A COOKBOOK FOR CODE

```
for photo in photos:
    rename(photo)

print("Done!")
```

Python for Busy Creators

Four short chapters to automate the boring parts of running a food account

WRITTEN FOR NADIA · MADE WITH CLAUDE

Contents

1 Code is just a recipe **1**
What a program is 1
Installing Python 2
Your first line of code 2
What just happened 3

2 Captions on autopilot **5**
Variables are labelled jars 5
Filling in a template 6
Why bother? 7
Two small tricks with text 7

3 Rename 50 photos in 2 seconds **9**
A list of things 9
The for loop 9
Renaming files: dry run first 11
Doing it for real 12

4 Which post did best? **14**
What a CSV file is 14
Reading it with Python 15
Sorting by saves 16
The average 17
Where to go next 18

CHAPTER 1

Code is just a recipe

It's Sunday evening. You have 40 new photos from this week's shoot, three captions to write, and a vague feeling that last month's Es Cendol video did really well. You don't know *how* well, because checking means scrolling through your insights one post at a time.

None of this is hard. It's just boring. And boring, repeated work is exactly what computers are good at.

This book teaches you just enough Python to hand those chores to your laptop. No computer science degree. Four short chapters, one idea at a time.

What a program is

You already know how to follow a recipe. "Heat the oil. Add the shallots. Stir for two minutes." Each step is small and clear, and the order matters.

A **program** is a recipe for a computer. The big difference: a computer follows the recipe *exactly*. It never guesses what you meant. If the recipe says "add salt" and forgets to say how much, a human cook uses common sense. A computer stops and complains.

That sounds annoying, but it's also the good news. When your program works once, it works the same way every single time,

whether it's renaming 5 photos or 5,000.

Python is the language we'll write our recipes in. It's free, it's popular, and it reads almost like English. That's why so many beginners start with it.

Installing Python

You only do this once.

- **Windows:** go to python.org, click *Downloads*, and run the installer. On the first screen, tick **"Add python.exe to PATH"** before you click *Install*. (PATH is just the list of places your computer looks for programs. Ticking the box means you can run Python from anywhere.)
- **Mac:** go to python.org and install the latest version for macOS.

To check it worked, open the **terminal**. That's the app where you type commands instead of clicking. On Windows it's called *Terminal* or *Command Prompt*. On a Mac it's called *Terminal*. Type this and press Enter:

```
python3 --version
```

On Windows, if that doesn't work, try `python --version`. You should see something like `Python 3.12.3`. Any version starting with 3 is fine.

Your first line of code

Make a new folder on your desktop called `creator-scripts`. Open a plain text editor (VS Code is free and friendly), and create a file in that folder called `hello.py`. The `.py` at the end tells the computer it's a Python recipe.

Type these two lines into the file:

```
print("Hello, Nadia!")
print("Nadia's Kitchen: simple food, big flavour")
```

Save it. Then in the terminal, go into the folder and run it:

```
cd Desktop/creator-scripts
python3 hello.py
```

Output:

```
Hello, Nadia!
Nadia's Kitchen: simple food, big flavour
```

That's it. You wrote a program and the computer followed it.

What just happened

`print()` is a **function**: a ready-made mini recipe that Python already knows. You give it something inside the brackets, and it shows that thing on the screen.

The words inside the quote marks are called a **string**. A string is just text. The quotes tell Python where the text starts and ends, so it doesn't try to read your tagline as instructions.

Python ran the lines from top to bottom, one after another. Same as a recipe.

> **Try it.** Change `hello.py` so it prints your own account name on the first line and your own tagline on the second. Run it again. Then delete one of the quote marks and run it once more, just to see what an error looks like. (It's fine. Errors are how the computer says "I didn't understand that step.")

Recap

- A program is a recipe the computer follows exactly, from top to bottom.
- You write Python in a `.py` file and run it with `python3 filename.py`.
- `print()` shows things on the screen, and text goes inside quote marks.

CHAPTER 2

Captions on autopilot

Look at your last ten captions. They probably follow the same pattern: the dish, how long it takes, a call to save the post, the same handful of hashtags. You write that pattern by hand every time, and sometimes you forget the hashtags.

In this chapter you'll write the pattern once and let Python fill it in.

Variables are labelled jars

In your kitchen you keep things in jars with labels: *sugar*, *salt*, *chilli flakes*. When a recipe says "add the sugar", you don't need to know which shelf it's on. You just look for the label.

