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

Звіт

про виконання лабораторної роботи №10

з курсу “Функціональне програмування ”

“**Функціональні конвеєри для даних (CSV/JSON)**”

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

Львів 2025

Мета роботи: pure-трансформації, point-free стиль, розшарування IO (I/O лише на “краях”).

Хід роботи

ETL-пайплайн: читання → трансформації → валідація → вивід, без мутацій.

1. Розшарування I/O: весь файловий ввід/вивід у lab10_etl_io.py, логіка - у lab10_etl_core.py.

lab10_etl_core.py

```python
def core_pipeline(records: Iterable[Raw]) -> Iterator[Out]:
    for raw in records:
        r_norm = normalize(raw)
        v = validate_all(r_norm)
        if isinstance(v, Ok):
            yield to_out(v.value)
```

lab10_etl_io.py

```python
def read_csv(path: str | Path) -> Iterator[Raw]:
    with open(path, encoding="utf-8", newline="") as f:
        reader = csv.DictReader(f)
        for row in reader:
            yield {
                "name": row.get("name",""),
                "age": row.get("age",""),
                "country": row.get("country",""),
                "email": row.get("email",""),
            }

def read_json(path: str | Path) -> Iterator[Raw]:
    data = json.loads(Path(path).read_text(encoding="utf-8"))
    for row in data:
        yield {
            "name": str(row.get("name","")),
            "age": str(row.get("age","")),
            "country": str(row.get("country","")),
            "email": str(row.get("email","")),
        }
```

```python
def write_csv(path: str | Path, rows: Iterable[Out]) -> None:
    fields = ["name","age","email","segment"]
    with open(path, "w", encoding="utf-8", newline="") as f:
        w = csv.DictWriter(f, fieldnames=fields)
        w.writeheader()
        for r in rows:
            w.writerow(r)

def write_json(path: str | Path, rows: Iterable[Out]) -> None:
    data = list(rows)  # матеріалізуємо тільки на краю
    Path(path).write_text(json.dumps(data, ensure_ascii=False, indent=2), encoding="utf-8")
```

2. Жодної мутації: кожний крок повертає нові dict-и; конвеєр - генератор (пам’ять не роздуває).

lab10_etl_core.py

```python
def normalize(raw: Raw) -> Result:
    age_r = safe_to_int(raw["age"])

    if isinstance(age_r, Err):
        return Err(f"Parsing failed for name={raw['name']!r}: {age_r.error}")

    return Ok({
        "name":    norm_name(raw["name"]),
        "age":     age_r.value,
        "country": to_cc(raw["country"]),
        "email":   norm_email(raw["email"]),
    })
```

```python
def core_pipeline(records: Iterable[Raw]) -> Iterator[Out]:
    for raw in records:
        r_norm = normalize(raw)
        v = validate_all(r_norm)
        if isinstance(v, Ok):
            yield to_out(v.value)
```

3. Валідація з коротким замиканням через Ok/Err і комбінатор and_then.

lab10_etl_core.py

```python
def and_then(r: Result, fn: Callable[[Any], Result]) -> Result:
    return fn(r.value) if isinstance(r, Ok) else r
```

```python
def validate_all(r: Result) -> Result:
    r = and_then(r, validate_age)
    r = and_then(r, validate_email)
    r = and_then(r, validate_cc)
    return r
```

4. Point-free цеглинки (strip/lower/title/only_digits, compose, pipeline) для читабельного складання.

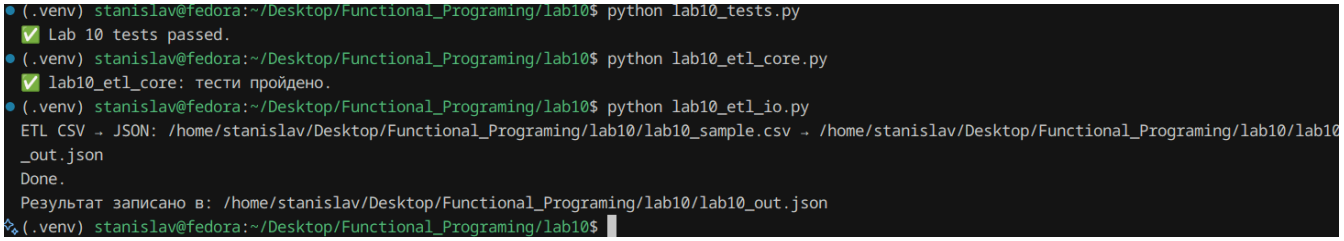
```python
def compose(*funcs: Callable[[Any], Any]) -> Callable[[Any], Any]:
    return lambda x: reduce(lambda acc, f: f(acc), reversed(funcs), x)

def pipeline(value: Any, *funcs: Callable[[Any], Any]) -> Any:
    return reduce(lambda acc, f: f(acc), funcs, value)

strip = methodcaller("strip")
lower = methodcaller("lower")
title = methodcaller("title")
```

```
80    def norm_email(s: str) -> str:
81        return lower(strip(s))
82
83    def norm_name(s: str) -> str:
84        return title(strip(s))
```

Результати:



Висновок: ця лабораторна навчила мене створювати надійні конвеєри для даних, які не виходять з ладу від помилок як винятки, а замість цього акуратно їх обробляють за допомогою спеціальних Result, щоб продовжувати роботу з коректними записами.