Object-Oriented Programming
50:198:113 (Spring 2017)

Homework Assignment 5

The assignment is due by 11:59PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student’s own work. Assistance should be sought or accepted only from the course instructor. Any violation of this rule will be dealt with harshly.

This assignment requires you to implement a class for polynomial equations and a class for quadratic equations as a subclass of polynomials. As usual, you are graded not only on the correctness of the code, but also on clarity and readability. I will deduct points for not following the guidelines for your class design, poor indentation, poor choice of object names, and lack of documentation.

Please read the submission guidelines at the end of this document before you start your work.

Problem 1 [60 points] | Polynomial objects. In this problem, you are asked to implement a class for polynomials. Please read the description carefully, as it will provide details about class methods and instance attributes.

A polynomial is an expression of the form

\[ c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_n x^n \]

where \( c_0, c_1, \ldots, c_n \) are real numbers. Each of \( c_i x^i \) is called a term of the polynomial, and \( c_i \) is called the coefficient of the term with exponent \( i \). Note that \( c_0 \) is simply the coefficient of the term with exponent 0. The maximum exponent with a non-zero coefficient is called the degree of the polynomial. For example, \( 6x^{14} + 9x^{11} - 12x^3 + 42 \) is a polynomial of degree 14. The polynomial \(-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9\) has degree 6. The following are standard operations on polynomials:

Scaling a polynomial: Given a polynomial \( p(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0 \) and a real value \( s \), scaling \( p(x) \) by \( s \) gives the polynomial \( s \cdot p(x) \) obtained by scaling the coefficient of every term in \( p(x) \) by the factor \( s \). For example, if \( p(x) = 6x^{14} + 9x^{11} - 12x^3 + 42 \), then \( 2p(x) = 12x^{14} + 18x^{11} - 24x^3 + 84 \).

Sum of two polynomials: The sum of two polynomials \( p_1(x) \) and \( p_2(x) \), denoted by \( p_1(x) + p_2(x) \), is the polynomial obtained by adding the terms of \( p_1(x) \) and \( p_2(x) \). For example, if \( p_1(x) = 6x^5 - 4x^3 + 10x^2 + 4 \) and \( p_2(x) = 3x^9 - 5x^7 - 3x^2 + 8x \), then \( p_1(x) + p_2(x) = 3x^9 - 5x^7 + 6x^5 - 4x^3 + 7x^2 + 8x + 4 \).

Difference of two polynomials: The difference of two polynomials \( p_1(x) \) and \( p_2(x) \) is the polynomial obtained by subtracting one from the other. Therefore, \( p_1(x) - p_2(x) \) is the polynomial obtained by subtracting the terms of \( p_2(x) \) from \( p_1(x) \). For example, if \( p_1(x) = 6x^5 - 4x^3 + 10x^2 + 4 \) and \( p_2(x) = 3x^9 - 5x^7 - 3x^2 + 8x \), then \( p_1(x) - p_2(x) = -3x^9 + 5x^7 + 6x^5 - 4x^3 + 13x^2 - 8x + 4 \).
Product of two polynomials: The product of two polynomials \( p_1(x) \) and \( p_2(x) \), denoted by \( p_1(x) \cdot p_2(x) \), is the polynomial obtained by the pair-wise multiplication of terms in \( p_1(x) \) and \( p_2(x) \). For example, if \( p_1(x) = x^4 + 4x^3 + 4x^2 \) and \( p_2(x) = 2x - 1 \), then \( p_1(x) \cdot p_2(x) = 2x^5 + 7x^4 + 4x^3 - 4x^2 \). This is obtained as follows: \((x^4 + 4x^3 + 4x^2)(2x) + (x^4 + 4x^3 + 4x^2)(-1) = 2x^5 + 8x^4 + 8x^3 - x^4 - 4x^3 - 4x^2 = 2x^5 + 7x^4 + 4x^3 - 4x^2 \).

In this program, you are asked to implement a `Polynomial` class. The main instance attribute for an object of this class is a list that stores the coefficients of the polynomial. A polynomial of degree \( d \) is stored in a list of length \( d + 1 \). For each term \( c_i x^i \), coefficient \( c_i \) is stored at index \( i \) in the list. Note that if a term of exponent \( i \) does not exist in the polynomial, its coefficient is 0 and hence the element at index \( i \) in that case will be 0.

Implement the following methods for the `Polynomial` class. Your implementation should appear in a module called `poly.py`. A test module called `test_poly.py` is also provided, which imports your `poly.py` module. When you are ready, test your implementation of the `Polynomial` class by typing `python3 test_poly.py`.

1. `__init__`: The parameters to the constructor are a number of term pairs, where each term pair is a 2-tuple of the form (coefficient, exponent). Note that the number of such pairs is unspecified, since a polynomial may have any number of terms. Hence, the `def` statement for the constructor will be as follows:

   ```python
def __init__(self, *termpairs):
```

   For example, to create a `Polynomial` instance for the polynomial \( 6x^{14} + 9x^{11} - 12x^3 + 42 \), we would do the following:

   ```python
P = Polynomial((6, 14), (9, 11), (-12, 3), (42, 0))
```

   Note that the term pairs can be specified in any order. Hence, `Polynomial((6, 14), (9, 11), (-12, 3), (42, 0))` and `Polynomial((42, 0), (6, 14), (-12, 3), (9, 11))` are the same polynomial. The constructor should create an instance attribute called `coeffs`, which is a list that stores coefficients of terms in the polynomial as described above.

