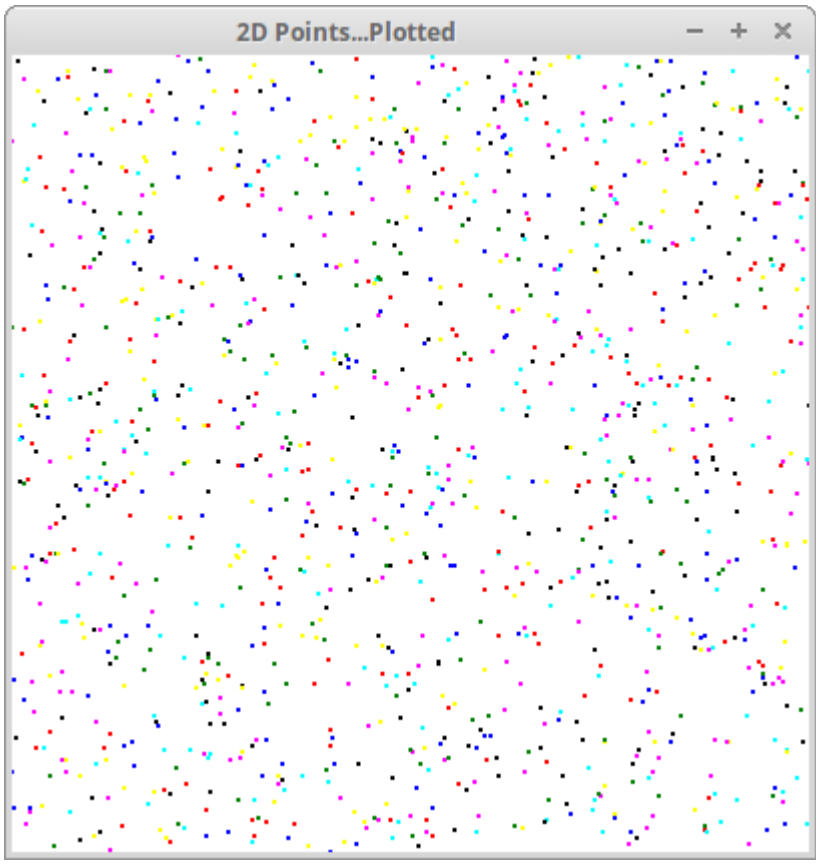
Program: 2D Points...Plotted

Using the template provided on the class web site, your task in this programming assignment is to implement a coordinate system class using the Tkinter Python library that will enable plotting points graphically. Clearly, you will need to re-implement the 2D point class from the previous programming assignment (since you will be plotting instances of points!).

Note that you must not modify the main part of the program, and your output should look somewhat like the following (note that it won't be exactly the same since the points are to be generated randomly):



First, re-implement your 2D point class. Recall the requirements:

- A 2D point is made up of an *x*-component and a *y*-component. Each component is a floating point value;
- A constructor should initialize a point either with specified values for the *x*- and *y*-components or the point (0.0,0.0) as default;
- Instance variables should be appropriately named (i.e., beginning with underscores);
- Accessors and mutators should provide read access of and write access to the instance variables;
- A function named `dist` should take **two points as input** and calculate and **return the floating point distance** between the two points;
- A function named `midpt` should take **two points as input** and calculate and **return the midpoint** of the two points; and

- A *magic* function should provide a string representation of a point in the format `(x,y)`.

Second, implement the coordinate system class. Note the following requirements:

