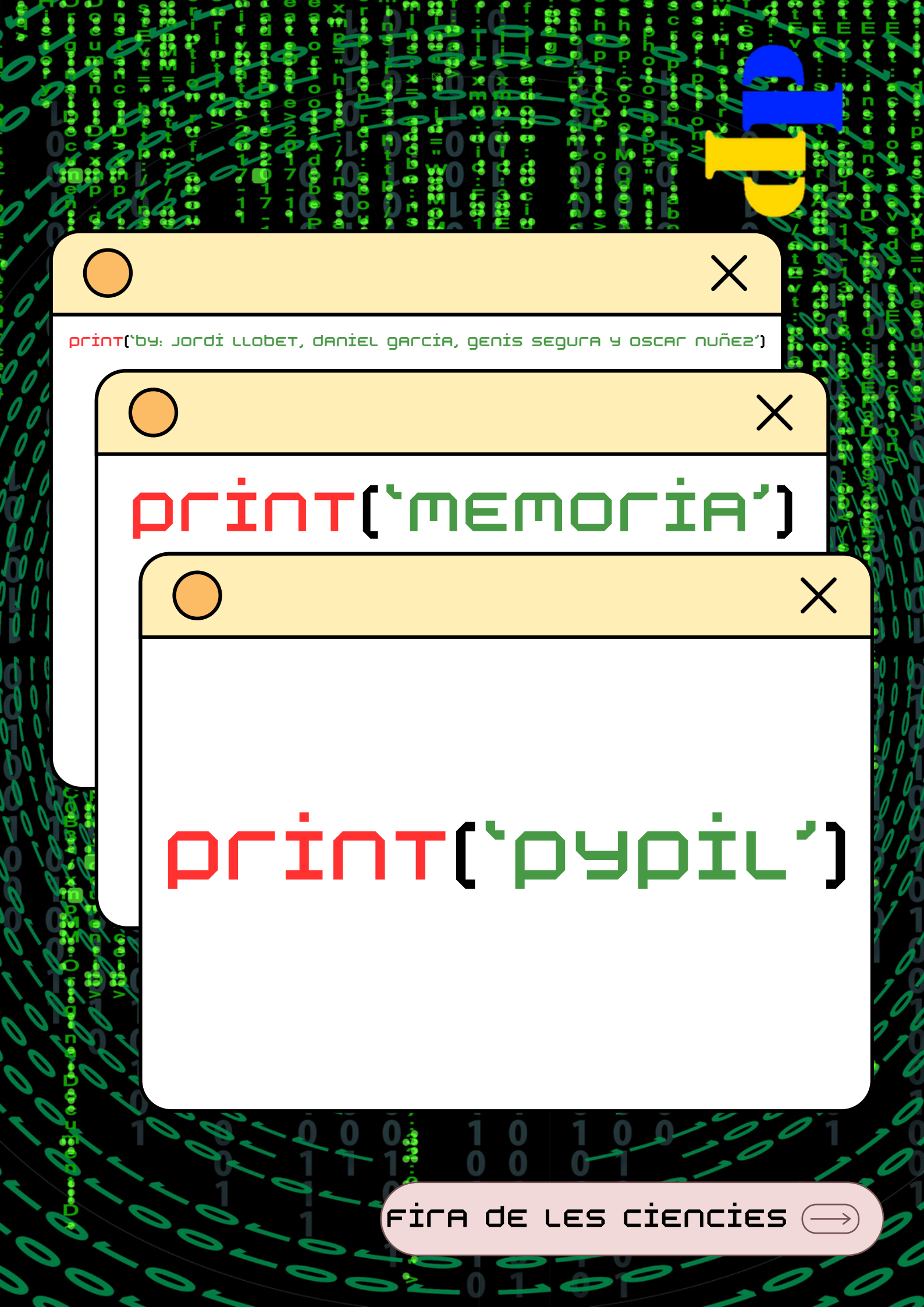
print('by: Jordi Llobet, Daniel Garcia, Genis Segura y Oscar Nuñez')
print('memoria')
print('pypil')
fira de les ciencies

Index/Índex/Índice

Index/Índex/Índice 1
English 2
1 Introduction 2
2 Problem 3
3 Objectives 3
4 Theoretical Framework 5
5 Implementation and Development 11
6 Analysis 15
Català 17
1 Introducció 17
2 Problema 18
3 Objectius 18
4 Marc teòric 20
5 Implementació i Desenvolupament 26
6 Anàlisi 31
Castellano 33
1 introducción 33
2 Problema 34
3 Objetivos 34
4 Marco teórico 36
5 implementación y Desarrollo 42
6 análisis 47
Readme 48

English

1 Introduction

“Note for GitHub users: Below, you will be able to see our R&D project. This has been developed over a week at our school, "Escola Mare de Déu de la Salut," and this document is entirely dedicated to it. To view the parameters, access the Readme, or you can contact us through our email at LibPypil@gmail.com .”

In the complex realm of programming, various functions can be found, and our program will delve into libraries, which are large collections of contrasting functions. These libraries serve a purpose within the programming world and can be quite intricate. They focus on reducing complexity, facilitating, and enhancing the speed of developers.

One common challenge for many programmers during software or application development is the lack of certain functions in programming languages. This deficiency often leads to the use of libraries that supplement the base code, accomplishing tasks that would otherwise be more tedious and complex.

Our research work will focus on the Python programming language. While not the most efficient or fastest for certain applications, Python is one of the most widely used languages today, boasting over 10 million users. It provides a simple user interface and accessibility to a broad audience. Our goal is to reach more people and make programming more accessible.
Pypil is a library primarily aimed at polynomials, with most of its functions centered around this mathematical concept. Nevertheless, it also encompasses other basic mathematical functions, such as equations, which enable programmers to develop various software and, in a way, improve the quality and ease of development.
In summary, due to the scarcity of polynomial libraries and some mathematical functions in Python, we have set out to develop a library that covers this domain in a straightforward manner.

2 Problem

Python incorporates various built-in functionalities in its core. Additionally, there are several libraries developed by the community that extend its power. Some of these tools are created by companies or research-focused universities, while others result from the collaboration of programmers. Often, these libraries emerge to address specific needs or to provide more specialized instruments in areas such as machine learning, data analysis, or visualization. The richness of the Python ecosystem lies in the multitude of available libraries that cover a wide range of applications. The choice to use specific tools depends on the purpose and preferences of each programmer who selects these functionalities according to their liking.

