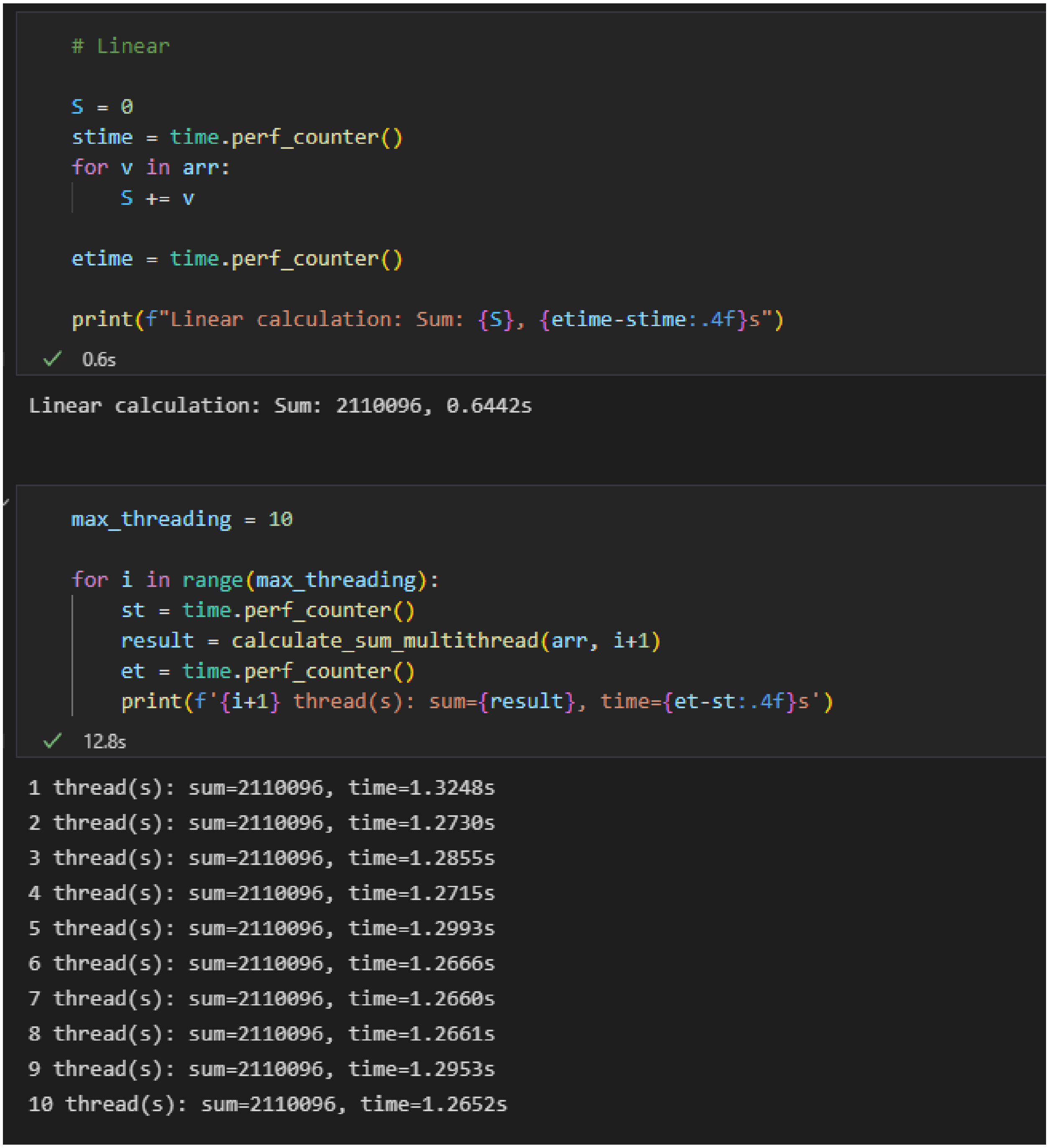
Project 2 - Multithreaded



```
# Linear

S = 0
stime = time.perf_counter()
for v in arr:
    S += v

etime = time.perf_counter()

print(f"Linear calculation: Sum: {S}, {etime-stime:.4f}s")
```

```
Linear calculation: Sum: 2110096, 0.6442s
```

```
max_threading = 10

for i in range(max_threading):
    st = time.perf_counter()
    result = calculate_sum_multithread(arr, i+1)
    et = time.perf_counter()
    print(f'{i+1} thread(s): sum={result}, time={et-st:.4f}s')
```

```
1 thread(s): sum=2110096, time=1.3248s
2 thread(s): sum=2110096, time=1.2730s
3 thread(s): sum=2110096, time=1.2855s
4 thread(s): sum=2110096, time=1.2715s
5 thread(s): sum=2110096, time=1.2993s
6 thread(s): sum=2110096, time=1.2666s
7 thread(s): sum=2110096, time=1.2660s
8 thread(s): sum=2110096, time=1.2661s
9 thread(s): sum=2110096, time=1.2953s
10 thread(s): sum=2110096, time=1.2652s
```