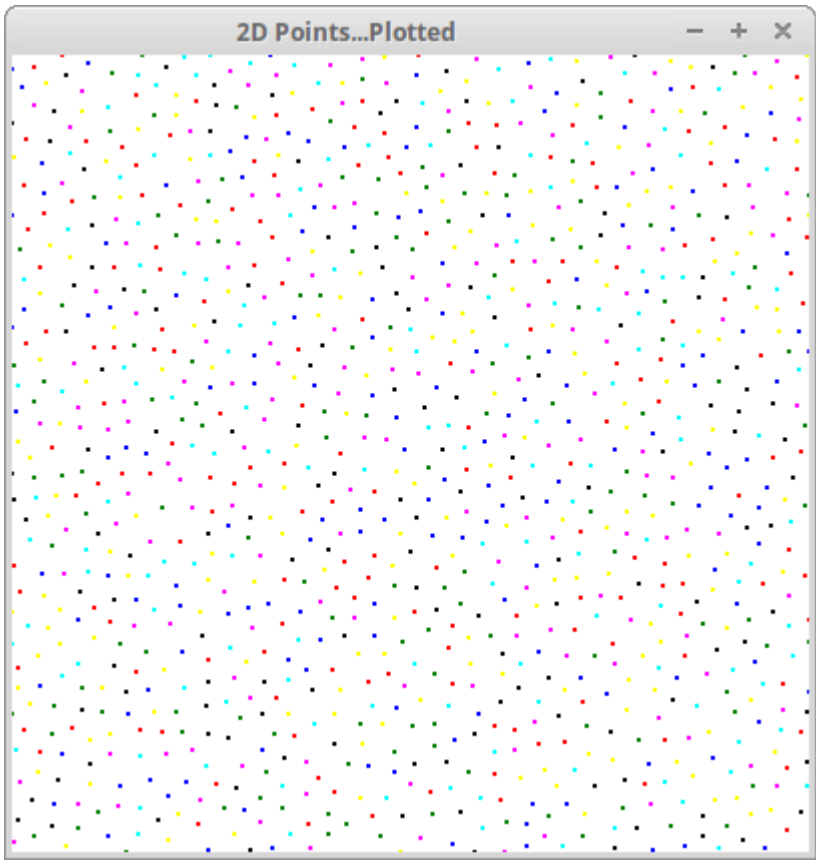
- The coordinate system class must inherit from Tkinter's canvas class and fill the entire window (except for the title bar);
- A function named `plotPoints` should take **the number of points to plot as input** (set to 5,000 by default in the main part of the program), and plot that many points to the canvas;
- Plotted points should be individual instances of your 2D point class, each with random *x*- and *y*-components **that are within the width and height of the canvas** (set to 800x800 by default in the main part of the program);
- Plotted points should have a radius of 0 (i.e., a point is made up of a single pixel);
- Points should be plotted in a random color from the following set of colors: black, red, green, blue, cyan, yellow, and magenta;
- The coordinate system class must include *class* variables for **point radius** and **point colors** – that are referred to as appropriate within the class; and
- You are free to define other functions in the coordinate system class as appropriate (e.g., a `plot` function that plots a single point of a specified radius in a specified color).

To help clarify, here are some specifics and/or constraints:

(1) Make sure that your 2D point class addresses all of the requirements listed above (note that a properly implemented 2D point class in the last programming assignment ensures this);
(2) Make sure that your coordinate system class addresses all of the requirements listed above;
(3) You must include a meaningful header, use good coding style, use meaningful variable names, and comment your source code where appropriate;
(4) Your output should look somewhat like the sample run shown above; and
(5) You must submit your source code as a single .py file.

An observation

Did you notice in the sample output above that there are small clusters of points in addition to gaps where no points exist (i.e., the points are not evenly distributed across the canvas)? Generally, plotting random points does not guarantee an even distribution of the points on the canvas. We can, however, implement what is known as a *Poisson disc* method that encourages points to "spread out" some. This is accomplished by setting a threshold for the minimum distance between points. If a randomly generated point is not at least the threshold distance from every other point, it is discarded – and a new point is randomly generated. Here's sample output with the Poisson disc method implemented:



Of course, this can be computationally intensive – especially when trying to fill the canvas with the maximum possible number of points. That is, once a good number of points have been plotted, it will be unlikely that a randomly generated point is far enough away from every other point on the canvas. It will possibly take many tries to find a valid point. And if too many points are plotted, it may be impossible to find a valid point. This motivates several strategies to attempt to address this:

(1) Pick an appropriate number of points based both on the size of the canvas and the threshold; and
(2) Set a limit on how many times to regenerate a point if one is not far enough away from every other point (i.e., after some number of tries, give up and go to the next point).

An appropriate number of points to generate can be calculated as follows:

$$n = \frac{WIDTH * HEIGHT}{THRESHOLD^2}$$

For an 800x800 canvas and a threshold of 10 pixels, for example, 6,400 points can be plotted. Note, however, that this would take a very long time. Reducing the number of points is recommended.

Similarly, a reasonable threshold can be calculated as follows:

$$t = \sqrt{\frac{WIDTH * HEIGHT}{n}}$$

For an 800x800 canvas on which 1,250 points are to be plotted, for example, a threshold of approximately 22 pixels should work. Again, this is likely to take a long time, and picking a smaller threshold is recommended.

Implementing a limit on how many times a point is regenerated if it is found to be too close to every other point is fairly simple. Simply set up a counter that increments each time a point is found to be invalid. Once a valid point is found, reset the counter and go to the next point. If the counter reaches the limit, skip the current point, reset the counter, and go on to the next point.

If you wish to implement the Poisson disc method, please ensure that the number of points plotted and the threshold are selected to ensure that the execution time of your program does not exceed approximately five seconds on a typical system. **Please note that this enhancement is not required!**