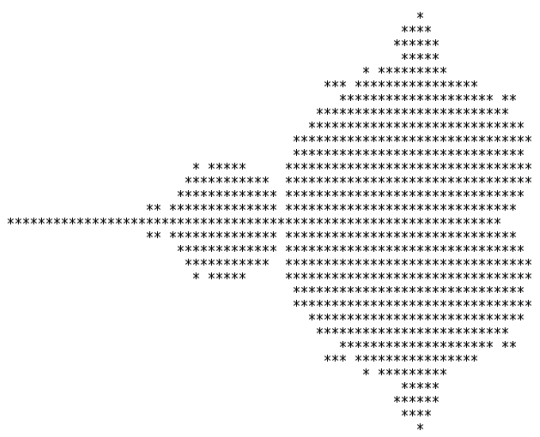
Mandelbrot set with Python

$z_0 = 0$
$z_1 = c$
$z_2 = z_1^2 + c$
...
$z_{n+1} = z_n^2 + c$



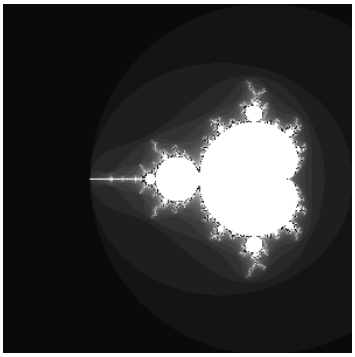
The first published picture of the Mandelbrot set, by Robert W. Brooks and Peter Matelski in 1978, IBM

sample-1 Python

```
import numpy as np
import matplotlib.pyplot as plt
length = 800
roop = 40
threshold = 1.1
image = [[i for i in range(length)] for j in range(length)
def init():
    for i in range(length):
        for j in range(length):
            image[j][i] = mandelbrot((i - (length/2)*1.3)/length*2, (j - (length/2))/length*2)

def mandelbrot(R, C):
    z = 0 + 0j
    _z = 0 + 0j
    C = C * 1j
    for i in range(roop):
        _z = z * z + R + C
        if (np.abs(_z) >= threshold):
            return 0
        z = _z
    return np.abs(_z)

if __name__ == "__main__":
    init()
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    plt.savefig("m1.png")

    refer GitHub, Inc.,  https://github.com/
```

sample-2 Python

```
from PIL import Image
max_iteration = 1000
x_center = -1.0
y_center =  0.0
size = 300
im = Image.new("RGB", (size,size))
for i in xrange(size):
    for j in xrange(size):
        x,y = ( x_center + 4.0*float(i-size/2)/size,
                y_center + 4.0*float(j-size/2)/size
              )
        a,b = (0.0, 0.0)
        iteration = 0
        while (a**2 + b**2 <= 4.0 and iteration < max_iteration):
            a,b = a**2 - b**2 + x, 2*a*b + y
            iteration += 1
        if iteration == max_iteration:
            color_value = 255
        else:
            color_value = iteration*10 % 255
        im.putpixel( (i,j), (color_value, color_value, color_value))
im.save("mandelbrot.png", "PNG")
```



*z*0* goes from -3 to 3 (step 0.1)
GIF: imetrics.co.jp/math3/MandelBrotSet.GIF
imetrics.co.jp/math3/image2017.pptx

