ECE 122: Introduction to Programming for ECE- Spring 2023

Project 1: A Polynomial Toolbox

A Basic Procedural Programming Example

Due Date: **Deadline: see website, class policy and Moodle for submission**

Description

The goal of the project is to design and implement a calculator tool for polynomial functions. This particular project is using procedural programming (stated otherwise, it must not use any object oriented programming concepts). The main code `project1.py` is provided (but you will need to understand its implementation and how does it work). All the needed functions must be implemented in the module `polynomial.py` used by the main code. The projects also contains a series of short testing program `test_option*.py` that will help you test/implement/debug step-by-step the needed functions. You are *not* allowed to change/edit the main program.

In this project, we will keep track of a list of polynomial functions. Each polynomial is itself represented by a list of floats. Let us consider the following polynomial of degree d:

$$p(x) = a_0 + a_1 x + \cdots + a_{d-1} x^{d-1} + a_d x^d,$$

it will be represented by the following list of length $d + 1$:

$$p = [a_0, a_1, \ldots, a_{d-1}, a_d]$$

where each a_i coefficient represents a float number.

In this project, you will be able to manipulate and perform some mathematical operations on polynomials such as: evaluate, derive, integrate, add/subtract/multiply two polynomials, etc.

The project is designed to be incremental, you can then debug, test and run your code after each new task/option is implemented. After Option 1 is completed all the other Tasks/options can be completed in any order (with some exceptions). Use your preferred IDE to read, write and save your files. Ideally you should use Visual Studio Code (VSCode). Do not use Google Colab...this is not a notebook assignment but a real programming one. Make sure your program can also be executed using the command prompt/terminal (this will happen automatically using VScode). Do not forget to comment your code. Make sure you obtain the **same output** for the **same input** in the examples (this includes syntax, blank spaces, and skipping blank lines). Your program will also be tested with different inputs by the graders. Finally, you are free to consider additional functions than the ones asked if you wish to do so. However, you **are not allowed** to use programming concepts different than the ones we have seen in class so far (beyond lecture 2.6). **No python methods** are permitted such as ".append" for list, ".split" for strings, any others methods, or any class. Basically, you should only use basic lists, if/elif/else, for and while loops, and basic functions. Also, you are not allowed to import other modules than the ones specified. If you have any doubts or questions about those requirements, please contact the TAs.

Submission/Grading Proposal

Only one file `polynomial.py` must be submitted on Moodle. This project will be graded out of 100 points:

1. You can do the project alone or by group of 2 max. If you do it in team, add your two names in the header of the file `polynomial.py`, and submit only ***one*** file per team (anyone from the two team members).
2. Your program should implement all basic functionality/Tasks and run correctly (90 points).
3. Overall programming style: program should be written in a clear way with all the proper comments. All functions should have a header doc-string as well (5 points).
4. Pre-submission up to at least Option-4 (5 points). The program does not have to run correctly for pre-submission. Only the act of pre-submission is graded and not the content of your submission. If needed, your pre-submission can be used as proof that you are effectively making steady progress (working from day 1).

1 Overview of the main program functionality

At the first execution of the program `project1.py`, the output should include a menu containing 11 options:

```
Polynomial Tool
===============

---------------
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

If no option is selected (just press Enter), the program stops. And it should print:

```
Thanks for using the polynomial tool
Come back soon!
```

All the command options that you must implement, will be reviewed in the following.

Option-1 [20pts]

Let us now see what should happen when option 1 is selected.

```
Enter Command: 1
Degree of polynomial? 2
Enter coef. of x^0: 1.5
Enter coef. of x^1: -3
Enter coef. of x^2: 2

---------------
1: 1.5-3.0x+2.0x^2
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

The program is asking to enter the degree of the polynomial you would like to insert into your calculator. It then proceeds by asking to enter the coefficients for each power (from smallest to largest power). Finally, the program displays the corresponding polynomial with its ID 1 (1 because it is the first polynomial entry so far). You can insert as many polynomials as you want. For example, if you execute option 1 again with the coefficients `1 5 4 -3.5`, you would get:

```
Enter Command: 1
Degree of polynomial? 3
Enter coef. of x^0: 1
Enter coef. of x^1: 5
Enter coef. of x^2: 4
Enter coef. of x^3: -3.5

---------------
1: 1.5-3.0x+2.0x^2
2: 1.0+5.0x+4.0x^2-3.5x^3
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

Each new polynomial will be listed 2 after the last one entered. Each polynomial is associated with an ID number (1 then 2 here). Let us insert a third one:

```
Enter Command: 1
Degree of polynomial? 1
Enter coef. of x^0: -3
Enter coef. of x^1: 3

---------------
```

```
1: 1.5-3.0x+2.0x^2
2: 1.0+5.0x+4.0x^2-3.5x^3
3: -3.0+3.0x
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

How to proceed? For option 1 to work, the file `test_option1.py` must be able to execute properly first. You will need to implement the following functions inside `polynomial.py`:

10pts the `get_expression` function, that accepts a list of float numbers as input argument and return an `str` expression of the polynomial function. Several examples are provided in the test file. For full credit you should obtain the fancy outputs. As you can see there are several corner (special) cases to consider. For example if a coefficient is negative, if it is equal to zero, etc. However, do not get stuck here, you can just do the "easy" output and move on (you could come back later to work on this function).

