Best Practices

- **Follow PEP 8** - Python style guide
- **Use meaningful variable names**
- **Write docstrings** for functions
- **Keep functions small and focused**
- **Use virtual environments**
- **Handle errors properly**
- **Write tests**
- **Use version control (Git)**

```python
def calculate_area(length, width):
    """
    Calculate the area of a rectangle.

    Args:
        length (float): Length of the rectangle.
        width (float): Width of the rectangle.

    Returns:
        float: Area of the rectangle.
    """
    return length * width
```

Assignment

Expense Tracker CLI (Core Python + Files)

Must cover

- Functions + modules (`tracker/` package)
- Lists/dicts, comprehensions, sorting/filtering
- File handling (CSV/JSON)
- Exceptions + input validation
- Logging
- CLI arguments (argparse)

Features

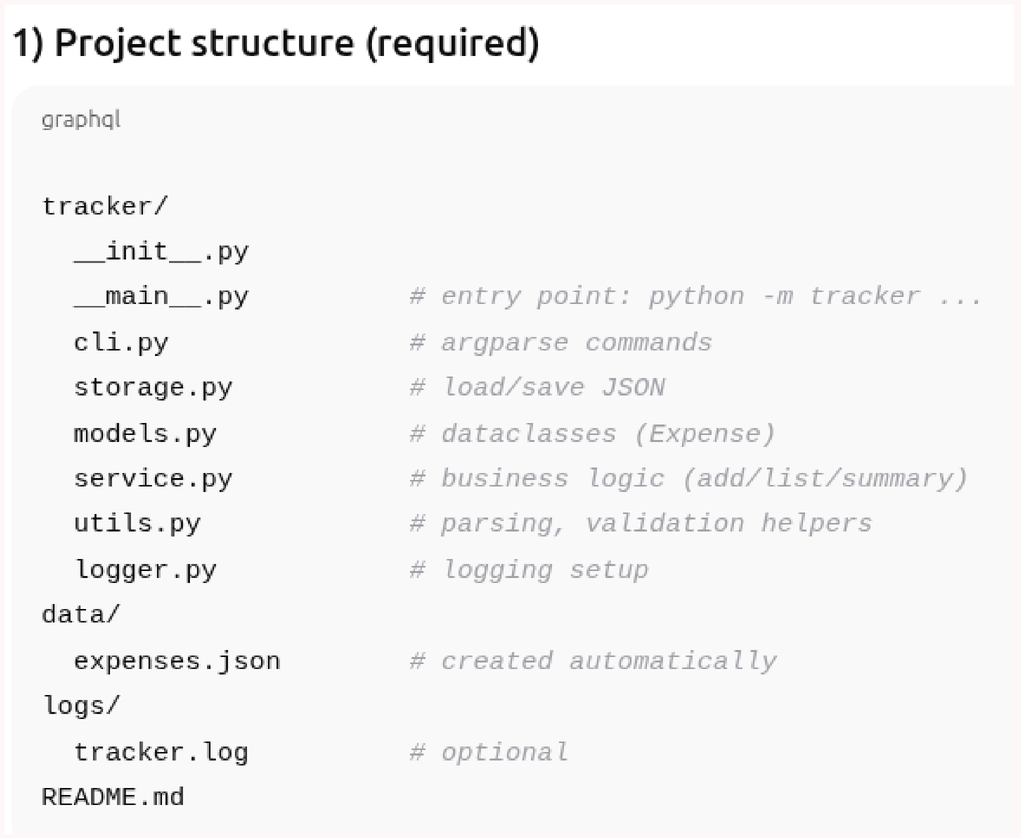
- `add` expense: date, category, amount, note
- `list` with filters: by month/category/min/max
- `summary`: totals per category + monthly total
- `export` to CSV
- Persistent storage in `data.json`

Acceptance

- Works from terminal: `python -m tracker add ...`, `list`, `summary`

Assignment Details

- `python -m tracker add ...`
- `python -m tracker list ...`
- `python -m tracker summary ...`

