Absolutely, Andy — love the consistency goal. Here's a structured **30-day, 1-hour-per-day Python plan** you can run entirely in **Cursor AI's IDE**, plus **step-by-step GitHub setup** and a daily workflow to upload your work. The plan balances fundamentals, data work (useful for Sales Engineering), automation, APIs, testing, and a mini-capstone.

What you'll get out of this month

- Solid Python fundamentals and the confidence to build small tools quickly.
- A clean GitHub repo with **30 dated folders** (great for your portfolio).
- Reusable utilities for CSV analysis, API calls, and simple automation.
- A small **capstone project** relevant to Sales Engineering.

One-time setup (20–40 min)

Do this before Day 1. You can do all steps inside Cursor's integrated terminal.

1) Install tools

- **Python 3.11+** → python.org
- **Git** → git-scm.com
- **Cursor IDE** → cursor.com

Verify:

```
python --version
git --version
```

2) Create a GitHub account

1. Go to **https://github.com/**
2. Click **Sign up**, enter email, pick a username and password.
3. Verify email and complete onboarding.

3) Create your repository

- Repo name: `python-30-days-with-cursor`

- Visibility: **Private** or **Public** (your call)
- Add a **README.md**
- Add a **.gitignore** → choose **Python**
- License (optional): MIT

4) Connect local project to GitHub

Option A — simpler (HTTPS + Personal Access Token)

1. Create a token: **GitHub > Settings > Developer settings > Personal access tokens > Tokens (classic)** → New token → scopes: `repo`.
2. In Cursor terminal:

```
# one-time setup
git config --global user.name "Andy Leung"
git config --global user.email
"your_email@example.com"

# clone your new repo (copy the HTTPS URL from
GitHub)
git clone https://github.com/&amp;lt;your-username&amp;gt;/
python-30-days-with-cursor.git
cd python-30-days-with-cursor
```

First push will prompt for username and token (use token as password).

Option B — SSH (recommended long-term)

```
# generate a key
ssh-keygen -t ed25519 -C "your_email@example.com"
# start agent &amp;amp; add key
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
# copy the public key and add it at GitHub &amp;gt;
Settings &amp;gt; SSH and GPG keys
cat ~/.ssh/id_ed25519.pub
# then clone via SSH
git clone git@github.com:&amp;lt;your-username&amp;gt;/
python-30-days-with-cursor.git
```

```
cd python-30-days-with-cursor
```

5) Project scaffold (run inside the repo)

```
python -m venv .venv
# macOS/Linux
source .venv/bin/activate
# Windows PowerShell
# .venv\Scripts\Activate.ps1

pip install --upgrade pip
pip install pandas numpy matplotlib seaborn requests
beautifulsoup4 pytest typer rich

# freeze initial deps
pip freeze &gt; requirements.txt

# baseline structure
mkdir data day-01
echo "# Python 30 Days with Cursor" &gt; README.md
echo "__pycache__/" &gt;&gt; .gitignore
```

Initial commit:

```
git add .
git commit -m "Initial scaffold: env, deps,
structure"
git push
```

Your daily 60-minute workflow (use this every day)

0–5 min: Create today's folder & plan

```
# replace XX with zero-padded day number, e.g. 01,
02...
mkdir day-XX
cp requirements.txt day-XX/requirements.txt  #
optional
```

5–45 min: Build the exercise in Cursor

- Use **Cursor Chat** to generate a starter function or test.
- Ask Cursor to "**write tests first**" or "**refactor for readability**".
- Run code in the integrated terminal.

45–55 min: Self-check & mini-writeup

- Create `day-XX/README.md` with:
  - What you built
  - How to run it
  - What you learned
- Add at least 1 test or assertion if applicable.

55–60 min: Commit & push

```
git add .
git commit -m "Day XX: &lt;short description&gt;"
git push
```

Optional: One branch per day.

```
git checkout -b day-XX
# work...
git push -u origin day-XX
# open a PR on GitHub, merge when done
```

30-Day plan (1 hour/day)

Each day lists **Goal → Exercise → Files → Run → Acceptance Criteria → Stretch**.\ Use prefixes like `day-01/` for all files.

Week 1 — Core Python fundamentals

Day 01 — Warm-up & I/O

- **Goal:** Variables, input/output, basic math.
- **Exercise: Temperature converter CLI** (C↔F).
- **Files:** `temp_converter.py`, `README.md`
- **Run:** `python temp_converter.py --to f --c 22`
- **Accept:** Correct conversions & input validation.
- **Stretch:** Support Kelvin; add `--precision`.

