A script another officer can run

Arguments instead of edits, logging instead of print, failing loudly instead of producing a wrong number — turning the analysis into something that runs next quarter without you in the room.

Pierre Robentz Cassion

Lesson 8 of 8

Python for Programme Data — Producing and handing over

What this lesson covers

- The handover test
- Arguments, not edits
- Fail loudly, early
- Logging, not print
- The shape of the script
- Testing the part that computes
- Making the run reproducible
- Handing it over
- Where this course ends

The handover test

- An analysis is finished when someone else can run it on next quarter's export without editing the code.

Arguments, not edits — In Python (cont.)

```python
# scripts/indicator_table.py
import argparse
from pathlib import Path

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="Compute GAM and SAM rates by commune from a screening register."
    )
    parser.add_argument(
        "--input", type=Path, required=True,
        help="Path to the screening register CSV.",
    )
    parser.add_argument(
        "--output", type=Path, required=True,
        help="Directory to write tables into. Created if missing.",
    )
```

Arguments, not edits — In Python (cont.)

```
    parser.add_argument(
        "--gam-threshold", type=int, default=125,
        help="MUAC cut-off in mm for global acute malnutrition (default: 125).",
    )
    parser.add_argument(
        "--min-assessment-rate", type=float, default=0.8,
        help="Communes below this assessment rate are flagged, not dropped.",
    )
    return parser.parse_args()
```

Arguments, not edits — Shell

```
uv run python scripts/indicator_table.py \
    --input data/raw/muac-screening-artibonite-2024.v1.csv \
    --output outputs/tables
```

Arguments, not edits

- Put the thresholds in arguments and the defaults in the code — A threshold that is only ever changed by editing a line...

Arguments, not edits — Shell

```
uv run python scripts/indicator_table.py --help
```

Fail loudly, early — In Python (cont.)

```python
def read_register(path: Path) -> pd.DataFrame:
    if not path.exists():
        raise SystemExit(f"Input file not found: {path}")

    muac = pd.read_csv(
        path,
        dtype={"child_id": "string", "commune": "string"},
        na_values={"muac_mm": ["-99"]},
    )

    expected = {"child_id", "commune", "screening_date", "muac_mm", "oedema"}
    missing = expected - set(muac.columns)
    if missing:
        raise SystemExit(f"Input is missing columns: {sorted(missing)}")

    if muac["child_id"].isna().any():
```

Fail loudly, early — In Python (cont.)

```
    raise SystemExit("Some rows have no child_id; refusing to continue.")

return muac
```

Fail loudly, early — In Python

```
import warnings
import pandas as pd

warnings.simplefilter("error", pd.errors.ChainedAssignmentError)
```

Logging, not print — In Python

```
import logging

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s %(levelname)-8s %(message)s",
    datefmt="%H:%M:%S",
)
log = logging.getLogger(__name__)

log.info("Read %d rows from %s", len(muac), args.input)
log.warning("%d communes below the assessment-rate floor", len(flagged))
log.info("Wrote %s", output_path)
```

Logging, not print — Example

```
09:14:02 INFO     Read 4218 rows from data/raw/muac-...v1.csv
09:14:02 WARNING  2 communes below the assessment-rate floor
09:14:02 INFO     Wrote outputs/tables/gam_by_commune.csv
```

The shape of the script — In Python (cont.)

```python
# scripts/indicator_table.py
"""Compute GAM and SAM rates by commune from a MUAC screening register."""

import argparse
import logging
from pathlib import Path

import pandas as pd

log = logging.getLogger(__name__)

GAM_DEFAULT, SAM_DEFAULT = 125, 115


def read_register(path: Path) -> pd.DataFrame:
    ...
```

The shape of the script — In Python (cont.)

```
def indicator_table(muac: pd.DataFrame, gam_mm: int, sam_mm: int) -> pd.DataFrame:
    measured = muac["muac_mm"].notna() | muac["oedema"].notna()
    gam = measured & ((muac["muac_mm"] < gam_mm) | (muac["oedema"] == True))
    sam = measured & ((muac["muac_mm"] < sam_mm) | (muac["oedema"] == True))

    table = (
        muac.assign(measured=measured, gam=gam, sam=sam)
        .groupby("commune", dropna=False, observed=True)
        .agg(
            screened=("child_id", "size"),
            assessed=("measured", "sum"),
            gam_cases=("gam", "sum"),
            sam_cases=("sam", "sum"),
        )
```

