МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет електроніки та комп'ютерних технологій

ЗВІТ

про виконання лабораторної роботи №3

з курсу "Функціональне програмування"

«Незмінні структури даних (immutability) у Python»

Виконав:

Студент 2 курсу

групи ФеП-23

Чепара Станіслав

Перевірив: Доцент Франів В.А.

Львів-2025

Мета роботи

1) Застосувати tuple, frozenset, dataclass(frozen=True), MappingProxyType.

2) Замінити мутації на чисті оновлення (повернення нових копій).

3) Порівняти час виконання тестів до/після через показ топ-повільних тестів.

Структура проєкту

```
lab03/
  model.py     # Item (NamedTuple), Order (@dataclass frozen), make_meta
  core.py      # add_item, add_tag, pay, top_expensive_items, freeze
  tests/
    test_immutability.py
README.md, pyproject.toml, requirements.txt
```

Короткі теоретичні відомості

• Чисті оновлення: жодних in-place змін, повертаємо нові об’єкти.

• Незмінні структури: tuple замість list, frozenset замість set, read-only MappingProxyType.

• Frozen dataclass: заборона зміни полів, зручні оновлення через replace().

Реалізація

• model.py: Item, Order(frozen, slots), make_meta(). Методи: subtotal(), add_tag(), with_item(), pay(), with_meta().

• core.py: чисті функції, що делегують у методи Order; утиліта freeze() для «глибокого» read-only.

• tests/test_immutability.py: перевірка відсутності мутацій, read-only meta, хешованості, типів колекцій, freeze().

Команди для перевірки

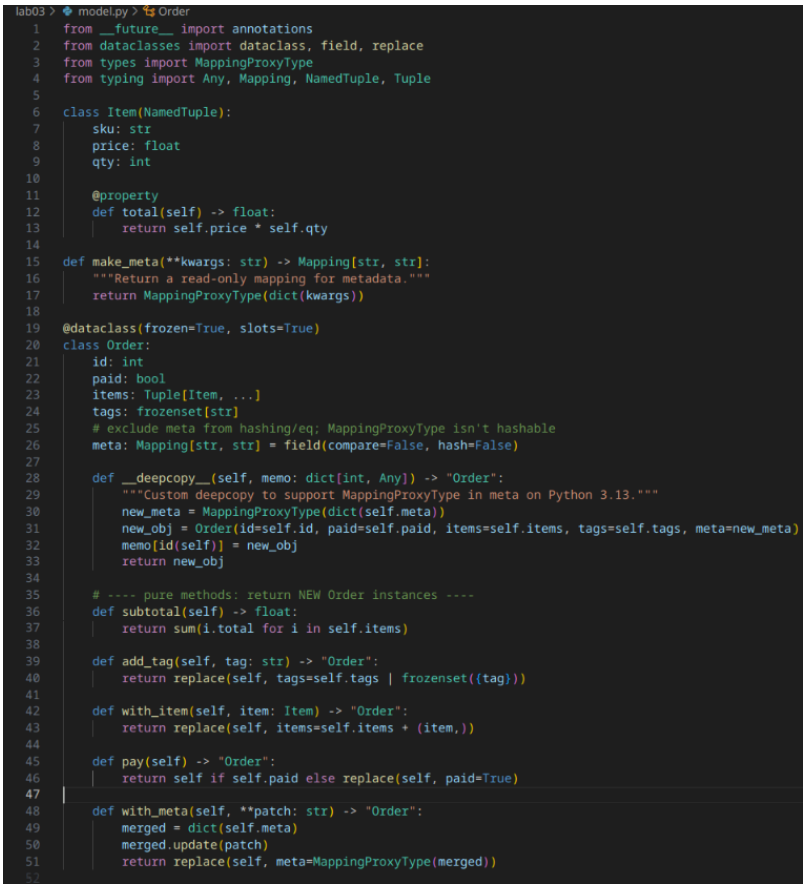
Створення середовища та запуск тестів:

```
python -m venv .venv

source .venv/bin/activate   # Windows: .venv\Scripts\activate

pip install -r requirements.txt

pytest -q
```

