Список лабараторних робiт

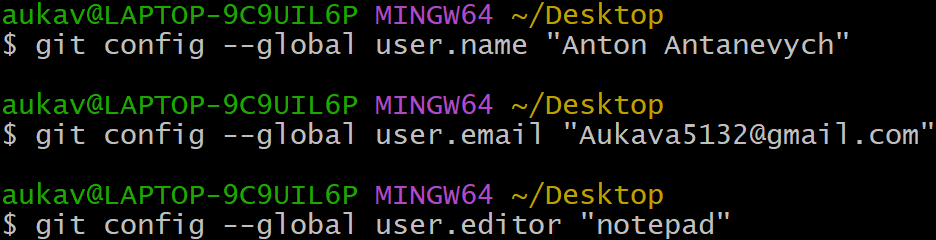
з предмету АСД

студента групи КН-24-1

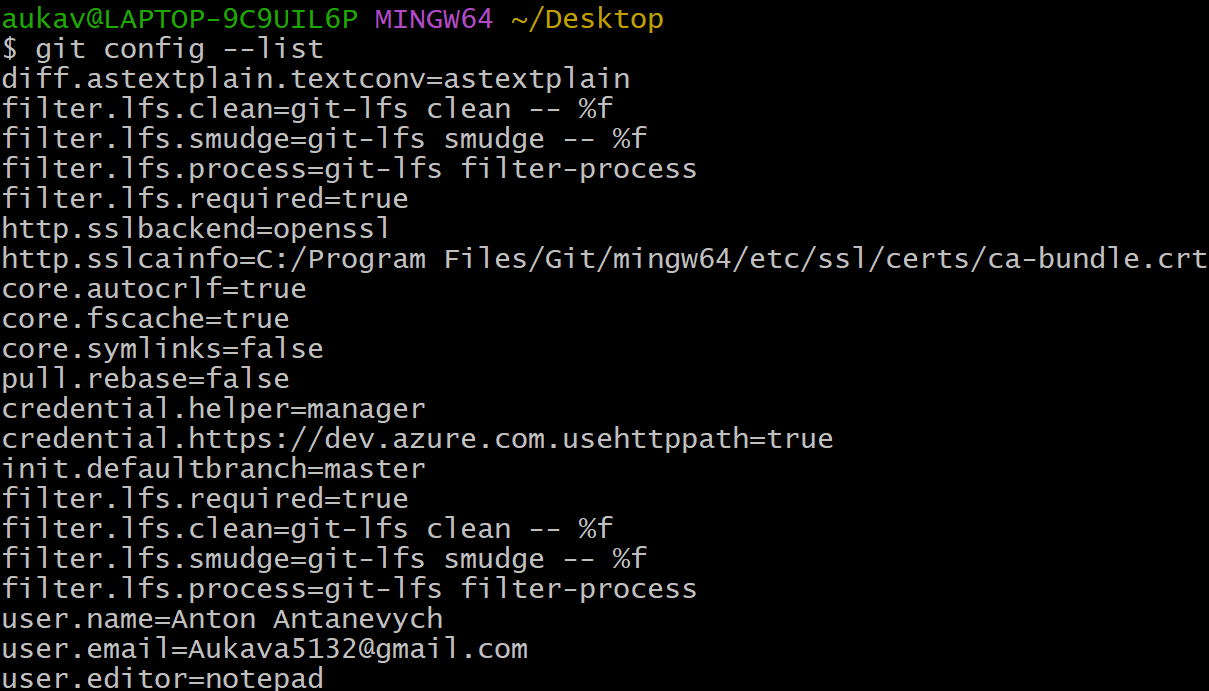
Антаневича Антона

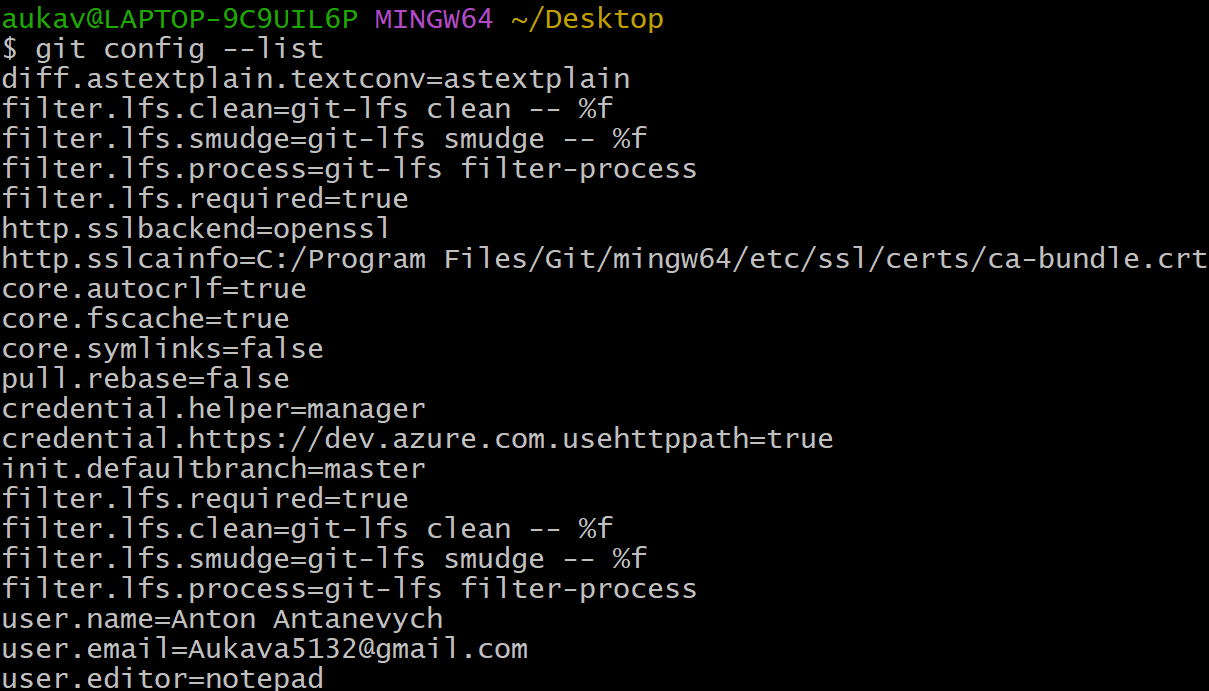