2. `_str_`: This method returns a printable version of a polynomial, which is a “nice” string representation of the polynomial. This means that only terms with non-zero coefficients must appear in the string, and the terms must appear from largest exponent to smallest. Furthermore, if the coefficient of a term is negative, the - (minus sign) must be embedded in the string.

3. `_repr_`: This method returns a meaningful string representation of the polynomial. For example, this could be the same string returned by the `_str_` method.

4. `degree`: This method returns the degree of the polynomial.

5. `evaluate`: This method returns the result of evaluating the polynomial at a real value \( x = X \). For example, if \( P \) is the polynomial \(-12x^6 + 5x^5 - 20x^4 + 8x^2 - 12x + 9 \), then \( P.evaluate(2) \) should return -911 (which we obtain by plugging in 2 as the value of \( x \)). This method has two parameters: `self` and \( X \) (the real value at which the polynomial is to be evaluated).

6. `addterm`: This method adds a term with coefficient `coeff` and exponent `exp` to the polynomial. If a term with this exponent already exists, the new term is simply added
on to it. For example, if \( P \) is the polynomial \(-4x^3 + 13x^2 + 4\), then \( P.addterm(6, 5) \) adds
the term \( 6x^5 \) to \( P \), so that \( P \) is now \( 6x^5 - 4x^3 + 13x^2 + 4 \). Subsequently, \( P.addterm(-3, 2) \) causes \( P \) to become \( 6x^3 - 4x^3 + 10x^2 + 4 \). Note the order of the parameters: we
specify the coefficient first and then the exponent.

7. \texttt{removeterm}: This method removes the term with exponent \texttt{exp} from the polynomial. If such a term does not exist, the polynomial remains unchanged. For example, if \( P \) is \( 6x^5 - 4x^3 + 10x^2 + 4 \), \( P.removeterm(3) \) causes \( P \) to become \( 6x^5 + 10x^2 + 4 \).

8. \texttt{scale}: This method returns the result of scaling the polynomial by a factor \texttt{s}. Note that a polynomial object is returned. Moreover, the activating polynomial itself should remain unchanged. For example, if \( P \) is \( 6x^5 + 10x^2 + 4 \), then \( P.scale(2.5) \) returns the polynomial \( 15x^5 + 25x^2 + 10 \). \( P \) itself is unchanged.

9. \texttt{__add__}: This method overloads the + operator and returns the sum of two polynomials, which is also a polynomial (as explained above). Note that \texttt{__add__} has two parameters, \texttt{self} and \texttt{other}, where \texttt{other} is also a polynomial. Implement this function by \textit{reusing} code already written, i.e., by calling methods already implemented. Important: Neither \texttt{self} nor \texttt{other} should be modified by this method.

10. \texttt{__sub__}: This method overloads the - operator and returns the difference of two polynomials, which is also a polynomial (as explained above). Comments on the \texttt{__add__} method apply here as well.

11. \texttt{__mul__}: This method overloads the * operator and returns the product of two polynomials, which is also a polynomial (as explained above). Again, neither \texttt{self} nor \texttt{other} should be modified by this method.

12. \texttt{__getitem__}: This method overloads the [] (indexing) operator. If \texttt{idx} is the parameter to the index operator, \texttt{__getitem__} must return the coefficient of the term with exponent \texttt{idx}. For example, if \( P \) is \( 6x^5 + 10x^2 + 4 \), \( P[5] \) should return 6, \( P[0] \) should return 4, and \( P[3] \) should return 0.

13. \texttt{__setitem__}: This method implements the index assignment operation. If \texttt{idx} and \texttt{value} are the two parameters to this method, then \texttt{value} should become the coefficient of the term with exponent \texttt{idx}. If a term with this exponent already exists, its coefficient is changed to \texttt{value}. If a term with this exponent does not exist, it should be created.

In addition to the above \texttt{Polynomial} class methods, you must implement the following two functions (these are regular functions, not methods):

- A function called \texttt{read_polynomial} with a single parameter, a string \texttt{polyfilename}, which is the name of a file that contains the terms of a polynomial in the following format: The file has as many lines as there are terms, and each line of the file contains a pair of numbers, the first being the coefficient of the term and the second being the exponent. This function opens the file for reading, creates a \texttt{Polynomial} instance from the data in the file, and then \texttt{returns} this instance.
  Two files called \texttt{poly1.txt} and \texttt{poly2.txt} containing polynomial data have been created for you. They are available on the Sakai site for this homework assignment. You may use these to test your implementation. Try creating some of your own as well for further testing.

- A function called \texttt{arith_ops_polys} with two parameters \( P \) and \( Q \), both of which are polynomial instances. This function simply prints (on separate lines) the degree of each
polynomial, and the polynomials resulting from adding, subtracting, and multiplying the polynomials P and Q.

