Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2023

Homework 5

Due Wednesday, February 22

Problem 1

What does the following function compute (in terms of x, y)? Explain why.

```
function fun1(x,y)
    if x == 0
        return y
    else
        return fun1(x - 1, x + y)
    end
end
```

The function computes the sum $y + \frac{x(x+1)}{2}$. This is because when $x = 0$, we get

$$\frac{x(x+1)}{2} = 0$$

and therefore output $y + 0 = y$. If not, then we compute

$$x + (x-1) + (x-2) + \cdots + 0 + y$$

which are the triangle numbers plus y.

Problem 2

What does the following function compute (in terms of a, b)? Explain why.

```
function fun2(a,b)
    if b == 0
        return 1
    elseif b % 2 == 0
        return fun2(a * a, b ÷ 2)
    else
        return fun2(a * a, b ÷ 2) * a
    end
end
```

The function computes a^b. This is because if we express b in base 2 representation and look from right to left at position $p = 0, 1, 2, \ldots, n$ with value

$$V(p) = \begin{cases} 0 & \text{if value at } p \text{ is } 0 \\ 1 & \text{if value at } p \text{ is } 1 \end{cases},$$

we have

$$b = \sum_{p=0}^{n} V(p) 2^p.$$

The indicator function V represents the if else statements for returning the function, where we return

$$V(p)(2^p)$$

This explains what the recursion does, and it will eventually reach the $n - th$ position, returning the product of the values, which is

$$a^{V(n)2^n} a^{V(n-1)2^{n-1}} \cdots a^{V(0)2^0} = a^b$$

Problem 3

Predict the output of the code below (try first without running it):

```
function fun3(x)
    if x > 0
        x -= 1
        fun3(x)
        print(x, " ")
        x -= 1
        fun3(x)
    end
end

fun3(5)
```

The function takes the previous output, prints it, then prints the value $x - 1$, then prints the value before the first one. The base case fun3(1) = 0.

For $n = 2$,

$$\text{fun3(1) 1} = 01$$

For $n = 3$

$$\text{fun3(2) 2 fun3(1)} = 0120$$

For $n = 4$

$$\text{fun3(3) 3 fun3(2)} = 0120301$$

For $n = 5$

$$\text{fun3(4) 4 fun3(3)} = 012030140120$$

In [1]:
```
function fun3(x)
    if x > 0
        x -= 1
        fun3(x)
        print(x, " ")
        x -= 1
        fun3(x)
    end
end

fun3(5)
```

```
0 1 2 0 3 0 1 4 0 1 2 0
```

Problem 4 - Mandelbrot set

The Mandelbrot set is the set of complex numbers $z_0 = C$ such that the quadratic recurrence equation

$$z_{n+1} = z_n^2 + C$$

does not tend to infinity.

To visualize the set, you will:

1. Create a matrix of points C in the complex plane
2. Iterate the recurrence for each point C until $|z_n| > 4$, and count the number of iterations n
3. For the points where the number of iterations exceeds `maxiter`, we will assume that the sequence is convergent and set $n = 0$
4. Visualize the set by an image plot of the n-values

Problem 4(a)

Write a function with the syntax

```
function mkCmatrix(xmin, xmax, ymin, ymax, nx, ny)
```

which computes `nx` equidistributed numbers x_k between `xmin` and `xmax`, `ny` equidistributed numbers y_j between `ymin` and `ymax`, and returns the `ny`-by-`nx` matrix C with complex entries $C_{jk} = x_k + iy_j$.

In [2]:
```
function mkCmatrix(xmin, xmax, ymin, ymax, nx, ny)
    x = [xmin + i*Float64(xmax - xmin)/(nx-1) for i = 0:(nx-1)]
    y = [ymin + i*Float64(ymax - ymin)/(ny-1) for i = 0:(ny-1)]
    C = [x[i] + y[j]im for j = 1:ny, i = 1:nx]
end
```

Out[2]:
```
mkCmatrix (generic function with 1 method)
```

Problem 4(b)

Write a function

```
function mandelbrot_set(C, maxiter)
```

which takes a matrix C as described above and an integer `maxiter` , and returns an integer matrix N of the same size as C containing the iteration counts n as described above.

