HOMEWORK 5

Due Thursday, Feb 23, at 11pm

Please enter your answers into a Jupyter notebook and submit by the deadline via canvas.

A Class for Polynomials. Design a class called `Polynomial` for polynomials in one variable. You should use a list to store the coefficients of the polynomial. I should be able to do the following:

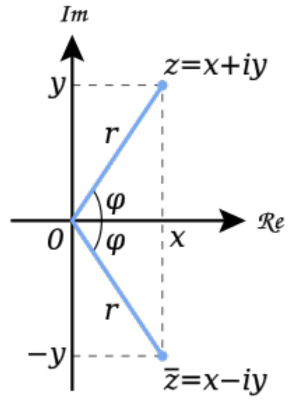
- Initialize my polynomial using a list of its coefficients: e.g. `p = Polynomial([1.0, 2.0, 3.0])`
- Print my polynomial: e.g. `print(p)` for the `p` above should give: `1.0x^0 + 2.0x^1 + 3.0x^2`. You should implement `__repr__(self)` in your class for this, which should return a string. (optional: make sure you print negative coefficients correctly)
- Add polynomials to get a new polynomial. (implement `__add__(self, other)` in your class for this, it should return a new polynomial)
- Evaluate the polynomial at a value x by running `p.eval(x)`. e.g. for the above polynomial, `p.eval(2.0)` should return `17.0`.

Complex Numbers in Python. Complex numbers are numbers of the form $a+bi$ where $a, b \in \mathbb{R}$. The number $i = \sqrt{1}$ is the formal square root of 1, which we pretend exists. So we have $i^2 = -1$, and $i^3 = -i$ and $i^4 = 1$. The rules for addition and multiplication of complex numbers follow the usual algebraic rules. For example:

$$(2+3i) + (1+5i) = 3+8i$$

$$(2+3i)(1+5i) = 2+10i+3i+15i^2 = 2+13i-15 = -13+13i$$

We can represent complex numbers on the plane as follows:



Here the number $z = x + yi$ is represented on the plane. $\bar{z} = x - yi$ is called the complex conjugate of z. We have

$$z\bar{z} = (x + yi)(x - yi) = x^2 - xyi + yxi - y^2i^2 = x^2 + y^2$$

So the norm (i.e. distance to the origin) of a complex number z is $|z| = \sqrt{x^2 + y^2} = \sqrt{z\bar{z}}$.

The angle φ between z and the x axis is called the phase of z. If z has phase φ, then $z = |z|(\cos(\varphi) + i\sin(\varphi))$.

We have the formula

$$e^{i\theta} = \cos\theta + i\sin\theta$$

So, for every complex number z, we have:

$$z = |z|e^{i\,\mathrm{phase(z)}}$$

This leads to the famous formula:

$$e^{i\pi} = -1$$

(math-tattoo anyone?)

Remark:

$$z_1z_2 = |z_1|e^{i\,\mathrm{phase(z_1)}}|z_2|e^{i\,\mathrm{phase(z_2)}} = |z_1||z_2|e^{i(\mathrm{phase}(z_1)+\mathrm{phase}(z_2))}$$