The shape of the script — In Python (cont.)

```
    )
    table["assessment_rate"] = (table["assessed"] / table["screened"]).round(3)
    table["gam_rate"] = (table["gam_cases"] / table["assessed"]).round(3)
    table["sam_rate"] = (table["sam_cases"] / table["assessed"]).round(3)
    return table.reset_index()


def main() -> None:
    args = parse_args()
    logging.basicConfig(level=logging.INFO, format="%(levelname)-8s %(message)s")

    muac = read_register(args.input)
    log.info("Read %d rows", len(muac))

    table = indicator_table(muac, args.gam_threshold, SAM_DEFAULT)
```

The shape of the script — In Python (cont.)

```
    thin = table.loc[table["assessment_rate"] < args.min_assessment_rate, "commune"]
    if len(thin):
        log.warning("Assessment rate below floor: %s", ", ".join(thin))

    args.output.mkdir(parents=True, exist_ok=True)
    destination = args.output / "gam_by_commune.csv"
    table.to_csv(destination, index=False)
    log.info("Wrote %s (%d communes)", destination, len(table))


if __name__ == "__main__":
    main()
```

The shape of the script

- Functions take arguments and return values — `indicator_table` does not read a file, does not know where output goes,...
- `main()` is the only place that does input and output — Reading, writing and logging live there; everything else is...
- The `if __name__ == "__main__"` guard — means importing this module from a notebook runs nothing
- Thin data is flagged, not dropped — A commune below the assessment-rate floor stays in the table with a warning beside...

Testing the part that computes — In Python (cont.)

```python
# tests/test_indicator_table.py
import pandas as pd
from indicator_table import indicator_table


def test_denominator_excludes_unmeasured_children():
    muac = pd.DataFrame({
        "child_id": ["a", "b", "c"],
        "commune": ["X", "X", "X"],
        "muac_mm": [110, 130, None],
        "oedema": [False, False, None],
    })

    table = indicator_table(muac, gam_mm=125, sam_mm=115)

    assert table.loc[0, "screened"] == 3
```

Testing the part that computes — In Python (cont.)

```
assert table.loc[0, "assessed"] == 2      # the unmeasured child is not a denominator
assert table.loc[0, "gam_rate"] == 0.5
```

Making the run reproducible — In Python (cont.)

```
import json
import subprocess
from datetime import datetime, timezone


def run_metadata(args) -> dict:
    return {
        "generated_at": datetime.now(timezone.utc).isoformat(timespec="seconds"),
        "input": str(args.input),
        "gam_threshold": args.gam_threshold,
        "pandas": pd.__version__,
        "commit": subprocess.run(
            ["git", "rev-parse", "--short", "HEAD"],
            capture_output=True, text=True,
        ).stdout.strip() or "not a git repository",
    }
```

Making the run reproducible — In Python (cont.)

```
(args.output / "run.json").write_text(json.dumps(run_metadata(args), indent=2))
```

Making the run reproducible — Shell

```
rm -rf outputs/
uv run python scripts/indicator_table.py --input data/raw/muac.csv --output outputs/tables
git status
```

Handing it over — Example

```
## Run

    uv sync
    uv run python scripts/indicator_table.py \
        --input data/raw/muac-screening-artibonite-2024.v1.csv \
        --output outputs/tables

Thresholds default to MUAC < 125 mm (GAM) and < 115 mm (SAM).
Pass --gam-threshold to change it; the value used is recorded in
outputs/tables/run.json.
```

Where this course ends

- You can build an environment that reinstalls offline, read any export without corrupting it, turn its codes into values, aggregate to a numerator and a denominator that came from the same operation, get the date arithmetic right, and hand the result to someone else as a script rather than as a favour.

Read the full lesson, with runnable code

https://data-analysis.cassion.dev/courses/python-for-programme-data/python-08-scripts-that-travel