That said, a common challenge for many developers and, based on user experience, is the lack of certain functions within the Python base code. For example, the basic Python code does not provide a built-in way to calculate roots. To address this, the Montreal Institute for Learning Algorithms (MILA) developed the library known as Math, which enables the calculation of roots. Similarly, as mentioned, there are many other libraries that offer similar functionalities.

One of the problems with Python is the lack of libraries for certain basic mathematical operations, such as polynomials. Therefore, our goal is to create a library that covers this area.

3 Objectives

The main goal of our research work is to develop a Python library that will allow users to work with various polynomial and basic mathematical operations, as well as to simplify and improve the lives of programmers during software development.

Furthermore, we aim to make our code accessible to everyone, which is why we have published our library on GitHub (https://github.com/), an online repository for managing projects and controlling code. This website is well-known among developers, as it has a vast database of software that enables programmers to collaborate or view different codes for use or understanding. Our profile name is LibPypil (https://github.com/LibPypil/PyPil/tree/main), and we hope to reach both beginner and advanced developers to enhance and

facilitate the development of specific applications.
During the week that we have been working, we have set the following objectives regarding the development of the library (the following are distinct functions).

- ✓ PolyAddition (addition) - Completed on 12/11/23.
- ✓ PolySubtraction (subtraction) - Completed on 12/12/23.
- ✓ PolyMultiplication (multiplication) - Completed on 12/12/23.
- ✓ PolyDivision (division) - Completed on 12/12/23.
- ✓ PolyRoot (root) - Completed on 13/12/23.
- ✓ Ruffini - Completed on 12/12/23.
- ✓ General Algorithm - Completed on 12/12/23.
- ✓ PolyPow (Power) - Completed on 13/12/23.
- ✓ EQ1 - Completed on development.
- ✓ EQ2 - Completed on 13/12/23.
- ✓ EQ3+ - Completed on 13/12/23.
- ✓ Readme.txt - Completed on 14/12/23.

As for the memory and poster, we have set the following objectives.

- ✓ Cover - Completed on 12/12/23.
- ✓ Index - Completed on 14/12/23.
- ✓ Introduction - Completed on 11/12/23.
- ✓ Problem Statement - Completed on 11/12/23.

- ✓ Objectives - Completed on 12/12/23.
- ✓ Theoretical Framework (code documentation) - Completed on 13/12/23.
- ✓ Implementation/Development - Completed on 13/12/23.
- ✓ Analysis - Completed on 14/12/23.

Below, I have attached the group's work schedule.

Date	Task
11/12/2023	Organize the calendar and work on the 5W's and SCQA methods.
12/12/2023	Code development, poster and memory (6h)
13/12/2023	Code development, poster and memory (6h)
14/12/2023	Code development, poster and report/application (5h)
15/12/2023	Finish pending objectives and prepare the oral presentation on Monday.

18/12/2023	Presentation
19/12/2023	Awards

4 Theoretical Framework

Here, you will find various concepts that we have needed to develop our research work or some necessary knowledge to understand the research correctly.

Furthermore, we will also present some concepts related to our "PyPil" library and how some of its functions work and what they do.

To begin, let us discuss what a programming language is and its function within software. A language is a set of specialized words and specific grammatical rules that instruct a device to perform activities such as displaying text or an image on the screen. These languages enable applications, websites, AI (Artificial Intelligence), neural networks, and even research and scientific development applications.

In addition to these specific words that form the foundation of any programming language, there are certain languages that allow you to use libraries. A library is a collection of functions and methods prescribed by someone else for you to use in your own code. Another way to explain it, if you are not familiar with the world of software development, is to imagine building a house and needing tools to do so. In this case, the "library" would be like a set of specialized tools that are already made and ready to use in your construction project.

The structure of a conventional Python library is based on:

- The main folder of the package.
- A file indicating that this folder is a Python package, usually called `__init__.py`.
- A module containing all the functions within the library.
- Some tests for the library.
- A file that configures the entire library and another that configures its installation.
- A file that provides all the basic information about the library and its usage.

Our "PyPil" library is specialized in the use of polynomials and monomials and includes the following functions:

- PolyAddition: Adds two monomials of similar degree.
- PolySubtraction: Subtracts two monomials of similar degree.
- PolyMultiplication: The multiplication is divided into two paths. 1) The polynomial is multiplied by a constant term. 2) The polynomial is multiplied by another polynomial in the classic way.
- PolyDivision: This function is divided into two ways of dividing polynomials.

GeneralAlgorithm: This function mimics the procedure of a general algorithm, acting as a translation for the machine to understand.

Ruffini: This function will only execute if in the division one of the two polynomials is of the 1st degree. The function takes the first monomial, multiplies it by the independent term of the second polynomial, then repeats this process until the end.

- PolyRoot: Given a polynomial, this function factors out the common factor (if possible), finds the divisors of the constant term, and tests Ruffini. When the degree is 2, it applies the quadratic formula, returning all roots.

- PolyPow: In this function, a polynomial is multiplied by itself "n" times.

- EQ1 (first-degree equations) - A first-degree equation consists of a sequence of numbers that can be accompanied by an unknown variable 'x,' which can only be raised to the power of 1. With that said, this algorithm is divided into 3 subsections: 1 - Equations without fractions, 2 - Equations with one fraction, and the third one is - More than one fraction.

1 - This is the section without fractions. What the function first does is differentiate between values with and without 'x.' After this, it swaps them to isolate 'x' as much as possible. Following this, it divides both sides of the equation, resulting in the value of 'x.'

2 - This is the section with one fraction. This function starts by classifying the types within the subsection (which are 4).The 4 types are: where 'x' is the numerator of a fraction, where 'x' is the denominator of a fraction and there is more than one 'x,' where 'x' is the denominator of a fraction and there are no more 'x,' and finally, where there is 'x' in both the numerator and denominator.

2.1 - These equations are where 'x' is in the numerator. In these examples, our algorithm

multiplies the entire equation by the denominator of the fraction. After this, the equation transforms into a type 1 equation and is solved following the same steps.

2.2 - These equations are where 'x' is in the denominator, and there is more than one 'x.' These are solved by multiplying the entire equation by 'x,' causing a second-degree equation that is solved using the quadratic equation.