Sample-2 C-programming

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdint.h>
int main(int argc, char* argv[])
{
  /* Parse the command line arguments. */
  if (argc != 8) {
    printf("Usage:   %s <xmin> <xmax> <ymin> <ymax> <maxiter> <xres>
<out.ppm>\n", argv[0]);
    printf("Example: %s 0.27085 0.27100 0.004640 0.004810 1000 1024
pic.ppm\n", argv[0]);
    exit(EXIT_FAILURE);
  }
  /* The window in the plane. */
  const double xmin = atof(argv[1]);
  const double xmax = atof(argv[2]);
  const double ymin = atof(argv[3]);
  const double ymax = atof(argv[4]);
  /* Maximum number of iterations, at most 65535. */
  const uint16_t maxiter = (unsigned short)atoi(argv[5]);
  /* Image size, width is given, height is computed. */
  const int xres = atoi(argv[6]);
  const int yres = (xres*(ymax-ymin))/(xmax-xmin);
  /* The output file name */
  const char* filename = argv[7];
  /* Open the file and write the header. */
  FILE * fp = fopen(filename,"wb");
  char *comment="# Mandelbrot set";/* comment should start with # */
  /*write ASCII header to the file*/
  fprintf(fp,
          "P6\n# Mandelbrot, xmin=%lf, xmax=%lf, ymin=%lf, ymax=%lf,
maxiter=%d\n%d\n%d\n%d\n",
          xmin, xmax, ymin, ymax, maxiter, xres, yres, (maxiter < 256 ? 256 :
maxiter));
  /* Precompute pixel width and height. */
  double dx=(xmax-xmin)/xres;
  double dy=(ymax-ymin)/yres;
  double x, y; /* Coordinates of the current point in the complex plane. */
  double u, v; /* Coordinates of the iterated point. */
  int i,j; /* Pixel counters */
  int k; /* Iteration counter */
  for (j = 0; j < yres; j++) {
    y = ymax - j * dy;
```

```
    for(i = 0; i < xres; i++) {
      double u = 0.0;
      double v= 0.0;
      double u2 = u * u;
      double v2 = v*v;
      x = xmin + i * dx;
      /* iterate the point */
      for (k = 1; k < maxiter && (u2 + v2 < 4.0); k++) {
            v = 2 * u * v + y;
            u = u2 - v2 + x;
            u2 = u * u;
            v2 = v * v;
      };
      /* compute  pixel color and write it to file */
      if (k >= maxiter) {
        /* interior */
        const unsigned char black[] = {0, 0, 0, 0, 0, 0};
        fwrite (black, 6, 1, fp);
      }
      else {
        /* exterior */
        unsigned char color[6];
        color[0] = k >> 8;
        color[1] = k & 255;
        color[2] = k >> 8;
        color[3] = k & 255;
        color[4] = k >> 8;
        color[5] = k & 255;
        fwrite(color, 6, 1, fp);
      };
    }
  }
  fclose(fp);
  return 0;
}
```

Usage:

```
  ./mandelbrot <xmin> <xmax> <ymin> <ymax> <maxiter> <xres> <out.ppm>
```

Example:

```
  ./mandelbrot 0.27085 0.27100 0.004640 0.004810 1000 1024 pic.ppm
```

another sample-3 Python encoding

```
Z = zeros((len(Y), len(X)))
return n
 sample-2
 from pylab import *
 from numpy import NaN
  def m(a):
 z=0
 for n in range(1, 100):
 z = z**2 + a
 if abs(z) > 2:
  return NaN
  X = arange(-2, .5, .002)
  Y = arange(-1,  1, .002)
   for iy, y in enumerate(Y):
 print (iy, "of", len(Y))
 for ix, x in enumerate(X):
 Z[iy,ix] = m(x + 1j * y)
  imshow(Z, cmap = plt.cm.prism, interpolation = 'none', extent =
 (X.min(), X.max(), Y.min(), Y.max()))
 xlabel("Re(c)")
 ylabel("Im(c)")
 savefig("mandelbrot_python.svg")
 show()
```

Complex Dynamics. http://imetrics.co.jp/math3/ComplexDynamics.pdf

The **Mandelbrot set** is the set of complex numbers
c for which the function $f(z) = z^2 + c$. does not diverge when iterated from z = 0, i.e., for which the sequence *f(0)*, *f(f(0))*, etc., remains bounded in absolute value. Its definition and name are due to Adrien Douady, in tribute to the mathematician Benoit Mandelbrot. The set is connected to a Julia set, and related Julia sets produce similarly complex fractal shapes.