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

Звіт

про виконання лабораторної роботи
№5 з курсу “Функціональне програмування”

«Ліниві обчислення: генератори та itertools»

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

Львів-2025

Мета роботи: зрозуміти generator expressions, yield/yield from, ітератори та лінивість; реалізувати: нескінченний стрім Фібоначчі, sliding window, take/drop/accumulate-практики.

Хід роботи:

1. Streams.py

streams.py > ...

```python
from __future__ import annotations
from typing import Iterable, Iterator, TypeVar, Tuple, Callable
from collections import deque
from itertools import islice, count as _count, accumulate as _accumulate, takewhile as _takewhile, dropwhile as _dropwhile
import operator

T = TypeVar("T")

# --------- Нескінченні потоки ---------
def naturals(start: int = 0, step: int = 1) -> Iterator[int]:
    """Нескінченний потік цілих: start, start+step, ..."""
    # делегуємо в itertools.count заради ефективності
    yield from _count(start, step)

def fib_stream() -> Iterator[int]:
    """Нескінченний потік Фібоначчі: 0,1,1,2,3,5,..."""
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

# --------- Взяття / Пропуск ---------
def take(n: int, it: Iterable[T]) -> list[T]:
    """Повернути список із перших n елементів iterable. Якщо n <= 0 -> []."""
    if n <= 0:
        return []
    return list(islice(it, n))

def drop(n: int, it: Iterable[T]) -> Iterator[T]:
    """Повернути ітератор, що пропускає перші n елементів. Якщо n <= 0 -> повертає початковий."""
    if n <= 0:
        # пропускати нічого не треба
        yield from it
        return
    it = iter(it)
    # промотуємо n кроків
    next(islice(it, n, n), None)
    # повертаємо решту
    yield from it

# Умовні аналоги поверх itertools
def takewhile(pred: Callable[[T], bool], it: Iterable[T]) -> Iterator[T]:
    """Повертати елементи, поки предикат істинний."""
    yield from _takewhile(pred, it)

def dropwhile(pred: Callable[[T], bool], it: Iterable[T]) -> Iterator[T]:
    """Пропускати елементи, поки предикат істинний, далі віддавати все."""
    yield from _dropwhile(pred, it)

# --------- Обгортки для accumulate ---------
def accumulate_sum(it: Iterable[int]) -> Iterator[int]:
    """Префіксні суми (1,2,3) -> (1,3,6)."""
    yield from _accumulate(it)

def accumulate_prod(it: Iterable[int]) -> Iterator[int]:
    """Префіксні добутки (1,2,3) -> (1,2,6)."""
    yield from _accumulate(it, operator.mul)

def accumulate_custom(it: Iterable[T], func: Callable[[T, T], T]) -> Iterator[T]:
    """Загальна версія з довільною бінарною функцією."""
    yield from _accumulate(it, func)

# --------- Ковзні вікна ---------
def sliding_window(it: Iterable[T], k: int) -> Iterator[tuple[T, ...]]:
    """Повертати послідовні вікна розміру k (кортежі) поверх iterable.
    Якщо k > len(it) для скінченної послідовності — нічого не віддаємо.
    Для нескінченних потоків вікна будуються ліниво.
```

```
    """
    if k <= 0:
        raise ValueError("k має бути додатним")
    it = iter(it)
    dq: deque[T] = deque(maxlen=k)

    # наповнюємо перше вікно
    for _ in range(k):
        try:
            dq.append(next(it))
        except StopIteration:
            return  # не вистачає елементів навіть на одне вікно

    # перше вікно
    yield tuple(dq)

    # наступні вікна
    for x in it:
        dq.append(x)  # deque автоматично витісняє найстаріше
        yield tuple(dq)

def moving_average(it: Iterable[float], k: int) -> Iterator[float]:
    """Лінива рухома середня з вікном k."""
    if k <= 0:
        raise ValueError("k має бути додатним")
    for win in sliding_window(it, k):
        yield sum(win) / k

