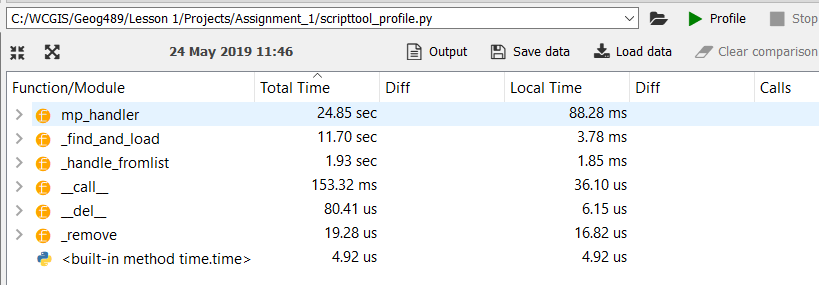
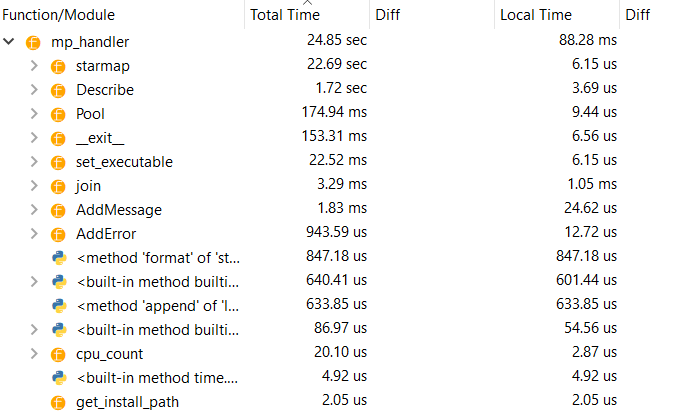
Lesson 1:

**Tool 1A**:

For this tool I left most of the original code unmodified. At my previous employer, I have used ArcGIS Pro to accomplish a task that needed 64-bit because I was running out of memory with ArcDesktop but was not aware of multiprocessing. I tested this in a few examples and published to a gist on my Github, so I had full understanding of how everything was working. To meet the guidelines of the assignment, I added another parameter for the user to select an output folder. I knew this needed to get passed to the worker function, so I added it to my jobs list, so each worker knew the folder that the files were getting exported to. I used os.path.basename and os.path.join for my output feature class name, and was able to get some practice throughout my tools with the os module.

These are my results in spyder for tool 1A. I included my (.prof and .run) files as well.





Total script time: about 25 seconds without find and load. About 23 seconds in ArcPro.

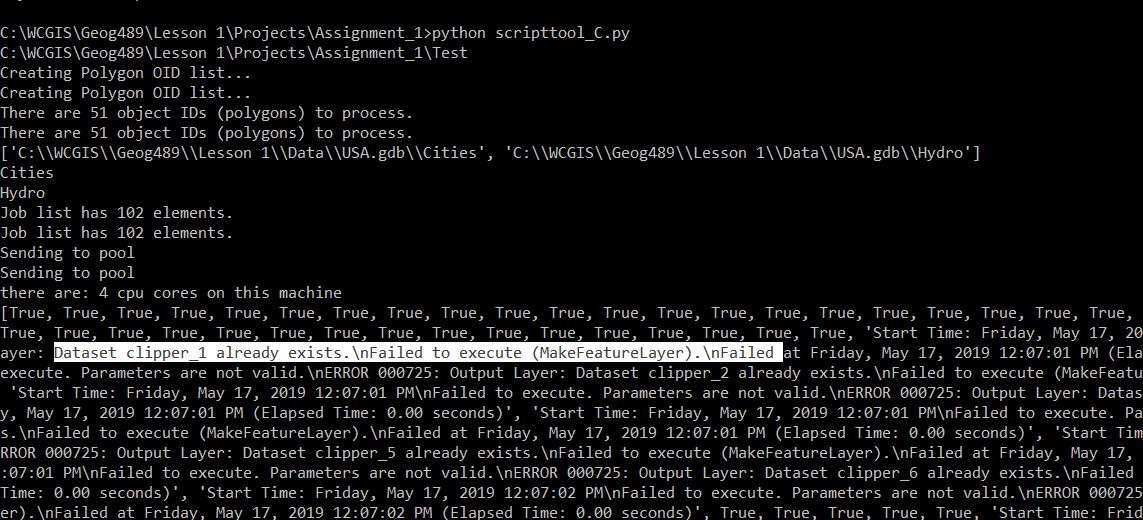
Most of the script time was spent in starmap where the workers are running, and the clip is happening. Describe is the second highest.

Starmap – 22.69 seconds.

Describe – 1.72 seconds.

**Tool 2B**: This tool includes the ability to handle multiple input feature classes. I created a script tool for this and since I used GetParameterAsText in 485 for multiple input, I wanted to try the GetParameter this time. Doing it this way compared to text, I found that there was a lot less conversion involved and seemed to be cleaner and easier to work with. Something else I added in here, was outputting a state name to the output name. I was able to include this as a user parameter and pass it to my worker function by appending the name of the state based off the index position of the object id. In turn, each list has the same identical amount of ids and states.

