Міністерство освіти і науки України
Львівський національний університет імені Івана Франка Факультет електроніки та комп’ютерних технологій

**Звіт**

про виконання лабораторної роботи №7

з курсу “Функціональне програмування ”

“**functools та operator**”

Виконав:
студент групи ФЕП-23
Чепара Станіслав
Перевірив:
доцент Франів В. А.

Львів 2025

**Мета роботи:** partial, reduce, singledispatch, total_ordering, itemgetter / attrgetter / methodcaller.

## Хід роботи

1. **Конвеєр**
   Створив функцію clean_and_rank(path_or_str, top=3), яка:

- завантажує дані про студентів через load_users (JSON із полями name, age, score),
- нормалізує (обрізання пробілів, title, типи),
- відфільтровує age >= 18,
- сортує за score ↓, name ↑,
- повертає список з top записів.
  Використайте partial + itemgetter/methodcaller + конвеєр.

```python
def pipeline(value: Any, *funcs: Callable[[Any], Any]) -> Any:
    """Проганяє value через послідовність функцій."""
    return reduce(lambda acc, f: f(acc), funcs, value)


def mapf(fn: Callable[[Any], Any]) -> Callable[[Iterable[Any]], Iterable[Any]]:
    """Повертає функцію, що застосує map(fn, iterable)."""
    return partial(map, fn)


def filt(pred: Callable[[Any], bool]) -> Callable[[Iterable[Any]], Iterable[Any]]:
    """Повертає функцію, що застосує filter(pred, iterable)."""
    return partial(filter, pred)


def normalize_row(r: Dict[str, Any]) -> Dict[str, Any]:
    """Повертає нормалізований словник з полями name:str, age:int, score:float."""
    return {
        "name": title(strip(get_name(r))),
        "age": int(get_age(r)),
        "score": float(get_score(r)),
    }


def build_clean_pipeline(rows: Iterable[Dict[str, Any]], top: int = 2) -> List[Dict[str, Any]]:
    """Сирі записи → нормалізація → фільтр повнолітніх → сортування → top-N."""
    adults = lambda r: r["age"] >= 18
    as_list = list
    # Сортування: score (спадання), name (зростання)
    sort_key = itemgetter("score", "name")

    def top_n(n: int) -> Callable[[Iterable[Any]], List[Any]]:
        def _take(it: Iterable[Any]) -> List[Any]:
            return list(it)[:n]
        return _take

    return pipeline(
        rows,
        mapf(normalize_row),
        filt(adults),
        as_list,
        partial(sorted, key=sort_key, reverse=True),
        top_n(top),
    )
```

lab07_functools_operator.py

2. **Поліморфний API**

Розширив load_users підтримкою CSV (рядок/шлях .csv). Переконався, що функція працює однаково для: JSON-рядка, шляху str, Path, відкритого файлу, списку словників, CSV.

```python
# B) Поліморфний API (singledispatch)
# ===========================

@singledispatch
def load_users(src) -> List[Dict[str, Any]]:
    """Завантажити користувачів у форматі list[dict] з різних типів джерела."""
    raise TypeError(f"Unsupported type: {type(src)!r}")


@load_users.register
def _(src: str) -> List[Dict[str, Any]]:
    """Приймає JSON-рядок або шлях (до .json/.csv)."""
    s = src.strip()
    if s.startswith('[') or s.startswith('{'):
        data = json.loads(s)
        if isinstance(data, dict):
            data = [data]
        return data
    # Інакше вважаємо, що це шлях
    return load_users(Path(s))


@load_users.register
def _(src: Path) -> List[Dict[str, Any]]:
    """Читає .json (json.loads) та .csv (csv.DictReader)."""
    suf = src.suffix.lower()
    if suf == ".json":
        txt = src.read_text(encoding="utf-8")
        data = json.loads(txt)
        if isinstance(data, dict):
            data = [data]
        return data
    elif suf == ".csv":
        rows: List[Dict[str, Any]] = []
        with src.open(encoding="utf-8", newline="") as f:
            reader = csv.DictReader(f)
            for row in reader:
                rows.append(dict(row))
        return rows
    else:
        raise ValueError(f"Unsupported file type: {suf}")


@load_users.register
```

```python
@load_users.register
def _(src: io.TextIOBase) -> List[Dict[str, Any]]:
    """Відкритий файл: JSON через json.load, CSV через DictReader (за розширенням name)."""
    name = getattr(src, "name", "") or ""
    if str(name).lower().endswith(".csv"):
        rows: List[Dict[str, Any]] = []
        reader = csv.DictReader(src)
        for row in reader:
            rows.append(dict(row))
        return rows
    else:
        data = json.load(src)
        if isinstance(data, dict):
            data = [data]
        return data


@load_users.register
def _(src: list) -> List[Dict[str, Any]]:
    """Уже-готовий список словників (мінімальна валідація)."""
    if src and not isinstance(src[0], dict):
        raise ValueError("Expected list of dicts with user records")
    return src

```

**lab07_functools_operator.py**

**3. Порівняння**

Зробив клас Box(w, h, d) з порядком за об’ємом, потім за площею фронтальної грані, потім за w. Використав @total_ordering. Перевірив sorted() і порівняння <=, >=.

```python
# C) Порядок порівняння (@total_ordering)
# ==========================

@total_ordering
class Box:
    """Порядок: (volume, front_area, w)."""
    def __init__(self, w: float, h: float, d: float):
        self.w, self.h, self.d = float(w), float(h), float(d)

    def __repr__(self) -> str:
        return f"Box(w={self.w}, h={self.h}, d={self.d})"

    @property
    def volume(self) -> float:
        return self.w * self.h * self.d

    @property
    def front_area(self) -> float:
        return self.w * self.h

    def __eq__(self, other: Any) -> bool:
        if not isinstance(other, Box):
            return NotImplemented
        # Порівнюємо за (об'єм, площа фронтальної грані, ширина)
        return (self.volume, self.front_area, self.w) == (other.volume, other.front_area, other.w)

    def __lt__(self, other: Any) -> bool:
        if not isinstance(other, Box):
            return NotImplemented
        # Порівнюємо за (об'єм, площа фронтальної грані, ширина)
        return (self.volume, self.front_area, self.w) < (other.volume, other.front_area, other.w)
```

**lab07_functools_operator.py**

**Результати:**

```
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab07$ python lab07_funtools_operator.py
Conveyor demonstration (top=2): [{'name': 'Denis', 'age': 20, 'score': 9.3}, {'name': 'Alice', 'age': 19, 'score': 8.5}]
All tests are passed!
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab07$
```

**Висновок:** в ході лабораторної я навчився складати довгі ланцюжки обробки даних (конвеєри) з малих частин (partial, reduce), не пишучи зайвих функцій. Також я тепер вмію створювати одну функцію (singledispatch), яка автоматично знає, як працювати з даними, незалежно від того, чи це текст, файл чи список.