Sample: Python - Fractals

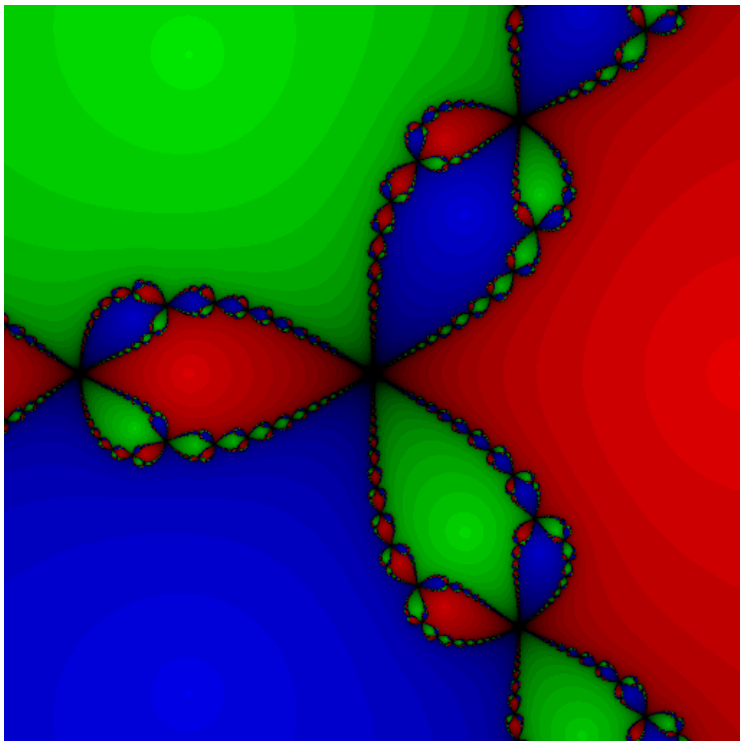
Code Listing

```
#!python3
import tkinter

IMAGE_SIZE = 512                                        # fractal image size
HALF_SIZE = IMAGE_SIZE / 2

CONV_CRITERIA = 10e-3                                   # the convergence criteria
MAX_ITER = 20                                  # the maximum number of iterations
H = 10e-4          # delta z value used to compute the derivative of a function

# Colors for different function roots
colors = ((255, 0,   0), (0,   255,   0), (0,   0, 255),
          (255, 255, 0), (255,   0, 255), (0, 255, 255))


# Converts image point coordinates to the complex plane coordinates
def convert_coords(x, y):
    x = (x - HALF_SIZE) / HALF_SIZE
    y = (y - HALF_SIZE) / HALF_SIZE
    return complex(x, y)


# Finds a root of a complex function given a starting point $z,
# an error bound $err and an upper iteration limit $iterCount
def find_root(f, z, err, iterCount):
    tmp = z + err + 1                                   # stores previous $z value
    iteration = 0                          # stores the number of iterations made

    while (abs(tmp - z) > err) and (iteration < iterCount):
        # Compute function value at $z point
        fz = f(z)
        # Compute the numerical derivative of the function at $z point
        df_dz = (f(z + complex(H, H)) - fz) / complex(H, H)

        # Check if $z point won't converge to any of the roots
        if df_dz == 0:
            return None

        tmp = z                     # save the previous $z complex number value
        z = z - fz / df_dz                              # perform an iteration
        iteration += 1
    return (z, iteration)
```

