Hands-On Python Lab: From Zero to Useful Scripts (Beginner → Intermediate)

Audience: New to Python but comfortable with computers.

Format: 6 modules (~60–90 min each) with checkpoints, hands-on tasks, and challenge extensions. Includes a mini-project capstone.
Prereqs: A laptop (macOS/Windows/Linux), ability to install software.

---

0) Setup (30 min)

Goal: Have an isolated Python workspace and a working editor.

1. **Install Python 3.12+**
2. macOS: `brew install python` (or download from python.org)
3. Windows: Download Python from python.org (check "Add python.exe to PATH").
4. Linux: Use your distro's package manager (ensure 3.12+).
5. **Create a project folder**

```
mkdir python-lab && cd python-lab
```

6. **Create a virtual environment**

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

7. **Install tooling**

```
python -m pip install --upgrade pip
pip install ruff pytest requests pandas matplotlib
```

8. **Editor**

9. VS Code with Python extension recommended.

Checkpoint: `python --version` shows 3.12+ and `pytest -q` returns no tests collected.

---

1) Python Basics (60–90 min)

Concepts: REPL, variables & types, expressions, f-strings, input/output.

1. Start the REPL: `python`
   Try: `1+1`, `'hello'.upper()`, `len([1,2,3])`.
2. Create `basics.py` with:

```
name = input("Your name: ")
age = int(input("Your age: "))
print(f"Hi {name}! In 5 years you will be {age + 5}.")
```

3. **Mini-tasks**
4. Convert Fahrenheit↔Celsius using variables and arithmetic.
5. Build a tip calculator (subtotal, tax %, tip % → grand total).

Challenge: Parse a string like `"x=  42  ,  y=7"` into ints using `.split`, `.strip`, `int`.

Success Criteria: You can write/run a script, accept input, and produce correct output.

---

2) Flow Control & Collections (75–90 min)

Concepts: `if/elif/else`, `for`/`while`, lists, tuples, dicts, sets, list/dict comprehensions.

1. Create `flow.py` to classify a score:

```
score = int(input("Score 0-100: "))
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
else:
    grade = 'C or below'
print(grade)
```

2. **Collections lab** (`collections_lab.py`):
3. Given: `nums = [5, 1, 2, 5, 10, 2]` → unique sorted list.
4. Map codes: `{'AL':'Alabama','AK':'Alaska',...}` → lookups with graceful fallback.
5. Build a frequency counter for words in a sentence.

6. **Comprehensions**: produce `[n*n for n in range(10) if n%2==0]`.

Challenge: Given log lines like `"2025-09-20 INFO user=alice action=login"`, build a dict of `{user: count}`.

Success Criteria: Comfortable looping over data and branching on conditions.

---

3) Functions, Modules, Files & Exceptions (75–90 min)

Concepts: `def`, parameters/returns, docstrings, imports, pathlib, file IO, `try/except`.

1. **Functions** (`utils.py`):

```python
def percent_change(old: float, new: float) -> float:
    """Return percent change from old to new."""
    if old == 0:
        raise ValueError("old cannot be zero")
    return (new - old) / old * 100.0
```

2. **File IO** (`io_lab.py`):
3. Write 3 lines to `notes.txt`, then read them back and number each line.
4. Use `pathlib.Path` to check if a file exists.
5. **Exceptions**: Wrap file reads in `try/except FileNotFoundError` and print a friendly message.

Challenge: Build `csv_sum.py` that sums a column from `data.csv` (comma-separated, header row), handling missing files and bad rows gracefully.

Success Criteria: You can organize code into modules and handle common runtime errors.

---

4) Testing, Packages, and CLI (75–90 min)

Concepts: `pytest`, assertions, arranging tests, simple command-line interfaces with `argparse`.

1. **Tests**
   Create `test_utils.py`:

```python
from utils import percent_change
import math

def test_percent_change_up():
    assert math.isclose(percent_change(100, 125), 25.0)

def test_percent_change_zero_old():
    import pytest
    with pytest.raises(ValueError):
        percent_change(0, 5)
```

Run: `pytest -q`.

2. **CLI** (`cli_tip.py`):

```
import argparse

p = argparse.ArgumentParser()
p.add_argument("subtotal", type=float)
p.add_argument("--tax", type=float, default=0.0)
p.add_argument("--tip", type=float, default=0.18)
args = p.parse_args()

total = args.subtotal * (1 + args.tax) * (1 + args.tip)
print(f"Total: {total:.2f}")
```

Example: `python cli_tip.py 50 --tax 0.06 --tip 0.2`

Challenge: Convert any earlier script into a CLI tool with flags and add 2 tests.