5pts You need to implement the function `display_list` that accepts a list of polynomial (namely a list of list of floats) as input argument, and display all the expression of all the polynomials that are in that list (with their ID first). This function must use the function `get_expression` that you just implemented. Obviously you will need a `for` loop to accomplish this.

5pts To make option 1 from the main program `project1.py` work, you now need to implement the function `get_polynomial` that has no argument but returns a list of float (that represents our polynomial). Inside this function, you will need to ask the user the question regarding the degree of the polynomial you wish to enter and proceed by asking the users the coefficients for each power. Look at the examples above and make sure you obtain the same behavior (output, etc.).

Option-2

Option 2 can be used to remove a particular polynomial from the list using its ID number.

```
Enter Command: 2
Which polynomial would you like to remove? 2

---------------
1: 1.5-3.0x+2.0x^2
2: -3.0+3.0x
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

How to proceed? Nothing to do, this one comes for free. Just understand what the main program is doing.

Option-3 [10pts]

Option 3 returns some info for each polynomial that are present in the list. Here is an example after inserting two more polynomial entries (so we have four polynomials in total).

```
---------------
1: 1.5-3.0x+2.0x^2
2: -3.0+3.0x
3: 1.0+3.0x^2
4: -5.0x-4.0x^3
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command: 3
1:  Degree is 2
2:  Degree is 1
3:  Degree is 2 and it is even
4:  Degree is 3 and it is odd

---------------
1: 1.5-3.0x+2.0x^2
2: -3.0+3.0x
3: 1.0+3.0x^2
4: -5.0x-4.0x^3
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

How to proceed? For option 3 to work, the file `test_option3.py` must be able to work properly first. For full credit the corner (special) cases should work too.

Write the function `info` that accepts a polynomial (represented by a list of floats), and returns a string that include various information about the degree of the polynomial (its highest power), and if the polynomial is odd or even. A polynomial $f(x)$ is even is $f(x) = f(-x)$ and it is odd if $f(-x) = -f(x)$. To find if a polynomial is even, you could just check if its coefs of odd powers $(x, x^3, x^5$, etc.) are all zeros; To find if a polynomial is odd, you could just check if its coefs of even powers $(x^0, x^2, x^4$, etc.) are all zeros.

Option-4 [10pts]

Option 4 is used to evaluate a particular polynomial $p(x)$ chosen by the user (from the list) for different x values (the program evaluates the polynomial at 13 distinct hard-coded values of x between $[-3, 3]$). For example:

```
Enter Command: 2
Which polynomial would you like to remove? 3

---------------
1: 1.5-3.0x+2.0x^2
2: -3.0+3.0x
```

```
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command: 4
Which polynomial would you like to evaluate from [-3,3]? 1
-3.0 28.5
-2.5 21.5
-2.0 15.5
-1.5 10.5
-1.0 6.5
-0.5 3.5
0.0 1.5
0.5 0.5
1.0 0.5
1.5 1.5
2.0 3.5
2.5 6.5
3.0 10.5

---------------
1: 1.5-3.0x+2.0x^2
2: -3.0+3.0x
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

How to proceed? For option 4 to work, the file `test_option4.py` must be able to work properly first.

Write the function `evaluate` that accepts two input arguments: a polynomial (represented by a list of floats), and a float number x_0. The function should evaluate and return the numerical value of $p(x_0)$ (a float number).

Option-5 [5pts]

Option 5 can be used to scale a particular polynomial (it means that it will multiply all its coefficients by a scaling factor chosen by the user).

```
Enter Command: 5
Which polynomial would you like to scale? 1
Which scaling factor? 0.5
0.5*p1(x)= 0.75-1.5x+1.0x^2

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

Remark: The program is also displaying the operation being performed and its result. The corresponding polynomial in the list is updated.
How to proceed? For option 5 to work, the file `test_option5.py` must be able to work properly first.

Write the function `scale` that accepts two input arguments: a polynomial (represented by a list of floats), and a scaling factor (float number). The function should evaluate and return a new polynomial (a list of float).