Day 02 — Data structures

- **Goal:** Strings, lists, dicts, sets.
- **Exercise: Mini contact book** (in-memory CRUD via CLI).
- **Files:** `contacts.py`
- **Run:** `python contacts.py add --name "Ada" --phone 123` etc.
- **Accept:** Add/list/find/delete works.
- **Stretch:** Save/load contacts to JSON.

Day 03 — Control flow

- **Goal:** Loops, conditionals, comprehensions.
- **Exercise: FizzBuzz+Primes**: print FizzBuzz 1–100; list primes ≤N.
- **Files:** `control_flow.py`, `tests/test_control_flow.py`
- **Run:** `pytest -q`
- **Accept:** Tests for edge cases.
- **Stretch:** Sieve of Eratosthenes.

Day 04 — Functions & tests

- **Goal:** Functions, docstrings, unit tests.
- **Exercise: Bill splitter** with tip %, rounding options.
- **Files:** `bill_splitter.py`, `tests/test_bill_splitter.py`
- **Run:** `pytest -q`
- **Accept:** All tests pass; helpful `--help`.
- **Stretch:** Split unevenly by weights.

Day 05 — Files & paths

- **Goal:** Read/write files, `pathlib`.
- **Exercise: Log analyzer**: parse `data/app.log` (you create it) counting levels INFO/WARN/ERROR and top keywords.
- **Files:** `log_analyzer.py`, `data/app.log`
- **Run:** `python log_analyzer.py data/app.log`
- **Accept:** Outputs counts & top 5 keywords.
- **Stretch:** Export JSON summary.

Day 06 — Exceptions & validation

- **Goal:** Try/except, custom errors.
- **Exercise: Robust CSV grader**: read `data/grades.csv` and compute avg/median/std; handle missing/invalid rows.
- **Files:** `grades.py`, `data/grades.csv`
- **Run:** `python grades.py data/grades.csv`
- **Accept:** Skips bad rows, reports stats.
- **Stretch:** Output per-student report.

Day 07 — Mini project

- **Goal:** Persisting state.
- **Exercise: CLI To-Do** with JSON storage (`todo.json`): add/list/done/remove.
- **Files:** `todo.py`, `todo.json`
- **Run:** `python todo.py add "Call client"`
- **Accept:** Tasks persist across runs.
- **Stretch:** Due dates & priority sorting.

Week 2 — Data wrangling & visualization

Day 08 — NumPy basics

- **Goal:** Arrays, vectorization, broadcasting.
- **Exercise: Array stats toolkit** with mean/median/std, normalize columns.
- **Files:** `numpy_tools.py`
- **Run:** `python numpy_tools.py`
- **Accept:** Matches `numpy` built-ins.
- **Stretch:** Z-score normalization.

Day 09 — pandas basics

- **Goal:** DataFrame create/load/select/groupby.
- **Exercise: Sales CSV analysis** on `data/sales.csv` with columns: `date, rep, region, product, units, price`.
- **Files:** `sales_analysis.py`, `data/sales.csv`
- **Run:** `python sales_analysis.py`
- **Accept:** Prints total revenue, top rep, top region.
- **Stretch:** Revenue by product as table.

Sample `data/sales.csv` starter (paste into your file):

```
date,rep,region,product,units,price
2025-01-02,Andy,APAC,Widget-A,12,39.9
2025-01-03,Ada,EMEA,Widget-B,5,59.0
2025-01-04,Bob,APAC,Widget-A,20,39.9
2025-01-05,Ada,AMER,Widget-C,7,79.0
```

Day 10 — Data cleaning

- **Goal:** Missing values, types, dedupe, outliers.
- **Exercise: Clean messy sales** (`data/sales_dirty.csv`) and write `data/sales_clean.csv`.
- **Files:** `clean_sales.py`
- **Accept:** Coerced types; dropped dupes; filled NAs.
- **Stretch:** Summary report of changes.

Day 11 — Datetimes & resampling

- **Goal:** Parse dates, resample/group by time.
- **Exercise: Daily revenue trend** with 7-day moving average.
- **Files:** `sales_time_series.py`
- **Accept:** Outputs CSV and prints last 7 days MA.
- **Stretch:** Weekly and monthly rollups.

Day 12 — Visualization

- **Goal:** Matplotlib/Seaborn basics.
- **Exercise: Sales charts**: line (revenue over time), bar (revenue by product), box (units by region).
- **Files:** `sales_charts.py`, saves `charts/`
- **Accept:** PNGs saved, axes labeled, readable.
- **Stretch:** Add Seaborn theme & annotations.

