МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет електроніки та комп’ютерних технологій

ЗВІТ

про виконання лабораторної роботи №2

з курсу “Функціональне програмування”

«Функції вищого порядку, lambda, композиція»

Виконав:

Студент 2 курсу

групи ФеП-23

Чепара Станіслав

Перевірив: Доцент Франів В.А.

Львів-2025

Мета роботи

1) Закріпити поняття чистих функцій та функцій вищого порядку.

2) Побудувати конвеєр перетворень над колекцією даних за допомогою композиції.

3) Виконати агрегування через reduce без мутацій вхідних структур.

4) Перевірити коректність рішення автоматичними тестами.

Постановка задачі

Реалізувати конвеєр обробки записів виду {"id": int, "name": str, "age": int, "city": str, "purchases": list[float]} з такими кроками: нормалізація імен, фільтрація повнолітніх, обчислення total, підсилення total для вибраного міста, сортування за спаданням, відбір Top-N.

Окремо реалізувати агрегування через reduce: count, sum_total, avg_total. Вхідні дані не мутаються. Забезпечити модульні тести.

Структура проєкту

```
lab2/
  __init__.py
  lab2.py        # чисті функції, HOF, pipeline, reduce_stats
  main.py        # CLI
  tests/
    test_lab2.py
  sample_data.json
pyproject.toml
requirements.txt
README.md
```

Короткі теоретичні відомості

• Чиста функція — для тих самих аргументів повертає той самий результат і не має побічних ефектів.

• Функції вищого порядку — приймають/повертають інші функції (map, filter, reduce, фабрики функцій).

• Композиція — поєднання дрібних кроків у послідовний конвеєр перетворень.

Реалізація (коротко)

• normalize_names() — охайні імена у Title Case.

• only_adults(min_age) — фільтр повнолітніх записів.

• with_total() — додає total = sum(purchases).

• boost_city(city, factor) — множить total для міста, округлення round(..., 10).

• sort_by_total_desc() — сортування за total спадно.

• take(n) — відбір Top-N.

• build_pipeline(...) — складання конвеєра через композицію.

• reduce_stats(...) — агрегування: count, sum_total, avg_total.

Запуск (CLI)

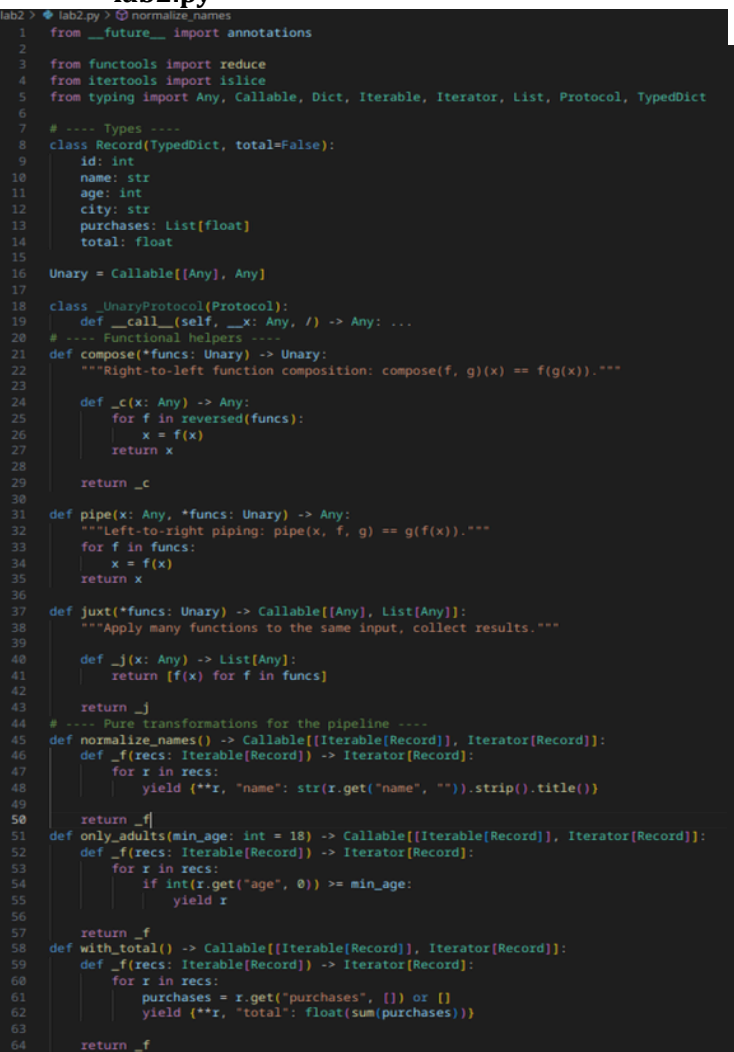
Команда:

python -m lab2.main --data lab2/sample_data.json --top 3 --boost-city Delhi --factor 1.1

Де --data — шлях до JSON; --top — кількість записів; --boost-city та --factor — параметри підсилення.