**Лабараторна робота №1**

Налагоджую профiль через Git Bash

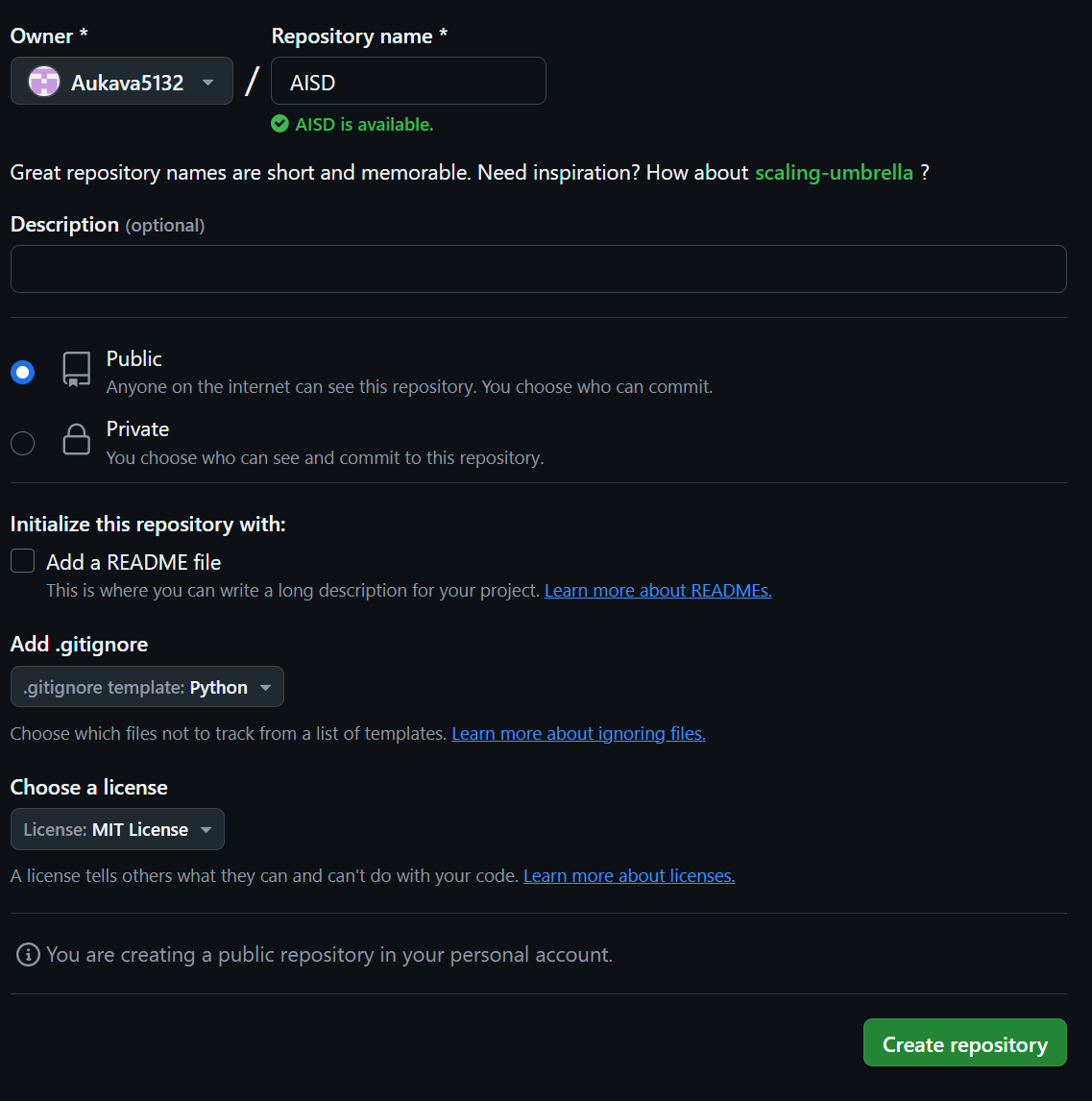
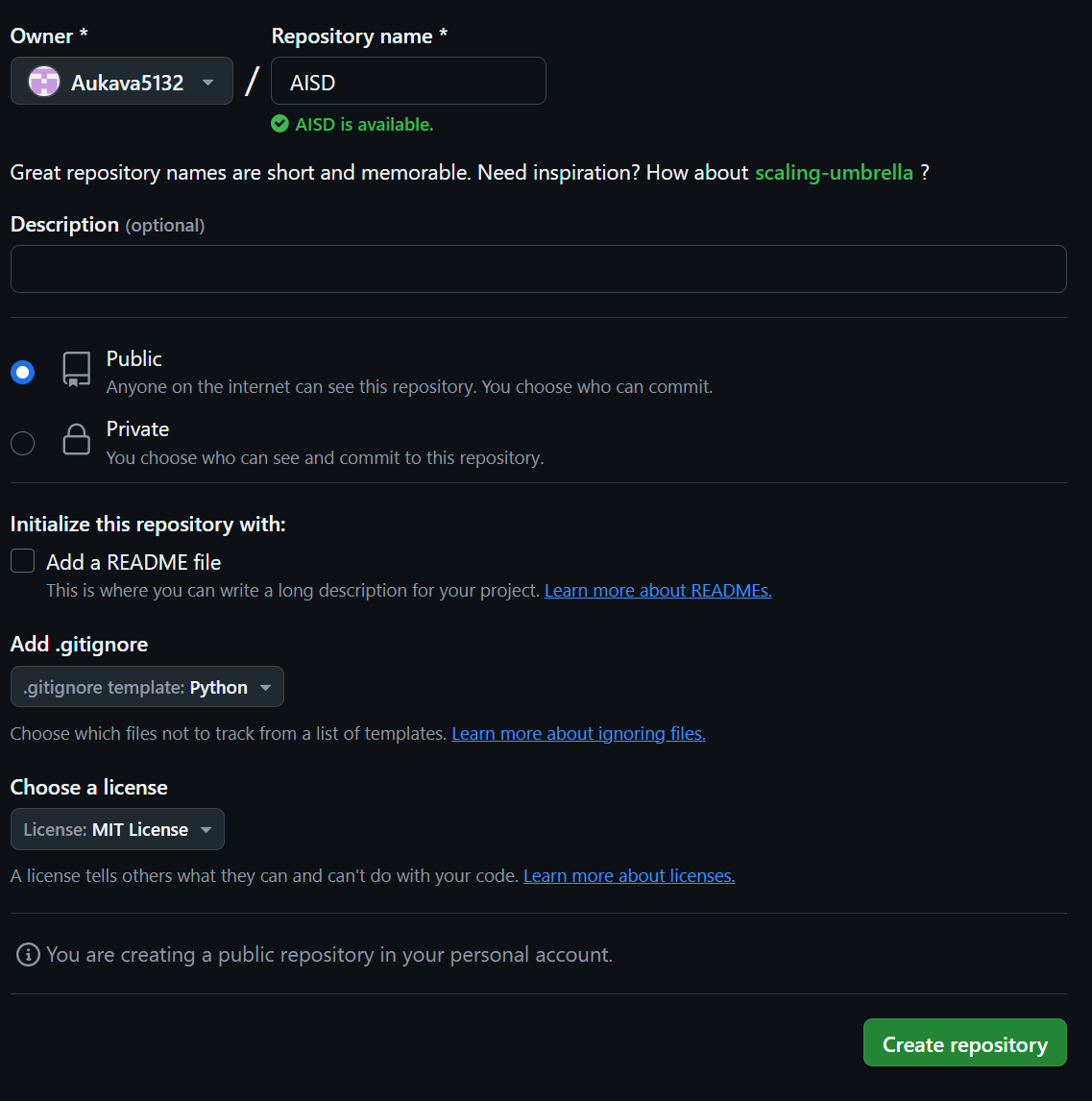
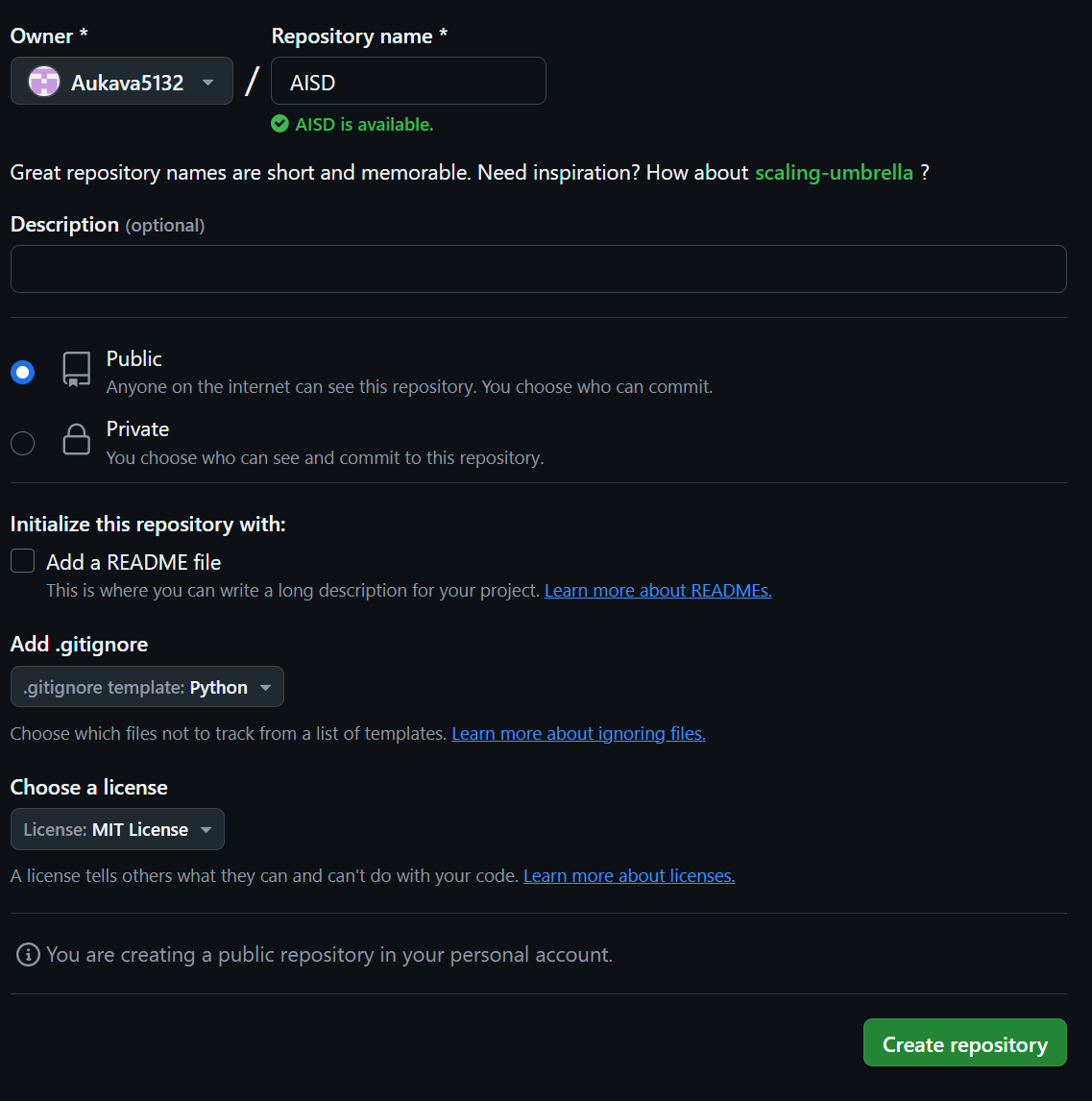


