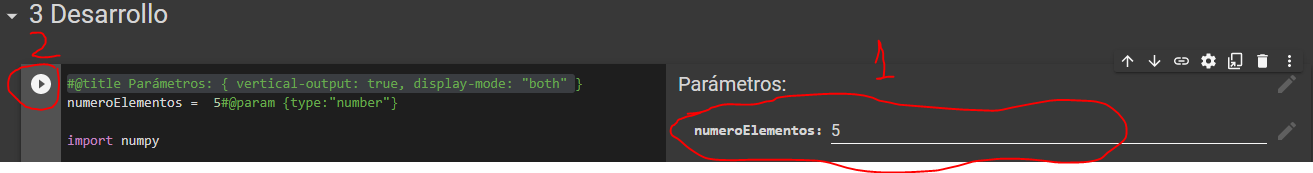
**Explicación general ejercicio 1:**

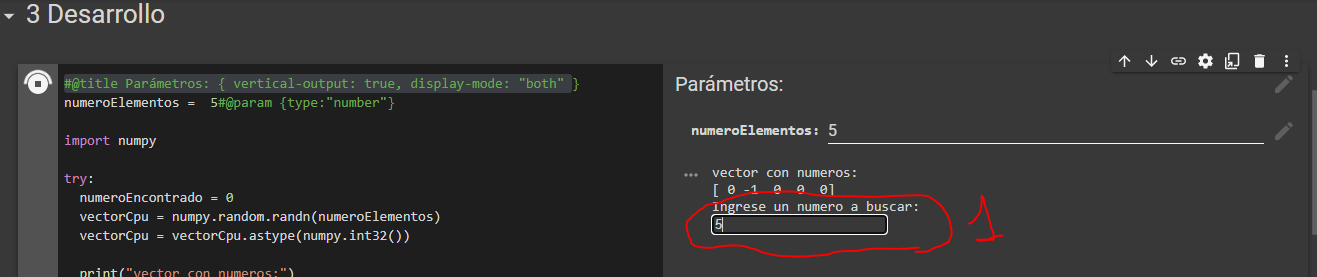
En este ejercicio se desarrolló el algoritmo de búsqueda secuencial tanto usando solamente la CPU de manera secuencial como usando la GPU aplicando paralelismo. En este ejercicio lo que se hace es ingresar la cantidad de elementos que deseamos que tenga un vector y luego ingresamos un numero que deseamos que se busque dentro del vector y se nos informe si se encontró o no. El objetivo buscado con este ejercicio es comparar los tiempos de ejecución de la manera secuencial contra la manera que utiliza paralelismo para ver cual resulta ser más eficiente.

**Manual de uso para la versión CPU ( secuencial):**

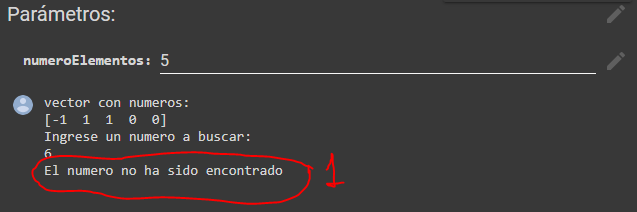
Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el numero de elementos que deseamos que tenga el vector donde se va a buscar un número, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” de la imagen.



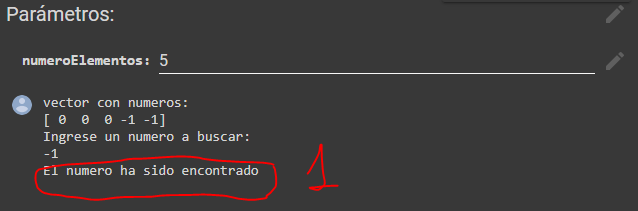
Paso2: Luego se nos solicitara por pantalla que ingresemos un numero a buscar en el vector creado previamente con números enteros al azar, ingresamos un numero como se indica donde dice “1 (en rojo)” y después presionamos enter



Paso3a: Si el número ingresado por teclado no fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)



Paso3b: Si el número ingresado por teclado fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)