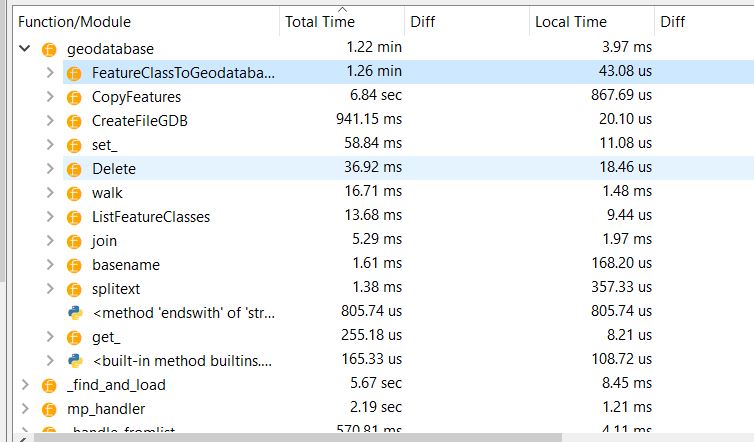
I had an issue with this tool, in that every time I ran the tool, at least ten workers were failing. I tried several things to figure out why these were failing, since I couldn’t get the output of any errors, I returned the messages instead of False.



My issue with why workers were failing involved the make feature layer. The workers were running with each other and outputting the layer with the same name. This was a quick fix and a slight oversight on my part, as a fix I added the name and name field to the output feature layer. I was glad this came up though, as it gave me an opportunity to find a way to output error messages with worker functions.

**Tool 2C**:

This tool converts feature classes into a geodatabase. I knew from the readings that anything involved with a geodatabase needed to happen outside of the worker function to avoid locks, but I also wanted to try and do everything in memory. My original test included trying to clip everything and put it in memory in my worker function. After running my script several times in the terminal, I had several issues with this. From reading up a bit on ‘memory’, ESRI suggested that this wouldn’t be possible. “A geoprocessing operation that writes to the in\_memory workspace holds the result of that operation in a particular piece of memory that is not available to other threads”. This was my problem, so I needed another way to increase my performance. I left this as is and went to my geodatabase function. Here I copied every feature class that was clipped to memory. My thought process here was to eventually create the geodatabase in memory and do everything there. I continued to learn about the limitations of ‘memory’ workspaces. I ran this through the spyder profiler and noticed that two items were taking the most amount of time. You can see that mp\_handler was taking very little time, but I wasn’t ever able to access the results as the workers couldn’t work together with memory.



Most of my efforts with this script was to cut down time, so I attempted to cut time where I could. FeatureClassToGeodatabase was really slowing my script down.



I took this out and decided to copy them in instead, to see if I could speed up the time of the script. With using arcpy.Copy, I was able to cut over ten seconds.



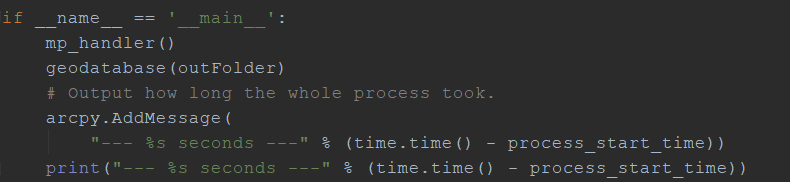
Looking at my profile some more, copying these to memory was taking some time, so I wanted to see if the tradeoff was worth it. Without copying to memory, I cut an additional six seconds.



I still wanted to use ‘memory’ where I could and thought about running the search cursor in memory, although as discussed in the lesson material these are already fast. For testing purposes and memory practice, I wanted to see if there was any speed increase. This increased my performance and I was able to cut seven more seconds.



Still looking over the profiling results, I thought of one more performance boost I could use, and I moved my geodatabase function inside the mp\_handler function as it was currently getting called after mp\_handler.



This ended up getting me the results I was looking for, and I ended cutting about thirty-six seconds overall from beginning to end.



This exercise really got me thinking about performance and how it can be cut. I learned a lot about memory workspaces and how the threads can’t talk to each other in situations, and what would work and what doesn’t.

Outside of profiling for Tool C, I wanted to work with the os module to get some practice with this. I kept the output folder organized by placing the feature classes in their own individual folders. This brought some issues up later on, as I couldn’t use arcpy.ListFeature classes with the way these were output. I could change my workspace for every folder, but I assumed that would come at a performance cost, and from what I’ve read this is bad practice. Doing some digging, os.walk would get me what I needed. I can loop through the workspace and get the files I needed. This will give me all the folders, files, and subdirectories. Originally, I was placing these into a list and using feature class to geodatabase. As I cut my performance, my code was also cut, and I was able to avoid this and just copy these in the loop. I also deleted any reference to ‘print’, since I was just running this in Pro and that can add some time, I also removed any arcpy.Addmessage or print references in my worker function since the user will never see those.

**Over and above**:

* For my over and above, I did the following:
* Added a field parameter and modified the output name to include a row from the field.
* Created a script tool for the multiple input parameters.
* Increased efficiency of code using profiling.
* Created a script tool to output to a geodatabase.
* Added in-tool documentation.
* Placed this write-up on my GitHub.