Day 13 — Join & merge

- **Goal:** Multi-file processing.
- **Exercise:** Merge `data/sales_clean.csv` with `data/targets.csv` → report attainment % by rep.
- **Files:** `merge_targets.py`, `data/targets.csv`
- **Accept:** Correct join logic & output.
- **Stretch:** Flag under/over achieve reps.

Day 14 — Review & refactor

- **Goal:** Organize code, functions, modules.
- **Exercise:** Refactor Days 9–13 into `saleslib/` package with reusable functions; add docstrings.
- **Files:** `saleslib/__init__.py`, `saleslib/*.py`
- **Accept:** `from saleslib import load_sales` works.
- **Stretch:** Add minimal tests.

Week 3 — CLI, APIs, scraping, automation

Day 15 — Packaging & venv hygiene

- **Goal:** Project structure, `setup.cfg`/`pyproject.toml` (lightweight).
- **Exercise:** Turn `saleslib/` into an installable local package.
- **Files:** `pyproject.toml`
- **Run:** `pip install -e .`
- **Accept:** Import from anywhere in repo.
- **Stretch:** Add `ruff` or `black` config.

Day 16 — CLI with Typer

- **Goal:** Build ergonomic CLI.
- **Exercise: CSV filter CLI**: filter rows by column=value and save.
- **Files:** `csv_filter.py`
- **Run:** `python csv_filter.py --in data/sales.csv --col region --val APAC --out data/apac.csv`
- **Accept:** Works with helpful `--help`.
- **Stretch:** Multiple predicates with AND/OR.

Day 17 — HTTP & GitHub API

- **Goal:** Make GET requests, parse JSON.
- **Exercise: GitHub repo stats**: input `owner/repo`, output stars, forks,

open issues.

- **Files:** `gh_stats.py`
- **Run:** `python gh_stats.py --repo pandas-dev/pandas`
- **Accept:** Handles 404 and rate limits gracefully.
- **Stretch:** Fetch top N contributors.

Day 18 — Web scraping (safe target)

- **Goal:** Requests + BeautifulSoup.
- **Exercise:** Scrape https://quotes.toscrape.com/ — get top 50 quotes and authors; save CSV.
- **Files:** `scrape_quotes.py`
- **Accept:** CSV created with quotes & tags.
- **Stretch:** Handle pagination, dedupe.

Day 19 — Caching & JSON

- **Goal:** Serialize/deserialize; cache to file.
- **Exercise:** Add a caching layer to Day 17: store API results to `cache/` with TTL.
- **Files:** `gh_cache.py`
- **Accept:** Second run uses cache if fresh.
- **Stretch:** Pluggable cache backends.

Day 20 — Scheduling

- **Goal:** Simple scheduling patterns.
- **Exercise:** Run a task hourly (simulated) to append a timestamped metric to CSV; capture exceptions.
- **Files:** `scheduler_demo.py`
- **Accept:** Clean shutdown, logs written.
- **Stretch:** Use `schedule` lib or `APScheduler`.

Day 21 — File utilities

- **Goal:** OS ops, globbing, renaming.
- **Exercise: Media organizer**: from `downloads/`, move files into `photos/`, `docs/`, `other/` by extension; dry-run mode.
- **Files:** `organizer.py`
- **Accept:** Dry-run shows intended moves.
- **Stretch:** Hash-based duplicate detection.

Week 4 — OOP, testing, perf, capstone

Day 22 — OOP basics

- **Goal:** Classes, methods, encapsulation.
- **Exercise: Account class**: deposit/withdraw/transfer with balance checks; custom `InsufficientFunds`.
- **Files:** `bank.py`, `tests/test_bank.py`
- **Accept:** Tests pass for normal & error cases.
- **Stretch:** Transaction history and CSV export.

Day 23 — Pytest & coverage

- **Goal:** Test suite & fixtures.
- **Exercise:** Add/expand tests for `saleslib/` and Day 16 CLI.
- **Files:** `tests/` (multiple files)
- **Run:** `pytest -q --maxfail=1 --disable-warnings`
- **Accept:** ~80% coverage on key modules (rough estimate).
- **Stretch:** Add GitHub Actions CI (optional).

Day 24 — Logging & config

- **Goal:** `logging` module & config files (YAML/INI).
- **Exercise:** Add structured logging to 2–3 tools; read settings from a config file.
- **Files:** `logging_config.yaml`, updates to scripts
- **Accept:** Info/warn/error to console & file.
- **Stretch:** Log rotation.