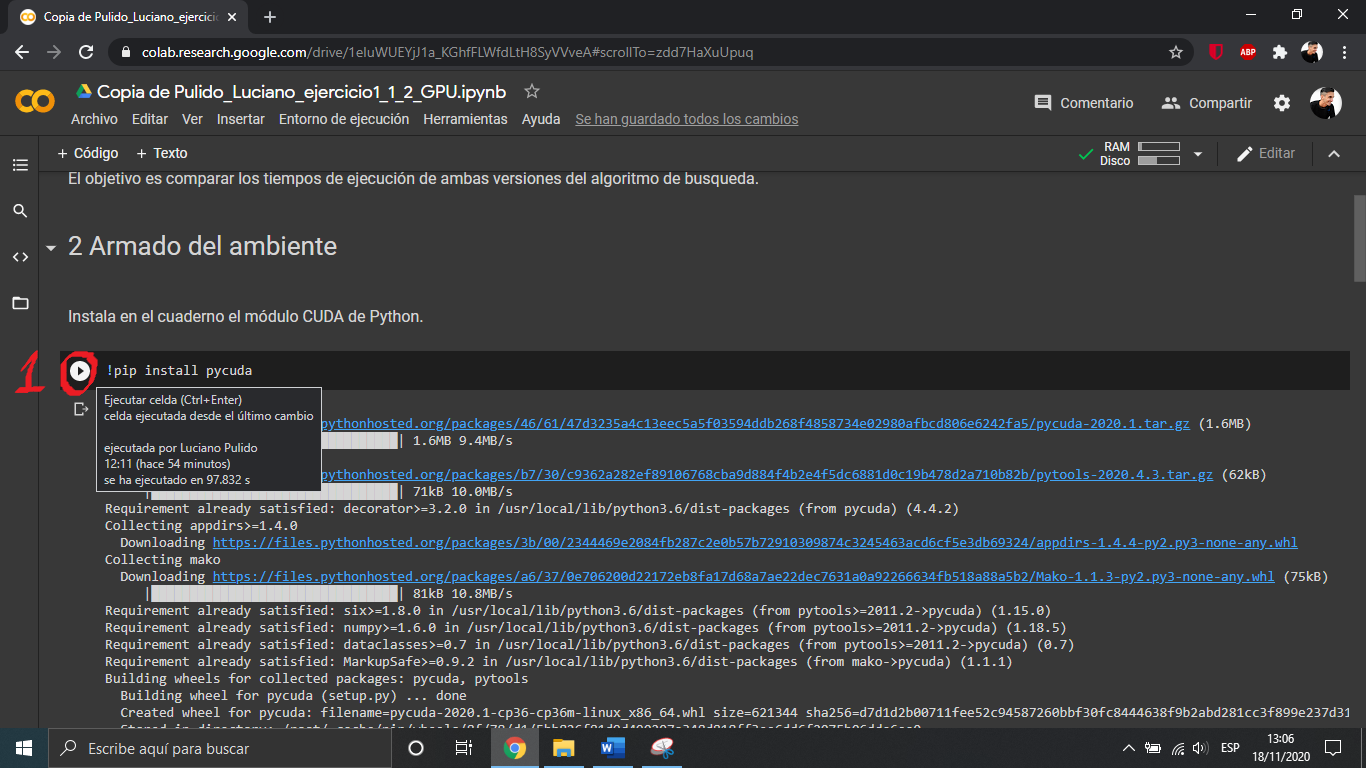


**link versión secuencial:** <https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido_Luciano_ejercicio1_1_2_CPU.ipynb>

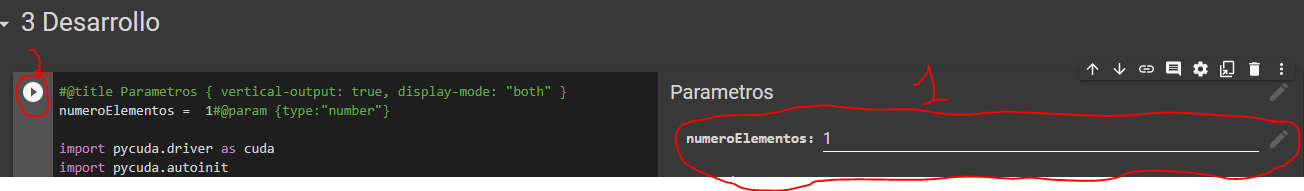
**Manual de uso para la versión GPU ( paralelismo):**

**Aclaración: esta versión debe ser ejecutada en el entorno de GPU.**

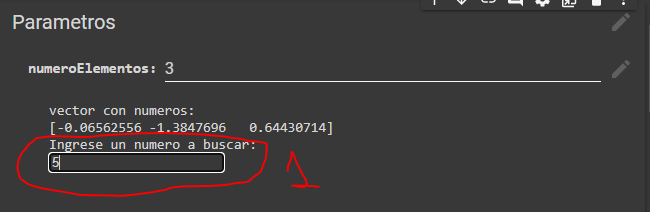
Paso1: : Dentro del cuaderno, debemos dirigirnos a la sección “Armado del ambiente” y luego debemos hacer click en el botón de “play” donde dice “1 (en rojo)” de la imagen.



