All assignment work will be done in the included template files on Github this week. Do not forget to adjust the README to indicate you have completed the assignment before your final commit!

Get Assignment link: https://classroom.github.com/a/3YKU17L_

1. The ***digital root*** of an integer n is define as the result of summing the digits repeatedly until only a single digit remains. For example, the digital root of 1729 can be calculated using the following steps:

$$\begin{array}{lrcr} \text{Step 1:} & 1+7+2+9 & \rightarrow & 19 \\ \text{Step 2:} & 1+9 & \rightarrow & 10 \\ \text{Step 3:} & 1+0 & \rightarrow & 1 \end{array}$$

   Because the total at the end of step 3 is a single digit (1), that value is the digital root. Your task for this problem is to write a function `digital_root` that returns this value for any provided number. `digital_rool` will take a single argument or input, which will correspond to the starting number. The template file `Prob1.py` sets up the basics of this function for you. You are free to define any other helper functions you might like, but make sure they are commented and explained. At the bottom of the file is a variable `test_input` which you can change to other values in order to easily test your function. By default it is set to the above example. Here are some other examples you can test against:

```
digital_root(1729)   → 1
  digital_root(45)   → 9
digital_root(2021)   → 5
digital_root(314159) → 5
```

   (*Hint: an example in Chapter 2 of the book may prove very useful here, but there are other ways to do this as well.*)

2. The German mathematician Gottfried Wilhelm von Leibniz discovered the rather remarkable fact that the mathematical constant π can be computed using the following mathematical relationship:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

   where the formula to the right of the equals sign represents an infinite series; each fraction represents a term in that series. If you start with 1, subtract one-third, add one-fifth, and so on for each of the odd integers, you get an umber that gets closer and closer to the value of $\frac{\pi}{4}$ as you go along.

   Your task here is to write a function which can return the approximate value of pi for any desired number of terms. That is, if we called the function and wanted it to only use a single term, it would give:

```
approximate_pi(1) → 4
```

whereas taking the first 5 terms as shown above gives:

`approximate_pi(5)` $\rightarrow 3.339682$

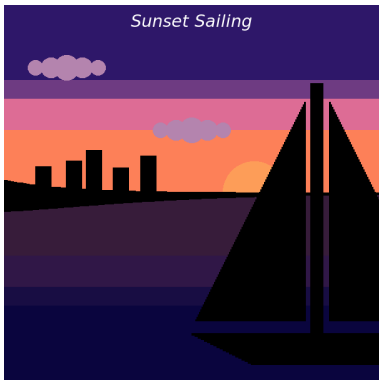
Your function should return the proper approximation of π for any number of terms, though I will not test it with term counts over 1 million.

3. While printing content or inputting content from the terminal is nice, often times you want to have more control over graphic elements of your program. To that end, we are using the `pgl` library in class this semester. To start things off in a very simple manner and to get you some more practice with the library, this week you will just need to draw a pretty picture of whatever you might like. A few qualifications though to get full points:

   - It must be a coherent picture. No purely abstract art comprised of random shapes.
   - You must use multiple colors
   - You must use multiple types of `GObjects` (ovals, rectangles, lines, etc).
   - You must define at least one function which groups together some code relating to a particular object (or objects) in your image (for instance, a function to draw a tree at some location, or a cloud, etc). The function must take some form of input. It can not be a helper function such as `draw_filled_rect` or similar.
   - You must use comments or docstrings to label what different functions or parts of your code are responsible for drawing.
   - You must use a loop to draw some repeating portion of your image.
   - You must title your masterpiece at the top or bottom using a `GLabel` centered horizontally within the window.

   If you need a list of known colors in `pgl` I've taken the time to give you a visual chart on the last page of this document! Or you can use something like a color picker to get the hex value for a color (starts with a # symbol) and provide that string (including the #) directly to the `set_color` method.