Success Criteria: You can write tests and ship small reproducible scripts.

---

5) Talking to the Outside World (APIs, Web, Data) (90 min)

Concepts: HTTP requests, JSON, basic data analysis, simple plots.

1. **HTTP & JSON** (`weather_demo.py`):
2. Use `requests.get("https://api.github.com/repos/python/cpython")`
3. Parse JSON → print repo `stargazers_count`, `open_issues`.
4. **Data wrangling** (`data_lab.py`):
5. Use `pandas` to load `sample.csv` (make a small 10-row sample).
6. Compute summary stats and filter rows.
7. **Plot** (`plot_lab.py`):
8. With `matplotlib`, plot a line chart of a small numeric series.

Challenge: Fetch a small public JSON (no auth), normalize into a DataFrame, and plot one metric.

Success Criteria: Comfortable fetching data, parsing JSON/CSV, and producing a basic chart.

---

6) Capstone Mini-Project (2–3 hrs)

Choose **one** and complete end-to-end (design, implement, test, README). Keep scope small and ethical; only operate on data/systems you own or are authorized to test.

Option A: Log Analyzer

- Input: A directory of `.log` files (you provide small samples).
- Output: Top N users, error rate over time, and a CSV summary.
- Features: CLI flags for `--topN`, `--since YYYY-MM-DD`, `--export summary.csv`.
- Tests: At least 3 `pytest` tests for parsing & aggregation.

Option B: Personal Finance Summarizer

- Input: Exported CSV from your bank (or a synthetic sample).
- Output: Monthly spend by category, top merchants, and a plot.
- Features: CLI filters by month; export a PNG chart.
- Tests: 2–3 tests for category mapping & totals.

Option C: Uptime Pinger (Safe/Local)

- Input: A list of allowed URLs/hosts you control (e.g., `http://localhost:8000`).
- Output: Latency stats and availability report.
- Features: CLI for interval, count; CSV export.
- Tests: Mock `requests.get` to simulate up/down.

Deliverables: Source code, tests, sample data, and a concise `README.md` with usage examples.

Best Practices & Style

- Use `ruff` to lint/format: `ruff check .` and `ruff format .`
- Prefer `pathlib.Path` for file paths.
- Write docstrings and type hints (`from __future__ import annotations` in 3.12+ projects).
- Keep functions short and single-purpose.
- Add `if __name__ == "__main__":` guards to scripts intended to run.

Checkpoints & Self-Assessment

- **After Module 2:** I can write loops and use dicts/sets confidently.
- **After Module 4:** I have at least 3 passing tests and a CLI script with flags.
- **After Module 6:** I shipped a mini-project with tests, data, and a README.

Extensions (Optional, 60–90 min each)

- **OOP:** Classes, dataclasses, and simple domain models.
- **Concurrency:** `concurrent.futures` for parallel I/O tasks.
- **Packaging:** Turn your CLI into a pip-installable package using `pyproject.toml` and `pipx`.

Appendix: Starter Files & Hints

Sample `data.csv`

```
name,amount
alpha,10
beta,15
alpha,7
```

`csv_sum.py` **hint**

```python
from pathlib import Path
import csv
import sys

path = Path(sys.argv[1]) if len(sys.argv) > 1 else Path("data.csv")
try:
    with path.open() as f:
        reader = csv.DictReader(f)
        total = sum(float(r["amount"]) for r in reader)
    print(f"Total: {total}")
except FileNotFoundError:
    print(f"File not found: {path}")
except (KeyError, ValueError) as e:
    print(f"Bad data: {e}")
```

Matplotlib hint

```python
import matplotlib.pyplot as plt
vals = [1, 3, 2, 5, 4]
plt.plot(vals)
plt.title("Sample Trend")
plt.xlabel("Index")
plt.ylabel("Value")
plt.savefig("trend.png", dpi=150)
print("Wrote trend.png")
```

Requests/JSON hint

```python
import requests
resp = requests.get("https://api.github.com/repos/python/cpython", timeout=10)
resp.raise_for_status()
info = resp.json()
print({"stars": info["stargazers_count"], "open_issues": info["open_issues"]})
```

Pytest tip

```
pytest -q
pytest -k "percent and not zero" -q
```

Ethics & Safety

- Only test against systems/data you own or have explicit permission to use.
- Keep credentials and secrets out of source control.
- Use `.gitignore` for virtualenvs, caches, and local data.