Скріншоти файлів

lab2.py



```python
from __future__ import annotations

from functools import reduce
from itertools import islice
from typing import Any, Callable, Dict, Iterable, Iterator, List, Protocol, TypedDict

# ---- Types ----
class Record(TypedDict, total=False):
    id: int
    name: str
    age: int
    city: str
    purchases: List[float]
    total: float

Unary = Callable[[Any], Any]

class _UnaryProtocol(Protocol):
    def __call__(self, __x: Any, /) -> Any: ...
# ---- Functional helpers ----
def compose(*funcs: Unary) -> Unary:
    """Right-to-left function composition: compose(f, g)(x) == f(g(x))."""

    def _c(x: Any) -> Any:
        for f in reversed(funcs):
            x = f(x)
        return x

    return _c

def pipe(x: Any, *funcs: Unary) -> Any:
    """Left-to-right piping: pipe(x, f, g) == g(f(x))."""
    for f in funcs:
        x = f(x)
    return x

def juxt(*funcs: Unary) -> Callable[[Any], List[Any]]:
    """Apply many functions to the same input, collect results."""

    def _j(x: Any) -> List[Any]:
        return [f(x) for f in funcs]

    return _j
# ---- Pure transformations for the pipeline ----
def normalize_names() -> Callable[[Iterable[Record]], Iterator[Record]]:
    def _f(recs: Iterable[Record]) -> Iterator[Record]:
        for r in recs:
            yield {**r, "name": str(r.get("name", "")).strip().title()}

    return _f
def only_adults(min_age: int = 18) -> Callable[[Iterable[Record]], Iterator[Record]]:
    def _f(recs: Iterable[Record]) -> Iterator[Record]:
        for r in recs:
            if int(r.get("age", 0)) >= min_age:
                yield r

    return _f
def with_total() -> Callable[[Iterable[Record]], Iterator[Record]]:
    def _f(recs: Iterable[Record]) -> Iterator[Record]:
        for r in recs:
            purchases = r.get("purchases", []) or []
            yield {**r, "total": float(sum(purchases))}

    return _f
def boost_city(city: str, factor: float) -> Callable[[Iterable[Record]], Iterator[Record]]:
    def _f(recs: Iterable[Record]) -> Iterator[Record]:
        for r in recs:
            base = float(r.get("total", 0.0))
            boosted = base * factor if r.get("city") == city else base
            yield {**r, "total": round(boosted, 10)}  # stabilize floats

    return _f
def sort_by_total_desc() -> Callable[[Iterable[Record]], List[Record]]:
    def _f(recs: Iterable[Record]) -> List[Record]:
        return sorted(recs, key=lambda r: (-float(r.get("total", 0.0)), int(r.get("id", 0))))

    return _f

def take(n: int) -> Callable[[Iterable[Record]], List[Record]]:
    def _f(recs: Iterable[Record]) -> List[Record]:
        return list(islice(recs, n))

    return _f
# ---- Reductions / aggregations ----
def reduce_stats(recs: Iterable[Record]) -> Dict[str, float]:
    """Return count, sum_total, avg_total without mutating inputs."""
    init: Dict[str, float] = {"count": 0.0, "sum_total": 0.0}
    acc = reduce(
        lambda a, r: {
            "count": a["count"] + 1.0,
            "sum_total": a["sum_total"] + float(r.get("total", 0.0)),
        },
        recs,
        init,
    )
    avg = acc["sum_total"] / acc["count"] if acc["count"] else 0.0
    return {"count": acc["count"], "sum_total": acc["sum_total"], "avg_total": avg}
# ---- Build pipeline ----
def build_pipeline(
    top_n: int = 2,
    city: str = "Delhi",
    factor: float = 1.1,
) -> Callable[[Iterable[Record]], List[Record]]:
    """Return a composite function that transforms a stream of records."""
    return compose(
        take(top_n),
        sort_by_total_desc(),
        boost_city(city, factor),
        with_total(),
        only_adults(18),
        normalize_names(),
    )
```