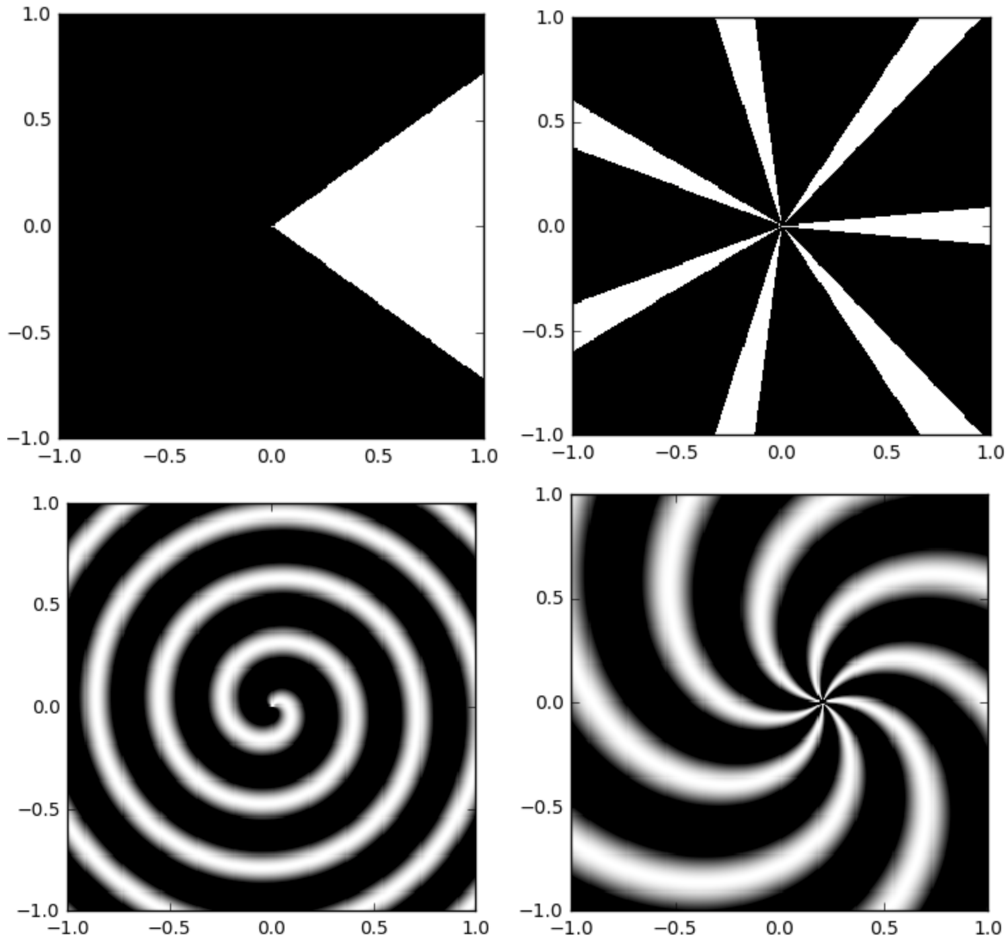
So when you multiply two complex numbers, you multiply the norms, and you add up the angles.

In Python, complex numbers are represented by expressions like `2+3j`. Here, it is important that there is no space between the `3` and the `j`.

If we set `z = 3 + 4j` then we can use `z.real` and `z.complex` to access the real and imaginary parts of `z`, namely `3` and `4` in this case.

- Write a function norm(z) that returns the norm of a complex number z. (as a float. e.g. (`norm(3+4j)` should return `5.0`))
- Understand the code below, and then complete the function below to produce the pictures shown.

```
import libhw01 as libhw
from cmath import phase
import math
from math import pi

# draws a picture for a function f: Complex -> Reals
def drawComplexFunction(f, imsize=300):
    g = lambda x,y: f(x+y*1j)
    libhw.drawfunction(g, imsize=imsize)

def norm(z):
    return math.sqrt(z.real * z.real + z.imag * z.imag)

def f_one(z):
    if # . . . something about phase(z) . . .
        return 1.0
    return 0.0

def f_two(z):
    # modify f_one, hint: take a power of z
    return 0.0

# modify this function to get the one in the third picture
def f_three(z):
    return math.sin(phase(z) + norm(z))

def f_four(z):
    # combine the ideas of f_two and f_three
    # and then figure out a trick to shift the picture slightly in one direction

drawComplexFunction(f_one)
drawComplexFunction(f_two)
drawComplexFunction(f_three)
drawComplexFunction(f_four)
```



The Julia Set in the Complex Plane. The Julia set J_c is defined as follows. Let $f(z) = z^2 + c$. Apply f repeatedly to a complex number z, i.e. take $f(z)$, $f(f(z))$. $f(f(f(z)))$, ... This is an example of a *dynamical system.*

As you do this, a typical point will lead to expenential growth as you keep squaring and adding c (i.e. the norm grows exponentially) . The Julia set J_c is the set of points which don't grow exponentially when you do this.

To compute the Julia set, we will do the following: we will start with z and compute $f(z)$, $f(f(z))$. $f(f(f(z)))$,... $f^k(z)$ for a fixed k, e.g. $k = 100$. If the result has norm $|f^k(z)| < 2$, we will assume z is in the Julia set.