Перевiряю збереження iнформацiї

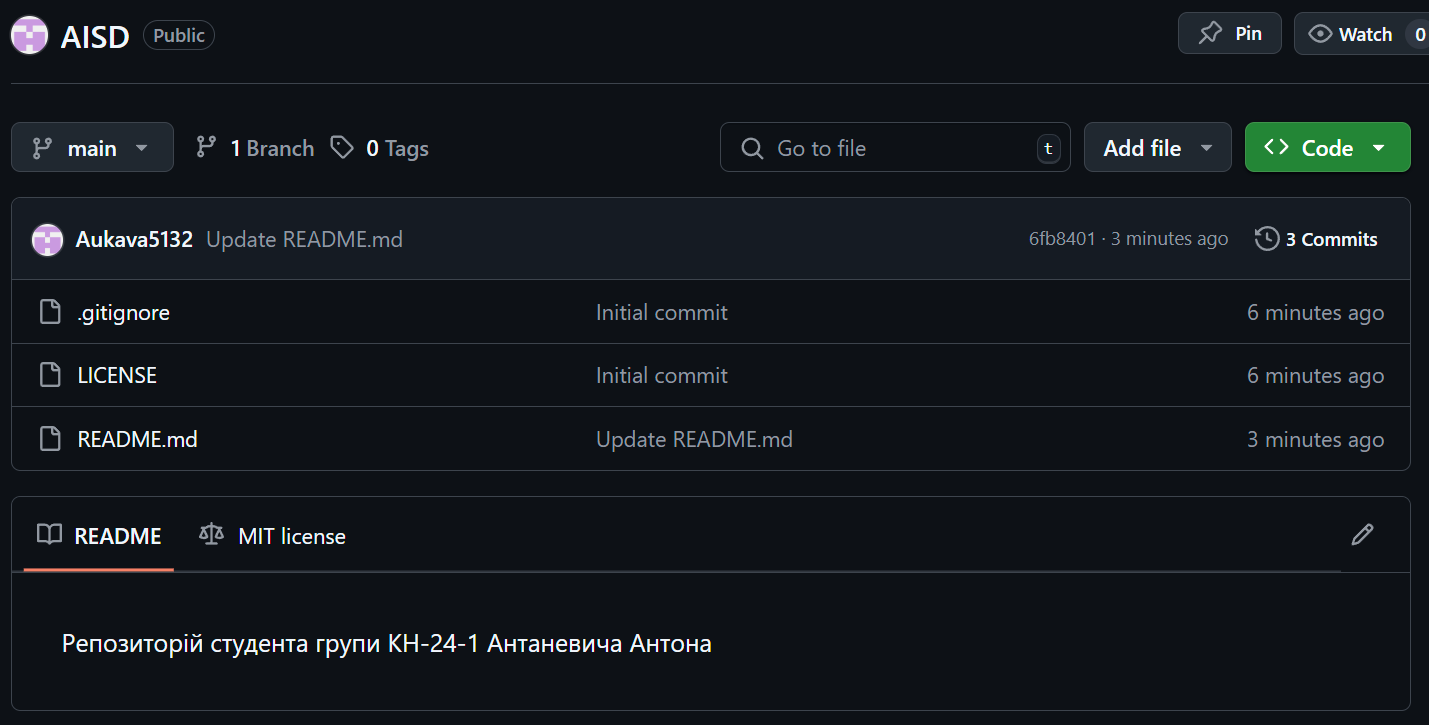




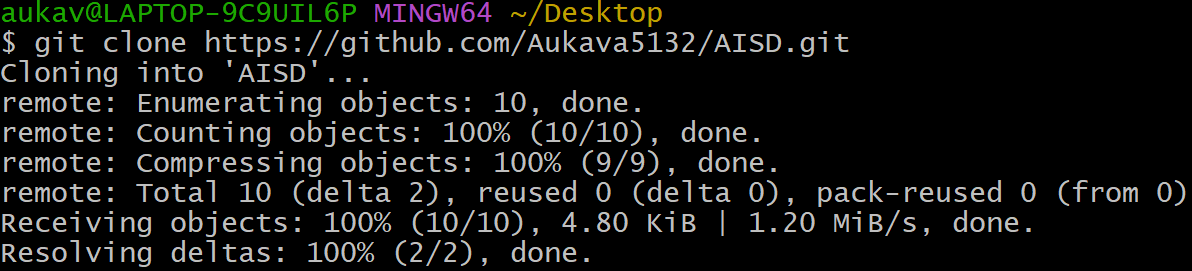
Створюю репозиторiй на GitHub



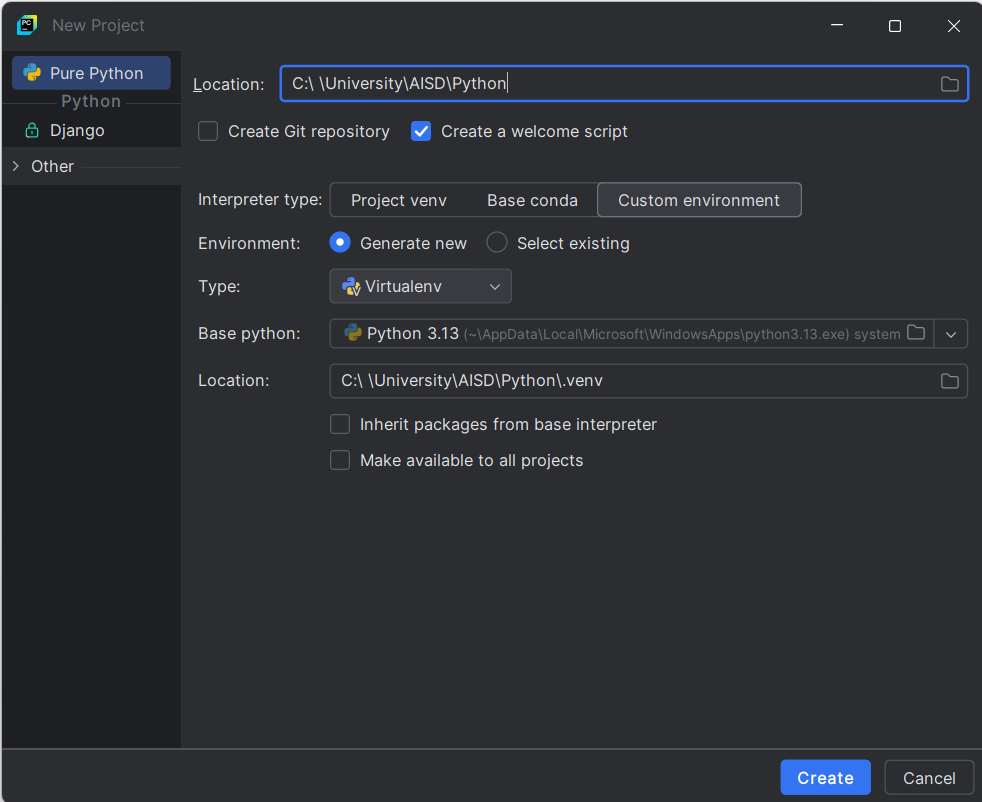
Створений репозиторiю