A **variable** is a labelled jar for a piece of information. You put something in it once, and after that you use the label.

```
account = "Nadia's Kitchen"
dish = "Sambal Matah"
minutes = 15

print(account)
print(dish)
print(minutes + 5)
```

Output:

```
Nadia's Kitchen
```

```
Sambal Matah
20
```

The `=` sign doesn't mean "equals" like in maths. It means "put this into the jar". So `dish = "Sambal Matah"` means: take the text *Sambal Matah* and put it in a jar labelled `dish`.

Notice that `minutes` has no quote marks. That makes it a **number**, not text, so Python can do maths with it. `minutes + 5` gave us 20.

Filling in a template

Now the useful part. Python has something called an **f-string**: a string with an `f` in front of it, where anything inside curly brackets `{ }` gets replaced by what's in that jar.

```
account = "Nadia's Kitchen"
dish = "Sambal Matah"
minutes = 15

caption = f"New on {account}: {dish}! Ready in {minutes}
↪  minutes. Save this for later."
print(caption)
print(len(caption), "characters")
```

Output:

```
New on Nadia's Kitchen: Sambal Matah! Ready in 15
↪  minutes. Save this for later.
79 characters
```

Your caption template now lives in one line. Tomorrow you change `dish` and `minutes`, run the file, and copy the result.

The last line uses another function, `len()`, which counts how long something is. Handy when a platform has a character

limit.

Why bother?

Fair question. Typing one caption by hand takes a minute.

But templates are where automation starts. Once your caption is a recipe instead of something you retype, you can make 30 of them at once, keep the hashtags consistent, or change the call to action across every caption with one edit. In the next chapter we'll learn how to repeat a recipe many times, and that's when this starts saving real time.

Two small tricks with text

Strings come with built-in tools. Put a dot after the string's jar and the tool's name:

- `dish.upper()` turns *Es Teler* into *ES TELER*. Good for a loud first word.
- `dish.lower()` does the opposite. Good for hashtags.

You can also start a new line inside a string with `\n` (a backslash and the letter n).

> **Try it.** Make a file called `caption.py` that builds a caption for tomorrow's post. Use four jars: `dish`, `main_ingredient`, `minutes`, and `hashtags`. Make the dish name loud with `.upper()` and put the hashtags on their own line.
>
> One possible answer:
>
> ```
> dish = "Es Teler"
> main_ingredient = "young coconut"
> minutes = 10
> hashtags = "#esteler #resepmudah #minumansegar"
> ```

```
caption = f"{dish.upper()} in {minutes} minutes! The
↪  secret is fresh {main_ingredient}.\n{hashtags}"
print(caption)
```

Output:

```
ES TELER in 10 minutes! The secret is fresh young
↪  coconut.
#esteler #resepmudah #minumansegar
```

Recap

- A variable is a labelled jar: `dish = "Sambal Matah"` puts text into a jar called `dish`.
- Text needs quote marks. Numbers don't, and you can do maths with them.
- An f-string like `f"Today: {dish}"` fills in a template from your jars.

CHAPTER 3

Rename 50 photos in 2 seconds

Your camera names photos like `IMG_0412.jpg`, `IMG_0413.jpg`, `IMG_0415.jpg`. Six months later you're looking for the sambal matah shots and every file looks the same.

Renaming them by hand is the definition of a boring chore: click, type, Enter, fifty times. Let's teach the computer to do it.

A list of things

So far each jar held one thing. A **list** is a jar that holds many things, in order. You write it with square brackets and commas:

```
dishes = ["nasi goreng", "soto ayam", "es teler"]
```

That's a list of three strings. It could just as easily be a list of 50 photo names.

The for loop

A **loop** repeats the same steps for every item in a list. Like a shopping list: for each item on the list, find it, put it in the basket.

```
dishes = ["nasi goreng", "soto ayam", "es teler"]
```

```
for dish in dishes:
    print("Filming:", dish)

print("Done for today!")
```

Output:

```
Filming: nasi goreng
Filming: soto ayam
Filming: es teler
Done for today!
```

Read the loop out loud: "for each dish in dishes, print *Filming:* and the dish." Each time around, Python puts the next item into the jar called `dish` and runs the indented lines again.