pytest -q --durations=5

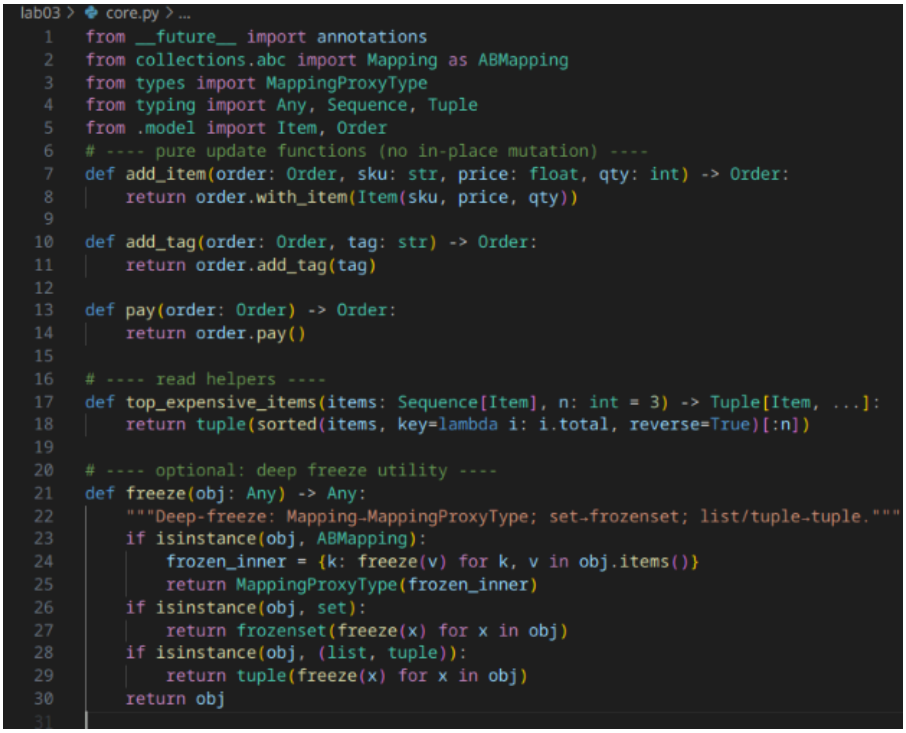
Скріншоти

model.py



```python
from __future__ import annotations
from dataclasses import dataclass, field, replace
from types import MappingProxyType
from typing import Any, Mapping, NamedTuple, Tuple

class Item(NamedTuple):
    sku: str
    price: float
    qty: int

    @property
    def total(self) -> float:
        return self.price * self.qty

def make_meta(**kwargs: str) -> Mapping[str, str]:
    """Return a read-only mapping for metadata."""
    return MappingProxyType(dict(kwargs))

@dataclass(frozen=True, slots=True)
class Order:
    id: int
    paid: bool
    items: Tuple[Item, ...]
    tags: frozenset[str]
    # exclude meta from hashing/eq; MappingProxyType isn't hashable
    meta: Mapping[str, str] = field(compare=False, hash=False)

    def __deepcopy__(self, memo: dict[int, Any]) -> "Order":
        """Custom deepcopy to support MappingProxyType in meta on Python 3.13."""
        new_meta = MappingProxyType(dict(self.meta))
        new_obj = Order(id=self.id, paid=self.paid, items=self.items, tags=self.tags, meta=new_meta)
        memo[id(self)] = new_obj
        return new_obj

    # ---- pure methods: return NEW Order instances ----
    def subtotal(self) -> float:
        return sum(i.total for i in self.items)

    def add_tag(self, tag: str) -> "Order":
        return replace(self, tags=self.tags | frozenset({tag}))

    def with_item(self, item: Item) -> "Order":
        return replace(self, items=self.items + (item,))

    def pay(self) -> "Order":
        return self if self.paid else replace(self, paid=True)

    def with_meta(self, **patch: str) -> "Order":
        merged = dict(self.meta)
        merged.update(patch)
        return replace(self, meta=MappingProxyType(merged))
```

