Part 2- Benchmarking functions in Python, with Multithreading

Solution format: A single Python file called `benchmark.py` containing the solutions to the following three exercises.

Exercise 4- A decorator for benchmarking

Define a Python decorator called `benchmark`. When invoking a function `fun` decorated by `benchmark`, `fun` is executed possibly several times (discarding the results) and a small table is printed on the standard output including the average time of execution and the variance.

The exact behaviour of the `benchmark` decorator is ruled by the following optional parameters:

- `warmups`: The number of warm-up invokations to `fun` (i.e. invokations whose timing must be ignored) (default: `warmups = 0`);
- `iter`: The number of times `fun` must be invoked and whose timing must be taken into account for the final metrics (default: `iter = 1`);
- `verbose`: Whether the execution should be verbose (i.e. if it must print the timing of each warm-up round and invokation) or not (default `verbose = False`);
- `csv_file`: A CSV file name where the benchmark information must be written. The header of the file will be in the form `run num, is warmup, timing` with the intuitive meaning (default `csv_file = None`, meaning that the benchmark information is only displayed on screen).

Exercise 5 - Testing the decorator with multithreading

Test your implementation by also evaluating the effectiveness of multhitreading in Python.

Using the [`threading`] module of the Standard Library and exploiting the `benchmark` decorator, write a function `test` that executes a function `f` (passed as parameter) with varying numbers of iterations and degrees of parallelism. More preciselsy, the `test` first runs `f` 16 times on a single thread, then 8 times on two threads, then 4 times on 4 threads, and finally 2 times on 8 threads. The program must write the benchmarking information for the four scenarions in corresponding files named `f_<numthreads>_<numiterations>`.

Run the program using a function that computes the `n`-th Fibonacci number in the standard, inefficent, double recursive way (choose `n` carefully) .

Discuss briefly the results in a comment in the Python file.