TOOL GUIDE

Building PDFs with WeasyPrint

How to install it, use it with Claude Code and Claude Desktop, and set up your brand kit so every PDF looks professional.

What Is WeasyPrint?

WeasyPrint is a free Python tool that turns an HTML file into a PDF. You write your document the same way you would build a webpage using HTML and CSS, then WeasyPrint converts it into a clean, printable PDF in seconds.

This is the tool used to build the document you are reading right now. Logo in the header, green accents, footers with page numbers, all done with basic HTML and CSS.

> **WHY IT MATTERS FOR AUTOMATION**
>
> If you are building systems that generate client proposals, reports, invoices, or onboarding docs automatically, WeasyPrint is how you make those outputs look professional without hiring a designer every time.

Where to Find It

WeasyPrint is open source and free. Here is where everything lives:

Resource	Where
Official docs	doc.courtbouillon.org/weasyprint
PyPI package	pypi.org/project/weasyprint
GitHub repo	github.com/Kozea/WeasyPrint
Install command	`pip install weasyprint`

How to Install It

WeasyPrint needs Python and one system library called Pango. Here is the full install for Mac.

Step 1: Install the Pango system library

Pango handles text rendering. You install it once using Homebrew:

```
# Install Homebrew first if you don't have it
# Go to brew.sh and follow their one-line install

# Then install Pango
brew install pango
```

Step 2: Install WeasyPrint via pip

```
pip install weasyprint

# Also install Jinja2 for templating (highly recommended)
pip install jinja2
```

Step 3: Verify it works

```
python3 -c "import weasyprint; print('WeasyPrint ready')"
```

MAC USERS ONLY

If you get a "cannot load library" error, add this before running your script: `DYLD_LIBRARY_PATH=/opt/homebrew/lib python3 your_script.py`. This tells Python where Homebrew's libraries live.

How It Works

The workflow is always the same three steps:

1. **Write an HTML template**

   Create a `.html` file with your layout, styles, and placeholders like `{{ client_name }}` for dynamic data.

2. **Inject your data with Jinja2**

   Use Python to load the template and fill in the placeholders with real values.

3. **Call WeasyPrint to render the PDF**

   One line of Python turns the filled HTML into a finished PDF file.

```
from weasyprint import HTML
import jinja2, pathlib

# Load your HTML template
template_path = pathlib.Path("/your/project/template.html")
template = jinja2.Template(template_path.read_text())

# Fill in the data
html_string = template.render(
    client="Acme Corp",
    date="March 2026",
    total="$4,800"
)

# Generate the PDF
HTML(string=html_string).write_pdf("/your/project/output/invoice.pdf")
```

Two Ways to Use It with Claude

You can have Claude help you build and run WeasyPrint in two different ways depending on your setup.

OPTION 1

Claude Code in the Terminal

This is the most direct option. You open your terminal, navigate to your project folder, and run `claude`. Claude Code reads your files and can write, edit, and run your WeasyPrint scripts directly. You approve each action before it happens.

OPTION 2

Claude Desktop with Claude Code connected

Claude Desktop is the regular chat app. When you connect Claude Code to it via MCP (see below), Claude can access your files from inside the desktop chat window. You get a friendlier interface but the same file access and code execution power.

Option 1: Using Claude Code in Terminal

1

Install Claude Code

You need Node.js installed first. Then run:

```
npm install -g @anthropic-ai/claude-code
```

2

Navigate to your project folder and start Claude

```
cd /your/project/folder
claude
```

3 **Tell Claude what you want in plain English**

Claude will write the template and script for you, then run it when you say so.

> WHAT TO SAY TO CLAUDE
>
> "Create a WeasyPrint script that generates a branded invoice PDF using these colors and logo"
> "Update the template to add a footer with page numbers"
> "Run the script and show me the output"

Option 2: Claude Desktop + MCP

This connects the Claude Desktop app to your computer's files and terminal through Claude Code.

1 **Download Claude Desktop**

Go to **claude.ai/download** and install it for Mac or Windows.

2 **Install Claude Code if you have not already**

```
npm install -g @anthropic-ai/claude-code
```

3 **Connect Claude Code to Claude Desktop**

```
claude mcp add claude-code
```

4

Restart Claude Desktop

Close it fully and reopen. You will see a tools icon at the bottom of the chat. That means it is connected and can read and run files on your computer.

> WHICH OPTION SHOULD YOU USE?
>
> Use **terminal + Claude Code** if you are comfortable in the terminal and want full control. Use **Claude Desktop + MCP** if you prefer a chat interface and are less technical. Both produce the same results.

Brand Kit Best Practices

The biggest difference between a PDF that looks professional and one that looks like a script made it is the brand kit. Here is the recommended setup.

Always embed your logo as base64

If you reference a logo by file path, it will break when the script runs on a different machine or a server. Convert it to base64 once at the top of your script and it will work everywhere.

```
import base64, pathlib

logo_path = pathlib.Path("/absolute/path/to/your/logo.png")
logo_b64  = base64.b64encode(logo_path.read_bytes()).decode()
logo_uri  = f"{logo_b64}"

# Pass logo_uri into your Jinja2 template as a variable
html = template.render(logo=logo_uri)
```

Define your brand colors as CSS variables

Put all your brand colors at the top of your CSS in one place. Change them once, they update everywhere.

```
:root {
  --brand-green:  #10b981;
  --brand-dark:   #047857;
  --text-primary: #111827;
  --text-mid:     #6b7280;
  --background:   #ffffff;
  --card-bg:      #f3f9f7;
  --border:       #e5e7eb;
}
```

Use CSS @page for headers and footers on every page

This is the proper way to get a logo and footer that show up on every page automatically, no matter how long the document is.

```
@page {
  size: A4;
  margin: 22mm 18mm 24mm 18mm;
  @top-center    { content: element(header); }
  @bottom-center { content: element(footer); }
}

#header {
  position: running(header);
  /* your header styles */
}

#footer {
  position: running(footer);
  /* page number: counter(page) */
}
```

Recommended file structure for any PDF project

```
your-pdf-project/
  template.html      # Your HTML layout and CSS
  generate.py        # Python script that runs WeasyPrint
  assets/
    logo.png         # Logo file (encoded to base64 at runtime)
  output/
    result.pdf       # Generated PDFs go here
```

Brand Kit Checklist

Before you send any PDF to a client or use it in public, run through this list:

- ✓ Logo appears in the header on every page
- ✓ Logo is embedded as base64, not a file path
- ✓ Footer shows company name, website, and page number on every page
- ✓ Brand colors are defined as CSS variables at the top of the stylesheet
- ✓ No blank pages (every page has content)
- ✓ Font size is 10pt or larger for body text
- ✓ Line height is at least 1.6 so text does not look cramped
- ✓ Code blocks and cards use page-break-inside: avoid so they do not split across pages
- ✓ Headings use page-break-after: avoid so they never sit alone at the bottom of a page
- ✓ Background is white or off-white for print readability

> **FINAL TIP**
>
> Keep your HTML template and your Python script completely separate. The template is a design file. The script is logic. Mixing them together makes both harder to maintain and harder to hand off to someone else.