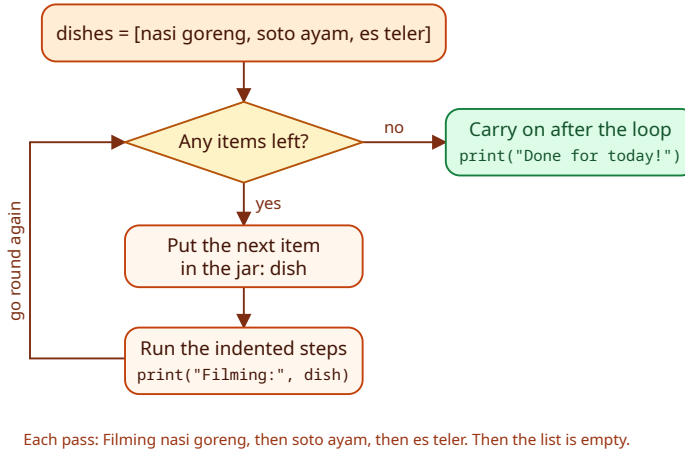

Figure 3.1: How a for loop works: take the next item, run the indented steps, repeat until the list is empty, then carry on.

Two details matter:

- The line with `for` ends with a **colon** `:`.
- The steps that repeat are **indented** (pushed right with 4 spaces). That's how Python knows what's inside the loop. `print("Done for today!")` isn't indented, so it runs once,

after the loop finishes.

Renaming files: dry run first

Now the real chore. Put a copy of a few photos in a folder called `photos` inside your `creator-scripts` folder. (A copy. We're careful with real files.)

When code changes files, always do a **dry run** first: a version that only *prints* what it would do, without doing it.

```
from pathlib import Path

folder = Path("photos")
photos = sorted(folder.glob("IMG_*.jpg"))

for number, photo in enumerate(photos, start=1):
    new_name = f"sambal-matah-{number:02}.jpg"
    print(photo.name, "->", new_name)
```

Output:

```
IMG_0412.jpg -> sambal-matah-01.jpg
IMG_0413.jpg -> sambal-matah-02.jpg
IMG_0415.jpg -> sambal-matah-03.jpg
IMG_0418.jpg -> sambal-matah-04.jpg
IMG_0421.jpg -> sambal-matah-05.jpg
```

New words, one at a time:

- `from pathlib import Path` borrows a tool for working with files. Python comes with lots of these tools; `import` takes one off the shelf.
- `folder.glob("IMG_*.jpg")` finds every file whose name starts with `IMG_` and ends with `.jpg`. The `*` means "anything here".

- `sorted(...)` puts them in order, so photo 1 really is the first one you shot.
- `enumerate(photos, start=1)` gives you a counter alongside each photo: 1, 2, 3…
- `{number:02}` writes the number with two digits, so `1` becomes `01`. That keeps the files in the right order when there are more than nine.

Doing it for real

The list looks right. Now swap the `print` line for the line that renames, and print the folder at the end so you can see the result:

```
from pathlib import Path

folder = Path("photos")
photos = sorted(folder.glob("IMG_*.jpg"))

for number, photo in enumerate(photos, start=1):
    new_name = f"sambal-matah-{number:02}.jpg"
    photo.rename(folder / new_name)

print(sorted(p.name for p in folder.iterdir()))
```

Output:

```
['sambal-matah-01.jpg', 'sambal-matah-02.jpg',
 ↪  'sambal-matah-03.jpg', 'sambal-matah-04.jpg',
 ↪  'sambal-matah-05.jpg']
```

Five files or five hundred, it takes the same two seconds.

> **Try it.** Make a test folder with a few copies of your photos. Change the script so the new names use today's dish and the date, like `es-teler-2026-10-26-01.jpg`. Do the dry run first. Only rename when the printed list looks right.

Recap

- A list holds many things in order: `["nasi goreng", "soto ayam"]`.
- A `for` loop runs the indented steps once for every item in the list.
- When code changes files, do a dry run that only prints, then do it for real.

CHAPTER 4

Which post did best?

Every creator has a gut feeling about what works. “Drinks do well.” “Sunday is a good day.” Sometimes the gut is right. Sometimes it just remembers the one post that went viral.

In this chapter you’ll let your own numbers answer the question.

What a CSV file is

Most platforms let you export your insights, and a common format is **CSV**. It stands for *comma-separated values*, and it’s simply a spreadsheet saved as plain text. Each line is a row, and commas separate the columns.

