Repository Guidelines

Project Structure & Module Organization

- `cashflow/`: Python package.
  - `core/`: models, ledger, validation.
  - `engines/`: solvers (`dp.py`, `cpsat.py`).
  - `io/`: plan loading and markdown rendering.
  - `cli.py`: Typer CLI entry point.
- `cashflow/tests/`: `unit/`, `property/`, `regression/` suites.
- Root: `plan.json` (example input), `Makefile`, `pyproject.toml`, `requirements.txt`.

Build, Test, and Development Commands

- `make setup`: install dependencies from `requirements.txt` (Python 3.11+).
- `make test`: run the full `pytest` suite.
- `make lint`: static checks with `ruff` and `black --check`.
- `make format`: auto-format with `black` and fix lints.
- `make type`: type checks with `mypy`.
- `make verify`: cross-check DP vs CP-SAT objective.
- Run CLI locally:
  - `python -m cashflow.cli show|solve|export|verify`
  - After editable install: `pip install -e .` then `cash solve`.

Coding Style & Naming Conventions

- Use Black formatting (4-space indent). Keep lines per Black defaults.
- Type hints required for public functions; run `make type` before PRs.
- Naming: modules/functions `snake_case`, classes `PascalCase`, constants `UPPER_SNAKE_CASE`.
- Lint with `ruff`; keep imports ordered; avoid unused code and broad exceptions.

Testing Guidelines

- Frameworks: `pytest` + `hypothesis` for property-based checks.
- Location: `cashflow/tests/{unit,property,regression}/`; files `test_*.py`, functions `test_*`.
- Common commands:
  - `pytest -q`
  - `pytest -q cashflow/tests/unit -k cli`
- No strict coverage gate; add unit tests with behavior changes and property tests when applicable.

Commit & Pull Request Guidelines

- Prefer Conventional Commits (e.g., `feat: add dp tie-break`, `fix: guard negative balance`).
- PRs should include: clear description, linked issue, rationale, sample CLI output (e.g., `python -m cashflow.cli verify`), and tests.

- Keep changes focused; update docs when user-facing behavior changes.

Architecture Overview

- DP engine (`engines/dp.py`) produces feasible 30-day schedules over integer cents.
- Validator (`core/validate.py`) enforces constraints; CP-SAT (`engines/cpsat.py`) verifies lexicographic optimality.
- Input is `plan.json` at repo root; renderers output markdown tables for human review.