Un script qu'un autre chargé peut exécuter

Des arguments au lieu de modifications, de la journalisation au lieu de print, un échec bruyant au lieu d'un chiffre faux — transformer l'analyse en quelque chose qui tourne au trimestre prochain sans vous dans la pièce.

Pierre Robentz Cassion

Leçon 8 sur 8

Python pour les données de programme — Produire et transmettre

Ce que couvre cette leçon

- Le test de la transmission
- Des arguments, non des modifications
- Échouer bruyamment, tôt
- Journaliser, non imprimer
- La forme du script
- Tester la partie qui calcule
- Rendre l'exécution reproductible
- Transmettre
- Où ce cours s'arrête

Le test de la transmission

- Une analyse est terminée quand quelqu'un d'autre peut l'exécuter sur l'export du trimestre prochain sans modifier le code.

Des arguments, non des modifications — En Python (suite)

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

def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="Calcule les taux de MAG et MAS par commune depuis un registre."
    )
    parser.add_argument(
        "--input", type=Path, required=True,
        help="Chemin du CSV du registre de depistage.",
    )
    parser.add_argument(
        "--output", type=Path, required=True,
        help="Repertoire ou ecrire les tableaux. Cree s'il manque.",
    )
```

Des arguments, non des modifications — En Python (suite)

```python
parser.add_argument(
    "--gam-threshold", type=int, default=125,
    help="Seuil de PB en mm pour la malnutrition aigue globale (defaut : 125).",
)
parser.add_argument(
    "--min-assessment-rate", type=float, default=0.8,
    help="Les communes sous ce taux d'evaluation sont signalees, non retirees.",
)
return parser.parse_args()
```

Des arguments, non des modifications — Shell

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

Des arguments, non des modifications

- Mettez les seuils dans les arguments et les valeurs par défaut dans le code — Un seuil qu'on ne change qu'en éditant. . .

Des arguments, non des modifications — Shell

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

Échouer bruyamment, tôt — En Python (suite)

```
def read_register(path: Path) -> pd.DataFrame:
    if not path.exists():
        raise SystemExit(f"Fichier d'entree introuvable : {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"Colonnes absentes de l'entree : {sorted(missing)}")

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

Échouer bruyamment, tôt — En Python (suite)

```
        raise SystemExit("Des lignes n'ont pas de child_id ; arret.")

    return muac
```

Échouer bruyamment, tôt — En Python

```python
import warnings
import pandas as pd

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

Journaliser, non imprimer — En Python

```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("Lu %d lignes depuis %s", len(muac), args.input)
log.warning("%d communes sous le plancher de taux d'evaluation", len(flagged))
log.info("Ecrit %s", output_path)
```

Journaliser, non imprimer — Exemple

```
09:14:02 INFO     Lu 4218 lignes depuis data/raw/muac-...v1.csv
09:14:02 WARNING  2 communes sous le plancher de taux d'evaluation
09:14:02 INFO     Ecrit outputs/tables/gam_by_commune.csv
```

La forme du script — En Python (suite)

```python
# scripts/indicator_table.py
"""Calcule les taux de MAG et MAS par commune depuis un registre de PB."""

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:
    ...
```

La forme du script — En Python (suite)

```
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"),
        )
```

La forme du script — En Python (suite)

```
    )
    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("Lu %d lignes", len(muac))

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

La forme du script — En Python (suite)

```
    thin = table.loc[table["assessment_rate"] < args.min_assessment_rate, "commune"]
    if len(thin):
        log.warning("Taux d'evaluation sous le plancher : %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("Ecrit %s (%d communes)", destination, len(table))


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

La forme du script

- Les fonctions prennent des arguments et renvoient des valeurs — `indicator_table` ne lit aucun fichier, ignore où va la...
- `main()` est le seul endroit qui fait des entrées-sorties — Lecture, écriture et journalisation y vivent ; tout le...
- Le garde `if __name__ == "__main__"` — fait qu'importer ce module depuis un notebook n'exécute rien
- Les données minces sont signalées, non supprimées — Une commune sous le plancher de taux d'évaluation reste dans le...

Tester la partie qui calcule — En Python (suite)

```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
```

Tester la partie qui calcule — En Python (suite)

```python
assert table.loc[0, "assessed"] == 2      # l'enfant non mesure n'est pas au denominateur
assert table.loc[0, "gam_rate"] == 0.5
```

Rendre l’exécution reproductible — En Python (suite)

```python
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 "pas un depot git",
    }
```

Rendre l'exécution reproductible — En Python (suite)

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

Rendre l’exécution reproductible — Shell

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

Transmettre — Exemple

```
## Executer

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

Les seuils valent par defaut PB < 125 mm (MAG) et < 115 mm (MAS).
Passez --gam-threshold pour changer ; la valeur employee est consignee
dans outputs/tables/run.json.
```

Où ce cours s'arrête

- Vous savez construire un environnement réinstallable hors ligne, lire n'importe quel export sans le corrompre, transformer ses codes en valeurs, agréger vers un numérateur et un dénominateur issus de la même opération, maîtriser l'arithmétique des dates, et remettre le résultat à quelqu'un d'autre sous forme de script plutôt que de service rendu.

Lire la leçon complète, avec le code exécutable

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