In [3]:
```
function mandelbrot_set(C, maxiter)


    function recurrence(c, z, iter, maxiter)
        if iter > maxiter
            iter = 0
        elseif abs(z) ≤ 4
            z = z^2 + c
            iter += 1
            return recurrence(c, z, iter, maxiter)
        end
        iter
    end


    N = []
    for a in C
        push!(N, recurrence(a, a, 0, maxiter))
    end


    return reshape(N, size(C,1), size(C,2))
end
```

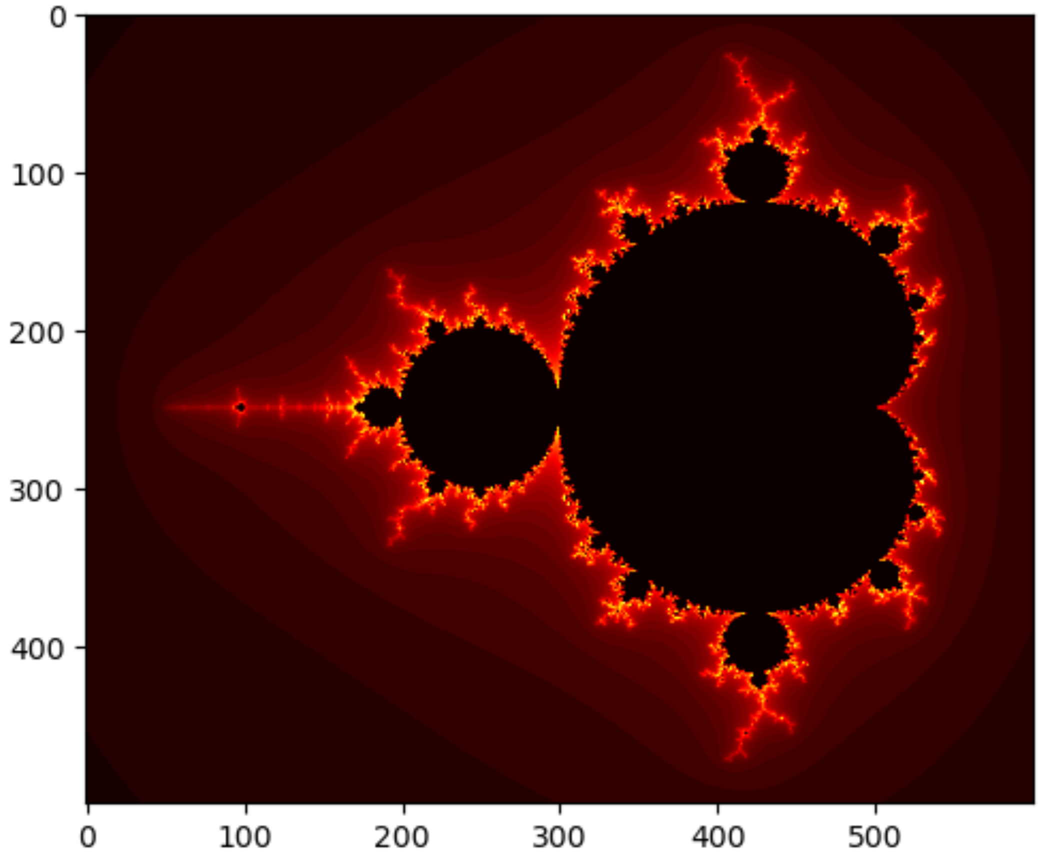
Out[3]:
```
mandelbrot_set (generic function with 1 method)
```

Problem 4(c)

Run the code below to visualize the set.

In [4]:
```
C = mkCmatrix(-2.25, 0.75, -1.25, 1.25, 600, 500)
maxiter = 50
N = mandelbrot_set(C, maxiter)

using PyPlot
imshow(N, cmap=ColorMap("hot"));
```



Problem 5 - Koch curve

A Koch curve between two points (x_1, y_1) and (x_2, y_2) can be defined as follows:

1. If `level` is zero, draw a straight line between the two points
2. Otherwise, define the following 3 additional points

$$
\begin{aligned}
\Delta x &= x2 - x1 \\
\Delta y &= y2 - y1 \\
x_3 &= x1 + \Delta x/3 \\
y_3 &= y1 + \Delta y/3 \\
x_5 &= x1 + 2\Delta x/3 \\
y_5 &= y1 + 2\Delta y/3 \\
x_4 &= (x1 + x2)/2 - \Delta y/2\sqrt{3} \\
y_4 &= (y1 + y2)/2 + \Delta x/2\sqrt{3}
\end{aligned}
$$

3. Draw Koch curves of level `level - 1` between the following pairs of points:

$$
\begin{aligned}
&(x_1, y_1) \text{ to } (x_3, y_3) \\
&(x_3, y_3) \text{ to } (x_4, y_4) \\
&(x_4, y_4) \text{ to } (x_5, y_5) \\
&(x_5, y_5) \text{ to } (x_2, y_2)
\end{aligned}
$$

Problem 5(a)

Write a function

```
function koch_curve(x1, y1, x2, y2, level)
```

which draws a Koch curve as described above.