2.3 - These equations are where 'x' is in the denominator of the fraction, but there is no more 'x.' These are solved by multiplying the entire equation by 'x,' isolating all 'x,' and setting it equal to 0.

2.4 - These fractions are those with 'x' in both the numerator and denominator. These 'x's will cancel each other out, resulting in 3 distinct types:

2.4.1 - If 'x' is larger in the numerator, it means we will follow type 2.1.

2.4.2 - If 'x' is larger in the denominator than in the numerator and there is no more 'x,' we will follow type 2.3.

2.4.3 - But if in the previous case, we do have more 'x,' we will follow type 2.2.

3 - This type consists of equations with more than 1 fraction. There are 3 cases.

3.1 - 'x' is in the numerator, and there is a number in the numerator. To solve it, the least common multiple (LCM) of the denominators is calculated and multiplied by the denominators.

3.2 - 'x' is in the denominators. The LCM of the

denominators (which is more complex) is calculated and multiplied by the numerators.

3.3 - Some have 'x' in the numerator, and others in the denominator. The methods from the previous cases are combined to solve it.

3.4 - This case is if there is 'x' in both the numerator and denominator. In this case, they will be divided, and depending on which one is larger, type 3.1, 3.2, or 3.3 will be followed.

- EQ2 (second-degree equations): In this function, you input three numerical values (a, b, c), and it starts the quadratic equation. It begins by calculating the discriminant of the equation to see the possible results. After this, the equation is split into two paths, one for addition and one for subtraction. After this, b is added/subtracted, and then everything is divided by 2, giving the two desired results.

- EQ3 (third-degree or higher equations): In this function, we have covered two types of equations of degree greater than 2. Those with only one x raised to a number and an independent term, and those with more than one x with different exponents.

For those with different exponents, we have applied the previously created root algorithm, and from there, we have obtained the answers. For the remaining ones, the Nth root is taken to the independent term.

Computational algebra is a software package that allows the manipulation of mathematical expressions similarly to doing it by hand.

A monomial is a simple mathematical expression consisting of a coefficient multiplied by a variable (or more) raised to a non-negative exponent. On the other hand, a polynomial is a set of monomials that are either added or perform an operation among themselves. Next, we will see how basic polynomial operations work:

- The addition of polynomials involves combining similar terms by summing

the coefficients of terms with the same variable and exponent. For example, in the sum (2x^2 + 3x + 5) and (4x^2 - 2x + 1), you combine the corresponding terms: ((2x^2 + 4x^2) + (3x - 2x) + 5 + 1), resulting in (6x^2 + x + 6).

- The subtraction of polynomials also involves combining similar terms, but in this case, you subtract the coefficients.

- Multiplication of polynomials involves distributing each term of the first polynomial over each term of the second polynomial and then combining like terms.

- Division of polynomials involves dividing the divisor polynomial into the dividend polynomial, like numerical division. The result is the quotient and possibly a remainder.

Next, we will show how roots are found and polynomials are factored. To find the roots of a polynomial, i.e., the values of the variable that make the polynomial equal to zero, you can use different methods. One of the most common methods is Ruffini and the quadratic formula. To factorize a polynomial, we need the Ruffini method (previously performed) and the quadratic formula. With these methods, if the polynomial is of a degree greater than 2, we will apply Ruffini until the polynomial is reduced to a degree of 2, at which point we will apply the quadratic formula. With all these results, we will combine them into a list of polynomials and return them to the user.
To easily solve first-degree equations in a single paragraph, follow these steps. Take an equation like 2x+5=11, for instance. First, subtract 5 from both sides to isolate the variable term: 2x=6. Then, divide both sides by 2 to find the variable's value: x=3. To verify your answer, substitute x=3 back into the original equation to ensure both sides are equal, in this case, 2(3) +5=11. If both sides are equal, you have successfully solved the equation.

To solve second-degree equations, various methods exist, but the primary and most used one is the quadratic formula:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The quadratic formula yields two distinct results, which must be substituted to verify if they are real solutions. If the substitution results in 0, the solution

is correct.

Third-degree equations, also known as cubic equations, are algebraic expressions in which the variable is raised to the power of three. They have the general form ax^3 + bx^2 + cx + d = 0, where (a), (b), (c), and (d) are constant coefficients, and (x) is the unknown variable. Solving a cubic equation involves finding the values of \(x\) that satisfy the equality. While there are specific formulas for solving third-degree equations, numerical methods or factorization can also be employed. These equations can have up to three real or complex solutions, and their study is fundamental in algebra and mathematical analysis.

GitHub is a collaborative development platform that enables developer teams to manage and share their source code conveniently. It operates using the distributed version control system Git. Developers can create repositories to store their projects, and each change made to the code is recorded as a "commit." GitHub facilitates collaboration by allowing multiple people to work simultaneously on a project, merge changes from different branches, and effectively track version history. Additionally, it provides tools like pull requests to streamline code review and integration, along with features for issue management, project tracking, and continuous deployment. In summary, GitHub simplifies the software development process, promoting effective collaboration and efficient source code management.

5 Implementation and Development

Our library can be used for many purposes, as it allows people to solve polynomial operations through various functions we have implemented. With the functions in our library, one could create applications like Photomath, which scans an operation, organizes the information, and solves polynomials quickly and easily. Another example could be a computational algebra software, enabling users to perform algebra similarly to pen-and-paper methods. The goal of PyPil is to enhance the quality and effectiveness of such applications, improving development speed and polynomial handling in Python software. Another common use could be aiding beginners in learning the language and expanding options for novice application creators.

Our library is publicly available for anyone who needs to use it. You can find it on our GitHub (https://github.com/LibPypil/PyPil/tree/main). One of our

objectives was to reach as many people as possible, and as developers, sharing our work with others. Therefore, we implemented a developer website on GitHub to reach a broader audience and make it accessible to anyone who wants to use it, taking our project beyond the classroom and giving it visibility.

Next, we will delve into the development of some functions in our "PyPil" library, discussing their initial design and how they were modified based on encountered challenges during development.

Addition: Our approach was that when introducing polynomials (with well-placed degrees), the similar degrees should be added, and the result printed on the screen. We encountered no problems with the addition, and everything went as planned.