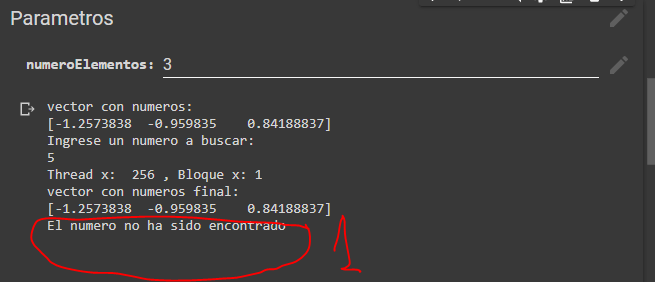
Paso2: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el número de elementos que deseamos que tenga el vector donde se va a buscar un número, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen



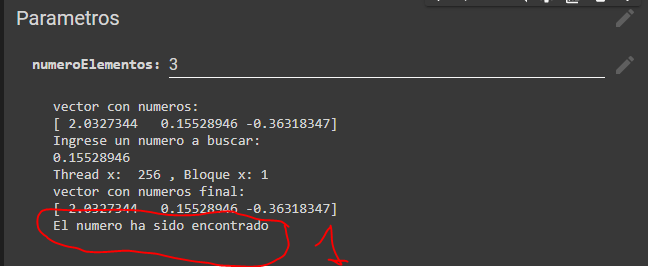
Paso3: Luego se nos solicitara por pantalla que ingresemos un numero a buscar en el vector creado previamente con números flotantes al azar, ingresamos un numero como se indica donde dice “1 (en rojo)” y después presionamos enter



Paso4a: Si el número ingresado por teclado no fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)



Paso 4b: Si el número ingresado por teclado fue encontrado en el vector se muestra un mensaje como se indica donde dice “1 (en rojo)



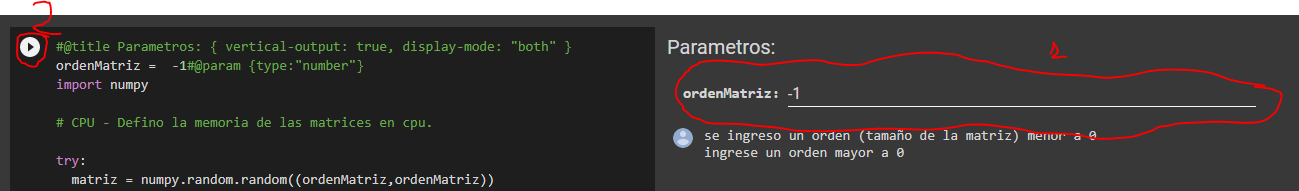
**link versión con paralelismo:** https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido\_Luciano\_ejercicio1\_1\_2\_GPU.ipynb

**Explicación general ejercicio 2:**

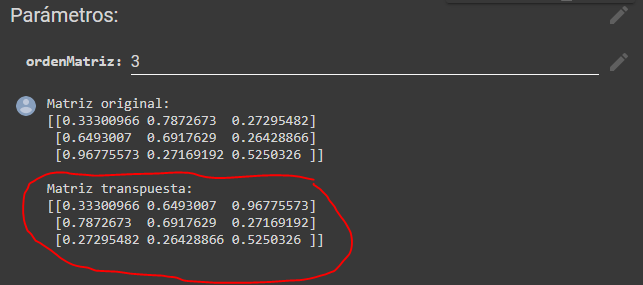
En este ejercicio se desarrolló el algoritmo para transponer una matriz tanto de manera secuencial usando solamente la CPU como usando la GPU aplicando paralelismo. En este ejercicio lo que se hace es ingresar el orden de la matriz (cantidad de filas y columnas iguales, es decir, solo admite matrices cuadradas) que deseamos que tenga una matriz y luego una vez ejecutado el algoritmo, se encarga de transponerla y nos informa la matriz transpuesta. El objetivo buscado con este ejercicio es comparar los tiempos de ejecución de la manera secuencial contra la manera que utiliza paralelismo para ver cual resulta ser más eficiente.

**Manual de uso para la versión CPU ( secuencial):**

Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el orden de la matriz que deseamos que tenga la matriz que deseamos transponer, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen.