Here’s a small export for eight posts, saved as `posts.csv`:

```
date,day,dish,likes,saves
2026-09-01,Mon,Sambal Matah,412,96
2026-09-03,Wed,Soto Ayam,388,71
2026-09-05,Fri,Es Teler,951,240
2026-09-07,Sun,Nasi Goreng Kampung,603,158
2026-09-08,Mon,Tempe Mendoan,297,44
2026-09-10,Wed,Martabak Telur,520,133
2026-09-12,Fri,Es Cendol,1104,301
2026-09-14,Sun,Ayam Geprek,734,187
```

Open it in a spreadsheet app and you’d see a normal table. The

first line is the **header**: the column names.

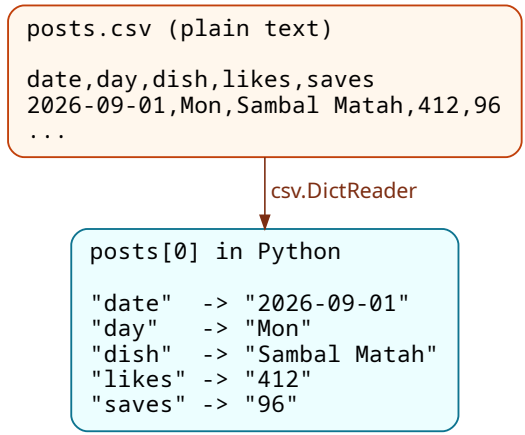

Figure 4.1: A CSV row becomes a dictionary in Python: each column name becomes a label you can look up.

Reading it with Python

Python has a built-in tool for CSV files. `csv.DictReader` reads each row into a **dictionary**: a set of jars inside one bigger jar, each labelled with a column name.

```
import csv

with open("posts.csv", newline="") as file:
    posts = list(csv.DictReader(file))

print(len(posts), "posts")
print(posts[0])
```

Output:

```
8 posts
{'date': '2026-09-01', 'day': 'Mon', 'dish': 'Sambal
↪  Matah', 'likes': '412', 'saves': '96'}
```

`posts[0]` means "the first post" (programmers count from 0, sorry). To get one value out, use its label: `posts[0]["dish"]` gives you `Sambal Matah`.

Look closely at `'saves': '96'`. The quote marks mean Python read the number as *text*. To do maths with it, you turn it into a number with `int()`, short for *integer* (a whole number).

Sorting by saves

Saves are a good signal: people save what they want to cook later. Let's find the top three.

```
import csv

with open("posts.csv", newline="") as file:
    posts = list(csv.DictReader(file))

def saves(post):
    return int(post["saves"])

posts.sort(key=saves, reverse=True)

for post in posts[:3]:
    print(post["dish"], "-", post["saves"], "saves")
```

Output:

```
Es Cendol - 301 saves
Es Teler - 240 saves
Ayam Geprek - 187 saves
```

`def` makes your own small function. `saves(post)` takes one post and gives back its saves as a number. `posts.sort(key=saves, reverse=True)` sorts the list by that number, biggest first. `posts[:3]` means "the first three".

Two of the top three are drinks. Your gut might have been right.

The average

One more useful number: the average saves per post, so you know what "normal" looks like.

```
import csv

with open("posts.csv", newline="") as file:
    posts = list(csv.DictReader(file))

total = 0
for post in posts:
    total = total + int(post["saves"])

print("Average saves per post:", total / len(posts))
```

Output:

```
Average saves per post: 153.75
```

Anything well above 154 saves is a post worth learning from.

> **Try it.** Find your best day to post. Group the saves by the `day` column and print the average for each day.
>
> One possible answer:
>
> ```
> import csv
>
> with open("posts.csv", newline="") as file:
>     posts = list(csv.DictReader(file))
> ```

```
by_day = {}
for post in posts:
    by_day.setdefault(post["day"],
↪   []).append(int(post["saves"]))

for day, numbers in by_day.items():
    print(day, sum(numbers) / len(numbers))
```

Output:

```
Mon 70.0
Wed 102.0
Fri 270.5
Sun 172.5
```

With only two posts per day this is a hint, not proof. Keep collecting and run it again next month.

Recap

- A CSV file is a spreadsheet saved as plain text, one row per line.
- `csv.DictReader` turns each row into a dictionary you read by column name.
- Numbers from a CSV arrive as text. Use `int()` before you do maths.

Where to go next

You now know variables, strings, lists, loops, files, and dictionaries. That's a real foundation. Some next steps that fit a creator's week:

- Generate a whole week of captions from a small CSV of dishes.
- Resize photos for each platform with the free `Pillow` library.

- Ask an AI assistant to explain any code you don't understand, line by line. You now know enough to follow along.