Option-6 [10pts]

Option 6 performs and displays the successive derivatives of a particular polynomial (until the derivative is 0).

```
Enter Command: 6
Which polynomial would you like to derive? 1
(1) -1.5+2.0x
(2) 2.0
(3) 0.0

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

Remarks:

- Here the (1), (2), (3) labels mean first, second and third derivatives respectively.
- From Calculus I, if we consider a polynomial of degree d:

  $$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{d-1}x^{d-1} + a_dx^d$$

  its derivative becomes the polynomial of degree $d-1$

  $$p'(x) = a_1 + 2a_2x + \cdots + (d-1) * a_{d-1}x^{d-2} + d * a_dx^{d-1}$$

  This process can be repeated until the derivative becomes 0.

How to proceed? For option 6 to work, the file `test_option6.py` must be able to work properly first.

Write the function `derive` that accepts one input argument: a polynomial (represented by a list of floats). The function should derive the polynomial and return a new polynomial (a list of float). Here what you could do to implement this function:

1. If the size L of your input polynomial list p1 is less or equal to 1, just return [0.0]
2. Initialize a new list p of containing the last L-1 float numbers of your input polynomial p1.
3. Scan this list from left to right:
   - Multiply the coefficients by the correct power index from the p1 polynomial.
4. Return the new list of float.

Option-7 [10pts]

Option 7 is performing the integration of a particular polynomial within user chosen lower bound a and upper bound b.

$$I = \int_a^b p(x)dx$$

```
Enter Command: 7
Which polynomial would you like to integrate? 1
enter lower bound: -3
enter upper bound: 4.5
int(-3.0,4.5) p1(x) dx= 36.5625

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

Remarks: The program is displaying the operation being performed and its result. From Calculus II, if we consider a polynomial of degree d:

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{d-1}x^{d-1} + a_dx^d$$

its anti-derivative becomes the polynomial of degree $d + 1$

$$P(x) = a_0x + \frac{a_1}{2}x^2 + \frac{a_2}{3}x^3 + \cdots + \frac{a_{d-1}}{d}x^d + \frac{a_d}{d+1}x^{d+1} + C$$

We then get to evaluate:

$$I = P(b) - P(a)$$

Remark: In practice, here we can safely choose $C = 0$ in the anti-derivative expression (so no need to consider it).

How to proceed? For option 7 to work, the file `test_option7.py` must be able to work properly first.

Write the function `integrate` that accepts three input arguments: a polynomial (represented by a list of floats), the lower bound and upper bound b of the integral. The function should compute the anti-derivative (new polynomial), evaluate the integral and return the result (a float number).

Option-8 [10pts]

Option 8 performs the summation between two polynomials, display the operation and its result, and create a new polynomial to be added onto the list. Remark: the 2nd polynomial may have the same ID than the first one.

```
Enter Command: 8
Enter 1st polynomial #: 1
Enter 2nd polynomial #: 2
p1(x)+p2(x)= -2.25+1.5x+1.0x^2

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
3: -2.25+1.5x+1.0x^2
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

How to proceed? For option 8 to work, the file `test_option8.py` must be able to work properly first.

Write the function `add` that accepts two input arguments: two polynomials (represented by a list of floats). The function should perform their summation and return a new polynomial (list of float).

Here what you could do to implement this function (just a suggestion with example):

1. Initialize a new list p of 0.0 float numbers (length of the list should be the max size between the two inputs p1 and p2 polynomials). Ex:

   ```
   p1=[1.0,2.0,3.0] and p2=[3.0,4.0]
   Remark:  max size is 3 (p1 has 3 coefs), min size is 2 (p2 has 2 coefs)
   p=[0.0,0.0,0.0]
   ```

2. Scan this list from left to right until the min size between p1 and p2 is reached, and add the corresponding coefs into p. Ex:

   ```
   p[0]=1.0+3.0=4.0
   p[1]=2.0+4.0=6.0
   ```

3. If max size is different than min size, insert into p the remaining coefs of the "longest" polynomial. Ex:

   ```
   p[2]=3.0   --> since p1[2]=3.0
   ```

4. Corner case: at the end, trim the list p to remove the end zeros. For example: if your p ends up being `p=[1.0,0.0,0.0]` you should trim the zeros to obtain `p=[1.0]`. Hint: use the `del` function.

5. Return the list of float p.

Option-9 [5pts]

Option 9 performs the subtraction between two polynomials, display the operation and its result, and create a new polynomial to be added onto the list. Remark: the 2nd polynomial may have the same ID than the first one.

```
Enter Command: 9
Enter 1st polynomial #: 3
Enter 2nd polynomial #: 2
p3(x)-p2(x)= 0.75-1.5x+1.0x^2

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
3: -2.25+1.5x+1.0x^2
4: 0.75-1.5x+1.0x^2
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

How to proceed? For option 9 to work, the file `test_option9.py` must be able to work properly first.