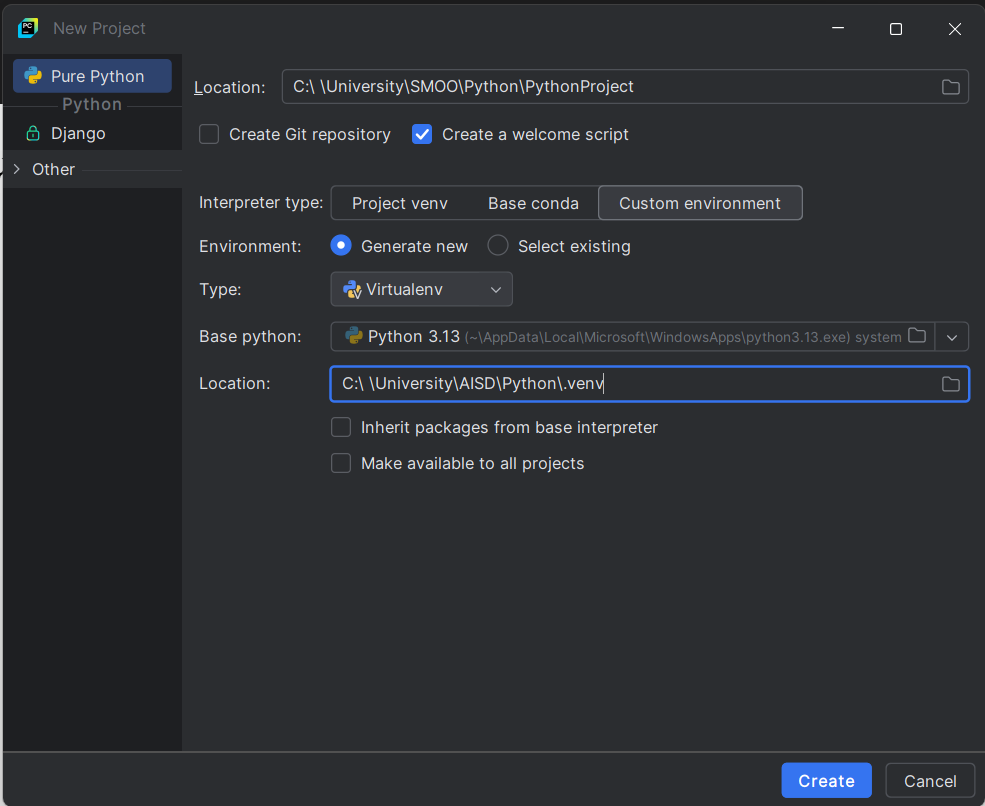


Клоную

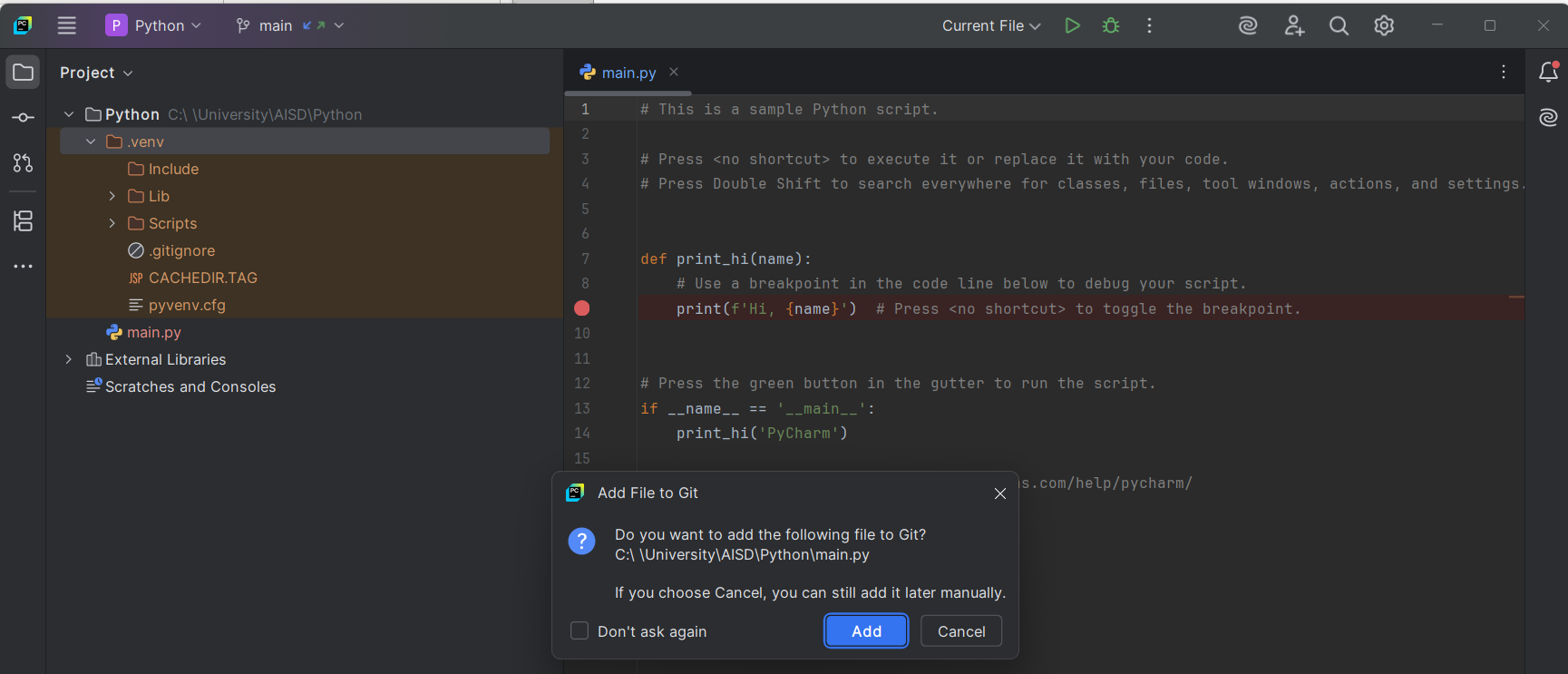


Створюю проект





Створений проект

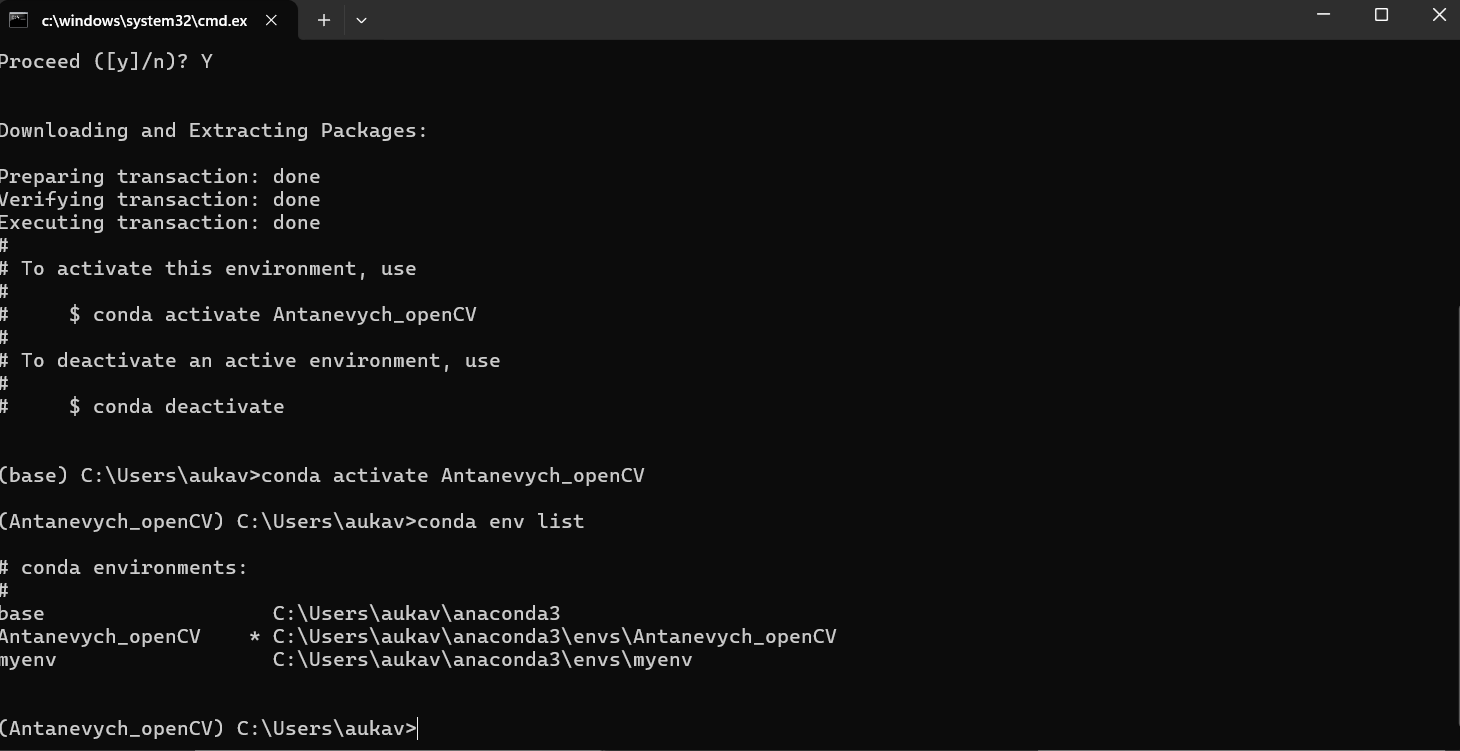


Оновлений репозиторiй