Thread	Running time
Linear (using basic loop)	0.6442 (s)
1 thread (using multiprocessing library)	1.3248 (s)
2 threads	1.2730 (s)
3 threads	1.2855 (s)
4 threads	1.2715 (s)
5 threads	1.2993 (s)

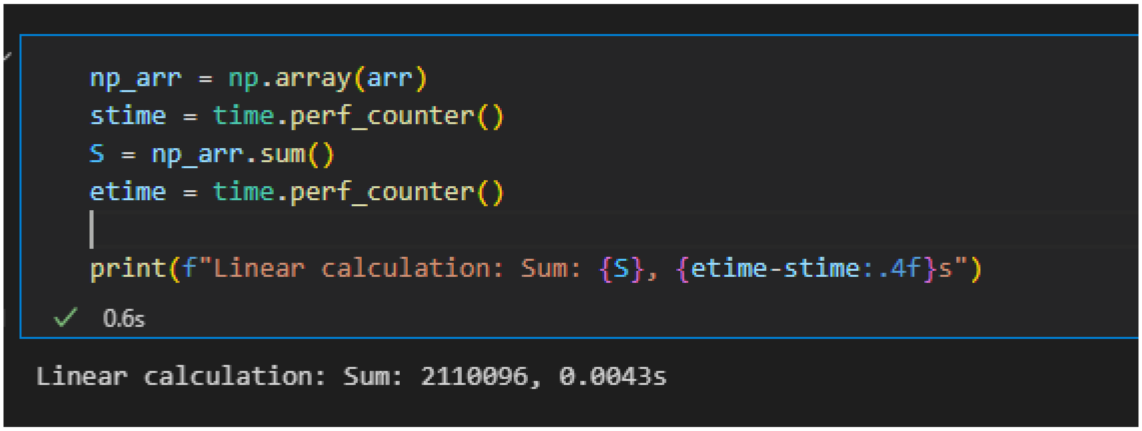
6 threads	1.2666 (s)
7 threads	1.2660 (s)
8 threads	1.2661 (s)
9 threads	1.2953 (s)
10 threads	1.2652 (s)

From the result above, we can see that the result is the same but the running time of linear calculation is lower than the running time of multi-thread **2 times**. it can be concluded that **multi-thread calculation doesn't improve the running time when increasing the number of threads**. On the contrary, it's even slower than linear calculation.

The reason comes from the thing called Python Global Interpreter Lock (GIL). GIL is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once.

Some long-running operations such as I/O, image processing, **and NumPy number crunching**, happen outside the GIL so those operations can run on multiple threads.

Let's check the calculated time when using NumPy library:



```
np_arr = np.array(arr)
stime = time.perf_counter()
S = np_arr.sum()
etime = time.perf_counter()

print(f"Linear calculation: Sum: {S}, {etime-stime:.4f}s")
```

```
Linear calculation: Sum: 2110096, 0.0043s
```

It runs really fast, just **0.0043s**. NumPy is always a strong Python-extended library for processing numbers, matrices, etc. It handles the multiple threads automatically for ourselves and integrates C, C++, and Fortran codes which have very little execution time compared to Python.