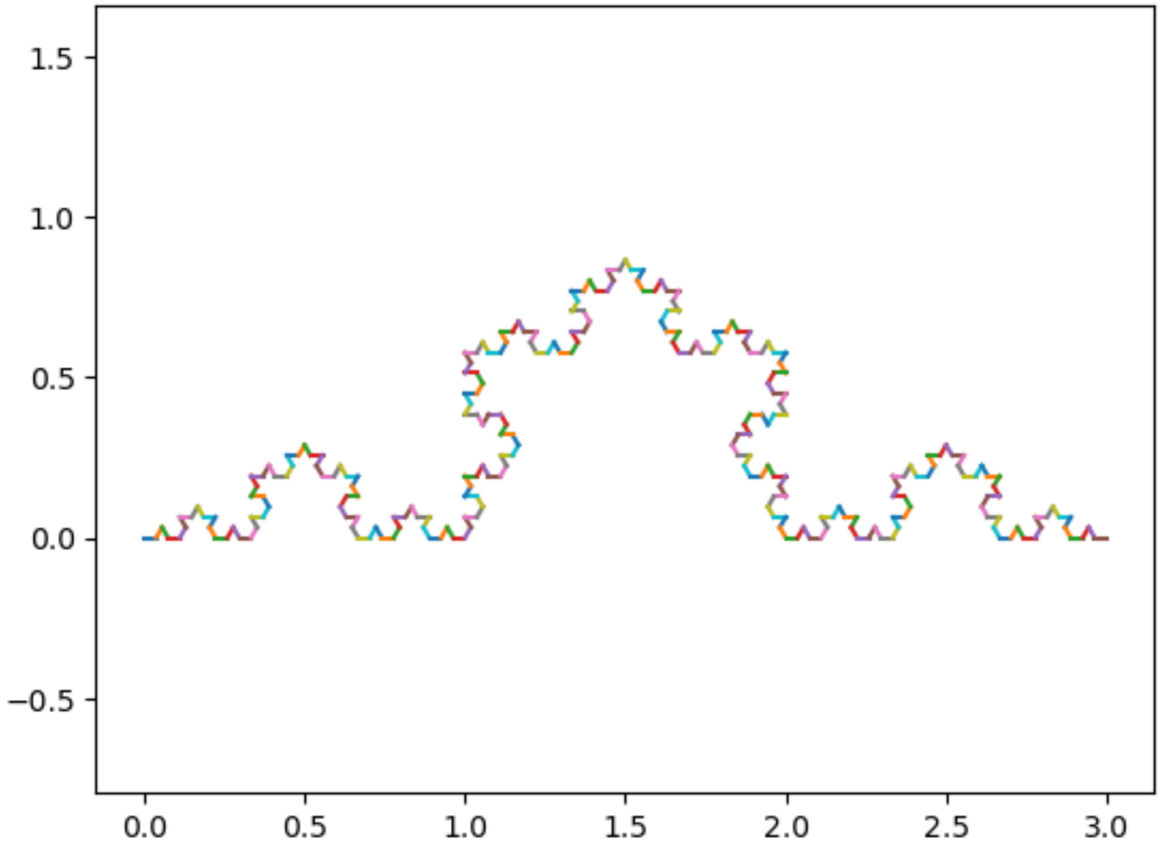
In [5]:
```
using PyPlot
function koch_curve(x1, y1, x2, y2, level)
    if level == 0
        x = [x1, x2]
        y = [y1, y2]
        plot(x, y)
    else
        Δx = x2 - x1
        Δy = y2 - y1
        x3 = x1 + Δx/3
        y3 = y1 + Δy/3
        x5 = x1 + 2*Δx/3
        y5 = y1 + 2*Δy/3
        x4 = (x1 + x2)/2 - Δy/(2*sqrt(3))
        y4 = (y1 + y2)/2 + Δx/(2*sqrt(3))
        koch_curve(x1, y1, x3, y3, level - 1)
        koch_curve(x3, y3, x4, y4, level - 1)
        koch_curve(x4, y4, x5, y5, level - 1)
        koch_curve(x5, y5, x2, y2, level - 1)
    end
end
```

Out[5]:
```
koch_curve (generic function with 1 method)
```

Problem 5(b)

Draw a Koch curve of level 4 between the points $(0, 0)$ and $(3, 0)$. Use `axis("equal")` .