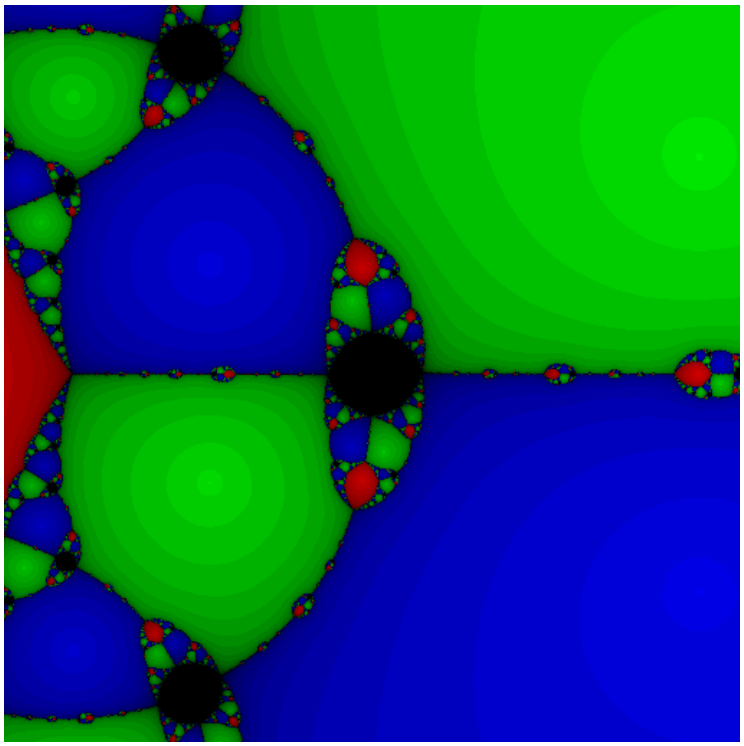
```
# Returns the number of function root
# to which a given $z complex number converges
def get_root_number(z, roots):
    minDistance = abs(z - roots[0])
    rootNumber = 0

    # Find a root which has the minimum distance to $z number
    for k in range(1, len(roots)):
        tmp = abs(z - roots[k])
        if tmp < minDistance:
            minDistance = tmp
            rootNumber = k

    return rootNumber


# Draws a Newton fractal of the specified complex function on a given image
def draw_fractal(fun, roots, image):
    for x in range(IMAGE_SIZE):
        for y in range(IMAGE_SIZE):
            # Find the root using the coordinates of current pixel
            # as the starting point for Newton's method
            res = find_root(fun, convert_coords(x, y), CONV_CRITERIA, MAX_ITER)
            if res == None:
                continue         # do not color the points that do not converge

            # Get the root found and the number of iterations made
            z, i = res
            # Get the color of the exact root the found root converges to
            rootColor = colors[get_root_number(z, roots)]

            # Get color depth
            depth = (MAX_ITER - i) / MAX_ITER
            # Multiply the components of the exact root color by depth
            RGB = tuple(int(depth * a) for a in rootColor)

            # Set the color of the image pixel at (x, y)
            image.put("#{0:02x}{1:02x}{2:02x}".format(*RGB), (x, y))
```

```
# Configures the window to display a fractal in
# Returns an image to draw the fractal on
def configure_window(window, title):
    window.title(title)                                    # set window title

    # Create a Canvas widget
    canvas = tkinter.Canvas(window, width = IMAGE_SIZE, height = IMAGE_SIZE)
    # Create a PhotoImage to draw fractal on
    img = tkinter.PhotoImage(width = IMAGE_SIZE, height = IMAGE_SIZE)

    canvas.create_image((HALF_SIZE, HALF_SIZE), image = img)
    canvas.pack()                              # put canvas widget on the window
    return img


# f(z) = z^3 - 1
def f1(z):
    return z ** 3 - 1
# f1 complex function roots
f1_roots = (1 + 0j, -(1j ** (1/3)), 1j ** (2/3))


# f(z) = z^3 - 2z + 2
def f2(z):
    return z ** 3 - 2 * z + 2
# f2 complex function roots
f2_roots = (-1.76929 + 0j, 0.88465 - 0.58974j, 0.88456 + 0.58974j)


# f(z) = z^5 - 1
def f3(z):
    return z ** 5 - 1
# f3 complex function roots
f3_roots = (-1 + 0j, 1j ** (1/5), -(1j ** (2/5)), 1j ** (3/5), -(1j ** (4/5)))


def main():
    # Create the toplevel window
    window_1 = tkinter.Tk()
    img_1 = configure_window(window_1, "f(z) = z^3 - 1")

    draw_fractal(f1, f1_roots, img_1)                  # draw a fractal for f1
    img_1.write("fractal_1.gif", "gif")              # save the fractal to file
```

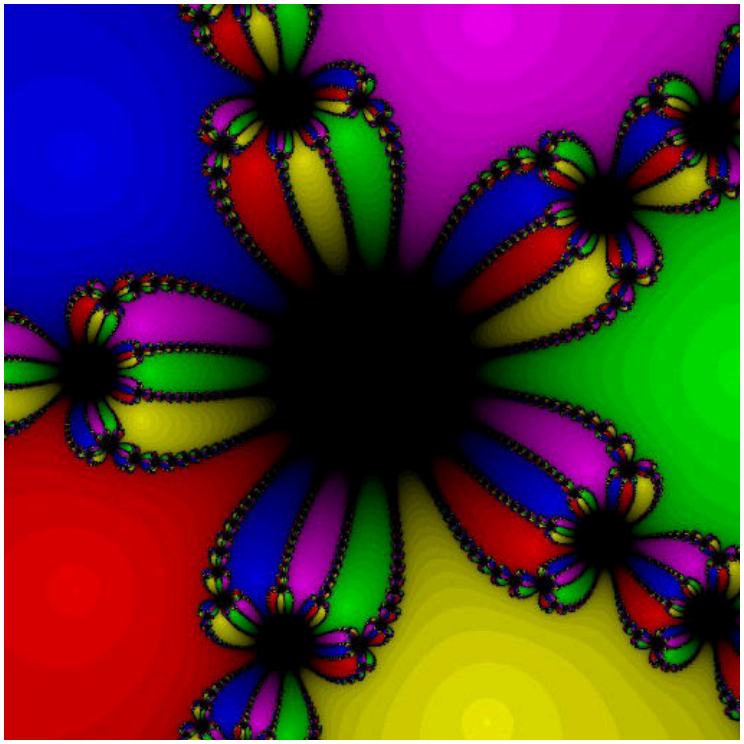
```
    # Create a window to display the second function fractal
    window_2 = tkinter.Toplevel()
    img_2 = configure_window(window_2, "f(z) = z^2 - 2z + 2")

    draw_fractal(f2, f2_roots, img_2)                    # draw a fractal for f2
    img_2.write("fractal_2.gif", "gif")                 # save the fractal to file

    # Create a window for showing the third function fractal
    window_3 = tkinter.Toplevel()
    img_3 = configure_window(window_3, "f(z) = z^5 - 1")

    draw_fractal(f3, f3_roots, img_3)                    # draw a fractal for f3
    img_3.write("fractal_3.gif", "gif")                 # save the fractal to file

    window_1.mainloop()                              # run the GUI application loop

if __name__ == '__main__':
    main()
```



f(z) = z^3 - 1



$f(z) = z^3 - 2z + 2$



$f(z) = z^5 - 1$