# Assignment 3: Modeling with Classes: Complex and Polynomials

EC602 Design by Software

Fall 2021

## Contents

## 1 Introduction

### 1.1 Assignment Goals

The assignment goals are to

- illustrate the advantages of modeling data using classes
- introduce complex numbers
- introduce python's class mechanism

For this assignment, the maximum group size is 2.

### 1.2 Due Date

This assignment is due 2021-10-7 at 11:59.

Late assignments will be accepted through a grace period of $H = 26$ hours.

The grade will be scaled on a linear scale from 100% at the deadline to 50% at the end of the grade period.

If the *natural grade* on the assignment is $g$, the number of hours late is $h$, and the grace period is $H$, then the grade is

```
grade = (h > H) ? 0 : g * (1- h/(2*H));
```

in C++ or

```
grade = 0 if h>H else g * (1- h/(2*H))
```

in python.

If the same assignment is submitted ontime *and* late, the grade for that component will be the maximum of the ontime submission grade and the scaled late submission grade.

### 1.3 Submission Link

Assignment Three Submission Page

Please use the checker locally before submitting.

### 1.4 Points

This assignment is worth 5 points.

## 2 Background

### 2.1 Polynomials

The standard way to write a polynomial in $x$ containing only positive powers of $x$ is

$$p(x) = \sum_{i=0}^{N} c_i x^i = c_N x^N + c_{N-1} x^{N-1} + \cdots + c_1 x^1 + c_0$$

If negative powers of $x$ are involved, then you can write

$$p(x) = \sum_{i=-M}^{N} c_i x^i = c_N x^N + c_{N-1} x^{N-1} + \cdots + c_1 x^1 + c_0 + c_{-1} x^{-1} + \cdots + c_{-M} x^{-M}$$

The coefficents can be integer, real, or complex. The two most common cases we will see are integer and real. The polynomial variable $x$ can also be integer, real, or complex (there are also more general types of polynomials on other algebraic systems).

## 2.2 Representing Numbers as Polynomials.

When we write an integer, say 451, or a real number, say 31.4159, we are actually using a shorthand for a polynomial in 10, as follows:

$$451 = 4x^2 + 5x^1 + 1x^0$$

where $x = 10$

and

$$31.4159 = 3x + 1 + 4x^{-1} + 1x^{-2} + 5x^{-3} + 9x^{-4}$$

where $x = 10$ again.

For convenience, to establish uniqueness, and by convention, when we write a number this way we also restrict the coefficients to be *integers* in the range

$$0 \leq c < x$$

Numbers can also be written in base 2 (binary), which is important because computer memory is binary.

For example, 16.125 can be represented as

$$16.125 = 1x^4 + 1x^{-3}$$

where $x = 2$

and so $16.125 = 1000.001_2$

## 2.3 Sequences in Python

A sequence is a generic term for a data type that has a "length" and so can be considered "one-dimensional."

In python, the primary sequence types are

- `str`: immutable sequence of characters. Each character is also a `str` of length 1.
- `list`: a mutable sequence with an `append` method.
- `tuple`: an immutable sequence

The key property of a sequence is that the data it holds can be looked up by *location* or *index*, so

```
>>> Apples = [6,8,9]  # create a 3-element list called Apples
>>> print(Apples[1])  # printout the middle value of Apples
8
```

The items in a sequence of length `N` are numbered 0, 1, 2, and so on up to `N-1`

Note that `list` and `tuple` can both hold any data type at each element, and these can be different.

For example,

```
Fridge = ['eggs',6,(1,2)]
```

is a list with a string, integer, and tuple as its elements.

## 2.4 Modeling data with Classes

The idea of a `class` is useful not only as a way of structuring code, but as a way of thinking about:

- modeling
- abstraction
- defining specifications
- defining expectations
- defining requirements
- organization
- relationships between parts of a larger system

More concretely, we can use classes to

- organize and collect related information
- define methods that operate on the object
- allow for integration into the language using operators.

In Python's standard library most code is provided in the form of families of related classes.

## 2.5 Classes and Objects

A class is like a type.

An object is like a variable.

In python, everything is an object.

## 2.6 Representing polynomials in python

We could represent a polynomial in $x$ using a `list`. The value of the polynomial $x$ is not stored, but instead is implicit.

Suppose the size of the list `v` is `N`, then the polynomial represented is

$$v[N-1]x^{N-1} + v[N-2]x^{N-2} + \cdots v[1]x^1 + v[0]$$

However, our model for a polynomial (using a list) has several issues

- the ordering needs to be understood by users of the data structure
- there is no way to represent negative powers of x
- inefficient in the case of sparse polynomials like $2x^{10001} + 1$
- functions are all separate and un-organized.
- can't type C = A + B , or D = A * B

## 2.7 Example: Complex Numbers

In python, the built in class `complex` models complex numbers.

You may use `j` as the imaginary number (not `i`), like this:

```
c = 3 + 4j
```

or

```
d = 4.3 + 4.2j
```

or define using the initializer

```
f = complex(3,4)
```

### 2.7.1 The `Complex` class.

*Read this section (and the code provided) carefully and fully: it will be necessary to complete the assignment.*

Here is a python re-implementation of the built-in `complex` class.

The new class is called `Complex`.

