

# Prompt, Package, Publish!

```
def ai_tools():
    generate_code()
    ± code
```

Using AI agents to create & publish a python package from scratch

Rhet Turnbull

HSV.py, 26 June 2025

# About Me

- Hobbyist programmer
- First code in Tandy BASIC on a TRS-80 Model III
- Reformed Perl hacker
- Pythonista since 2018
- [github.com/RhetTbull](https://github.com/RhetTbull)



ComputerMuseum / Dylan van Voorthuijsen

# The Problem I was Trying to Solve

- Maintain documentation for my `osxphotos` project with `Sphinx` and `sphinx-click`
- I use custom help methods for my `click` documentation
- `sphinx-click` doesn't support custom help methods

But who wants to write a plugin for Sphinx?

# Let AI Write The Code!

Lots of AI tools! Here's a sample of some of the tools I've tried:

- **Github Copilot**: plug-in for your editor (VSCode, Neovim, etc.)
- **Lovable**: create apps & websites
- **Cursor**: AI powered editor
- **Claude Code, Codex**: AI powered agents in your terminal!

# Claude Code

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

# OpenAI Codex

```
npm i -g @openai/codex
```

# Gemini CLI

```
npm install -g @google/gemini-cli
```

claude

▶ hsvpy\_ppp on ⚡ main [?] via 🐍 v3.13.2 (hsvpy\_ppp) took 7m6s

› claude

\* Welcome to **Claude Code!**

*/help for help, /status for your current setup*

cwd: /Users/rhet/code/hsvpy\_ppp

Tips for getting started:

1. Run `/init` to create a `CLAUDE.md` file with instructions for Claude
2. Use Claude to help with file analysis, editing, bash commands and git
3. Be as specific as you would with another engineer for the best results
4. ✓ Run `/terminal-setup` to set up terminal integration

> A python package that checks <https://www.meetup.com/hsv-py/> to let me know when the next HSV.py meetup is



claude

⌘2

Let me start by planning this task.

- **Update Todos**

- └
  - Examine the current project structure and files
  - Check the meetup page to understand its structure
  - Create the main meetup checker module
  - Add requirements/dependencies file
  - Create a simple CLI interface

- **Update Todos**

- └
  - **Examine the current project structure and files**
  - Check the meetup page to understand its structure
  - Create the main meetup checker module
  - Add requirements/dependencies file
  - Create a simple CLI interface

\* Scraping... (14s · ✘ 342 tokens · **esc** to interrupt)

&gt; █

? for shortcuts

claudie

- Create the main meetup checker module
- Add requirements/dependencies file
- Create a simple CLI interface

- **List(.)**
  - └ Listed **8** paths (ctrl+r to expand)
- **Fetch(<https://www.meetup.com/hsv-py/>)**
  - └ Fetching...

**Fetch**

<https://www.meetup.com/hsv-py/>  
Claude wants to fetch content from [www.meetup.com](https://www.meetup.com)

Do you want to allow Claude to fetch this content?

› 1. Yes

2. Yes, and don't ask again for [www.meetup.com](https://www.meetup.com)

3. No, and tell Claude what to do differently (esc)



claudie

⌘2

### └ Next HSV.py Meetup:

Title: Talk: Building Ubuntu for AWS / Building Python Packages from Scratch w/ AI

Date: Thursday, June 26, 2025 at 10:30 PM UTC

... +2 lines (ctrl+r to expand)

### ● Update Todos

- └  Examine the current project structure and files
- └  Check the meetup page to understand its structure
- └  Create the main meetup checker module
- └  Add requirements/dependencies file
- └  Create a simple CLI interface

- Created a Python package that checks the HSV.py meetup page. Run `python3 hsvpy_checker.py` to see the next meetup details. The script extracts structured data from the meetup page and displays the event title, date, time, and location.