Subtraction: Similarly, when introducing polynomials (with well-placed degrees), similar degrees are subtracted, and the result is printed on the screen. There have been no problems with subtraction since the code is the same as addition; we only modified some operators, and everything worked as expected.

Multiplication: This function is divided into two parts (polynomial x number, polynomial x polynomial). During development, we only had issues with the second part (polynomial x polynomial) due to difficulties with degree multiplication. One significant error was mistakenly adding degrees instead of multiplying them.
The solution was to change a "+" to "*" (multiplication sign in Python) at the end of the function. A small mistake that complicated development but was corrected in time.

- *Division:* This function is divided into two parts.
  Ruffini: We approached this function by breaking it into simpler functions since Ruffini involves complex sums and multiplications. Many errors were due to the polynomial not inverting as the program needed, and the remainder of the operation appeared as part of the resulting polynomial. To solve the remainder problem, we created a variable that isolated the remainder and printed it separately.

  General Algorithm: This has undoubtedly been the most challenging algorithm to program. The initial approach was to

first perform the first iteration as done on paper: find the multiple, multiply, and subtract it from the initial polynomial. I indented a code block incorrectly, delaying progress a bit. Once we had this, the next step was to create a loop that repeated the process until the final polynomial was smaller than the divisor (the remainder). Once done, we returned the coefficient and remainder to the user.

- *PolyPow (Exponentiation)*: This function has only one configuration (polynomial x integer). We encountered no errors, either major or minor, during the development of this function. It simply multiplies a polynomial by itself "n" times.

- *Polyroot (Roots)*: We approached this function similarly to how we do it on paper: first extract the common factor (no problem), perform Ruffini (more complex as it required testing divisors, and the loop gave us quite a few issues), and finally, when the polynomial is of degree 2, apply the EQ2 function, which we had previously developed.

- EQ1 (1st Degree Equations):

1- For these first-degree equations, what we did was write one and analyse it very slowly. Everything we did was added in a note. After this, we thought about how to translate it into code (Python) and began trying separate ways to accomplish it. There were a few errors in the code that prevented us from making progress; for instance, one of our errors was that it did not change the symbol for numbers and did not switch sides. Another error we had was that initially, the code did not separate the two sides of the equation to work on them separately. Later, we fixed this, and now it separates, calculates, and finally rejoins them.

2- For this section, we began by writing all the possible variables (there are 3).

1- There are no fractions: This case was the easiest since we only had to separate the x's from the numbers and finally divide them.

2- There is 1 fraction:

2.1- There's an x in the numerator; we only had to multiply the denominator by all the monomials, and that is it.

2.2- This was more complex because there were x's in the denominators. We thought about multiplying the entire equation by x and then applying a quadratic equation (because if there are more x's, there will be an x^2).

2.3- This is like the previous one but simpler since there were no more x's. We did the same thing, but instead of using a quadratic equation, we used a recursive function with a regular equation.

2.4- This case deals with fractions that have x is both in the numerator and denominator. Our approach here was to create conditions to classify if the numerator is larger than the denominator. If that is the case, it will follow 2.1. If the condition is not met, it will check for more x's. If there are, it will execute 2.2; if not, it will execute 2.3.

3- There is more than 1 fraction.

3.1- In this case, we thought that simply finding the least common multiple (LCM) of the denominators (which will be only numbers) was the easiest way.

3.2- This was more challenging to program because the x's were in the denominator. The mistake I made was not considering finding the LCM with x, as it is different from regular numbers. I simply had to address that, implement polynomial multiplication for multiplying polynomials, and that is it.

3.3- This was simple; we just had to use the last two methods to solve the equation fragments. It was not a problem.

3.4- This is like 2.4, depending on where the x appears larger, we would execute 3.3, which would also execute 3.1 and 3.2.

- *EQ2 (2nd Degree Equations)*: In this function, we planned to use the quadratic formula in reverse (starting from within the equation). First, calculate the discriminant (what is inside the square root).

Thanks to the discriminant, we could determine the number of solutions for a second-degree equation.
If it is negative, the equation has no solutions; if it is zero, there is one solution, and if it is positive, there are two solutions.

- *EQ3p (3rd+ Degree Equations)*: We planned this in two ways: case 1 (only one term with a literal part) and case 2 (more than 1 term with a literal part). For case 2, the solution was easy—simply apply the root algorithm we had previously created, which did not cause any problems. For case 1, I had to put some thought into it since Python has no built-in method for cube roots or higher. With a bit of research, I remembered that in math class, we learned that a square root is the same as that number raised to a fraction (where the base is how much the number is raised, and the denominator is the index of the root). As Python does have a method for exponentiation, I applied that theory, and it worked.

6 Analysis

PyPil is a Python library dedicated to polynomials and equation solving. It was developed over a week with around 28 hours of work and aims to provide tools for software developers and make polynomial operations easier for novice programmers.

The goal of Pypil is to be a library that aids developers, and after a whole week, we passionately believe that it fulfils its purpose. Throughout the development and its completion, along with the small demo we have created, we have been able to verify that it solves the problem we had in mind. Not only that, but we have also been able to develop first-degree equations. Although they are not yet finished, we will continue working on them and developing them to eventually publish them in the official version in the future.

Also, below, I leave the progress of our work and how we have been progressing.

Activities	December 2023														
	1 week							2 week							
	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Organize the work group															
Select project topic															
Group task planning															
Definition of project objectives															
Draft memory															
Search information															
Library creation															
Pruebas librería															
Library implementation															
Final report writing															
Prepare Presentation															
Presentation															

Our library has over 700 lines of code, and while the operations may seem straightforward, in the programming realm, creating functions like these can become quite complex. They cannot be approached from a purely mathematical standpoint but must be visualized from a highly logical and intricate perspective.

As developers, we believe PyPil can be a great asset for certain software, and we also think it can be a useful tool for novice developers to learn certain basics and take advantage of our library.

However, we do not see this as the end of the project. We plan to keep working on it to continue improving and bring future updates to this library with many more functionalities beyond the polynomial scope.
The idea is for PyPil to keep evolving and encompass all basic mathematical functions, making it more widely used by developers and well-known.

We have been working, and we will continue to do so, to make PyPil as accessible as possible for everyone and to fill a gap in the Python library ecosystem, improving this expanding field.