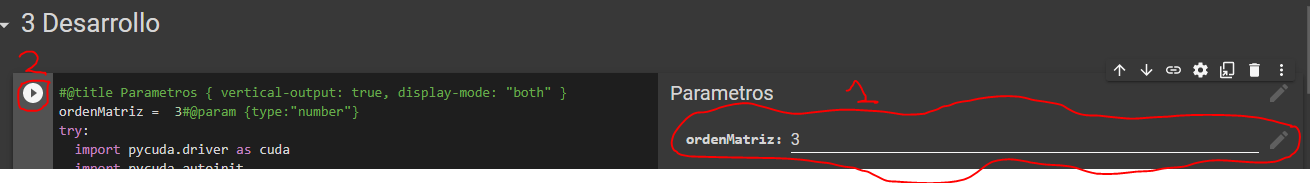
Paso2 : se informa la matriz transpuesta como en la siguiente imagen



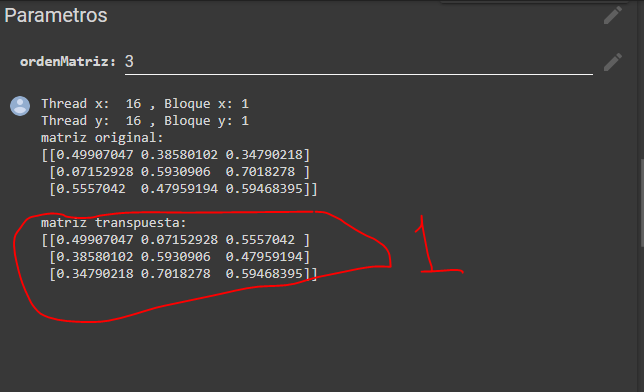
**link versión secuencial**: https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido\_Luciano\_Ejercicio1\_1\_3\_CPU.ipynb

**Manual de uso para la versión GPU ( paralelismo):**

Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el orden de la matriz que deseamos que tenga la matriz que deseamos transponer, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen.



Paso2: se informa la matriz transpuesta como en la siguiente imagen



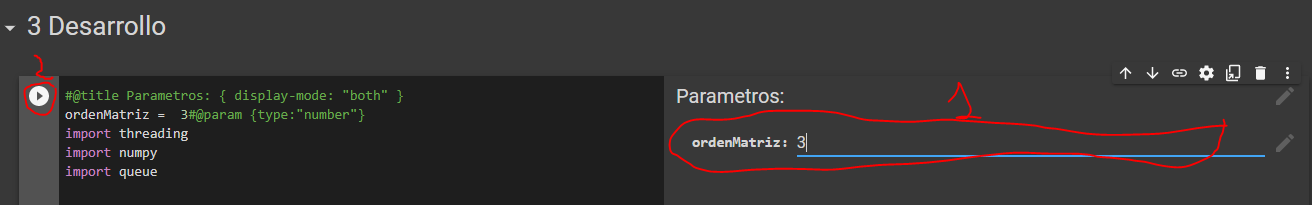
**Aclaración: esta versión debe ser ejecutada en el entorno de GPU.**

**link versión con paralelismo**: https://github.com/lucianopulido/EA2-Luciano-Pulido/blob/master/HPC/Pulido\_Luciano\_Ejercicio1\_1\_3\_GPU.ipynb

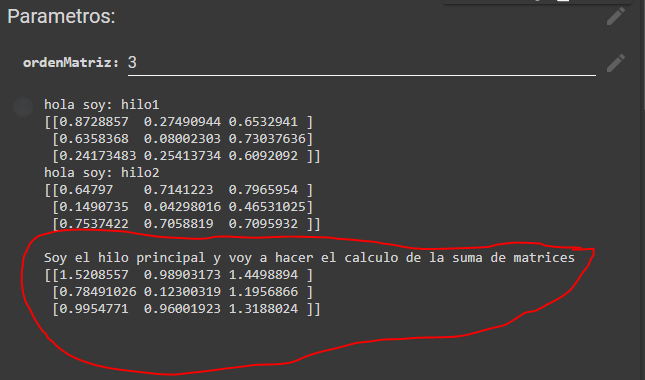
**Explicación general ejercicio 3:**

En este ejercicio se desarrolló el algoritmo de una suma de matrices usando solamente la CPU y aplicando paralelismo mediante Threads. En este ejercicio lo que se hace es ingresar el orden de la matriz (cantidad de filas y columnas iguales, es decir, solo admite matrices cuadradas) que deseamos que tenga 2 matrices que se van a sumar y luego una vez ejecutado el algoritmo, se encarga de sumar las matrices y nos informa la matriz resultante de las sumas de las mismas.

**Manual de uso ( paralelismo):**

Paso1: Dentro del cuaderno, debemos dirigirnos a la sección “Desarrollo” y luego ingresar el orden de la matriz que deseamos que tengan la matrices que deseamos sumar, como se indica donde dice “1 (en rojo)” señalado en la imagen, luego debemos hacer click en el botón de “play” donde dice “2 (en rojo)” en la imagen.

Paso2:



**link versión con paralelismo:** https://github.com/lucianopulido/EA2-Luciano-Pulido/commit/41df74b057f98992bbd4655a9315011bf2cb2807

**Resumen general de todo el trabajo:**

**Autoevaluación:**