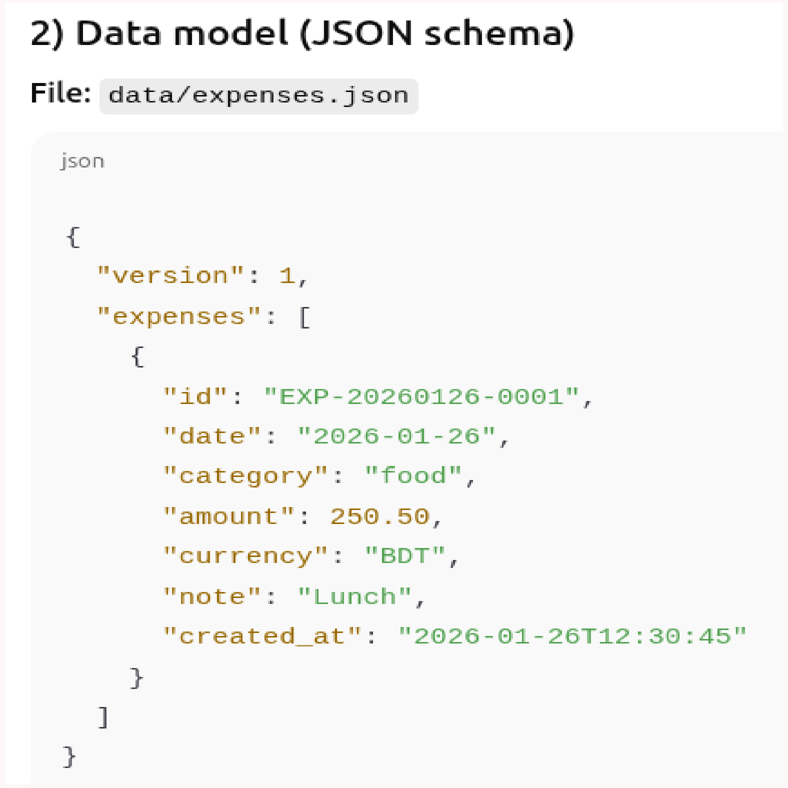


1) Project structure (required)

graphql

```
tracker/
  __init__.py
  __main__.py          # entry point: python -m tracker ...
  cli.py               # argparse commands
  storage.py           # load/save JSON
  models.py            # dataclasses (Expense)
  service.py           # business logic (add/list/summary)
  utils.py             # parsing, validation helpers
  logger.py            # logging setup
data/
  expenses.json        # created automatically
logs/
  tracker.log          # optional
README.md
```

Assignment Details



2) Data model (JSON schema)

File: `data/expenses.json`

json

```json
{
  "version": 1,
  "expenses": [
    {
      "id": "EXP-20260126-0001",
      "date": "2026-01-26",
      "category": "food",
      "amount": 250.50,
      "currency": "BDT",
      "note": "Lunch",
      "created_at": "2026-01-26T12:30:45"
    }
  ]
}
```

Rules

- `date` format: `YYYY-MM-DD`
- `amount` > 0 (float allowed)
- `category` is a string (recommend lowercase)
- `currency` default `"BDT"` (can be configurable)
- `id` unique (auto-generated)

Assignment Details

A) add — add an expense

Command

```
python -m tracker add --date 2026-01-26 --category
   food --amount 250.5 --note "Lunch"
```

Arguments

- `--date` (optional) default: today
- `--category` (required) e.g., `food`, `transport`, `rent`
- `--amount` (required) positive number
- `--note` (optional) default: empty string
- `--currency` (optional) default: BDT

Expected output

```
Added: EXP-20260126-0001 | 2026-01-26 | food |
   250.50 BDT | Lunch
```