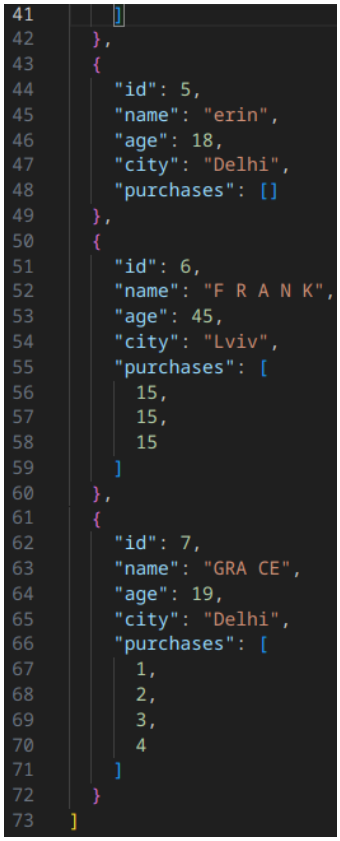
Main.py

```python
from __future__ import annotations

import argparse
import json
from typing import List, Any
from .lab2 import build_pipeline, reduce_stats, Record

def parse_args() -> argparse.Namespace:
    p = argparse.ArgumentParser(description="Lab 2: HOF pipeline demo")
    p.add_argument("--data", required=True, help="Path to JSON list of records")
    p.add_argument("--top", type=int, default=3, help="Top-N to take")
    p.add_argument("--boost-city", default="Delhi", help="City to boost totals for")
    p.add_argument("--factor", type=float, default=1.1, help="Boost factor")
    return p.parse_args()

def main() -> None:
    args = parse_args()
    with open(args.data, "r", encoding="utf-8") as f:
        data: List[Record] = json.load(f)

    pipeline = build_pipeline(top_n=args.top, city=args.boost_city, factor=args.factor)
    top = pipeline(data)
    stats = reduce_stats(top)

    print("Top records:")
    for r in top:
        print(json.dumps(r, ensure_ascii=False))

    print("\nStats:", json.dumps(stats, ensure_ascii=False))

if __name__ == "__main__":
    main()

```

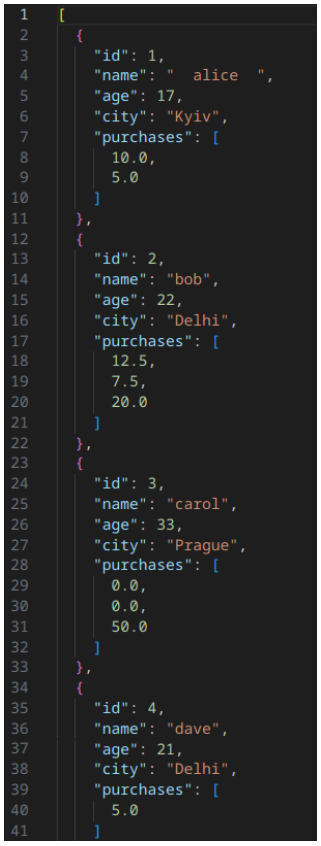
Test_lab2.py

```python
from __future__ import annotations

import copy
from typing import List
from lab2.lab2 import build_pipeline, reduce_stats, Record

def test_pipeline_top_and_sort():
    data: List[Record] = [
        {"id": 1, "name": "a", "age": 19, "city": "X", "purchases": [5, 5]},
        {"id": 2, "name": "b", "age": 19, "city": "Delhi", "purchases": [5, 6]},
        {"id": 3, "name": "c", "age": 17, "city": "Y", "purchases": [100]},
        {"id": 4, "name": "d", "age": 19, "city": "Delhi", "purchases": [1]},
    ]
    pipeline = build_pipeline(top_n=2, city="Delhi", factor=1.1)
    out = pipeline(data)
    # Adults only: ids 1,2,4. Totals: 10, 11*1.1=12.1, 1*1.1=1.1 -> sorted: 2,1 then 4 (but top 2 taken)
    assert [r["id"] for r in out] == [2, 1]
    assert out[0]["total"] == 12.1

def test_no_mutation():
    data: List[Record] = [
        {"id": 10, "name": " aa ", "age": 30, "city": "Delhi", "purchases": [1, 2, 3]},
    ]
    original = copy.deepcopy(data)
    pipeline = build_pipeline(top_n=1, city="Delhi", factor=2.0)
    _ = pipeline(data)
    assert data == original, "Input must not be mutated"

def test_reduce_stats():
    data: List[Record] = [
        {"id": 1, "name": "x", "age": 19, "city": "Delhi", "purchases": [10]},
        {"id": 2, "name": "y", "age": 19, "city": "Z", "purchases": [5]},
    ]
    out = build_pipeline(top_n=5, city="Delhi", factor=1.1)(data)
    stats = reduce_stats(out)
    # records: 2. totals: (10*1.1)=11, 5 -> sum=16, avg=8
    assert stats["count"] == 2.0
    assert abs(stats["sum_total"] - 16.0) < 1e-9
    assert abs(stats["avg_total"] - 8.0) < 1e-9

