# Asynchronous Processing in Python

## Overview

In this lab, you'll enhance an existing application so that it performs time-consuming tasks concurrently using the Python *asyncio* library.

## Source folders

Student folder : C:\AdvPythonDev\Student\08-Async

Solution folder: C:\AdvPythonDev\Solutions\08-Async

## Roadmap

There are 5 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Getting ready to use the *asyncio* library
2. Implementing the directory-search as a coroutine
3. Scheduling the directory-search coroutine
4. Awaiting for all tasks to complete
5. Additional techniques (if time permits)

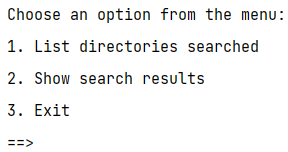
## Familiarization with the application

In the *student* folder, run main.py. When prompted, enter a directory name and press *Enter*. The application asks you to enter another directory name, so do so. Repeat this a few times, and then enter *done*.

The application then searches each directory, to find all the files and sub-directories in each directory. It performs this search sequentially:

* First it searches your first directory, and when this is complete it displays a message to indicate as such.
* Then it starts searching your second directory, and displays a message when this is complete.
* And so on, for each of your directories.

When all the searches are complete, the application displays a menu that enables you to review the results:



Choose option 1 to see all your directories. Then choose option 2 and specify one of your directories - the application will display all the files and sub-directories in that directory recursively.

## When you’re comfortable with how the application works, take a look at the code in main.py and make sure you understand how it works. Feel free to ask the instructor about anything.

## Exercise 1: Getting ready to use the asyncio library

To get ready to use the *asyncio* library, follow these steps:

* At the top of main.py, import the asyncio library.
* Prefix the main() function with the async keyword. This transforms it from a "regular function" into a coroutine, i.e. something that can be scheduled to execute in an *asyncio* event loop.
* At the end of main.py, locate the code that calls main(). You need to modify this statement because main() is now a coroutine, and you can't just call a coroutine like that - instead you must schedule it for execution on an *asyncio* event loop. Your application doesn't currently have an *ayncio* event loop, so call asyncio.run() to start one. Pass main() as a parameter, to schedule main() for execution on the event loop. That all sounds quite tricky, but here's all you actually need:

asyncio.run(main())

**Exercise 2: Implementing the directory-search as a coroutine**

Locate the search\_directory() function and prefix it with the async keyword. This transforms the function into a coroutine, so it can be scheduled to execute concurrently in the event loop (you'll do this in Exercise 3).

The code inside the search\_directory() function runs in a tight loop. If you ran this code in a coroutine, it would hog the application's main thread, so the thread would never get a chance to peek in the event loop to schedule any other coroutines. In such a case, a coroutine should yield control periodically so that the application thread can get a chance to schedule other coroutines. The easiest way to do this is to call asyncio.sleep(0) in one of the loops; prefix this call with await, so that your coroutine continues execution after its sleep.

**Exercise 3:** **Scheduling the directory-search coroutine**

Locate start\_all\_searches(). Currently this function calls search\_directory() as a regular function, but that won't work now because search\_directory() is now a coroutine.

Instead, you need to schedule the coroutine for execution on the *asyncio* event loop. Here's the code you need:

task = asyncio.create\_task(search\_directory(directory\_name))

create\_task() returns a Task object that represents the coroutine. The idea is that you can use the Task object to manage the coroutine.

After you've made this change, your application now has a separate task for each directory search. In order to coordinate all these tasks, we suggest you create a list and add each task to the list. Return the list of tasks from your start\_all\_searches() function.

**Exercise 4:** **Awaiting for all tasks to complete**

Modify main() so that it receives the return value from start\_all\_searches() and then awaits for all the tasks to complete. After the tasks have completed, you can then proceed to call display\_results() to display the results as before.

Run the application and enter some directory names. You should now find that the directory searches are interspersed concurrently, rather than running consecutively.

**Exercise 5 (If time permits): Additional techniques**

Explore some of the additional techniques covered in the chapter, e.g. polling for task completion, cancelling tasks, etc.