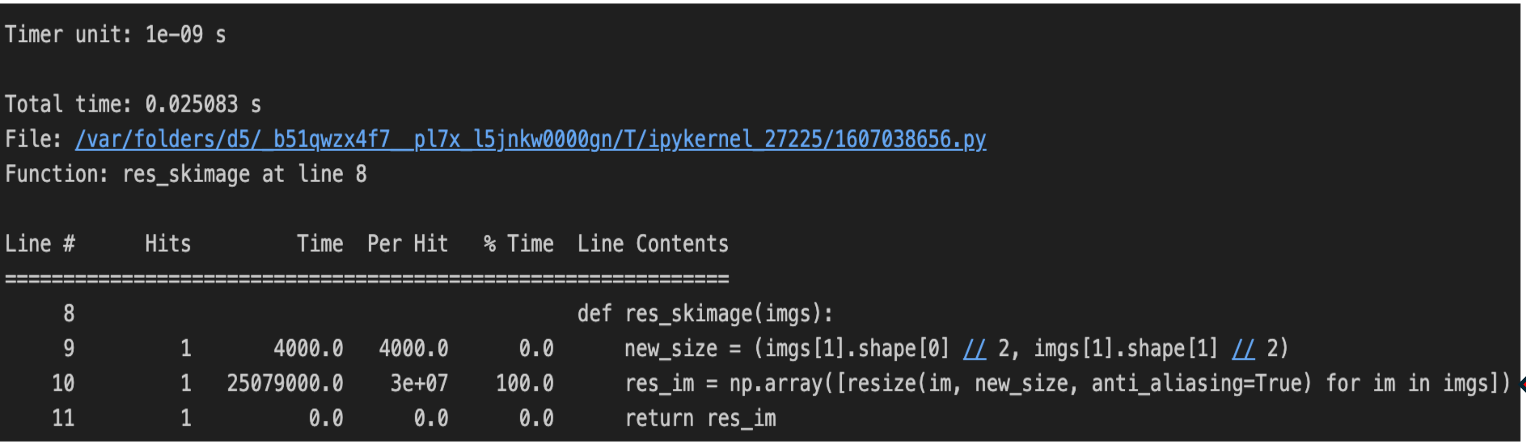
Task 1 :



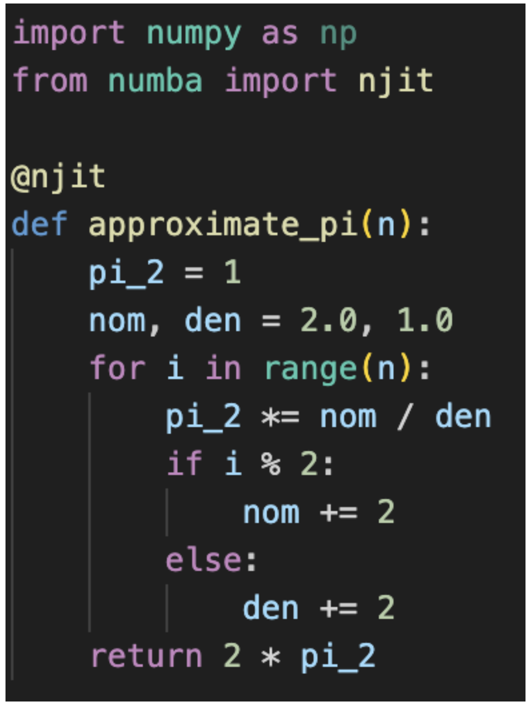
```
Timer unit: 1e-09 s

Total time: 0.025083 s
File: /var/folders/d5/_b51qwzx4f7__pl7x_l5jnkw0000gn/T/ipykernel_27225/1607038656.py
Function: res_skimage at line 8

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     8                                           def res_skimage(imgs):
     9         1       4000.0   4000.0      0.0      new_size = (imgs[1].shape[0] // 2, imgs[1].shape[1] // 2)
    10         1   25079000.0    3e+07    100.0      res_im = np.array([resize(im, new_size, anti_aliasing=True) for im in imgs])
    11         1          0.0      0.0      0.0      return res_im
```

Task 2 :

I used multiprocessing, because it allows each function call to run on a separate CPU core. In multithreading, the difference is that threads run in the same memory space, while in multiprocessing the processes have separate memory. This makes it much faster.

Sequential call of the function ran in **21.59 seconds**.
Multiprocessing call of the function ran in **10.51 seconds**.
Performance improved by **51.32%**.

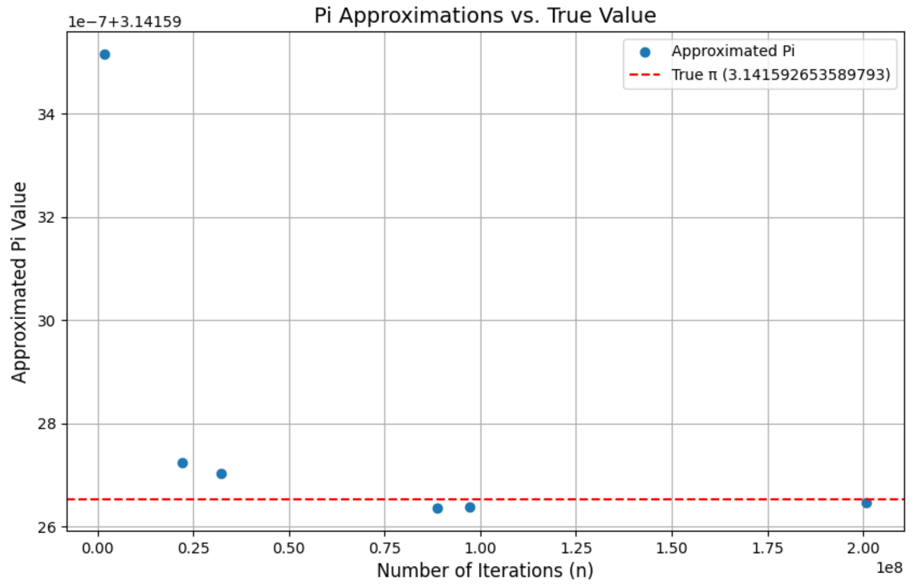
Task 3 :



```
import numpy as np
from numba import njit

@njit
def approximate_pi(n):
    pi_2 = 1
    nom, den = 2.0, 1.0
    for i in range(n):
        pi_2 *= nom / den
        if i % 2:
            nom += 2
        else:
            den += 2
    return 2 * pi_2
```

By using Numba the function is **2294.90%** faster.

Task 4 :



Rohit Potdukhe, M. N.: 22985091, IdM: yx49uxym