The functions starting with two underscores `__` are special names that allow python to map operations to them. So, for example, if `z1` and `z2` are of type `Complex`, then

```
total = z1 + z2
```

is equivalent to

```
total = z1.__add__(z2)
```

and is also equivalent to

```
total = Complex.__add__(z1,z2)
```

Some people call this *syntactic sugar* since it makes the code *sweeter*.

Two underscores is pronounced *dunder* in the python world, short for *Double UNDERscore*.

Note that the class `Complex` is incomplete. Some operations have not been implemented.

```
class Complex():
    "Complex(real,[imag]) -> a complex number"

    def __init__(self,real=0,imag=0):
        self.real=real
        self.imag=imag

    def __abs__(self):
        "abs(self)"
        return (self.real**2 +self.imag**2)**0.5

    def __add__(self,value):
        "Return self+value."
        if hasattr(value,'imag'):
            return Complex(self.real+value.real,self.imag+value.imag)
        else:
            return Complex(self.real+value,self.imag)

    def __mul__(s,v):
        "Return s*v."
        if hasattr(v,'imag'):
            x = s.real * v.real - s.imag * v.imag
            y = s.real * v.imag + v.real * s.imag
            return Complex(x,y)
        else:
            return Complex(v*s.real,v*s.imag)

    def __rmul__(s,v):
        "Return s*v"
        if hasattr(v,'imag'):
            x = s.real * v.real - s.imag * v.imag
            y = s.real * v.imag + v.real * s.imag
            return Complex(x,y)
        else:
            return Complex(v*s.real,v*s.imag)

    def __radd__(self,value):
        "Return self+value"
        if hasattr(value,'imag'):
            return Complex(self.real+value.real,self.imag+value.imag)
        else:
            return Complex(self.real+value,self.imag)

    def __str__(self):
        if self.real==0:
            return "{}j".format(self.imag)
        else:
            sign="-" if self.imag<0 else "+"
            return "({}{}{}j)".format(self.real,sign,abs(self.imag))

    def __repr__(self):
        return str(self)

    def __pow__(self,value):
        "Return self ** value"
        raise NotImplementedError('not done yet')
```

The full program along with some test code is here: model_complex.py

Another test program is provided here: fancy_tester.py

## 3 The assignment

Write a python program `model_poly.py` that implements polynomials by defining a class Polynomial.

*You may not import any modules.*

Here are the requirements:

- implement a constructor which takes a sequence and assigns the coefficients in the natural (descending order). So `Polynomial([4,-9,5.6])` should make $4x^2 - 9x + 5.6$
- implement addition and subtraction of polynomials using `+` and `-`
- implement multiplication of polynomials using `*`
- implement testing for equality of polynomials using `==`
- implement an efficient mechanism for handling sparse polynomials
- implement negative powers in the polynomial, i.e. you should be able to handle $p(x) = x^{-1}$
- implement evaluation of the polynomial using a `eval` method, like this: p.eval(2.1)
- implement accessing and modifying the coefficients using `[]`. So `p[2]` should be the coefficient of $x^2$ and `p[8] = 12` should set the coefficent of $x^8$ to 12.
- implement a derivative method `p.deriv()` which returns the derivative of the polynomial.

During testing, the coefficients and variable `x` may be integers, float, or complex. All three cases should work.

Your program should be *importable* which means it does absolutely nothing when imported except define the class `Polynomial`.

Here is the basic template for doing this: modeling_template.py shown below:

```
class Polynomial():
    pass

def main():
    pass

if __name__=="__main__":
    main()
```

Your program should be called `model_poly.py`

When we test your code, it will be done like this:

```
from model_poly import Polynomial
```

and so no code gets executed except that the class `Polynomial` will be defined.

## 3.1 Assignment Checker

You can check your work prior to submission using the assignment checker, which is a python program.

- hw3_ec602_poly_checker.py

The current version of the checker is 1.0, and it uses version 3.1 of curl_grading.py

which it will automatically download if necessary.

## 3.2 Notes and Hints

### 3.2.1 Output and testing

You do not need to implement any output facilities for your class, but it will be very difficult for you to test your own code if you do not.

Furthermore, the checker will use the functions `str` and `repr` (which you can implement with `__str__` and `__repr__`) to provide you with useful information about what is or is not working.

### 3.2.2 Dealing with negative polynomials.

The constructor does not provide any mechanism for creating a Polynomial with negative exponents. This can only be done using `[]`, like this

```
q = Polynomial( [] )
q[-2] = 5.1
```

The first line creates `q`, a Polynomial with no values. Hence $q(x) = 0$. The second line sets the coefficient of $x^{-2}$ to `5.1`, so $q(x) = 5.1x^{-2}$

### 3.2.3 Dealing with missing exponents.

Using the constructor, you must specify missing coefficients by adding zeros to the sequence. For example, to construct $3.1x^2$, you could use

```
p = Polynomial([3.1, 0, 0])
```

or

```
p = Polynomial([])
p[2] = 3.1
```

### 3.2.4 About sequences

In python, a *sequence* is any collection that has elements that can be accessed by index starting at 0 all the way up to N-1 where N is the length of the list.

Examples of sequences are

- string
- list
- tuple
- numpy.ndarray

If `s` is a sequence, then `s[i]` is the element at position `i`, the valid values for `i` being 0 through `N-1`. Negative numbers are also allowed, they specify count from the end. So `s[-1]` is the last element in the sequence.

The input to the constructor of `Polynomial` must be a sequence with numeric values.