Aicha cheridi : 23138371

Task 2:

```
# multithreading
%%timeit
threads = []
for n in nums:
    thread = Thread(target=approximate_pi, args=(n,))
    thread.start()
    threads.append(thread)

for thread in threads:
    thread.join()
```
✓ 3m 15.7s

```
24.5 s ± 62.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
# Multiprocessing
%%timeit
processes = []
for n in nums:
    process = multiprocessing.Process(target=approximate_pi, args = (n,))
    process.start()
    processes.append(process)
for process in processes:
    process.join()
```
✓ 2m 3.7s

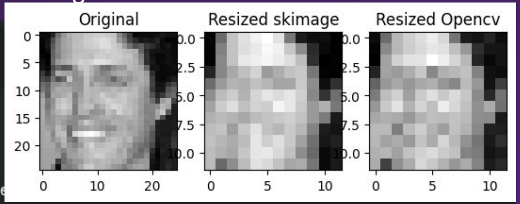
```
15.4 s ± 178 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

The code for task 2 benefits from Multiprocessing because the pi approximation function is computations heavy task and benefit more from exploiting all the cpus core through a multiprocessing. It runs 37% faster than using Multithreading

The code for task 3 benefits from Multithreading because we are reading numpy arrays from the disk i.e I/O bound tasks that benefit more from multithreading since the threads run concurrently (when one thread is idle the other can run). It runs 86% faster than using Multiprocessing

Task 1:

I used the function cv2.resize with cv2.INTER_LANCZOS4 interpolation that is 91% faster than skimage:




```
%lprun -f res_cv2 res_cv2(imgs)
```
[9] ✓ 0.1s

```
Timer unit: 1e-09 s

Total time: 0.002324 s
File: /var/folders/rj/ls_3x1nx2q96t6562rd051_00000gn/T/ipykerne
Function: res_cv2 at line 2

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     2                                           def res_cv2(imgs):
     3         1       6000.0   6000.0      0.3      new_size = (imgs[1].shape[0]//2, imgs[1].shape[1]//2)
     4         1          0.0      0.0      0.0      res_im = []
     5       200      53000.0    265.0      2.3      for im in imgs:
     6       200    2170000.0  10850.0     93.4          image_resized = cv2.resize(im, new_size, interpolation = cv2.INTER_LANCZOS4)
     7       200      47000.0    235.0      2.0          res_im.append(image_resized)
     8         1      48000.0  48000.0      2.1      return np.asarray(res_im)
```

Task 3:

```
# multithreading
%%timeit
threads = []
for file_name in numpy_files_names:
    thread = Thread(target=load_array, args=(file_name,))
    thread.start()
    threads.append(thread)

for thread in threads:
    thread.join()
```
✓ 2.6s

```
292 ms ± 18.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
# Multiprocessing
%%timeit
processes = []
for file_name in numpy_files_names:
    process = multiprocessing.Process(target=load_array, args = (file_name,))
    process.start()
    processes.append(process)
# Wait for all processes to complete
for process in processes:
    process.join()
```
✓ 17.5s

```
2.2 s ± 88.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```