```

Sample_data.json

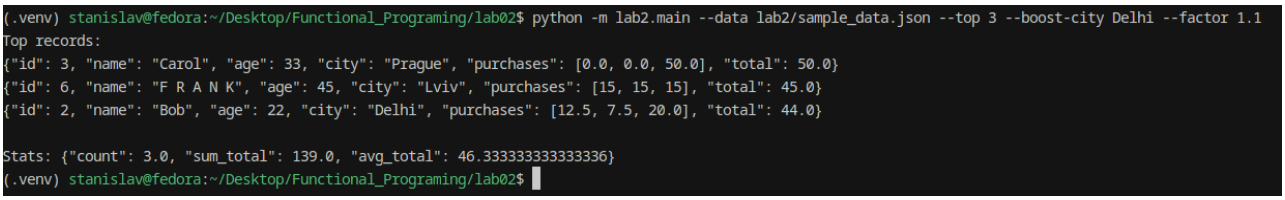


```
    ]
  },
  {
    "id": 5,
    "name": "erin",
    "age": 18,
    "city": "Delhi",
    "purchases": []
  },
  {
    "id": 6,
    "name": "F R A N K",
    "age": 45,
    "city": "Lviv",
    "purchases": [
      15,
      15,
      15
    ]
  },
  {
    "id": 7,
    "name": "GRA CE",
    "age": 19,
    "city": "Delhi",
    "purchases": [
      1,
      2,
      3,
      4
    ]
  }
]
```

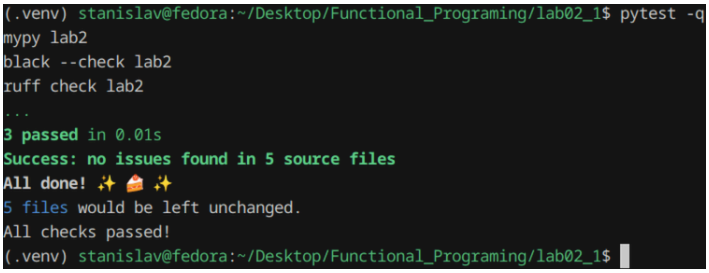


```
[
  {
    "id": 1,
    "name": "  alice  ",
    "age": 17,
    "city": "Kyiv",
    "purchases": [
      10.0,
      5.0
    ]
  },
  {
    "id": 2,
    "name": "bob",
    "age": 22,
    "city": "Delhi",
    "purchases": [
      12.5,
      7.5,
      20.0
    ]
  },
  {
    "id": 3,
    "name": "carol",
    "age": 33,
    "city": "Prague",
    "purchases": [
      0.0,
      0.0,
      50.0
    ]
  },
  {
    "id": 4,
    "name": "dave",
    "age": 21,
    "city": "Delhi",
    "purchases": [
      5.0
    ]
  }
```

запуск програми (вивід) і тестів



```
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab02$ python -m lab2.main --data lab2/sample_data.json --top 3 --boost-city Delhi --factor 1.1
Top records:
{"id": 3, "name": "Carol", "age": 33, "city": "Prague", "purchases": [0.0, 0.0, 50.0], "total": 50.0}
{"id": 6, "name": "F R A N K", "age": 45, "city": "Lviv", "purchases": [15, 15, 15], "total": 45.0}
{"id": 2, "name": "Bob", "age": 22, "city": "Delhi", "purchases": [12.5, 7.5, 20.0], "total": 44.0}

Stats: {"count": 3.0, "sum_total": 139.0, "avg_total": 46.333333333333336}
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab02$
```



```
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab02_1$ pytest -q
mypy lab2
black --check lab2
ruff check lab2
...
3 passed in 0.01s
Success: no issues found in 5 source files
All done! ✨ 🍰 ✨
5 files would be left unchanged.
All checks passed!
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab02_1$
```

Висновки

Реалізовано конвеєр обробки даних за допомогою чистих функцій та функцій вищого порядку, побудовано композицію кроків без мутацій вхідних структур і підтверджено коректність автоматичними тестами.