usage
●○○○○

Next steps
○○

Contribution
○○

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



Introduction
○○○○○

Examples
○○○○

Usage
●○○○

Next steps
○○

Contribution
○○

```
~/repo
├── /doctor
│   └── ...
├── /document
│   ├── main.tex
│   ├── ...
│   ├── book.pdf
│   └── slides.pdf
├── /tests
│   └── ...
├── app.py
└── venv
```



Introduction
○○○○○

Examples
○○○○

Usage
○○●○○

Next steps
○○

Contribution
○○

```
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 )
```



Introduction
○○○○○

Examples
○○○○

Usage
○○○●○

Next steps
○○

Contribution
○○

```
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
○○○○○

Examples
○○○○

Usage
○○○○●

Next steps
○○

Contribution
○○

Document class API



Introduction

Examples

Usage

Next steps

Contribution

Next steps

– Package

Introduction
○○○○○

Examples
○○○○

Usage
○○○○○

Next steps
●○

Contribution
○○

- ▶ Refine and publish existing PoC test library
- ▶ Deliver testing fundamentals workshop
- ▶ Build robust testing suite for **Doctor**



Package – Deployment

21/24

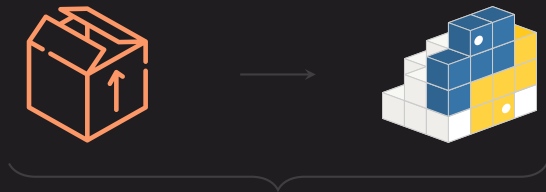
Introduction
○○○○○

Examples
○○○○

Usage
○○○○○

Next steps
○●

Contribution
○○



```
$ pip install doctor
```



Introduction

Examples

Usage

Next steps

Contribution

Contribution

– GitHub


Introduction
○○○○○


Examples
○○○○

Usage
○○○○○

Next steps
○○

Contribution
●○

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

 /ellsphillips/doctor



● TeX ● Python



Introduction
○○○○○

Examples
○○○○

Usage
○○○○○

Next steps
○○

Contribution
○○●

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!





Doctor

Elliott Phillips

 /ellsphillips/doctor

24/03/2022