In summary, we believe PyPil has turned out as we had envisioned and planned. We are satisfied with our results and believe that, with time, it will continue to evolve and improve.

Català

1 Introducció

En el complex àmbit de la programació, es poden trobar diverses funcions dins d'aquest món, el nostre programa aprofundirà en les llibreries, les quals són grans agrupacions de diferents funcions, tenen un ús dins d'aquest món i d'una certa manera poden arribar a ser molt complexes, les llibreries se centren en reduir aquesta complexitat i a facilitar i acréixer la velocitat dels desenvolupadors.

Logo PyPil

Un dels problemes amb els quals es troben molts programadors durant el desenvolupament d'algun programari o aplicació és l'escassetat d'algunes funcions que uns certs llenguatges de programació no et proporcionen, provocant així l'ús de les llibreries que et suplementen el codi basi complint algunes tasques que, sense aquestes llibreries, serien molt més monòtones i complexes.

El nostre treball de recerca treballa amb el llenguatge de programació Python, que malgrat no ser el més eficaç ni el més veloç per a unes certes aplicacions és un dels més utilitzats avui dia, tenint més de 10 milions d'usuaris i un dels pocs llenguatges que proporciona un ús i una interfície senzilla per a l'usuari, amb això el nostre objectiu és arribar a molta més gent i donar molta més accessibilitat a la programació.

Pypil és una llibreria que va dirigida principalment als polinomis i la gran majoria de les seves funcions estan centrades en això, així i tot, tindrà altres funcions relacionades amb la matemàtica bàsica, com l'equació, la qual permetran al programador desenvolupar diferent programari, i d'una certa manera millorar la qualitat i facilitat del desenvolupament.

En resum per culpa de la escassetat de llibreries polinòmiques i de algunes funcions matemàtiques a Python ens em proposat elaborar una llibreria que cobreixi aquet àmbit de manera senzilla.

2 Problema

Python compta amb una àmplia varietat de funcionalitats integrades en la seva base. No obstant això, també existeixen diverses biblioteques desenvolupades per la comunitat que permeten ampliar encara més la seva potència. Algunes d'aquestes eines són creades per empreses o universitats dedicades a la recerca, mentre que unes altres són fruit de la col·laboració entre programadors. Amb freqüència sorgeixen per a donar solució a necessitats puntuals o per a proveir instruments més específics en àrees com l'aprenentatge automàtic, l'anàlisi de dades o la visualització. La riquesa de l'ecosistema de Python es basa en la multitud de biblioteques disponibles, els quals cobreixen una àmplia selecció d'aplicacions. L'elecció d'utilitzar determinades eines dependrà del propòsit i de les idees de cada programador el qual triarà a gust aquestes funcionalitats.

Dit això, el principal problema de molts desenvolupadors i per experiència de molts usuaris és la falta d'unes certes funcions dins d'un codi, per exemple; el codi base de Python no et permet calcular arrels, per això l'Institut d'Algorismes d'Aprenentatge de Mont-real (MILA) va desenvolupar la llibreria coneguda com Math la qual et permet calcular-les, però igual que esmento aquesta hi ha moltes altres llibreries que et permeten això.

Un del problemes de Python es la falta de llibreries per certes operacions matemàtiques bàsiques com podrien ser els polinomis per això el nostre objectiu es crear una llibreria que cobreixi aquet àmbit.

3 Objectius

L'objectiu principal del nostre treball de recerca és desenvolupar una llibreria de Python la qual permetrà a l'usuari treballar amb diferents operacions polinòmiques i de matemàtica bàsica, a més de facilitar i millorar la vida del programador durant el desenvolupament de programari.

D'altra banda, volem que el nostre codi sigui accessible per a tothom per això hem publicat la nostra llibreria en GitHub (https://github.com/) el qual és un repositori en línia per a gestionar projectes i controlar el codi, aquesta web és molt coneguda entre desenvolupadors ja que té una gran base de dades de diferents softwares que permet als programadors cooperar entre si, o bé poder visualitzar diferents codis per al seu ús personal o per a treure un coneixement d'això. El nom del nostre perfil es LibPypil (https://github.com/LibPypil/PyPil/tree/main) i volem que arribi a molts desenvolupadors principiants i avançats per a millorar i facilitar el desenvolupament d'aplicacions concretes.

Durant la setmana que hem estat treballant, ens hem marcat els següents objectius, quant al desenvolupament de la llibreria (El que veuràs a continuació són diferents funcions).

- ✓ PolyAddition (suma) - Finalitzat el 11/12/23
- ✓ PolySubstraction (resta) – Finalitzat el 12/12/23
- ✓ PolyMultiplication (multiplicació) – Finalitzat el 12/12/23
- ✓ PolyDivision (divisió) – Finalitzat el 12/12/23
- ✓ PolyRoot (arrel) - Finalitzat el 13/12/23
- ✓ Ruffini - Finalitzat el 12/12/23
- ✓ GeneralAlgorithm (algoritme general) – Finalitzat el 12/12/23
- ✓ EQ1 – En desenvolupament
- ✓ EQ2 - Finalitzat el 13/12/23
- ✓ EQ3+ - Finalitzat 13/12/23
- ✓ Readme.txt - Finalitzat el 14/12/23

I en quant a memòria y pòster ens em marcat los següents objectius.

- ✓ Portada – Finalitzat el 12/12/23
- ✓ Índex - Finalitzat el 14/12/23
- ✓ Introducció - Finalitzat el 11/12/23
- ✓ Plantejament del problema – Finalitzat el 11/12/23
- ✓ Objectius – Finalitzat el 12/12/23
- ✓ Marco teòric (documentació codi) - Finalitzat el 13/12/23
- ✓ Implementació/ Desenvolupament – Finalitzat el 13/12/23
- ✓ Anàlisis - Finalitzat el 14/12/23

A continuació, deixo adjunto l'horari de treball del grup.

