COSC 2P91 – Assignment 4 (Mini-Project 2) – Part A

(Note: This is a placeholder until the entire project is up)
Assignment 4 will deal with combining multiple techniques we've covered across the course:

- Writing a library in C
- Joining that library with a Python script
- Threading
- File IO
- Data management

Our final task will be to create a simple *ray tracer* – a program that, when configured with information about shapes within a virtual space, will be able to render an image of some view within that space. Ray tracers rely on determining the intersections between rays and the shapes.

Specifically, a ray tracer fires a ray for each pixel of the image, into the field, to calculate the colour for assignment to that pixel, based on the lights in the scene and the material of the intersected shape.

Before getting to that, however, we first need to understand how we'll be using both C and Python to generate images.

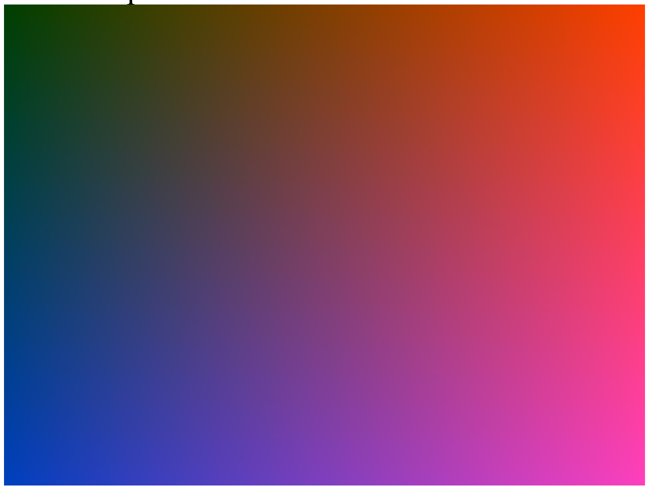
Task:
You'll be generating an image of user-defined width, height, and filename. The image will be a Portable Pixmap. The program must allow for multiple images to be generated (i.e. the program will loop until the user chooses to quit).

The generated image will always be the same in principle:

- The red will start at 0 on the left, and increase to full intensity* at the right
- The green channel will always be around 63 (25% brightness/intensity)
- The blue will start at 0 at the top, and increase to full intensity* at the bottom

*For all of these calculations, rounding errors are fine. It doesn't matter if the green is 62 or 66, the red and blue can stop at 254, etc.

This will produce a colour field as shown:



Python:
You *main* program will be written in Python. Call it `imagegeneration.py`. The Python script will cover the following:

- Interaction with the user, based on a text interface (i.e. `print` and `raw_input`)
- Saving the generated images to disk
- Coordinating with the C component (covered below)

C:
The C component will be handling all of the actual image generation/rendering. It's basically the number-cruncher. You're free to write a Python module in C, or to write a Shared Object library (and use `ctypes` in Python to load the library for use).
If you want to save yourself a lot of time and effort, just write it as a C Shared Object library (entirely oblivious to Python), and rely on `ctypes`.
Specifically, the C portion is responsible for:

- Rendering the image
  - This should be broken down into a separate task for calculating individual pixel data
- In later parts, it will also need information for the scene (to aid in rendering)

File IO:
You'll be saving your images as Portable Pixmaps (the Netpbm format). I'd suggest type P3.

- The file size will be atrocious, and you may not have an image viewer that can view them. Refer to the tips below for advice on this

Threading:
This part of the project does not require threading. You're free to generate the image in a single thread. However, for the sake of making later work easier, you're highly encouraged to go straight to threading.

- It's up to you whether you do the threading in Python or C. If you're looking for a suggestion: pthreads in C are very well-suited for these problems, and reduce the difficulty significantly
- For the later parts, you'll need the option to explicitly choose the number of threads to allocate for rendering. In case it's important for your design, you're allowed to assume the number of rendering threads will never exceed 8 (if you wish)

Tips:

- Use proper design: for the sake of encapsulation, you may wish to create a class for an image; otherwise you'll want a struct. If using an image, remember to document it. Consider using properties or proper getter/setters. Also, use methods when appropriate. Ensure that C functions that aren't intended to be accessed outside of that library/module are declared static.
- It has already been suggested to use `ctypes`. Specifically, try allocating space for the image in Python
  - You can create a c-friendly array using asterisk:
    - c_int*5 creates a class for producing 5-integer arrays
    - This means you need to actually call its constructor
      - e.g. `(c_int*5)()` or `(c_int*5)(1,3,5,7,9)`
- Even though you're effectively generating 3D data (width, height, and colour channel), you're strongly encouraged to keep it simple and just use a single-dimensional array
- Don't ignore the instruction to create a task (i.e. function) to calculate a single pixel. This will safe you time later on

- Thought for the future: your final task will require proper documentation (including docstrings). You may find it easier to just start including them now
- As mentioned, PPMs are big and esoteric. If you wish to be able to view them directly, consider using either MobaXTerm or Xming, and then using `eog` on sandcastle. Alternatively, you can use imagemagick on sandcastle to convert it to a more common format
    - e.g. assuming a filename of `dealie.ppm`:
        - `convert dealie.ppm dealie.png`
        - You can then download the .png for traditional viewing

Resources:
https://docs.python.org/2/library/ctypes.html
http://netpbm.sourceforge.net/doc/ppm.html
http://people.sc.fsu.edu/~jburkardt/data/ppma/ppma.html