Doctor

Elliott Phillips

 /ellsphillips/doctor

23/03/2022

Introduction

– **About**

– **Users**

– **Design**

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

About – Mission statement

3/22

Introduction
○●○○○○

Examples
○○

Usage
○○○○○

Next steps
○○

Contribution
○○

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

Transparent and opinionated, without the WYSIWYG faff.

About – Journey

4/22

Introduction
○○●○○○

Examples
○○

Usage
○○○○○

Next steps
○○

Contribution
○○

Personal side project

- ► LaTeX 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 demoes, assembled cross-ONS development team

- ► Coach colleagues in software design and effective versioning

About – Journey

4/22

Introduction
○○●○○○

Examples
○○

Usage
○○○○○

Next steps
○○

Contribution
○○

Personal side project

- ► LaTeX 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 demoes, assembled cross-ONS development team

- ► Coach colleagues in software design and effective versioning

Introduction
○○●○○○

Examples
○○

Usage
○○○○○

Next steps
○○

Contribution
○○

Personal side project

- ► LaTeX 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 demoes, assembled cross-ONS development team

- ► Coach colleagues in software design and effective versioning

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

BRR demo

Introduction
○○○○○○●

Examples
○○

Usage
○○○○○

Next steps
○○

Contribution
○○

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 '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 is not only has classes, but also tuples, dictionaries, Protocols and data classes.

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' reflecting 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 register factories without modifying the original code interface.

Figure 4.1: Factory pattern Unified Modelling 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.

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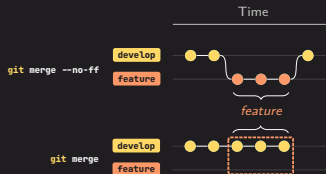
This user can choose to use the fast, high or master quality exporter to render their video and audio files, of which these are subclasses of the exporter factory. To facilitate these different factories, we create an object a dictionary `FACTORIES` and define a method `read_factory()` to read the user's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

```python
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

7/22

Introduction
○○○○○●

Examples
○○

Usage
○○○○○

Next steps
○○

Contribution
○○

Model – Feature branches

16/21

About
○○○○

Basics
○○○○○

Branches
○○○●○○

Closing
○○

Time

git merge --no-ff

develop

feature

feature

git merge

develop

feature

Git Good
Elliott Phillips · 10/11/2021

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

Examples

- **Table**
- **Plot**

Introduction
oooooo

Examples
●o

Usage
ooooo
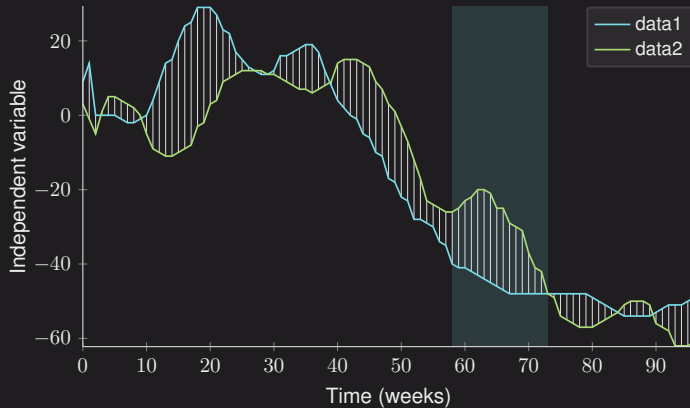
Next steps
oo

Contribution
oo

Introduction
○○○○○○

Examples
●○

Usage
○○○○○

Next steps
○○

Contribution
○○

Source: company
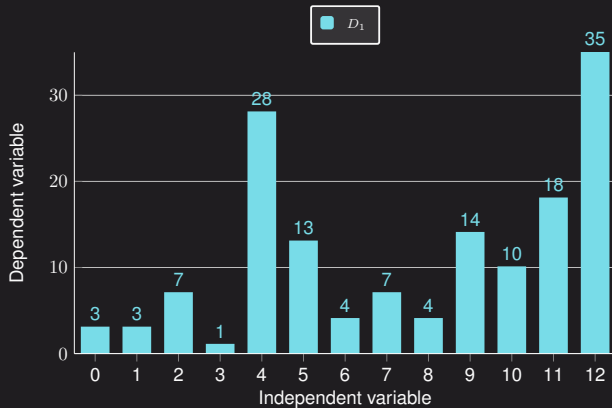
Plot – bar

10/22
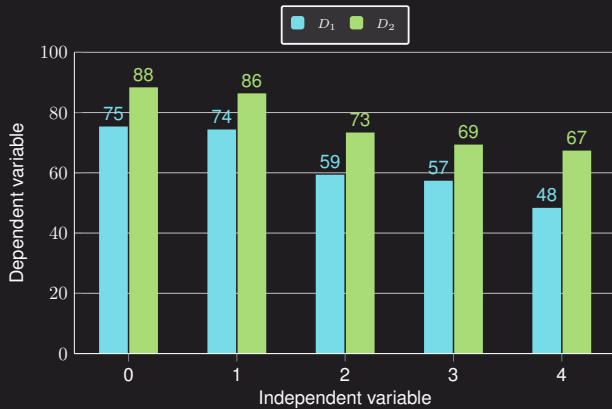
Introduction
000000

Examples
○●

Usage
00000

Next steps
00
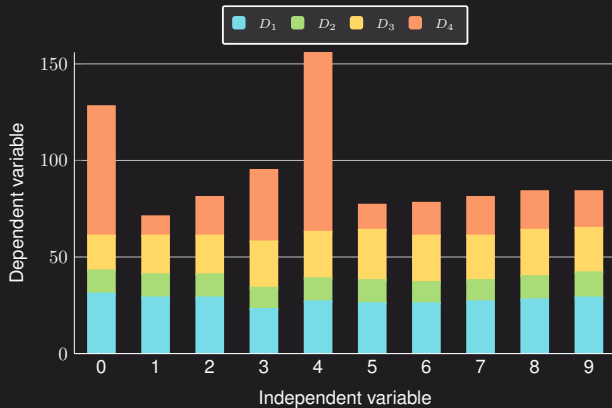
Contribution
00

Introduction
○○○○○○

Examples
○●

Usage
○○○○○

Next steps
○○

Contribution
○○

Usage

- **Installation**

- **API**

Installation – Project

12/22

Introduction
oooooo

Examples
oo

Usage
●oooo

Next steps
oo

Contribution
oo