Data	Tasques
11/12/2023	Organitzar el calendari i el treball en els mètodes 5W i SCQA
12/12/2023	Desenvolupament de codi, pòster i memòria (6h)
13/12/2023	Desenvolupament de codi, pòster i memòria (6h)
14/12/2023	Desenvolupament del codi, pòster i memòria/Possible aplicació (5h)
15/12/2023	Acabar els objectius pendents i preparar l'exposició oral dilluns

18/12/2023	Presentació
19/12/2023	Entrega de premis

4 Marc teòric

A continuació, podreu veure diferents conceptes els quals hem necessitat per desenvolupar el treball d'investigació o bé alguns coneixements necessaris per entendre correctament la investigació.
D'altra banda, també es mostraran alguns conceptes de la nostra llibreria "PyPil" i com funcionen i què fan algunes de les seves funcions.
Per començar, parlaré del que és un llenguatge de programació i de la funció que té dins d'un programari. Un llenguatge és un conjunt de paraules especialitzades i de regles gramaticals específiques que ordenen a un

dispositiu realitzar activitats com mostrar per pantalla un text o una imatge.

Aquests llenguatges possibiliten aplicacions, pàgines web, IA (Intel·ligència Artificial), xarxes neuronals i fins i tot aplicacions d'investigació i desenvolupament científic.

A més d'aquestes paraules específiques que tenim com a base en tot llenguatge de programació, hi ha certs llenguatges que et permeten l'ús de llibreries. Una llibreria és un conjunt de funcions i mètodes prescrits que algú més ha creat perquè tu els utilitzis en el teu propi codi... Una altra manera d'explicar-ho, si no tens coneixement en el món del desenvolupament del programari, és imaginar-te que estàs construint una casa i necessites eines per fer-ho. En aquest cas, la "llibreria" seria com un conjunt d'eines especialitzades que ja estan fetes i llestes per utilitzar en el teu projecte de construcció.

L'estructura d'una llibreria de Python convencional es basa en:

- Carpeta principal del paquet
- Un fitxer que indica que aquesta carpeta és un paquet de Python normalment anomenat **__init__.py**
- Un mòdul que conté totes les funcions que hi ha dins de la llibreria
- Algunes proves per a la llibreria
- Un fitxer que configura tota la llibreria i un altre que configura la instal·lació d'aquesta
- Fitxer que proporciona tota la informació bàsica sobre la llibreria i el seu ús

La nostra llibreria "PyPil" està especialitzada en l'ús de polinomis i monomis i conté les següents funcions:

- PolyAddition: Al detectar dos monomis de grau similar els suma.

- PolySubstraction: Al detectar dos monomis de grau similar els resta.

- PolyMultiplication: La multiplicació es divideix en dos camins. 1r el polinomi es multiplica per un terme independent. 2n el polinomi es multiplica per un altre polinomi de la manera clàssica.

- PolyDivision: Aquesta funció es divideix en dues maneres de dividir polinomis.

    GeneralAlgorithm: Aquesta funció imita el procediment que un realitza en realitzar un algoritme general és a dir, és com una traducció, però feta de manera que la màquina entengui.

    Ruffini: Aquesta funció només s'executarà si en la divisió un dels dos polinomis és de 1r grau. El que fa la funció és agafar el primer monomi i multiplicar-lo pel terme independent del segon polinomi; després el suma amb el següent monomi i es repeteix fins al final.

- PolyRoot: Amb un polinomi donat, aquesta funció treu el factor comú del polinomi (si és que es pot), trobarà els divisors del terme independent, i anirà provant Ruffini perquè quan sigui de grau 2, aplicar la fórmula quadràtica, retornant totes les arrels.

- PolyPow: En aquesta funció un polinomi es multiplica per si mateix "n" vegades.

- EQ1 (equacions de primer grau) - Una equació de primer grau consisteix en una seqüència de nombres que poden anar acompanyats d'una incògnita "x", la qual només pot estar elevada a 1. Dit això, aquest algorisme es divideix en 3 subseccions. 1 - Equacions sense fraccions, 2 - Equacions amb una fracció i la tercera és - Més d'una fracció.

    1 - Aquesta és la secció sense fraccions, el que fa la funció primer és diferenciar entre valors amb i sense "x", després d'això els canvia de costat per aïllar la "x" el màxim possible, després d'això divideix els dos costats de l'equació i acabem amb el valor de "x".

2 - Aquesta és la secció amb una fracció. Aquesta funció comença classificant els tipus que hi ha dins de la subsecció (els quals són 4), els 4 tipus són: on la "x" és el numerador d'una fracció, on la "x" és denominador d'una fracció i hi ha més d'una "x", on la "x" és el denominador d'una fracció i no hi ha més "x" i finalment on hi ha "x" tant en numerador com en denominador.

2.1 - Aquestes equacions són les que la "x" està en el numerador.

2.2 - Aquestes equacions són les que la "x" està en el denominador i hi ha més d'una "x", aquestes es resolen multiplicant tota l'equació per "x", la qual cosa provoca una equació de segon grau que es resol amb l'equació quadràtica.
2.3 - Aquestes equacions són en les quals la "x" està en el denominador de la fracció, però no hi ha més "x", aquestes es resolen multiplicant tota l'equació per "x" després d'això s'aïllen totes les "x" i s'iguala a 0.

2.4 - Aquestes fraccions són les que tenen x tant en el numerador com en el denominador. Aquestes "x" s'esborraran entre elles i donaran lloc a 3 tipus diferents:

2.4.1 - Si és que la "x" és més gran en el numerador, significa que hauràs de fer el tipus 2.1.

2.4.2 - Si la "x" és més gran en el denominador que en el numerador i no hi ha més "x", farem el tipus 2.3.

2.4.3 - Però si en el cas anterior sí que tenim més " x", faríem el cas 2.2.

3 - Aquest tipus són les equacions que tenen més d'1 fracció. Hi ha 3 casos.

3.1 - La x està en el numerador i hi ha un número en el numerador. Per resoldre-la hauràs de fer el m.c.m dels denominadors i multiplicar els denominadors.

3.2 - Les x estan en els denominadors. Es fa el m.c.m dels denominadors (el m.c.m és més complex) i es multiplica pels numeradors.

3.3 - En uns hi ha x en el numerador, i en d'altres en el denominador. Es combinen els mètodes anteriors per resoldre-la.

3.4 - Aquest cas és per si hi ha "x" tant en el numerador com en el denominador. En aquest cas es dividiran entre sí i depenent de qui sigui més gran, es farà el tipus 3.1, el 3.2 o el 3.3.