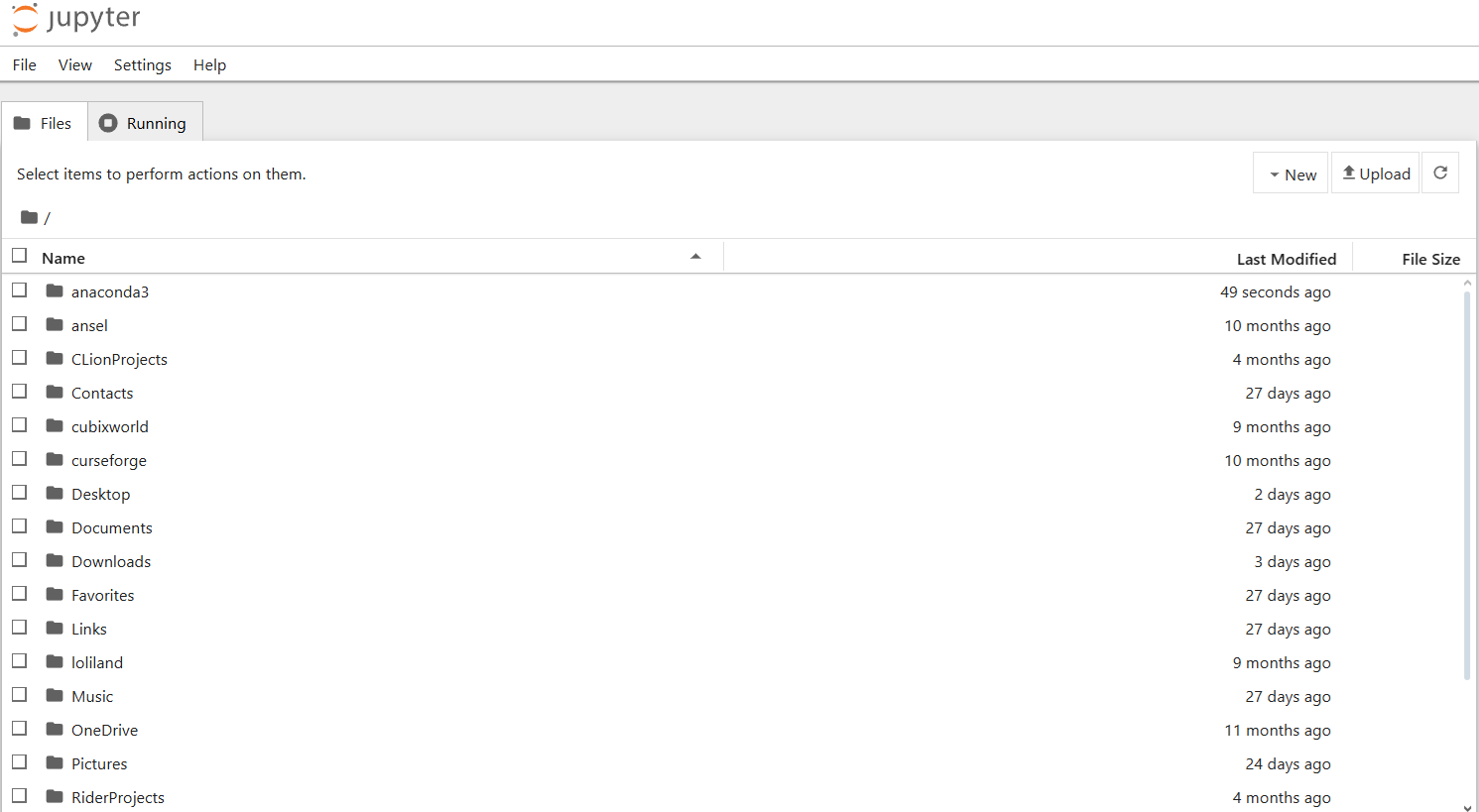


**Лабараторна робота №2**

Налаштування Conda



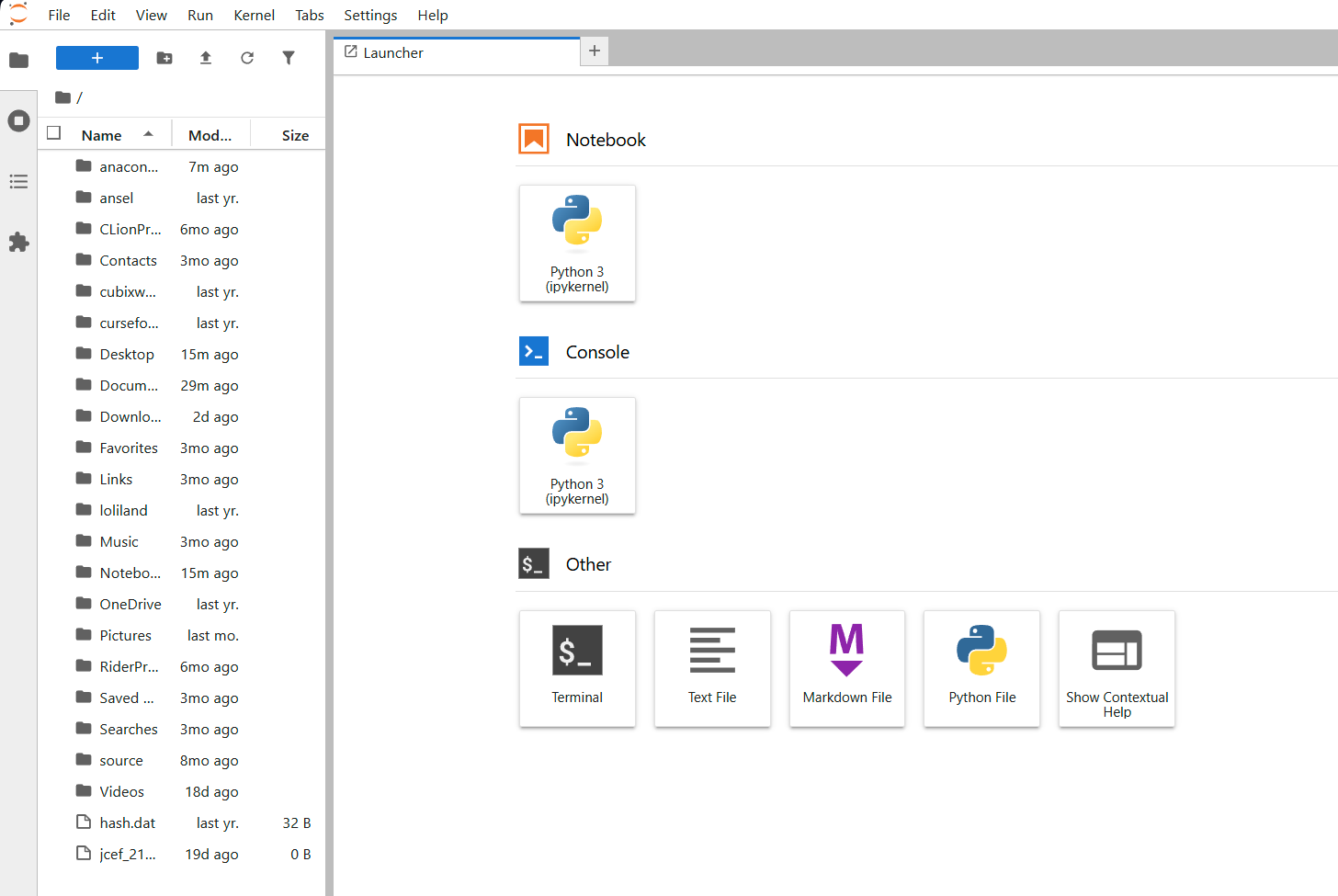
Jupiter Notebook



Запуск JupiterLab

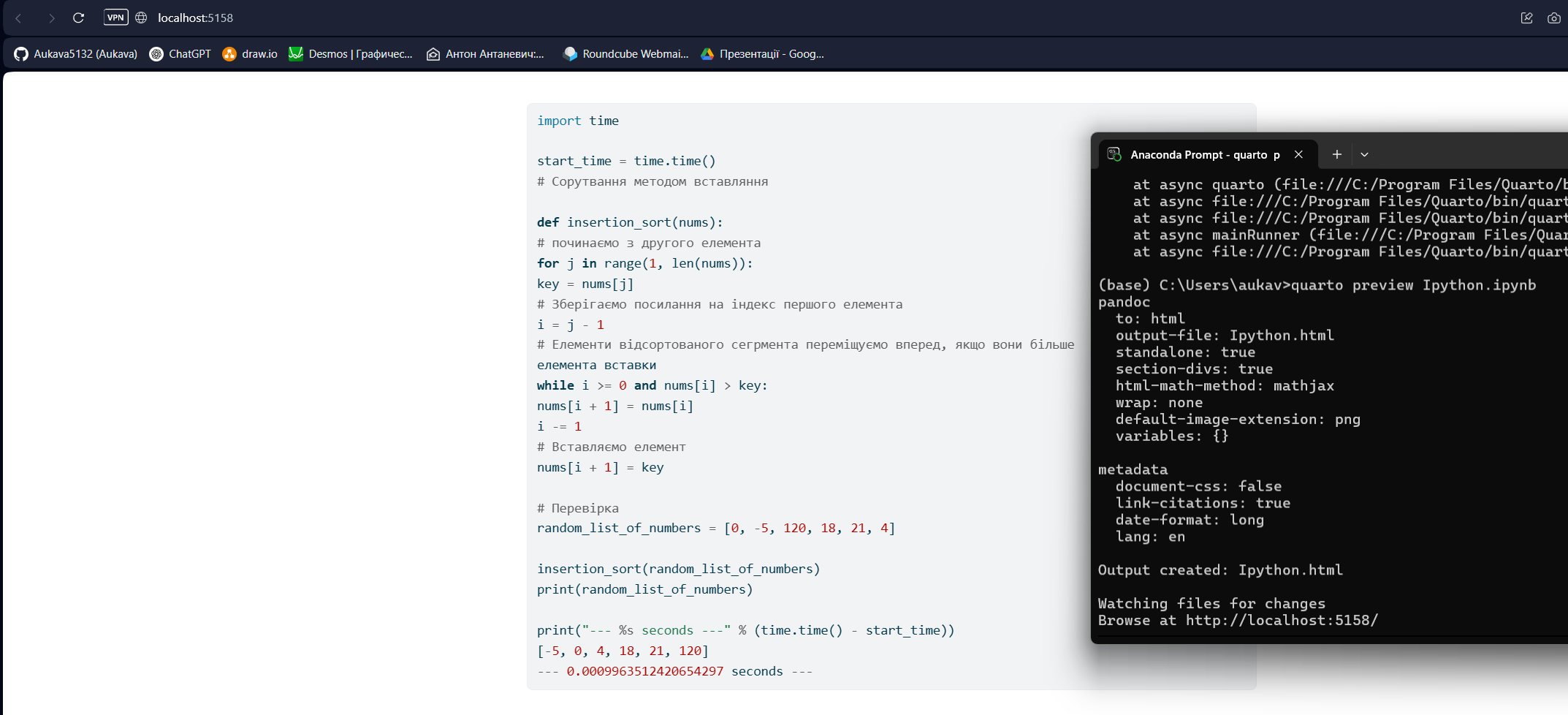
