Coding Example: The Mandelbrot Set (Python approach)

In this lesson, we are going to look at the Python implementation of the Mandelbrot set case study.

WE'LL COVER THE FOLLOWING

Python Implementation

A pure python implementation is written as:

main.py

tools.py

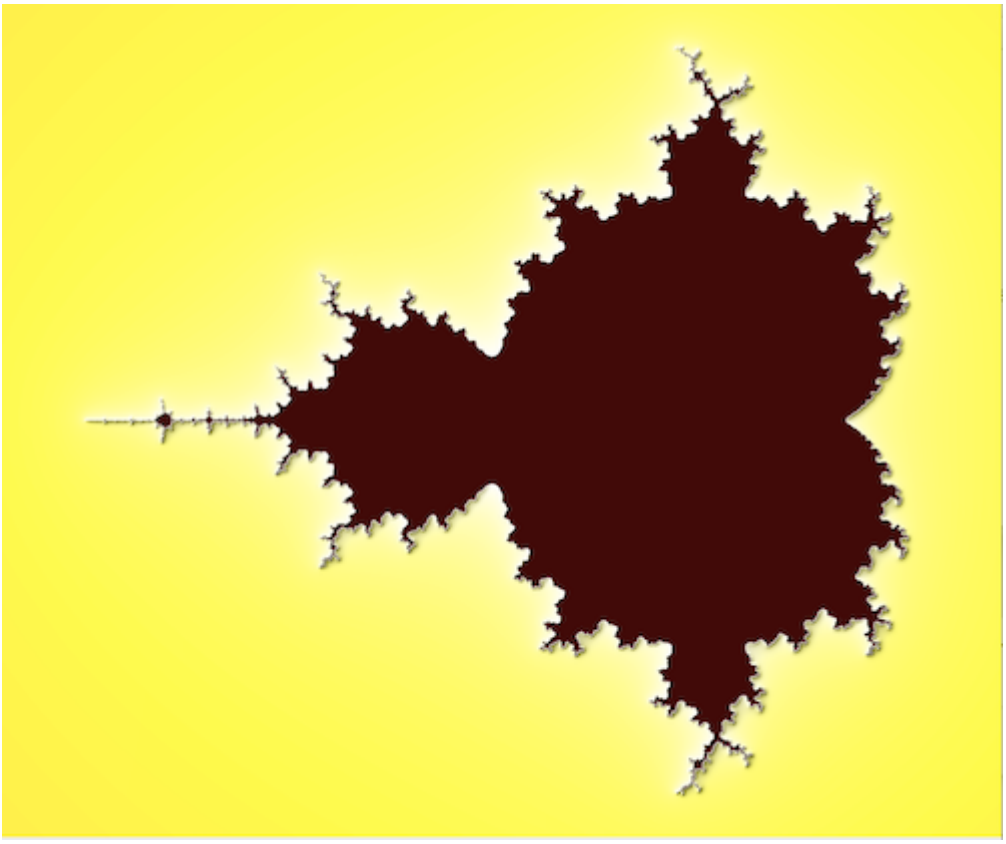
```
# ----------------------------------------------------------------------------
# From Numpy to Python
# Copyright (2017) Nicolas P. Rougier - BSD license
# More information at https://github.com/rougier/numpy-book
# ----------------------------------------------------------------------------
import numpy as np

def mandelbrot_python(xmin, xmax, ymin, ymax, xn, yn, maxiter, horizon=2.0):
  def mandelbrot(z, maxiter):
    c = z
    for n in range(maxiter):
      if abs(z) > horizon:
        return n
      z = z*z + c
    return maxiter
  r1 = [xmin+i*(xmax-xmin)/xn for i in range(xn)]
  r2 = [ymin+i*(ymax-ymin)/yn for i in range(yn)]
  return [mandelbrot(complex(r, i),maxiter) for r in r1 for i in r2]


def mandelbrot(xmin, xmax, ymin, ymax, xn, yn, itermax, horizon=2.0):
    # Adapted from
    # https://thesamovar.wordpress.com/2009/03/22/fast-fractals-with-python-and-numpy/
    Xi, Yi = np.mgrid[0:xn, 0:yn]
    Xi, Yi = Xi.astype(np.uint32), Yi.astype(np.uint32)
```

```
    Xi, Yi = Xi.astype(np.uint32), Yi.astype(np.uint32)
    X = np.linspace(xmin, xmax, xn, dtype=np.float32)[Xi]
    Y = np.linspace(ymin, ymax, yn, dtype=np.float32)[Yi]
    C = X + Y*1j
    N_ = np.zeros(C.shape, dtype=np.uint32)
    Z_ = np.zeros(C.shape, dtype=np.complex64)
    Xi.shape = Yi.shape = C.shape = xn*yn

    Z = np.zeros(C.shape, np.complex64)
    for i in range(itermax):
        if not len(Z): break

        # Compute for relevant points only
        np.multiply(Z, Z, Z)
        np.add(Z, C, Z)

        # Failed convergence
        I = abs(Z) > horizon
        N_[Xi[I], Yi[I]] = i+1
        Z_[Xi[I], Yi[I]] = Z[I]

        # Keep going with those who have not diverged yet
        np.logical_not(I,I)
        Z = Z[I]
        Xi, Yi = Xi[I], Yi[I]
        C = C[I]
    return Z_.T, N_.T


if __name__ == '__main__':
    from matplotlib import colors
    import matplotlib.pyplot as plt
    from tools import timeit

    # Benchmark
    xmin, xmax, xn = -2.25, +0.75, int(3000/3)
    ymin, ymax, yn = -1.25, +1.25, int(2500/3)
    maxiter = 200
    timeit("mandelbrot_python(xmin, xmax, ymin, ymax, xn, yn, maxiter)", globals())

    # Visualization
    xmin, xmax, xn = -2.25, +0.75, int(3000/2)
    ymin, ymax, yn = -1.25, +1.25, int(2500/2)
    maxiter = 20
    horizon = 2.0 ** 40
    log_horizon = np.log(np.log(horizon))/np.log(2)
    Z, N = mandelbrot(xmin, xmax, ymin, ymax, xn, yn, maxiter, horizon)

    # Normalized recount as explained in:
    # http://linas.org/art-gallery/escape/smooth.html
    M = np.nan_to_num(N + 1 - np.log(np.log(abs(Z)))/np.log(2) + log_horizon)

    dpi = 72
    width = 10
    height = 10*yn/xn

    fig = plt.figure(figsize=(width, height), dpi=dpi)
    ax = fig.add_axes([0.0, 0.0, 1.0, 1.0], frameon=False, aspect=1)

    light = colors.LightSource(azdeg=315, altdeg=10)
    plt.imshow(light.shade(M, cmap=plt.cm.hot, vert_exag=1.5,
                           norm = colors.PowerNorm(0.3), blend_mode='hsv'),
               extent=[xmin, xmax, ymin, ymax], interpolation="bicubic")
```

```
    ax.set_xticks([])
    ax.set_yticks([])


    plt.savefig("output/mandelbrot.png")
    plt.show()
```

Output

The output of the above code would look like this:



The interesting (and slow) part of this code is the mandelbrot function that actually computes the sequence $f_c(f_c(f_c\ldots))$. The vectorization of such code is not totally straightforward because the internal return implies a differential processing of the element. Once it has diverged, we don't need to iterate any more and we can safely return the iteration count at divergence. Now the problem is to do the same in NumPy.

In the next lesson, we'll solve this example using the numpy approach.