In [6]:
```
koch_curve(0, 0, 3, 0, 4)
axis("equal")
```

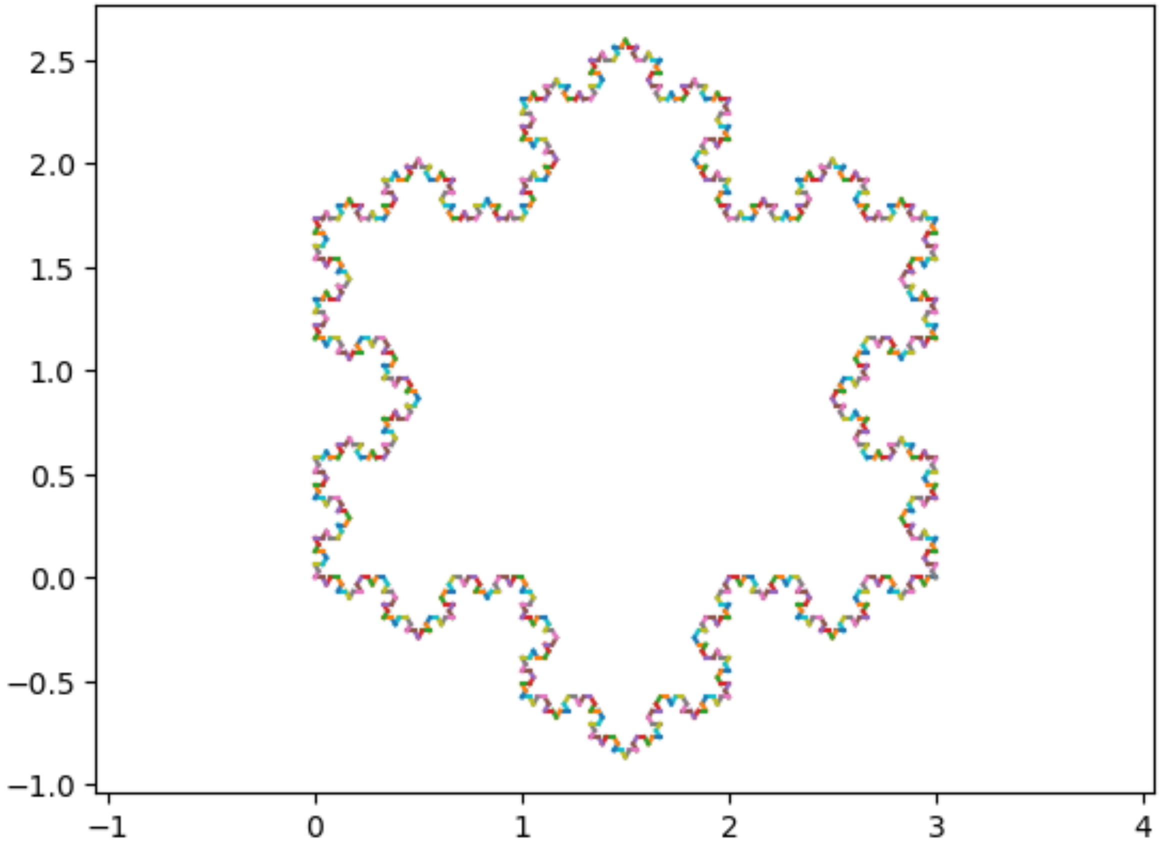


Out[6]:
```
(-0.15000000000000002, 3.15, -0.04330127018922194, 0.9093266739736606
)
```

Problem 5(c)

Draw three Koch curves of level 4 to make the outline of a snowflake. This can be done by generating Koch curves around each edge of an equilateral triangle.

In [7]:
```
koch_curve(3, 0, 0, 0, 4)
koch_curve(0, 0, 1.5, 3/2*sqrt(3), 4)
koch_curve(1.5, 3/2*sqrt(3), 3, 0, 4)
axis("equal")
```



Out[7]:
```
(-0.1500000000000002, 3.1500000000000004, -1.0392304845413265, 2.7712812921102037)
```