conda activate myenv

jupyter lab



Робота з Quardo

quarto preview ipython.ipynb



**Лабараторна робота №3**

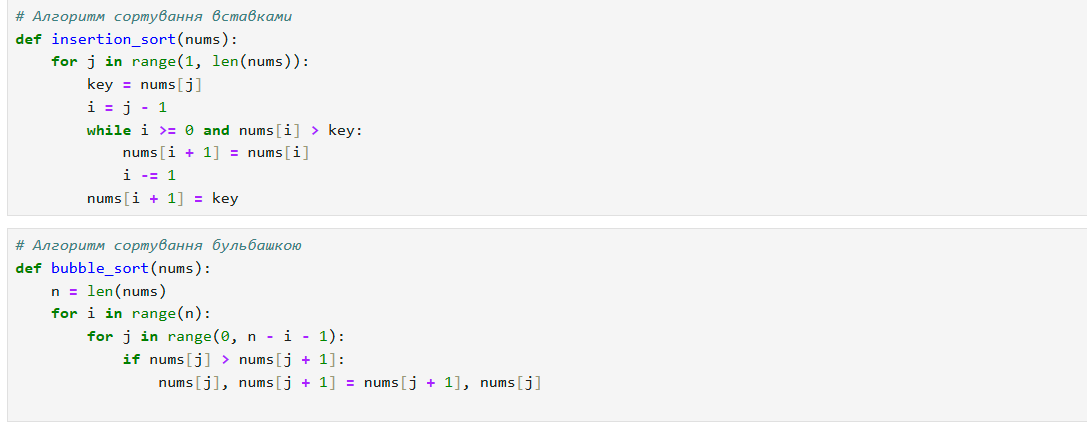
Визначення асимптотичної складності алгоритмів сортування:

* Сортування вставками (Insertion Sort): O(n2)
* Сортування бульбашкою (Bubble Sort): O(n2)

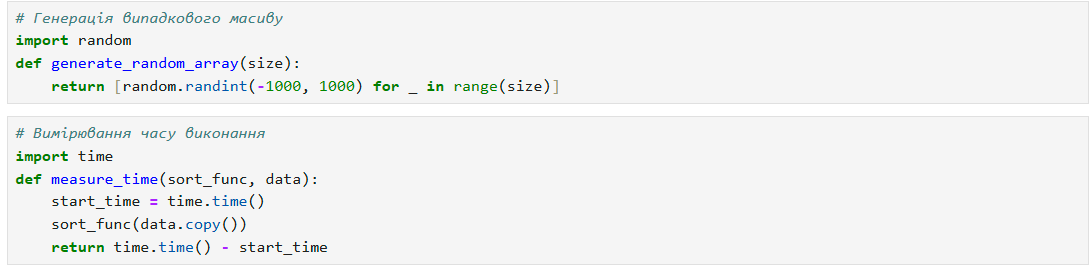
Це означає, що зі збільшенням розміру вхідних даних (n*2*) у 2 рази, час виконання зростає приблизно в 4 рази.

Реалізація алгоритмів:

Реалiзацiя на Python:



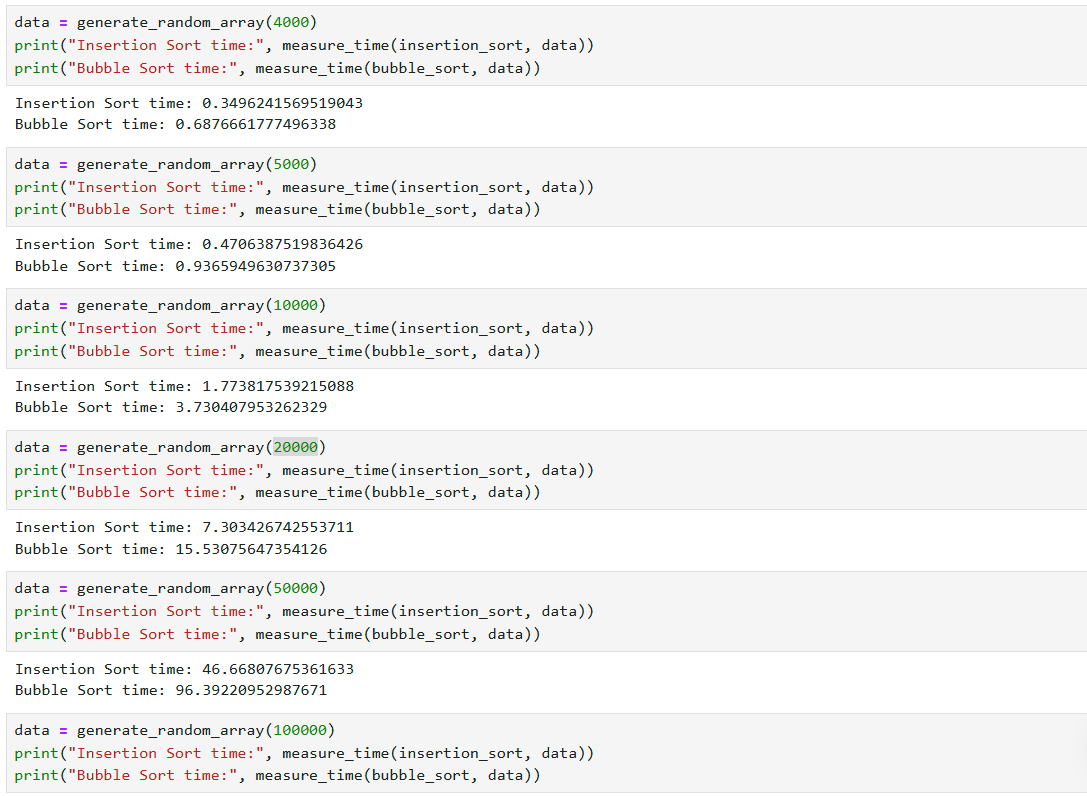
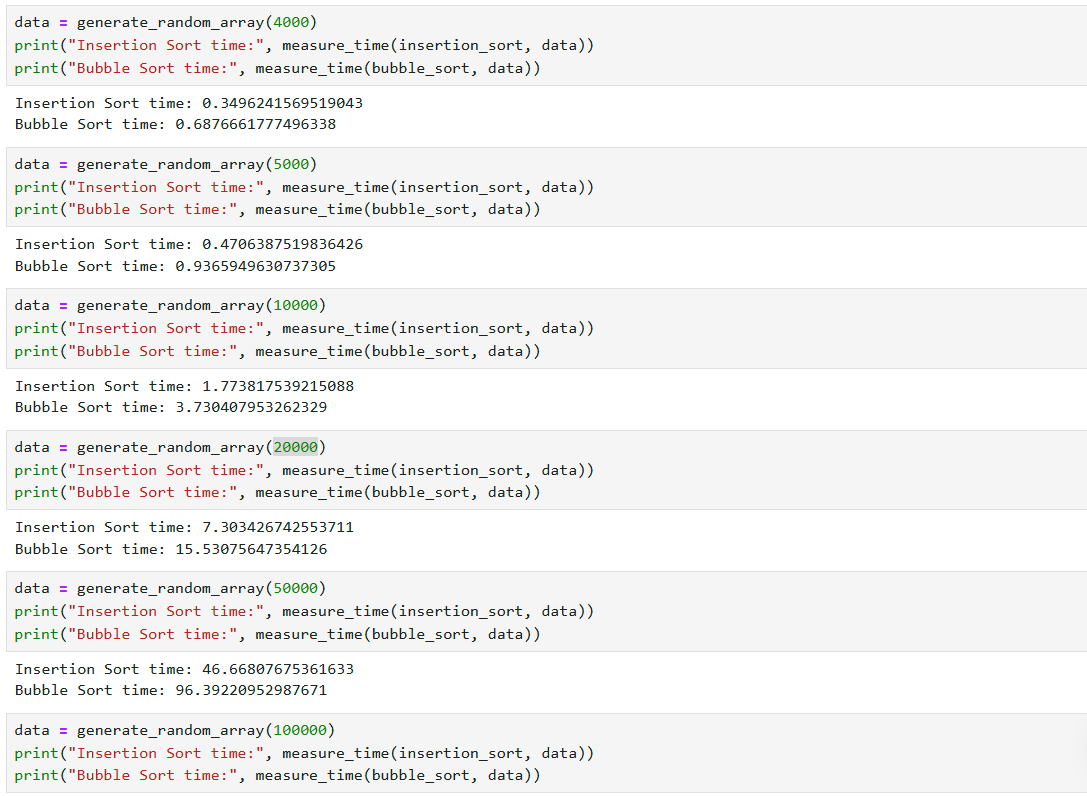
Вимірювання часу виконання:

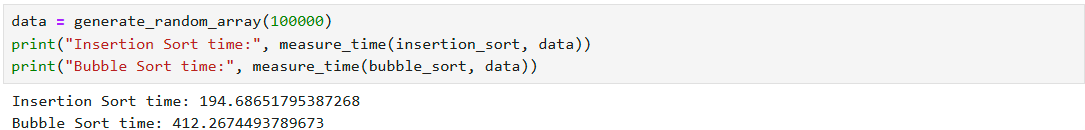


Виміряв час виконання кожного алгоритму за допомогою модуля time:





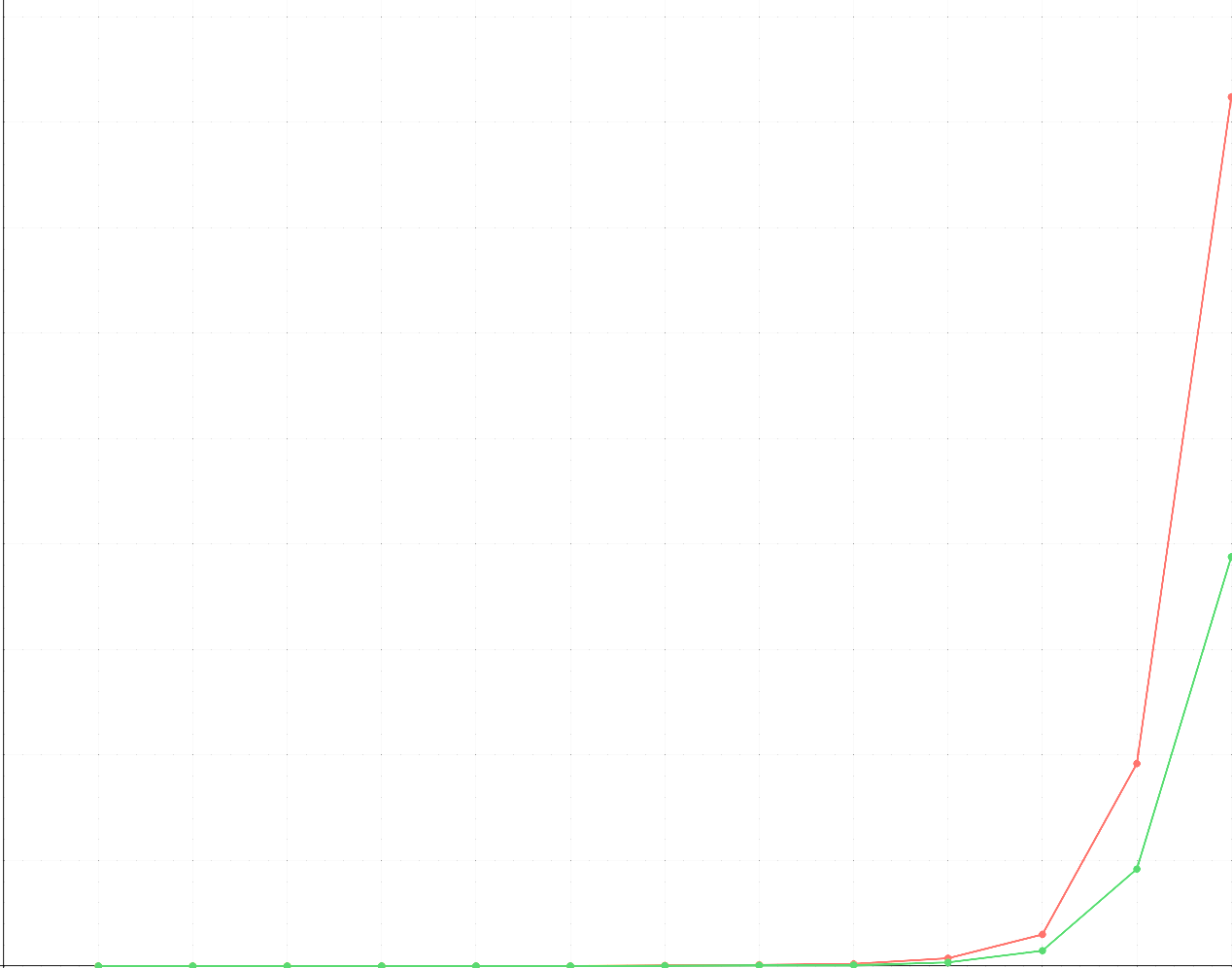




Результати та графіки

Після проведення експериментів я отримав наступні дані:

* Для малих масивів (n<1000*n*<1000) обидва алгоритми працюють швидко, але сортування вставками виявилось ефективнішим.
* Для n=5000*n*=5000 час виконання зріс до 0.5 секунд для insertion\_sort і 1 секунди для bubble\_sort.

Графік залежності часу від розміру масиву:  


46

412

194

92

15

Секунди

Кiлькiсть

5 10 50 100 1000 2000 4000 5000 10000 20000 50000 100000

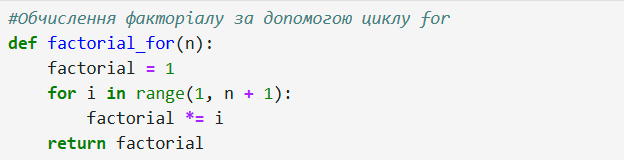
Висновки:

1. Обидва алгоритми мають квадратичну складність, що підтверджується експериментально.
2. Сортування вставками працює швидше за бульбашкове через меншу кількість операцій обміну

quarto preview Lab3\_Antanevych.ipynb

**Лабараторна робота №4**

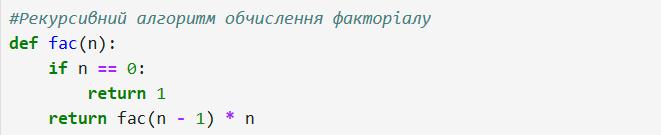
Обчислення факторіалу за допомогою циклу for



Асимптотична складність:

* Алгоритм виконує n ітерацій, кожна з яких має константну складність O(1).
* Отже, загальна складність: O(n).

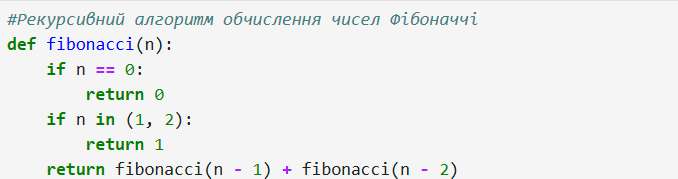
Рекурсивний алгоритм обчислення факторіалу



Асимптотична складність:

* Кожен виклик має складність O(1) (якщо не враховувати рекурсію).
* Алгоритм викликає себе n разів. Отже, загальна складність: O(n)*.*

Рекурсивний алгоритм обчислення чисел Фібоначчі



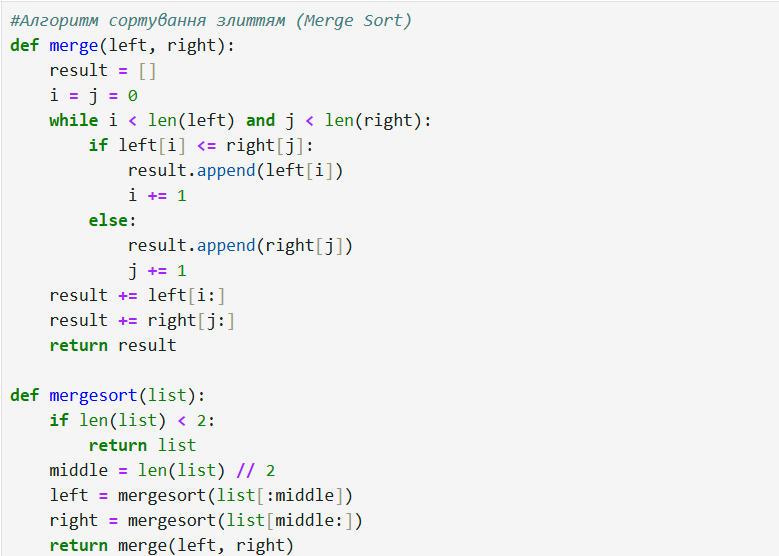
Асимптотична складність:

* Алгоритм має експоненційну складність O(2n), оскільки кожен виклик породжує два нових виклики (за винятком базових випадків).
* Це неефективний алгоритм, оскільки він багаторазово обчислює одні й ті ж значення.

Покращення:

Збереження проміжних результатів.

Алгоритм сортування злиттям (Merge Sort)



Асимптотична складність:

* a = 2 (дві підзадачі),
* b = 2 (розмір кожної підзадачі зменшується вдвічі),
* d = 1 (складність об'єднання merge ­– O(n)).

Оскільки logиa = log22 = 1 = d, то:

T(n) = O(ndlogn) = O(nlogn)

Висновок:  
Сортування злиттям має складність O(nlogn) у всіх випадках (найгіршому, середньому, найкращому).

quarto preview Lab4\_Antanevych.ipynb