```

2. Main.py

main.py > ...

```
from itertools import islice
from streams import naturals, fib_stream, take, drop, sliding_window, moving_average, accumulate_sum

def demo():
    print("Перші 10 натуральних з 1:", list(islice(naturals(1), 10)))
    print("Перші 10 чисел Фібоначчі:", list(islice(fib_stream(), 10)))

    # Приклад take/drop над одним потоком
    xs = naturals(1)
    print("Перші 5 елементів:", take(5, xs))         # зсуває ітератор до 6
    print("Наступні 3 елементи:", take(3, xs))       # продовжуємо від 6

    # Пропустити перші 10 і взяти наступні 5
    ys = take(5, drop(10, naturals(0)))
    print("Пропустили 10, взяли 5:", ys)

    # Суми за ковзним вікном на натуральних
    win2 = sliding_window(naturals(1), 2)
    first5_window_sums = list(islice((sum(w) for w in win2), 5))
    print("Перші 5 сум вікна (k=2) над натуральними:", first5_window_sums)

    # Рухоме середнє на скінченному списку
    print("Рухоме середнє k=3 для [1,2,3,4,5]:", list(moving_average([1,2,3,4,5], 3)))

    # Префіксні суми через accumulate
    print("Префіксні суми для [1,2,3,4]:", list(accumulate_sum([1,2,3,4])))

if __name__ == "__main__":
    demo()

```

4. Test_lazy_behavoir.py

tests > test_lazy_behavior.py > ...

```python
from itertools import islice
from streams import take

def noisy_iter(n):
    # yields 0..n-1 and counts how many times we pulled
    for i in range(n):
        yield i

def test_take_is_lazy():
    it = noisy_iter(1000)
    out = take(5, it)
    assert out == [0,1,2,3,4]
    # Iterator should now be at position 5; taking next 3 yields 5,6,7
    assert take(3, it) == [5,6,7]
```

5. Test_fib_stream.py

tests > test_fib_stream.py > ...

```python
from itertools import islice
from streams import fib_stream

def test_first_10_fib():
    assert list(islice(fib_stream(), 10)) == [0,1,1,2,3,5,8,13,21,34]
```

6. Test_take_drop.py

tests > test_take_drop.py > ...

```python
from itertools import count, islice
from streams import take, drop, naturals

def test_take_and_stream_progress():
    xs = naturals(1)
    assert take(5, xs) == [1,2,3,4,5]
    # xs has advanced
    assert take(3, xs) == [6,7,8]

def test_drop_then_take():
    xs = drop(10, count(0))
    assert list(islice(xs, 3)) == [10,11,12]

def test_negative_take_drop():
    xs = naturals(0)
    assert take(-5, xs) == []
    ys = drop(-3, [1,2,3])
    assert list(ys) == [1,2,3]
```

7. Test_sliding_window.py

tests > test_sliding_window.py > ...

```python
from itertools import islice, starmap
from streams import sliding_window, naturals, moving_average

def test_sliding_window_basic():
    assert list(sliding_window([1,2,3,4], 3)) == [(1,2,3),(2,3,4)]

def test_sliding_window_edge():
    assert list(sliding_window([1,2], 3)) == []
    assert list(sliding_window([7], 1)) == [(7,)]

def test_sliding_window_infinite_is_bounded():
    sw = sliding_window(naturals(1), 2)
    first5_sums = list(islice((sum(w) for w in sw), 5))
    assert first5_sums == [3,5,7,9,11]

def test_moving_average():
    assert list(moving_average([1,2,3,4,5], 3)) == [2.0,3.0,4.0]

```

11. Пройшли тести, запустив main.py

```
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab05/lab05$ pytest -q
...........
11 passed in 0.03s
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab05/lab05$ python main.py
Перші 10 натуральних з 1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Перші 10 чисел Фібоначчі: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Перші 5 елементів: [1, 2, 3, 4, 5]
Наступні 3 елементи: [6, 7, 8]
Пропустили 10, взяли 5: [10, 11, 12, 13, 14]
Перші 5 сум вікна (k=2) над натуральними: [3, 5, 7, 9, 11]
Рухоме середнє k=3 для [1,2,3,4,5]: [2.0, 3.0, 4.0]
Префіксні суми для [1,2,3,4]: [1, 3, 6, 10]
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab05/lab05$
```

Висновок: виконуючи дану лабораторну роботу я зрозумів generator expressions, yield/yield from, ітератори та лінивість та реалізував нескінченний стрім Фібоначчі, sliding window, take/drop/accumulate-практики.