This project required that around a thousand GeoTIFFs (~120GB total) be rendered into raster tiles according to the OSM Slippy Maps standard (XYZ Tiles) for use in a Leaflet web map. While the files could have been processed one at a time using available open-source command line scripts, it seemed more prudent from the outset to program a reusable bulk-processing solution. So, I wrote a Python module that allows its user to render raster tiles for an entire directory of GeoTIFFs with one simple command.

```
from tileTools.makeTiles import make_tiles

make_tiles(input_folder="tests/TIFFs",
           output_folder="tests/output")
```

In addition to simplifying and speeding up the process, the module also implements a robust solution to the problem of deciding how many zoom levels to render from each GeoTIFF (detailed at the end of this article) and provides the option to have metrics from the rendering process written to an sqlite3 database file. Visit the murf-tiling GitHub page for more detail about all the available arguments.

The tools that make up the core of the tiling process are gdal2tiles, which creates raster tiles from GeoTIFFs; and gdal_translate, which converts the original GeoTIFF file encoding to an 8-bit encoding suitable for use with gdal2tiles. Both of these tools are included in the Geospatial Data Abstraction Library (GDAL), a collection of software sponsored by the Open Source Geospatial Foundation (OSGeo). A GDAL Python package is also available via conda-forge or via unofficial Python wheels hosted by the University of Irvine's Laboratory for Fluorescence Dynamics (UCI-LFD). My solution to the problem of deciding how many zoom levels to render also depends on a third-party Python library called Rasterio, which provides access to some of GDAL's functionality in an convenient, object-oriented wrapper. Rasterio is likewise available via conda-forge or via the UCI-LFD unofficial wheels page linked above. The rest of the software tools that I used (including sqlite3) are available with every distribution of Python via the Python Standard Library.

In general terms, my Python module selects all the files in the supplied directory that have the ".tif" file extension and processes them one at a time. First it uses gdal_translate to create a byte encoded version of the file. If it encounters any errors in this process, it logs the error and moves on to the next file so that the process can complete for as many files as possible without requiring a manual restart. After a byte encoded version of the file is successfully created, the module selects the optimum maximum zoom level for which to create tiles by finding the first zoom level whose spatial resolution is equal to or greater than the spatial resolution of the original GeoTIFF, ensuring that all the original visual detail is available on the web map while avoiding the exponential increase in processing time that results from rendering unnecessary subsequent zoom levels (see below for the equation I use). Finally, armed with an appropriate maximum zoom level, the module creates a tile set via gdal2tiles - either with a set of default arguments (including multi-processing using the machine's full number of available cores) or with alternatives supplied by the user. Similarly to the translation step, if the tiling step fails for a map, the error is logged and the module moves on to the next GeoTIFF.

This tidy Python module was the result of much trial and error, a large portion of which had to do with finding the appropriate max zoom level. In my original trials I rendered each zoom level independently and recorded the duration of every step of the process. I quickly discovered that the render time increased exponentially for each zoom level, going from something like 3 minutes for one zoom level to over 15 minutes for the next and then potentially hours for the next. Multiply these rendering times by 1000 GeoTIFFs and you will see that the process could quickly escalate from taking 50 hours (3 min/file) to 250 hours (15 min/file) to 1000 hours (1 hr/file). Selecting a single maximum zoom level for every map was not an option because each map has a different spatial resolution, meaning that while some maps could be rendered fully at a max zoom of 12, for example, others would be barely visible until zoom level 16 or greater. My first solution to this problem was to simply abort the tiling process after the first zoom level that took over a minute to render. This strategy gave fairly consistent results initially, but failed during the second batch of files due to the gdal2tiles script taking over a minute to initialize when rendering the first zoom level for reasons unknown. Even if it did continue to work, though, it was entirely dependent on the context of one particular computer, something that weighed heavy on my mind.

Through much research and pondering, I eventually devised a way to calculate the zoom level equivalence of the original GeoTIFF's spatial resolution. Since the tiles are being rendered according to the OSM Slippy Map Tile (XYZ) Standard, I was able to locate and rearrange the equation used to determine the width of each pixel in meters of an XYZ tile at a given zoom level so that I could instead determine the "zoom level" (returned as a float value and rounded up to the next integer) of a GeoTIFF given the width of its pixels. Then all that was required to render all the original GeoTIFF's visual detail without wasting time on unnecessary resampling was to use its "zoom level" as the maximum zoom level in the gdal2tiles script. I have reproduced my math below for anyone who might want to use a similar strategy.

1. Variables:

   $w =$ pixel width

   $C =$ equatorial circumference of the earth in meters $= 40,075,016.686$

   $l =$ latitude of the GeoTIFF in radians

   $z =$ zoom level

2. OSM equation for horizontal width per pixel given zoom level:

   $w = \frac{C \times \cos(l)}{2^{z+8}}$

3. Rearrangement of the equation:

   $w \times 2^{z+8} = C \times \cos(l)$

   $2^{z+8} = \frac{C \times \cos(l)}{w}$

   $log_2(2^{z+8}) = log_2(\frac{C \times \cos l}{w})$

   $z + 8 = log_2(\frac{C \times \cos(l)}{w})$

4. Resultant equation (zoom level given pixel width):

   $z = log_2(\frac{C \times \cos(l)}{w}) - 8$