- EQ2 (equacions de segon grau)- En aquesta funció, tu introdueixes tres valors numèrics (a, b, c), amb aquests valors comença l'equació quadràtica, comença per calcular el discriminant de l'equació per veure els possibles resultats, després d'això es divideix en dos camins l'equació, el de sumar i el de restar, després d'això es suma/resta la b i després es divideix tot entre 2, això donarà els dos resultats desitjats.

- EQ3 (equacions de tercer grau o més) - En aquesta funció, hem cobert 2 tipus d'equacions de grau més que 2. Les que tenen només una x elevada a un nombre i un terme independent, i les que tenen més d'una x amb diferents elevats. Per a les que tenen diferents elevats, hem aplicat l'algoritme creat anteriorment de les arrels, i d'allà ja n'hem tret les respostes. I per a les altres que queden, es fa l'arrel N al terme independent.

L'àlgebra computacional és un paquet de programari que permet la manipulació d'expressions matemàtiques de manera semblant a com si fos a mà.

Un monomi és una expressió matemàtica simple que consta d'un coeficient multiplicat per una variable (o més) elevada a un exponent no negatiu; d'altra banda, un polinomi és un conjunt de monomis que es sumen o bé realitzen una operació entre ells.

A continuació, podrem veure com funcionen les operacions polinòmiques bàsiques:

- La suma de polinomis implica combinar termes semblants.

Sumes els coeficients dels termes que tenen la mateixa variable i exponent. Per exemple, en la suma (2x^2 + 3x + 5) i (4x^2 - 2x + 1), combines els termes corresponents: ((2x^2 + 4x^2) + (3x - 2x) + 5 +1), resultant en (6x^2 + x + 6).

La resta de polinomis també implica combinar termes semblants, però en aquest cas, restes els coeficients.

La multiplicació de polinomis implica distribuir cada terme del primer polinomi sobre cada terme del segon polinomi i després combinar
La divisió de polinomis implica dividir el polinomi divisor en el polinomi dividend, de manera similar a la divisió numèrica. El resultat és el quocient i, possiblement, un residu.

Seguidament es mostrarà com es troba l' arrel i factoritzen els polinomis. Per trobar les arrels d'un polinomi, és a dir, els valors de la variable que fan que el polinomi sigui igual a zero, pots utilitzar diferents mètodes. Un dels mètodes més comuns d'utilitzar és el de Ruffini i la formula quadràtica. Per poder factoritzar un polinomi necessitem el mètode Ruffini (que hem realitzat prèviament) i la formula quadràtica. Amb aquests mètodes, si el polinomi és major de grau 2, aplicarem Ruffini, fins que ens surti que el polinomi sigui de grau 2, quan passi, aplicarem la formula quadràtica. Amb tots aquests resultats, els unirem en una llista de polinomis i els ho retornarem a l'usuari.
Per resoldre equacions de primer grau fàcilment en un sol paràgraf, segueix aquests passos. Pren una equació com 2x+5=11, per exemple. Primer, resta 5 a tots dos costats per aïllar el terme amb la variable: 2x=6. Després, divideix tots dos costats per 2 per trobar el valor de la variable: x=3. Per verificar la teva resposta, substitueix x=3 novament a l'equació original per assegurar-te que tots dos costats siguin iguals, en aquest cas, 2(3)+5=11. Si tots dos costats són iguals, has resolt l'equació.

Per resoldre equacions de segon grau, hi ha diversos mètodes, però el principal i més utilitzat és la fórmula quadràtica:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

La fórmula quadràtica dona dos resultats diferents, els quals has de substituir per verificar si són solucions reals. Si la substitució dona com a resultat 0, la solució és correcta.

Les equacions de tercer grau, també conegudes com a equacions cúbiques, són expressions algebraiques en les quals la variable està elevada a l'exponent tres. Tenen la forma general ax^3 + bx^2 + cx + d = 0, on (a), (b), (c) i (d) són coeficients constants, i (x) és la variable desconeguda. Resoldre una equació cúbica implica trobar els valors de \(x\) que satisfan la igualtat. Tot i que hi ha fórmules específiques per resoldre equacions de tercer grau, també es poden utilitzar mètodes numèrics o factorització. Aquestes equacions poden tenir fins a tres solucions reals o complexes, i el seu estudi és fonamental en àlgebra i anàlisi matemàtica.

GitHub és una plataforma de desenvolupament col·laborativa que permet als equips de desenvolupadors gestionar i compartir el seu codi font de manera còmoda. Funciona utilitzant el sistema de control de versions distribuït Git. Els desenvolupadors poden crear repositoris per emmagatzemar els seus projectes, i cada canvi realitzat en el codi es registra com a "commit". GitHub facilita la col·laboració permetent que diverses persones treballin simultàniament en un projecte, fusionin canvis de diferents branques i facin un bon seguiment de l'historial de versions. A més, proporciona eines com ara sol·licituds d'extracció (pull requests) per facilitar la revisió i la integració del codi, així com funcions per a la gestió de problemes, seguiment de projectes i desplegament continu. En resum, GitHub simplifica el procés de desenvolupament de programari, promovent la col·laboració efectiva i la gestió eficient del codi font.

5 Implementació i Desenvolupament

La nostra llibreria es pot arribar a utilitzar per a moltes coses, ja que permet a la gent poder resoldre les operacions polinòmiques gràcies al fet que hem implementat diferents funcions a la nostra llibreria, gràcies a totes les funcions que hem fet es podrien fer les següents coses a partir de les nostres llibreries

(tenint en compte que només en formés part, no que es pot crear a partir del nostre): es pot crear una aplicació bastant semblant a la de Photomath que escaneja l'operació, i a base de l'escanejat,
ordeni la informació i depenent de les dades obtingudes dels polinomis de manera senzilla i ràpida, un altre exemple podria ser un programari d'àlgebra computacional, l'àlgebra computacional permet a l'usuari realitzar àlgebra de manera similar a com si fos en paper i llapis. L' objectiu de la PyPil és millorar la qualitat i l' efectivitat d' algunes aplicacions com les esmentades anteriorment, de tal manera que millori així la velocitat de desenvolupament d' aquest tipus d' aplicacions i que millori la disposició dels polinomis en el desenvolupament de programari amb Python. Un altre ús més comú que se li podria donar seriosa per ajudar els principiants en l'aprenentatge del llenguatge i ampliar les opcions dels novaiorquesos en crear aplicacions.