```python
import doctor as dr


def main() -> None:
    ...


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

Installation – Module

13/22

Introduction
oooooo

Examples
oo

Usage
o●oooo

Next steps
oo

Contribution
oo

```
~/repo
  └ 📁 /doctor
       └ 📄 ...
  └ 📁 /document
            └ 📄 main.tex
            └ 📄 ...
            └ 📄 book.pdf
            └ 📄 slides.pdf
  └ 📁 /tests
       └ 📄 ...
  └ app.py
  └ venv
```

Introduction
oooooo

Examples
oo

Usage
oo●oo

Next steps
oo

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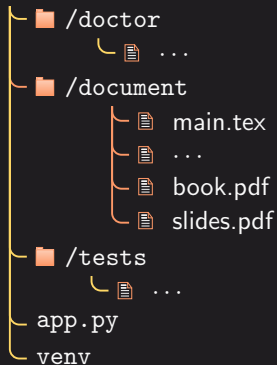
```
1  tabular = dr.table(
2      dr.data.table(
3          [
4                  dr.data.numeric.rand(10),
5                  dr.data.text.lorem(10),
6          ]
7      ),
8      columns=["numbers", "words"]
9  )
```

API – Plot

15/22

Introduction
○○○○○○

Examples
○○

Usage
○○○●○

Next steps
○○

Contribution
○○

```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",
    },
)
```

Introduction
oooooo

Examples
oo

Usage
ooooo●

Next steps
oo

Contribution
oo

Document class API

Next steps

- Package

Package – Testing

18/22

Introduction
oooooo

Examples
oo

Usage
ooooo

Next steps
●o

Contribution
oo

- Refine and publish existing PoC test library

- Deliver testing fundamentals workshop
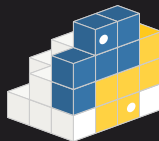
- Build robust testing suite for **Doctor**

```
$ pip install doctor
```

Contribution

- **GitHub**

Introduction
oooooo

Examples
oo

Usage
ooooo

Next steps
oo

Contribution
●o

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

 /ellsphillips/doctor

●TeX ●Python

GitHub – Sharing is caring

22/22

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

23/03/2022