Day 25 — Performance & profiling

- **Goal:** `timeit`, simple profiling.
- **Exercise:** Compare naive vs vectorized operations on a mid-size dataset; document results.
- **Files:** `perf_compare.py`, `day-25/README.md`
- **Accept:** Clear timing table & takeaway.
- **Stretch:** Use `cProfile` & `line_profiler` (if installed).

Day 26 — Dataclasses & typing

- **Goal:** `@dataclass`, type hints, `mypy` (optional).
- **Exercise: Inventory items** with price, cost, margin; load/save JSON; enforce types.
- **Files:** `inventory.py`
- **Accept:** Type-safe ops; simple tests.
- **Stretch:** Add `mypy.ini` and run `mypy .`.

Day 27 — Capstone (design)

- **Goal:** Plan a relevant tool for Sales Engineering.
- **Exercise:** Pick one:
  1. **Sales performance dashboard**: load `sales_clean.csv`, compute attainment vs target, and produce charts/HTML.
  2. **Deal pipeline analyzer**: ingest opportunities CSV, stage durations, conversion rates.
  3. **Renewal risk screener**: flag accounts by usage/decline heuristics.
- **Files:** `capstone/PLAN.md` (problem, data schema, user stories, CLI entrypoints, success criteria).
- **Accept:** Clear plan with 2–3 MVP features.
- **Stretch:** Wireframe CLI and file structure.

Day 28 — Capstone build (MVP)

- **Goal:** Implement core features per plan.
- **Exercise:** Build minimal CLI that ingests CSV and outputs key metrics to console and CSV.
- **Files:** `capstone/app.py`, `capstone/utils.py`, `capstone/data/`
- **Accept:** End-to-end run works on sample data.
- **Stretch:** Add simple charts to `capstone/outputs/`.

Day 29 — Capstone build (UX & polish)

- **Goal:** Better UX & resilience.
- **Exercise:** Add logging, input validation, nicer tables (use `rich`), helpful `--help`.
- **Files:** `capstone/…`
- **Accept:** Error messages clear; docs updated.
- **Stretch:** Config file support.

Day 30 — Capstone finalize & showcase

- **Goal:** Packaging & documentation.
- **Exercise:** Add `README.md` with screenshots (charts), usage examples, and a short video/GIF if you like. Tag a release.
- **Files:** `capstone/README.md`, `capstone/requirements.txt`
- **Accept:** `python capstone/app.py --help` works; repo is tidy.
- **Stretch:** Create a GitHub Release & badges.

Using Cursor effectively (quick tips)

- Start a file and ask:\ *“In this file, scaffold a Typer CLI with a `filter` command that loads a CSV and filters by a column=value.”*
- Ask Cursor to **generate tests first** for your function signatures.
- Use **inline chat** to **refactor** (e.g., “extract function,” “add type hints,” “add docstrings”).
- Keep prompts specific; paste a failing error into chat and ask for a fix suggestion.

Daily GitHub upload — checklists & commands

Checklist

- [ ] Code runs locally
- [ ] At least one test or assertion (where applicable)
- [ ] `day-XX/README.md` updated (what/why/how)
- [ ] Commit & push done

Commands

```
git add .
git commit -m "Day XX: &lt;your summary&gt;"
git push
```

If you forget to pull on another machine:

```
git pull --rebase
```

Optional quality-of-life

- **Pre-commit hooks** with `black`, `ruff`, `pytest` (later on).
- **GitHub Actions** to run tests on push (Day 23 stretch).
- Maintain a root `PROGRESS.md` with a checkmark table and links to each day.

Suggested repo layout (evolves through the month)

```
python-30-days-with-cursor/
├─ README.md
├─ requirements.txt
├─ data/
├─ day-01/ … day-30/
│   ├─ README.md
│   └─ (scripts, tests, data for that day)
├─ saleslib/           # from Week 2 refactor
├─ capstone/           # Week 4
└─ tests/              # shared tests (Week 3/4)
```

Want me to generate Day-01 files to jumpstart?

I can create the folder + starter script + a tiny test so you can run it immediately in Cursor.

Before we kick off:

- Do you prefer **HTTPS** or **SSH** for GitHub?
- Are you on **Windows**, **macOS**, or **Linux** (so I can tailor commands)?
- For the capstone, do you want to focus on **sales attainment**, **pipeline**, or **renewal risk**?