To get a little nicer a picture of the Julia set, we will do the following: $|f^k(z)| < 2$ after k iterations, the pixel will have white value (i.e. your function should output 1), but if $|f^i(z)| > 2$ at the ith iteration and not before, then the value will be i/k, so that it appears more white the closer it is to the actual Julia set.

- Complete the code below to draw the Julia set for $c = 0.28 + 0.008i$.

- Find another c which gives an interesting picture (you will have to do this by trial and error).

```
def julia(c,z):
  k = 100
  # in a loop, compute f^i(z) and see when it moved out of the circle of radius
     2
  # when it leaves, return i/k

def my_julia(z):
    return julia(0.28 + 0.008j, z)

drawComplexFunction(my_julia, imsize=600)    # imsize = 600 for a little better
    resolution
```

The Mandelbrot Fractal. Which c's give interesting Julia sets J_c? One way to analyze that is the following question: for which c's will the sequence $f(0), f(f(0)), \ldots, f^k(0), \ldots$ not grow exponentially? If we draw the answer to this, we get the Mandelbrot fractal. i.e. the Mandelbrot fractal is the set of points in the complex plane for which $f(0), f(f(0)), \ldots, f^k(0), \ldots$ does not grow exponentially.

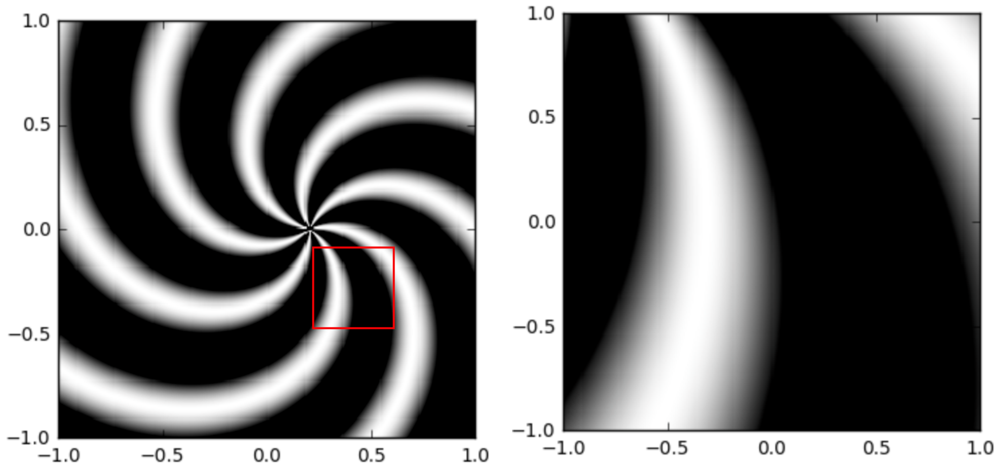
- Complete the code below to draw the Mandelbrot fractal.

```
def mandel(c):
    k = 100
    # . . . should return 1.0 if c is in the Mandelbrot set, and 0.0 otherwise .
         . .

drawComplexFunction(mandel)
```

For values c closer to the edge of the Mandelbrot set, you get the most interesting Julia sets J_c. Using this, you can explore more nice values of c for the previous question (optional).

Zooming in/out. We drew these fractals but we would like to frame them nicely and/or zoom into them. At the moment, `drawComplexFunction(mandel)` has the top left corner at $-1+i$ and the bottom-right at $1-i$. Write a function `reframe(f, z_top_left, z_bottom_right)` which takes a function $f : \mathbb{C} \to \mathbb{R}$ and zooms into the square whose top left corner is `z_top_left` and and bottom-right corner is `z_bottom_right`

For example `drawComplexFunction(reframe(f_four, 0.2 - 0.1j, 0.6 - 0.5j))` should give the picture on the right.



```
def reframe(f, z_1, z_2):
    def g(z):
        new_z = ???
        return f( new_z )
    return g
```

Hint: There are two ways to do this. The first is to get the real and imaginary parts and solve it as a problem for functions $\mathbb{R}^2 \to \mathbb{R}$. The other, more difficult way is to solve the problem for for real numbers and intervals and then generalize to complex numbers.

- Use the reframe function you wrote to zoom in (a lot, e.g. 100x, i.e. your top left and bottom right should differ by a complex number of norm around 0.01) to a nice looking part of the Mandelbrot or Julia fractals (you may need to increase k to get enough detail).