

# CS 172 - Programming Assignment 2

Mark Boady and Matthew Burlick - Drexel University

April 16, 2018

## 1 Overview

This project is about operator overloading. We will create a class for **Complex Numbers**. By overloading common operators like addition, we can make our class work just like the built in data types.

Once the class is built, we can use it to do computations with complex numbers. In particular we'll show that given complex number  $c$  and a positive integer  $k$ , that the  $\sum_{n=0}^k c^n = \frac{1-c^{k+1}}{1-c}$ .

## Contents

1 Overview	1
2 Complex Numbers	1
3 Using our Complex Class and Command Line Testing	3
3.1 Command-Line Arguments . . . . .	3
3.2 Your main program . . . . .	3
4 Grading	4
5 Resources	5

## 2 Complex Numbers

A **Complex Number** is a pair of two floating point numbers. One value is the real part and one is the imaginary part. The imaginary part of the number is multiplied by  $i$ .

A **Complex Number** has the form


$$x + yi \tag{1}$$

In our project,  $x$  and  $y$  will be floating point numbers.

Some example **Complex Numbers** are

$x + yi$ has the values	$(x, y)$
$2 + 5i$ has the values	$(2, 5)$
$1.92 - 7.86i$ has the values	$(1.92, -7.86)$
$28$ has the values	$(28, 0)$
$i$ has the values	$(0, 1)$

Create your own **Complex Number** class in the file **complex.py**. Your class **must** implement the following methods. You may add additional helper methods.

- Constructor `--init--(self,x,y)`
- String Method `--str--(self)`
- Accessor for Real Part `getReal(self)`
- Accessor for Imaginary part `getImaginary(self)`
- Addition `--add--(self,other)` 
- Subtraction `--sub--(self,other)`
- Multiplication `--mul--(self,other)`
- Division `--truediv--(self,other)`
- Exponent `--pow--(self,other)`

The formula for each operation (other than exponent) is given below.

$$\text{Addition : } (a + bi) + (x + yi) = (a + x) + (b + y)i \quad (2) \quad \img alt="speech bubble icon" data-bbox="688 638 718 660"/>$$

$$\text{Subtraction : } (a + bi) - (x + yi) = (a - x) + (b - y)i \quad (3)$$

$$\begin{aligned} \text{Multiplication : } (a + bi) * (x + yi) &= ax + ayi + bxi + byi^2 \\ &= (ax - by) + (ay + bx)i \end{aligned} \quad (4)$$

$$\text{Division : } (a + bi)/(x + yi) = \left(\frac{ax + by}{x^2 + y^2}\right) + \left(\frac{-ay + bx}{x^2 + y^2}\right)i \quad (5)$$

In your **complex.py** add the following code to test your complex number class:

```

if __name__=="__main__":
    c1 = ComplexNumber(4,3)
    c2 = ComplexNumber(2,8)
    print(c1)           #prints 4+3i
    print(c1+c2)        #prints 6+11i
    print(c1*c2)        #prints -16+38i
    print(c1/c2)        #prints 0.47 - 0.38i
    print(c1**3)        #prints -44+117i

```

Now you should be able to run this script directly to see if you get the desired results!

### 3 Using our Complex Class and Command Line Testing

Write a script called **hw2.py**. In this script write two functions,  $a(c, k)$  and  $b(c, k)$ , where  $c$  is a Complex Number object and  $k$  is an integer. These functions are defined as:

- $a(c, k) = \sum_{n=0}^k c^n$
- $b(c, k) = \frac{1-c^{k+1}}{1-c}$

We want to prove that  $a(c, k) = b(c, k)$

To run our program, instead of getting in the parameters for  $c$  and  $k$  interactively from a user, we're going to use a different approach: *command-line arguments*.

#### 3.1 Command-Line Arguments

When calling your script from the command line, you may optionally put additional information after the script name. For example **python hw2.py 2 4 4**. If your script *imports* the **sys** module, then a variable **sys.argv** (argument vector) will be available to you. This variable contains a *list* of the things entered in the command line after the word *python*. So in this example `sys.argv[0]` is "hw2.py", `sys.argv[1]` is "4", etc..

#### 3.2 Your main program

Your program should run and output the values computed by your functions  $a(c, k)$  and  $b(c, k)$  given parameters provided by via the command line. The first two additional parameters will be the real and imaginary components of  $c$ , respectively. The last parameter will be  $k$ . For instance **python zn.py 1 3 20** would allow us to construct a complex number with real component 1 and imaginary component 3 and to obtain  $k = 20$ .

Here is an example output for the command line *python hw2.py 2 3 10*:

```
For complex number 2.0+3.0i and k=10:  
a(2.0+3.0i,10) = -419378.0 - 57777.0i  
b(2.0+3.0i,10) = -419378.0 - 57777.0i
```

## 4 Grading

You will be graded on both the functionality and quality of your design.

- Complex Number Class (50 points)
  - Constructor Method (5 points)
  - String Method (5 points)
  - Attribute names are “mangled” (5 points)
  - Accessor Methods (5 points)
  - Add Method (5 points)
  - Subtract Method (5 points)
  - Multiply Method (5 points)
  - Divide Method (5 points)
  - Exponent Method (5 points)
  - `__main__` Section (5 points)
- Complex Geometric Sum (40 points)
  - a function (15 points)
  - b function (15 points)
  - Command line argument parsing (10 points)
- Well Formatted Output (1 point)
- File is well commented (3 points)
- Name in Comments (2 points, 1 for each file)
- Section Number in Comments (2 points, 1 for each file)
- `complex.py` named correctly (1 point)
- `hw2.py` named correctly (1 point)

If your code has any runtime errors, a 50 point deduction will be taken. Only portions of the code that execute without errors will be graded.

## 5 Resources

Additional Resources

<http://www.math.wisc.edu/~waleffe/M321/complex.pdf>