Exercice 4 : Tâches en parallèle via le multi-threading

Objectifs

Multi-threading
Synchronization des accès

Enoncé

Cette exercice a pour objectif de compresser les images d'un répertoire en faisant le traitement de chaque image dans un thread afin de minimiser le temps d'exécution et maximiser l'utilisation des cores. (le répertoire ne contient que 4 images, donc on aura au plus 4 threads simultanement !)

Ce traitement sera aussi l'occasion de mesurer les tailles des fichiers avant/aprés compression pour, en fin de traitement, afficher un taux de compression pour l'ensemble des fichiers (ce chiffre pourrait être le rapport des tailles de fichiers aprés / avant traitement)

es fichiers sont disponibles dans un répertoire `IMAGES` (voir l'archive IMAGES.tar à downloader puis uploader vers le notebook)
Les fichiers compressés seront mis dans un répertoire `COMPRESSED_IMAGES`, à créer s'il n'existe pas, par un des threads de traitement d'image
Penser à synchroniser les sections critiques (création du répertoire de destination, cumul des tailles de fichiers)

Exemple de résultat d'exécution du code

```
15:52:35.598    ProcessImages   MainThread INFO    Start processing images from di
rectory IMAGES
15:52:35.598 CompressOneImage     Thread-1 INFO    Creating directory COMPRESSED_I
MAGES
15:52:35.598 CompressOneImage     Thread-1 INFO    Start compressing image img1.jp
g
15:52:35.599 CompressOneImage     Thread-2 INFO    Start compressing image img2.jp
g
15:52:35.599 CompressOneImage     Thread-3 INFO    Start compressing image img3.jp
g
15:52:35.600 CompressOneImage     Thread-4 INFO    Start compressing image img4.jp
g
15:52:36.073 CompressOneImage     Thread-1 INFO    End compressing image, new imag
e saved as compressed_img1.jpg
15:52:36.089 CompressOneImage     Thread-2 INFO    End compressing image, new imag
e saved as compressed_img2.jpg
15:52:36.114 CompressOneImage     Thread-3 INFO    End compressing image, new imag
e saved as compressed_img3.jpg
15:52:36.125 CompressOneImage     Thread-4 INFO    End compressing image, new imag
e saved as compressed_img4.jpg
15:52:36.127    ProcessImages   MainThread INFO    End processing images
15:52:36.127    ProcessImages   MainThread INFO    Here are some results :
 Total files size                 3149598
 Total compressed files size      1191717
 Compression rate                 0.4
 Processing duration (ms)         528.9
```

Quand l'exercise sera (quasi) fini, vous pourriez essayer de faire le traitement des fichiers en mode séquentiel i.e. sans utiliser les threads pour se persuader de ce qu'apporte le multithreading en terme de gain de temps d'exécution sur cette exemple là.

Indications

- Faire une programmation en orienté objet et en utilisant le logging comme dans l'exercice précedent (un logger par classe et avec affichage de la classe mais aussi du thread via `%(threadName)12s` dans le pattern de formattage)
  On aura une classe `CompressOneImage`, dont le but est de compresser une image, et ceci dans un thread (cette classe dérivera donc de Thread)
  On aura une classe `ProcessImages`, dont le but est de scanner le répertoire IMAGES/ et de lancer un thread par image à traiter
  Et enfin un code d'utilisation de cette classe

.../...

- Le scan du répertoire `IMAGES/` peut se faire comme ceci

```
# filenames will be a list of all files found in directory
for (dirpath, dirnames, filenames) in walk(CompressImage.IMAGES_DIRNAME):
  pass
```

  Comme dit dans le commentaire, filenames contiendra tous les fichiers du répertoire.
  Il faudrait donc filter les noms selon leur extension pour ne retenir que ceux suffixés '.jpg' (l'occasion d'utiliser les "listes comprehensions" !)

- Le traitement d'images sera fait avec module **Image** du package **PIL**
  La lecture d'une image, l'obtention de la taille du fichier, et la sauvegarde de l'image compressée seront fait selon ces indications

```
# opening and getting image
picture = Image.open(image_filepath)

...

# get image filesize (in B)
image_filesize = os.stat(image_filepath).st_size

...

# compress image
compressed_image_filepath = "..."
picture.save(compressed_image_filepath, "JPEG", optimize=True, quality=30)

...
```

- La création du répertoire COMPRESSED_IMAGES, sera faite selon ces instructions
  (Penser à synchroniser les accès concurrents si cette étape est faite dans un thread comme celà est demandé)

```
if not os.path.exists(compressed_image_directory):
  os.makedirs(compressed_image_directory)
```

- Les calculs et cumul des tailles des fichiers seront fait dans les threads, et stockés dans des variables globales
  Penser à synchroniser les accès concurrents

- Pour démarrer, un aperçu de la classe `CompressOneImage`

```
import logging

import threading
from threading import Thread

import sys
import os
from os import walk
from PIL import Image # available with 'pip install pillow'

import time   # for time.time() which gives time in seconds since the epoch

class CompressOneImage(Thread):
  # When specifying datefmt, (see below), ms are lost unless you add then with
 %(msecs) pattern
  MY_FORMAT = "%(asctime)s.%(msecs)03d %(name)16s %(threadName)12s %(levelname
)-6s  %(message)s"
  logging.basicConfig(format=MY_FORMAT, datefmt='%H:%M:%S', level=logging.INFO
)

  IMAGES_DIRNAME = "IMAGES"
  COMPRESSED_IMAGES_DIRNAME = "COMPRESSED_IMAGES"

  # will hold the cumulative sizes of processed files
  IMAGES_FILESIZE = 0
  COMPRESSED_IMAGES_FILESIZE = 0

...
```