Validation errors

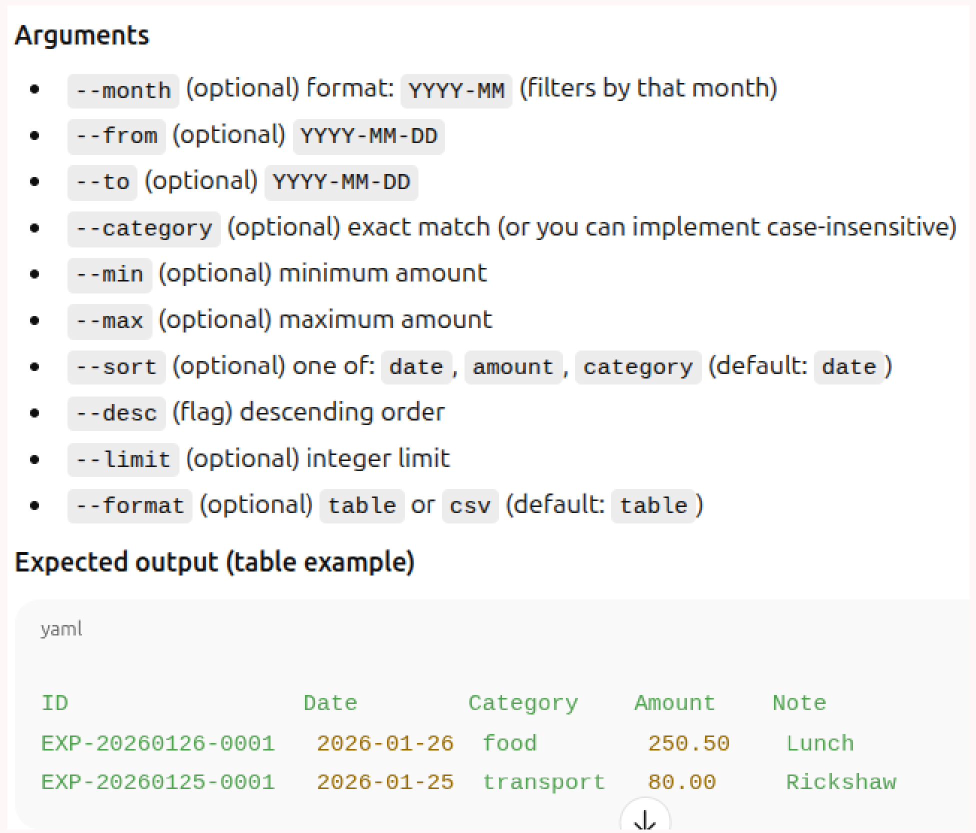
- Invalid date → `Error: date must be YYYY-MM-DD`
- Amount <= 0 → `Error: amount must be > 0`
- Missing category/amount → argparse error

Assignment Details

B) `list` — show expenses (with filters)

Command examples

- `python -m tracker list`
- `python -m tracker list --month 2026-01`
- `python -m tracker list --category food`
- `python -m tracker list --min 100 --max 1000`
- `python -m tracker list --sort amount –desc`
- `python -m tracker list --limit 20`



Arguments

- `--month` (optional) format: `YYYY-MM` (filters by that month)
- `--from` (optional) `YYYY-MM-DD`
- `--to` (optional) `YYYY-MM-DD`
- `--category` (optional) exact match (or you can implement case-insensitive)
- `--min` (optional) minimum amount
- `--max` (optional) maximum amount
- `--sort` (optional) one of: `date`, `amount`, `category` (default: `date`)
- `--desc` (flag) descending order
- `--limit` (optional) integer limit
- `--format` (optional) `table` or `csv` (default: `table`)

Expected output (table example)

yaml

```
ID                   Date        Category    Amount    Note
EXP-20260126-0001    2026-01-26  food        250.50    Lunch
EXP-20260125-0001    2026-01-25  transport   80.00     Rickshaw
```