Write the function `subtract` that accepts two input arguments: two polynomials (represented by a list of floats). The function should compute their subtraction and return a new polynomial. Hint: It is pretty straightforward to implement this function once the `add` function in Option 8 has been implemented. You could simply observe that $p = p1 - p2$ is the same as $p = p1 + p3$ with $p3 = -p2$. **Remark:**: The way the test is currently setup, requires the creation of a new list ($p3$ here). Indeed, lists are mutable so you cannot simply change the sign of all the $p2$ coefficients since it will also be changed in the main program calling the function (and we do not want that). So just make a copy (not an alias, see lecture notes) of $p2$ and change its sign.

Option-10 [10pts]

Option 10 performs the multiplication between two polynomials, display the operation and its result, and create a new polynomial to be added onto the list. Remark: the 2nd polynomial may have the same ID than the first one.

```
Enter Command: 10
Enter 1st polynomial #: 4
Enter 2nd polynomial #: 3
p4(x)*p3(x)= -1.6875+4.5x-3.75x^2+1.0x^4

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
3: -2.25+1.5x+1.0x^2
4: 0.75-1.5x+1.0x^2
5: -1.6875+4.5x-3.75x^2+1.0x^4
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
```

How to proceed? To implement this option, you must first have Option 5 and Option 8 implemented correctly. Thereafter, for this option 10 to work, the file `test_option10.py` must be able

to work properly first.

Write the function `multiply` that accepts two input arguments: two polynomials `p1` and `p2` (represented by a list of floats). The function should compute their product and return a new polynomial.

Here is a pseudocode you could do to implement this function:

1. Initialize a new list containing just [0.0] (let us call it `p=[0.0]`)
2. Scan the coefficients of polynomial p1 from left to right:
   - With the current p1 coefficient, use your function `scale` on polynomial p2, to create a new polynomial (let us call it p3)
   - Depending on the 'power' associated with the current coefficients p1, keep appending [0.0] on the left of p3
   - Using your function `add`, sum p and p3 together and assign it to p again.
3. Return p.

I guess you know how laborious it could be to make multiplications of polynomial by hand, now things become much easier with your own personal polynomial tool!

```
Enter Command: 10
Enter 1st polynomial #: 5
Enter 2nd polynomial #: 5
p5(x)*p5(x)= 2.84765625-15.1875x+32.90625x^2-33.75x^3+10.6875x^4+9.0x^5-7.5x^6+1.0x^8

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
3: -2.25+1.5x+1.0x^2
4: 0.75-1.5x+1.0x^2
5: -1.6875+4.5x-3.75x^2+1.0x^4
6: 2.84765625-15.1875x+32.90625x^2-33.75x^3+10.6875x^4+9.0x^5-7.5x^6+1.0x^8
---------------
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command: 10
Enter 1st polynomial #: 6
Enter 2nd polynomial #: 6
p6(x)*p6(x)= 8.109146118164062-86.49755859375x+418.071533203125x^2-1191.744140625x^3+
2168.84619140625x^4-2494.546875x^5+1526.34375x^6+98.71875x^7-981.17578125x^8+668.25x^9
-13.5x^10-202.5x^11+77.625x^12+18.0x^13-15.0x^14+1.0x^16

---------------
1: 0.75-1.5x+1.0x^2
2: -3.0+3.0x
3: -2.25+1.5x+1.0x^2
4: 0.75-1.5x+1.0x^2
5: -1.6875+4.5x-3.75x^2+1.0x^4
6: 2.84765625-15.1875x+32.90625x^2-33.75x^3+10.6875x^4+9.0x^5-7.5x^6+1.0x^8
7: 8.109146118164062-86.49755859375x+418.071533203125x^2-1191.744140625x^3+
2168.84619140625x^4-2494.546875x^5+1526.34375x^6+98.71875x^7-981.17578125x^8+668.25x^9
-13.5x^10-202.5x^11+77.625x^12+18.0x^13-15.0x^14+1.0x^16
---------------
```

```
1-Insert; 2-Remove; 3-Info; 4-Evaluate; 5-Scaling; 6-Derive; 7-Integrate
8-Summation; 9-Subtract; 10-Multiply; 11-Divide
Enter Command:
```

Option-11- Bonus [5pts]

Option 11 performs the division between two polynomials (numerator with highest power), and display the factorization. Some examples here.

```
If p1=[4.0,-9.0] and p2=[2.0,3.0]
Your program should display:

4.0x^2-9.0 = (2.0x+3.0)(2.0x-3.0)
```

```
If p1=[2.0,-5.0,1.0] and p2=[1.0,-3.0]
Your program should display (2.0 is the remainder):

2.0x^2-5.0x-1.0 = (1.0x-3.0)(2.0x+1.0) + 2.0
```

No help/guidance will be provided by the TAs for bonus questions.