core.py

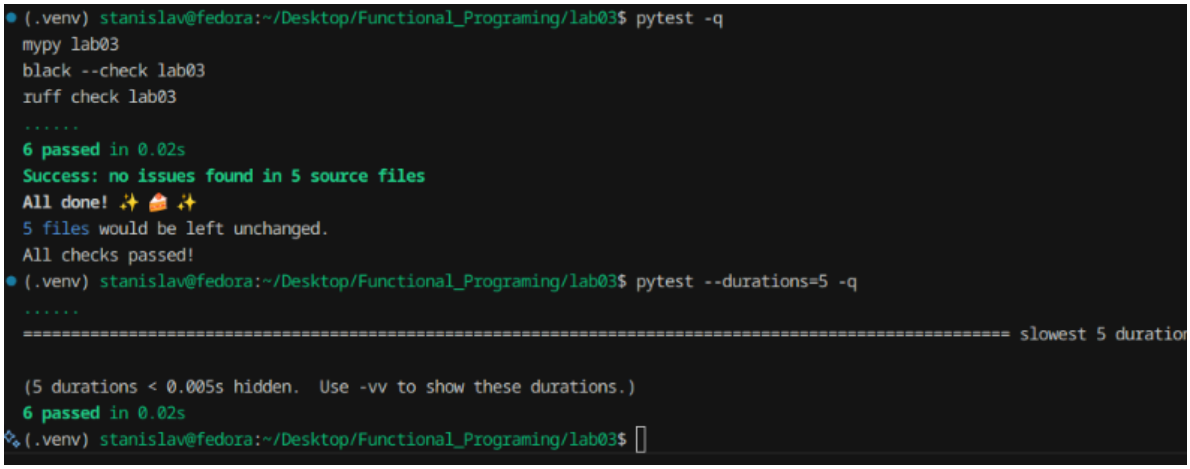


```python
from __future__ import annotations
from collections.abc import Mapping as ABMapping
from types import MappingProxyType
from typing import Any, Sequence, Tuple
from .model import Item, Order
# ---- pure update functions (no in-place mutation) ----
def add_item(order: Order, sku: str, price: float, qty: int) -> Order:
    return order.with_item(Item(sku, price, qty))

def add_tag(order: Order, tag: str) -> Order:
    return order.add_tag(tag)

def pay(order: Order) -> Order:
    return order.pay()

# ---- read helpers ----
def top_expensive_items(items: Sequence[Item], n: int = 3) -> Tuple[Item, ...]:
    return tuple(sorted(items, key=lambda i: i.total, reverse=True)[:n])

# ---- optional: deep freeze utility ----
def freeze(obj: Any) -> Any:
    """Deep-freeze: Mapping→MappingProxyType; set→frozenset; list/tuple→tuple."""
    if isinstance(obj, ABMapping):
        frozen_inner = {k: freeze(v) for k, v in obj.items()}
        return MappingProxyType(frozen_inner)
    if isinstance(obj, set):
        return frozenset(freeze(x) for x in obj)
    if isinstance(obj, (list, tuple)):
        return tuple(freeze(x) for x in obj)
    return obj
```

tests/test_immutability.py

```python
from __future__ import annotations
from copy import deepcopy
from typing import Any, cast
import pytest
from lab03.core import add_item, freeze, top_expensive_items
from lab03.model import Item, Order, make_meta

def make_sample_order() -> Order:
    items = (Item("A1", 50.0, 2), Item("B2", 20.0, 1))
    tags = frozenset({"delhi", "priority"})
    meta = make_meta(source="fb_ads", campaign="summer")
    return Order(id=10, paid=False, items=items, tags=tags, meta=meta)

def test_no_mutation_on_add_item() -> None:
    o1 = make_sample_order()
    o1_copy = deepcopy(o1)
    o2 = add_item(o1, "C3", 15.0, 2)

    # original is unchanged
    assert o1 == o1_copy
    # new object returned
    assert o2 is not o1
    # business effect preserved
    assert pytest.approx(o2.subtotal(), rel=1e-9) == o1.subtotal() + 30.0

def test_mappingproxy_is_read_only() -> None:
    o = make_sample_order()
    with pytest.raises(TypeError):
        cast(Any, o.meta)["source"] = "changed"  # MappingProxyType forbids assignmen

def test_hashability_and_set_usage() -> None:
    o1 = make_sample_order()
    o2 = add_item(o1, "C3", 15.0, 2)
    orders_set = {o1, o2}
    assert len(orders_set) == 2  # frozen dataclass + hashable fields

def test_tuple_and_frozenset_types() -> None:
    o = make_sample_order()
    assert isinstance(o.items, tuple)
    assert isinstance(o.tags, frozenset)

def test_top_expensive_items() -> None:
    o = make_sample_order()
    top = top_expensive_items(o.items, n=1)
    assert len(top) == 1
    assert top[0].sku == "A1"

def test_freeze_deep() -> None:
    deep = {"a": [1, 2, {"k": {1, 2}}], "b": {"x": 1}}
    f = freeze(deep)
    with pytest.raises(TypeError):
        cast(Any, f["b"])["x"] = 2  # inner mapping is read-only
```

Консоль



```
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab03$ pytest -q
mypy lab03
black --check lab03
ruff check lab03
......
6 passed in 0.02s
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/lab03$ pytest --durations=5 -q
......
============================================================================================== slowest 5 duration

(5 durations < 0.005s hidden.  Use -vv to show these durations.)
6 passed in 0.02s
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab03$
```

Висновки

Перехід на незмінні структури унеможливив приховані мутації, спростив тести та дав стабільні результати.