Team Exercise #2

David Kallemeyn

Last Revised Spring 2024

Due: 4/10/2024 by 10:00 PM

Submission: github repo link uploaded in canvas. The repo should contain your **team_ex_2.py** file and documentation of your process (you can use github, this document, a readme file, or other method).

Assignment type: Team, In-class

Points: Complete/Incomplete. No points are assigned. Extra credit points may be awarded to teams based on their approach and solutions.

INSTRUCTIONS

The file **team_ex_2.py** contains a naive solution to the Pair Exercise #4 assignment, which you are familiar with having recently submitted a solution. The code uses the *wikipedia* package and downloads references for related pages to a specified topic. This file has 4 functions:

1. a function to execute the task sequentially
2. a function to execute with multiple threads
3. a function to execute with multiple processes
4. a helper function to convert objects to

strings General advice/resources:

- you will need a virtual environment with the wikipedia package
- installed you may want to use wrapper functions or other abstractions

Problem

Your task is to review the code in the **team_ex_2.py** file, discuss it as a team, and refactor the code to be more efficient and adaptable. Your review process should include:

- identification of possible
- issues creation of tasks
- assignment of tasks
- description of each issue and how it was resolved

In addition to issues/areas for improvement discovered in the codebase, the refactored solution must add the following new functionality:

- allow the user to specify their own search term
- if the user input search term is less than 4 characters, it should default to "generative artificial intelligence"
- create a new directory named "wiki_dl" in which to store the created .txt files

Working Space / Notes

If you would like, feel free to use the template below to assist with the problem. However you decide to approach the problem as a team, be sure to document your review process.

Issues

- ➔ Issue 1: Code duplication
- ➔ Issue 2: Error handling
- ➔ Issue 3: Scalability
- ➔ Issue 4: Documentation
- ➔ Issue 5: Code inefficiency due to redundant operations

Tasks

task #	description	assignment	solution
1	The dl_and_save_thread and dl_and_save_process functions are nearly identical, performing almost identical tasks.	Anjan	Merged dl_and_save_thread and dl_and_save_process functions into a single dl_and_save function. Now, both concurrent executor types, ThreadPoolExecutor and ProcessPoolExecutor, use the same function to download and save references.
2	There is no error handling present. If there were any problems while downloading a page or writing to a file, the program would crash without providing any feedback.	Mike	Added a try-except block within the dl_and_save function to catch any exception that may occur. Additionally, in case of errors, the program can now report an error message without crashing abruptly.
3	The problem with ThreadPoolExecutor and ProcessPoolExecutor is that if the tasks are doing a lot of processing, they could slow down the code execution, use too much memory, or even hit limits set by the OS.	Roberta	While this issue may require further optimization, the new code addresses the mentioned issues by providing more flexibility and specifying the executor type - either ThreadPoolExecutor or ProcessPoolExecutor - and the number of workers, so threads or processes, to be used concurrently.
4	The code lacks proper documentation/comments explaining the purpose of each function and/or their	Haifa	Comments have been added to describe how the functions work more clearly. Additionally, the function names were changed to clearly reflect their purposes.

	parameters.		
5	The original code performed a Wikipedia search operation multiple times using different functions. This led the code to perform redundant network calls and be extremely inefficient in its search execution.	Anjan	The search operation was moved to the `__main__` block for it to execute only once. The search results now pass as an argument to other functions (“wiki_sequentially”, “concurrent_threads”, and “concurrent_process”), reducing the amount of network calls made and improving the overall code performance.

```
import time
import wikipedia
from concurrent.futures import ThreadPoolExecutor, ProcessPoolExecutor

# Convert objects produced by wikipedia package to a string var for saving to text file
def convert_to_str(obj):
 if type(obj) == list:
   mystr = '\n'.join(obj)
   return mystr
 elif type(obj) in [str, int, float]:
   return str(obj)

# IMPLEMENTATION 1: sequential example
def wiki_sequentially(results):
 t_start = time.perf_counter()
  def dl_and_save(item):
   page = wikipedia.page(item, auto_suggest=False)
   title = page.title
   references = convert_to_str(page.references)
   out_filename = title + ".txt"
   with open(out_filename, 'wt') as fileobj:
     fileobj.write(references)

 for item in results:
   dl_and_save(item)

 print('\nsequential function:')
 t_end = time.perf_counter()
 print(f'code executed in {t_end - t_start} seconds')
```

```
# IMPLEMENTATION 2: concurrent example w/ threads
def concurrent_threads(results):
 t_start = time.perf_counter()
  def dl_and_save_thread(item):
   page = wikipedia.page(item, auto_suggest=False)
   title = page.title
   references = convert_to_str(page.references)
   out_filename = title + ".txt"
   with open(out_filename, 'wt') as fileobj:
     fileobj.write(references)

 with ThreadPoolExecutor() as executor:

   executor.map(dl_and_save_thread, results)

 print('\nthread pool function:')
 t_end = time.perf_counter()
 print(f'code executed in {t_end - t_start} seconds')

# IMPLEMENTATION 3: concurrent example w/ processes
def concurrent_process(results):
 t_start = time.perf_counter()
  def dl_and_save_process(item):
   page = wikipedia.page(item, auto_suggest=False)
   title = page.title
   references = convert_to_str(page.references)
   out_filename = title + ".txt"
   with open(out_filename, 'wt') as fileobj:
     fileobj.write(references)

 with ProcessPoolExecutor() as executor:
   executor.map(dl_and_save_process, results)
 print('\nprocess pool function:')
 t_end = time.perf_counter()
 print(f'code executed in {t_end - t_start} seconds')

if __name__ == "__main__":
 search_results = wikipedia.search("general artificial intelligence")
 wiki_sequentially(search_results)
 concurrent_threads(search_results)
```

```
concurrent_process(search_results)
```

Sequential function:
code executed in 5.750310671981424 seconds

Thread pool function:
code executed in 1.0442203721031547 seconds

Process pool function:
code executed in 0.8030342811252922 seconds