>

▶▶ auto-accept edits on (shift+tab to cycle)

# HSV.py Meetup Checker

```
› python hsvpy_checker.py
```

Next HSV.py Meetup:

Title: Talk: Building Ubuntu **for** AWS / Building Python Packages from Scratch w/ AI

Date: Thursday, June 26, 2025 at 10:30 PM UTC

Location: GigaParts, Inc., 6123 University Dr, Huntsville, AL, Huntsville, AL

URL: <https://www.meetup.com/hsv-py/events/308413632/>

# A More Complex Example

This sample python project uses Click to build a simple CLI. It uses sphinx and sphinx\_click to generate documentation. However, the sample CLI uses a custom Click.Command class that has a custom get\_help() method to customize the help. This works fine when run from the command line (python3 cli.py --help) but the sphinx documentation (generated with sphinx\_click) shows only the standard help from the entry point docstring not the custom generated help. Create a sphinx plugin, sphinx\_click\_custom, that correctly generates the documentation from the custom Command class get\_help() method. Use flit to package the plugin. Write tests for the plugin using pytest. Use pytest stype methods, not unittest classes. Add a README.md for the project.

- I'll help you create a custom Sphinx plugin `sphinx_click_custom` that generates documentation from your custom Command class's `get_help()` method. Let me start by examining your current setup and understanding how `sphinx_click` works.
- Update Todos
  - └  Examine existing `cli.py` to understand custom Command class implementation
  - Research `sphinx_click` source code to understand how it extracts help text
  - Create `sphinx_click_custom` plugin that uses `get_help()` method
  - Check current Sphinx documentation setup in `docs/`
  - Update Sphinx configuration to use new plugin
  - Test documentation generation with custom plugin

*Note: I also used claude to generate the example `cli.py` that used a custom `get_help()` function to get this started.*

# Tips for Prompting the Assistant

- Treat the AI like a junior developer
- Be explicit
- Break complex tasks into smaller steps (CLI agents are getting good at this)
- Use iterative refinement (add tests, documentation, refactor, logging, etc.)

# Testing

- Why You Shouldn't Use AI To Write Your Tests
- Tends to write tests that are implementation specific
- Misses edge cases
- Wants to use unittest even when I specified pytest

- Now let me run all the tests to make sure the pytest-style functions work correctly:
- Bash(`python -m pytest tests/ -v`)
  - └ ===== test session starts ======platform darwin -- Python 3.13.2, pytest-8.4.0, pluggy-1.6.0 -- /Users/rhet/code/sphinx\_click\_custom/.venv/bin/pythoncachedir: .pytest\_cacherootdir: /Users/rhet/code/sphinx\_click\_custom... +56 lines (ctrl+r to expand)
- Perfect! Let me run with coverage to verify we still have good coverage:
- Bash(`python -m pytest tests/ --cov=sphinx_click_custom --cov-report=term-missing`)
  - └ ===== test session starts ======platform darwin -- Python 3.13.2, pytest-8.4.0, pluggy-1.6.0rootdir: /Users/rhet/code/sphinx\_click\_customconfigfile: pytest.ini... +19 lines (ctrl+r to expand)

# "Ensure mypy runs cleanly"

- I'll fix the mypy type errors identified in the `sphinx_click_custom/ext.py` file. Let me start by running mypy to see the current errors:
- **Bash**(cd /Users/rhet/code/sphinx\_click\_custom && python -m mypy sphinx\_click\_custom/ext.py)
  - └ Error: 1 ~code/sphinx\_click\_custom/docs  
0 ~code/sphinx\_click\_custom  
sphinx\_click\_custom/ext.py:116: error: Incompatible types in assignment (expression has type "bool | None", variable has type "bool | str") [assignment]  
sphinx\_click\_custom/ext.py:298: error: Item "None" of "str | None" has no attribute "upper" [union-attr]  
sphinx\_click\_custom/ext.py:310: error: Argument 1 to "\_format\_envvar" has incompatible type "Parameter"; expected "Option | Argument" [arg-type]  
sphinx\_click\_custom/ext.py:446: error: Name "\_should\_use\_custom\_help\_entirely" is not defined [name-defined]  
sphinx\_click\_custom/ext.py:545: error: Argument 1 to "\_format\_help" has incompatible type "str | None"; expected "str" [arg-type]  
Found 5 errors in 1 file (checked 1 source file)
- Now let me fix these type errors one by one:

- **Update**(sphinx\_click\_custom/ext.py)
  - └ Updated `sphinx_click_custom/ext.py` with 14 additions and 9 removals
    - 114           show\_default = opt.show\_default
    - 115       else:
    - 116           show\_default = ctx.show\_default
    - 117 +
    - 118 +     # Ensure show\_default is boolean or string, not None
    - 119 +     if show\_default is None:
    - 120 +       show\_default = False
    - 121
    - 122       if isinstance(show\_default, str):
    - 123           # Starting from Click 7.0 show\_default can be a string. This is
    - ...  
294       if auto\_envvar\_prefix is not None:
    - 295           params = []
    - 296           for param in ctx.command.params:
    - 297 -           if not param.envvar:
    - 297 +           if not param.envvar and param.name:
    - 298               param.envvar = f"{auto\_envvar\_prefix}\_{param.name.upper()}"
    - 299           params.append(param)

# Task Completed!

**sphinx\_click\_custom 0.2.0**

`pip install sphinx_click_custom` 

Released: Jun 21, 2025

Sphinx extension for documenting Click commands with custom `get_help()` methods.

**Navigation**

- Project description**
- Release history
- Download files

**Verified details**   
These details have been [verified by PyPI](#)

**Maintainers**

 RhetTbull

**Project description**

## sphinx\_click\_custom

A Sphinx extension for documenting Click CLI commands that use custom `get_help()` methods.

### The Problem

The standard `sphinx_click` extension is excellent for documenting Click commands, but it has a significant limitation: it only uses the command's docstring and standard Click help output and ignores custom `get_help()` methods.

I write a lot of CLI applications that override the `get_help()` method to provide enhanced help text with additional context, examples, or custom formatting. However, when trying to document these commands with `sphinx_click`, you lose all the custom help content.

[https://github.com/RhetTbull/sphinx\\_click\\_custom](https://github.com/RhetTbull/sphinx_click_custom)

# Simple Custom Help Example:

cli

CLI Overview

This is a sample CLI with custom help text.

Sample CLI with custom click command

```
cli [OPTIONS]
```

## Options

**--name <name>**

Name to greet

This help text will appear after the command description.

```
> /cost
└ Total cost:          $15.43
  Total duration (API): 1h 17m 27.5s
  Total duration (wall): 8h 43m 49.4s
  Total code changes:    4175 lines added, 1212 lines removed
  Token usage by model:
    claude-3-5-haiku: 322.1k input, 23.8k output, 0 cache read, 0 cache write
    claude-sonnet:   936 input, 115.1k output, 17.9m cache read, 2.1m cache write
```

## Worked Well

- Fulfilled the task with just a small amount of intervention
- Working package, documented, and published to pypi
- Basic tests, mypy runs cleanly, basic README
- Solved my problem!

## Didn't Work as Well

- Tests are very implementation specific
- Missed a few edge cases (some unique to my use case)
- Tends to solve edge cases with overly specific implementations
- Can get expensive (I was using metered API)

# Tips

- Always start with a clean git repo & new branch
- Customize your setup (CLAUDE.md, AGENT.md)
- Curate the list of allowed tools
- Install GitHub gh CLI (Claude knows how to use it)
- Ask the agent to read specific files (direct it to the problem)
- Ask the agent to make a plan to solve the problem
- YOLO! claude --dangerously-skip-permissions

# This Talk

<https://github.com/RhetTbull/hsvpypp>