Problem 2 [40 points] Quadratic Equations. A quadratic expression of one variable is an expression of the form $ax^2 + bx + c$, where $a$, $b$, and $c$ are real numbers, called the coefficients. In particular, $a$ is called the quadratic coefficient, $b$ is called the linear coefficient, and $c$ is called the constant coefficient. Since a quadratic expression is simply a polynomial of degree 2, in this problem you are asked to create a Quadratic class as a subclass of the Polynomial class. Implement your Quadratic class in a module called quad.py, which imports the Polynomial class from the module poly.py I have also provided a test module called test_quad.py (which imports your quad.py module). Simply type python3 test_quad.py to test your implementation of the Quadratic class.

For a quadratic expression, a real root is any real value $x$ such that $ax^2 + bx + c = 0$. In other words, a real root is a value of $x$ for which the quadratic expression evaluates to 0. For example, $2x^2 - 5x + 2$ has two real roots: $x = 2.0$ and $x = 0.5$ because $2x^2 - 5x + 2$ evaluates to 0 at both those values of $x$. The real roots of any quadratic expression can be found by applying the following rules:

1. If $a$, $b$, and $c$ are all zero, then every value of $x$ is a real root.
2. If $a$ and $b$ are zero but $c$ is nonzero, then there are no real roots.
3. If $a$ is zero and $b$ is nonzero, then there is only one real root, which is $x = -c/b$.
4. If $a$ is nonzero and $b^2 < 4ac$, then there are no real roots.
5. If $a$ is nonzero and $b^2 = 4ac$, there is one real root, which is $x = -b/(2a)$.
6. If $a$ is nonzero and $b^2 > 4ac$, there are two real roots, which are $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

In this problem, you are asked to implement a Quadratic class as a sub-class of the Polynomial class. We first describe below those methods of the Polynomial class that are customized for the Quadratic class. Next, we describe methods that extend the Quadratic class. Keep in mind that you should reuse methods of the Polynomial class whenever possible when customizing (or extending) methods for the Quadratic class.

Customize: The following methods of the Polynomial class are to be customized for the Quadratic class.

- **init**: Since a quadratic expression is always of the form $ax^2 + bx + c$, the parameter list for the constructor could consist simply of the quadratic, linear, and constant coefficients. Hence, the constructor for the Quadratic class has three real numbers as parameters, which are the quadratic, linear, and constant coefficients, in that order. As shown in the examples in class, in order to avoid unnecessary code repetition, the constructor should be implemented by calling the Polynomial class constructor with the suitable parameter list. Hence, an example of creating a Quadratic instance for the quadratic equation $2x^2 - 5x + 2$ would be as follows:

```
Q = Quadratic(2, -5, 2)
```

- **addterm**: We need to customize this method for Quadratic objects because only terms with exponents 2, 1, or 0 can be added to a Quadratic object in order to maintain it as a Quadratic (adding a term with higher exponent would imply that
the object is not a quadratic equation any more). Therefore, this method must first check that the exponent of the term being added is legal. If it is not, an exception should be raised. If it is, then the term may be added. Once again, call the Polynomial method suitably instead of repeating code.

- **add**: Since the sum of two quadratics is also a quadratic, the customization required here is that the add method for Quadratic objects should return a Quadratic as well. Implement this method to ensure that a Quadratic object is returned.

- **sub**: For the same reasons described above in the add method, you will need to customize the sub method as well so that it returns a Quadratic.

- **mul**: When two quadratics are multiplied, the result may or may not be a quadratic. Customize the mul method for the Quadratic class so that it returns a Quadratic object if the degree of the result is at most 2, and a Polynomial object otherwise.

- **scale**: Once again, since the result of scaling a quadratic is a quadratic, you must customize the scale method for the Quadratic class so that it returns a Quadratic.

**Extend**: The following method is specific to quadratic equations, and hence we extend the Quadratic class with this method.

- **roots**: This method returns a list containing the real roots of the quadratic equation. If the quadratic equation has no real roots, an empty list is returned. If the quadratic equation has one real root, a list containing that single root is returned. If the quadratic equation has two real roots, a list containing both roots is returned. Finally, if the quadratic equation has infinitely many roots (rule #1 above), a list of length 3, containing all zeros, is returned.

**Submission Guidelines**

Implement the first problem in a module called poly.py and the second one in a module called quad.py. Your name and RUID should appear as a comment at the very top of each module. Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Sakai as follows:

1. Use your web browser to go to the website sakai.rutgers.edu.

2. Log in by using your Rutgers login id and password, and click on the 'OBJECT-ORIENTED PROG S17' tab.

3. Click on the 'Assignments' link on the left and go to 'Homework Assignment #5' to find the homework file (hw5.pdf), and the test modules test.poly.py and test_quad.py. In addition, sample data files poly1.txt and poly2.txt for Problem 1 are also provided.

4. Use this same link to upload your two homework files (poly.py and quad.py). Please note that you are allowed only two re-submissions, so ensure that you are completely done with your program before you submit.

You must submit your assignment at or before 11:55PM on April 28, 2017.