   As a bit of an example, below is my creation:



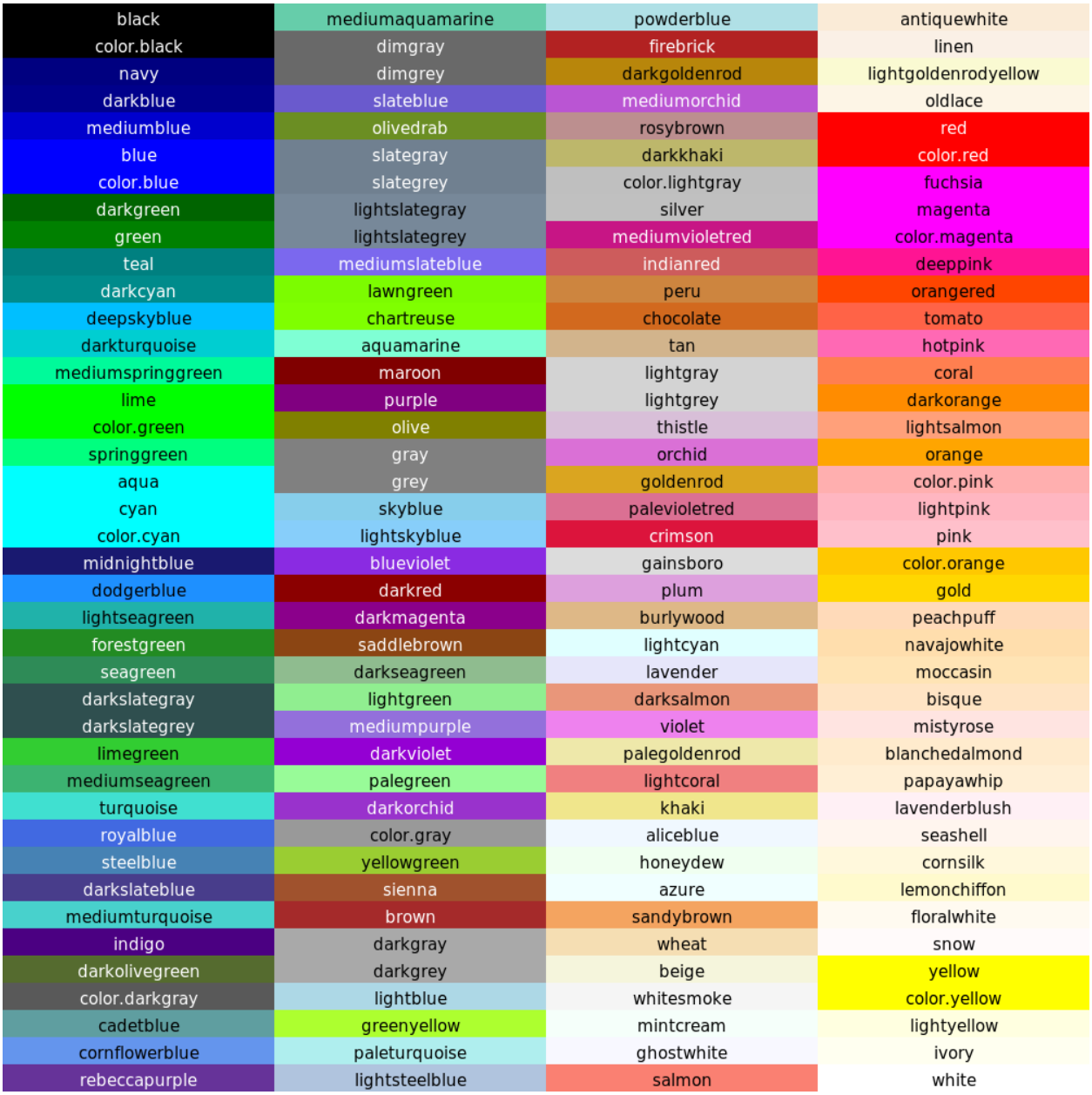


Figure 1: All available named colors in `pgl.py`