# comma

***Release 0.5.3***

**Jérémie Lumbroso**

**May 17, 2022**

# CONTENTS:

**1 Why?** **1**

**2 Indices and tables** **3**

CHAPTER

ONE

# WHY?

Although Python, fortuitously, is “batteries included”, on occasion, some of the libraries end up being designed with APIs that don’t map well to what turns out to be the most common usage patterns. This is what happened with the various `urllib` libraries, incredibly powerful, but limiting users by its complexity—it was not straightforward, for instance, to use cookies: One of several problems that the requests library by @ken-reitz addressed. Indeed, `requests` abstracts power beneath simplicity, smart defaults, and discoverability.

For the CSV format, we are confronted with a similar situation. While both the JSON and YAML formats have packages that provide, one-command means to load content from files in those respective formats to a nested Python object, for the CSV format, the standard library has you use an iterator to access the data. Many details require significant syntax change (for instance the difference between have lists or dictionaries depends on the class that is used to read the file).

Since then, we also have several excellent libraries that, by providing great auto-detection (of dialect, file format, encoding, etc.) allow for hiding many details from the end user.

All this to say, *comma* will try to do exactly what you want when you do:

```
import comma
data = comma.load("file.csv")
data[0]["field"] = "changed value"
comma.dump(data, filename="file_modified.csv")
```

# CHAPTER TWO

# INDICES AND TABLES

- genindex
- modindex
- search