Assignment Details

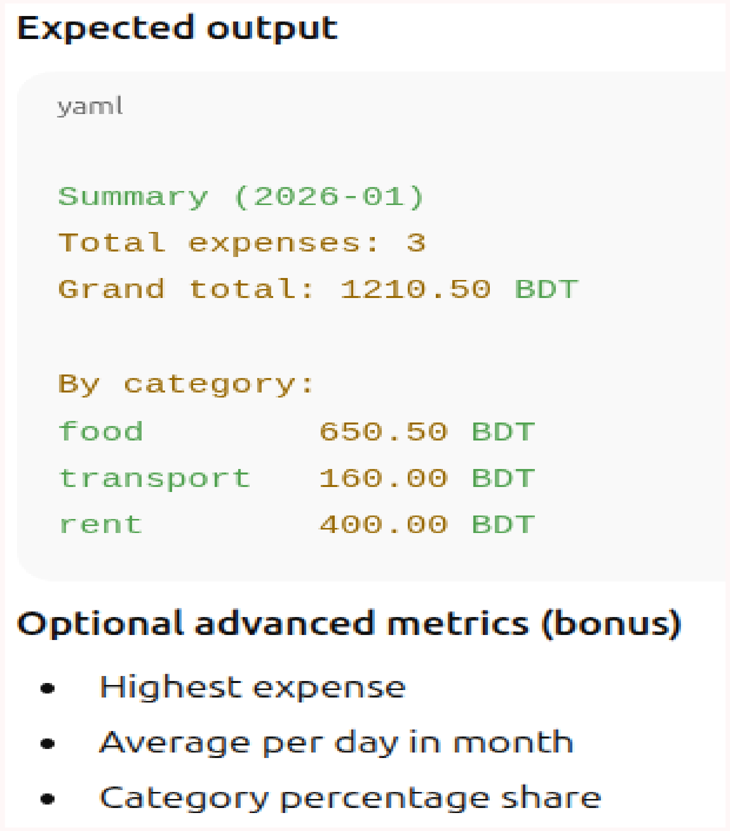
C) `summary` — totals & breakdown

Command examples

- `python -m tracker summary`
- `python -m tracker summary --month 2026-01`
- `python -m tracker summary --from 2026-01-01 --to 2026-01-31`
- `python -m tracker summary --category food`
-

Arguments

- Same filters as `list` (month/from/to/category)



Expected output

yaml

```
Summary (2026-01)
Total expenses: 3
Grand total: 1210.50 BDT

By category:
food           650.50 BDT
transport      160.00 BDT
rent           400.00 BDT
```

Optional advanced metrics (bonus)

- Highest expense
- Average per day in month
- Category percentage share

Assignment Details

D) (Optional but recommended) `delete` and `edit`

```
python -m tracker delete --id EXP-20260126-0001

python -m tracker edit --id EXP-20260126-0001 --amount 300
    --note "Lunch+coffee"
```

4) Business rules (service layer)

Implement in `service.py`:

- `add_expense(expense: Expense) -> Expense`
- `list_expenses(filters...) -> list[Expense]`
- `summary(filters...) -> dict`:
  - `count`, `grand_total`, `totals_by_category`

  Filters should be reusable between list and summary.

5) Logging (required)

- Log file: `logs/tracker.log`
- Log at least:
  - command called
  - validation failures
  - file read/write errors

Assignment Details

6) Acceptance checklist (how you'll grade it)

- Runs via `python -m tracker` …
- Creates `data/expenses.json` if missing
- add stores valid record and prints confirmation
- `list` supports at least: month + category + min/max + sorting
- `summary` prints grand total + category totals
- Handles empty dataset gracefully ("No expenses found")
- Validations + clean error messages
- No crashes on missing/corrupted file (show error, don't stack trace)

7) Quick test script (manual)

Run these in order:

```
python -m tracker add --date 2026-01-25 --category transport --amount 80 --
    note "Rickshaw"

python -m tracker add --date 2026-01-26 --category food --amount 250.5 --
    note "Lunch"

python -m tracker add --date 2026-01-26 --category rent --amount 400 --
    note "Room rent"


python -m tracker list

python -m tracker list --month 2026-01 --sort amount --desc

python -m tracker summary --month 2026-01
```