- La nostra llibreria està publicada i oberta per a totes les persones que requereixin del seu ús, la podràs trobar al nostre GitHub (https://github.com/LibPypil/PyPil/tree/main).

- Un dels nostres objectius era arribar a les màximes persones possibles, i com a desenvolupadores compartir el nostre treball amb els altres per això hem implementat en el nostre treball "GitHub" una web de desenvolupadors que ens permet arribar a les màximes persones possibles a més de donar-li accessibilitat a tot que vulgui utilitzar aconseguint així portar el nostre projecte fora de les aules i donar-livisibilitat.

- A continuació, es mostrarà el desenvolupament que han tingut algunes funcions de la nostra llibreria "PyPil", com el seu plantejament i com ha anat sent modificada segons els inconvenients que ens hem trobat durant el desenvolupament d'aquestes.

- Al nostre plantejament ha estat que en introduir els polinomis (amb els graus ben posats) se sumin els graus similars i el resultat sigui imprès per pantalla, amb la suma no hem tingut cap problema i tot ha sortit com teníem plantejat.

- El nostre plantejament ha estat que en introduir els polinomis (amb els graus ben posats) es resten els graus similars i el resultat sigui imprès per pantalla, amb la resta no hi ha hagut cap problema

ja que el codi és gairebé el mateix que el de la suma únicament hem modificat alguns operadors i tot ha funcionat com esperàvem.

Aquesta funció està dividida en dues parts (polinomi x número, polinomi x polinomi), durant el desenvolupament només vam tenir problemes amb la segona (polinomi x polinomi), ja que teníem bastants inconvenients amb la multiplicació dels graus, un dels errors més gran durant el desenvolupament d'aquesta funció va ser que en comptes de multiplicar els graus, els sumava. La solució a aquest problema va ser canviar uns operadors matemàtics al final de la funció, vam haver de canviar un + per * (signe de multiplicació a Python). Un error bastant tou que complico el desenvolupament d'aquesta funció però que puguem solucionar a temps.

- Division (divisió): Aquesta funció es divideix en dues parts.

  Ruffini (Ruffini): Per plantejar aquesta funció, nosaltres pensem a dividir-les en funcions més simples, ja que el Ruffini és una barreja de sumes i multiplicacions complexes. Molts dels errors que vam cometre al Ruffini van ser a causa del polinomi no s'invertia com el programa necessitava i, que el residu de l'operació apareixia com a part del polinomi resultant. Per solucionar el problema del residu, creem una variable que al final aïllava el residu i l'imprimeix a part.

  General Algorithm (Algoritme general): Aquest ha estat sens dubte l'algoritme més difícil de programar, el meu plantejament inicial va ser: primer fer la primera iteració que fem al paper, trobar el múltiple, multiplicar el que ens doni i restar-lo amb el polinomi inicial. Indentar (tabular) un bloc de codi mal, cosa que va retardar el progrés una mica. Un cop tenim això, el següent va ser crear un loop que anés repetint el mateix procés fins que el polinomi final sigui més petit que el divisor (seria la resta). Un cop fet això, li retornem el coeficient i la resta a l'usuari.

- PolyPow (Exponenciació): Aquesta funció només té 100.000 euros (polinomi x sencer), en aquesta funció no vam tenir cap error ni greu ni lleu, el que fa és multiplicar un polinomi sobre si mateix "n" vegades. No hem tingut cap problema durant el desenvolupament d'aquest i tot ha sortit com ho plantegem en un principi.

- Polyroot (Arrels): Ho vaig plantejar igual que nosaltres ho fem en el paper, primer extreure factor comú (no va ser un problema), fer Ruffini (va ser més complex ja que calia provar els divisors, i el loop em va donar bastants problemes) i, finalment, quan el polinomi és de grau 2, apliqués la funció d'EQ2, que ja teníem anteriorment feta.

- EQ1 (Equacions de 1r grau): Una equació de primer grau consisteix en una seqüència de números que poden anar acompanyats d'una incògnita "x", la qual només pot estar elevada a 1 Dit això, aquest algoritme es divideix en 3 subseccions. 1- Equacions sense fraccions, 2 - Equacions amb una fracció i la tercera és - Més d'una fracció.

    1- Aquesta és la secció sense fraccions. El que fa la funció primer és diferenciar entre valors amb i sense "x". Després d'això, els canvia de banda per aïllar "x" el màxim possible. Després d'això, divideix els dos costats de l'equació i acabem amb el valor de "x".

    2- Hi ha 1 fracció:

        2.1- Hi ha una x en el numerador, només calia multiplicar el denominador per tots els monomis i llest.

        2.2- Aquesta va ser més complex, ja que hi havia x en els denominadors, el que pensem és multiplicar tota l'equació per x, i després aplicar una equació de segon grau (ja que, si hi ha més x, si o si hi haurà un x^2).

        2.3- Aquet és com l'anterior, però més fàcil, ja que no hi havia més x, vam fer el mateix, però en comptes de fer una EQ2, vam fer una funció recursiva amb una equació normal.

        2.4- Aquest cas es tracten d'aquestes fraccions que tenen x tant en el numerador i denominador, la qual cosa pensem aquí, va ser realitzar unes condicions perquè ens classifiqués si el numerador és més gran que el denominador, si això passa, es farà el tipus 2.1. En el cas que no es compleixi la condició, es comprovarà si hi ha més x, si n'hi ha, em farà la 2.2, si no, la 2.3.

3 - Hi ha més d'1 fracció.

3.1- En aquest cas, pensem que amb simplement fer el m.c.m dels denominadors (que seran sol números) era la manera més fàcil.

3.2- Est va ser més difícil de programar ja que les x estaven en el denominador. L'error que vaig tenir és no pensar a fer el m.c.m amb x, ja que és diferent als de sempre. Simplement vaig haver d'arreglar això, implementar el mòdul multiplicació per a multiplicar polinomis i llest.

3.3. Aquest va ser relativament senzill, simplement calia usar els 2 últims mètodes per a solucionar els trossos d'equació, no va ser cap problema.

3.4. Aquest és semblant al 2.4, depenent d'on la x queda més gran, es caldria executar la 3.3 que està també executés la 3.1 i la 3.2.

.

- EQ2 (Equacions de 2n grau): En aquesta funció, nosaltres plantegem usar la formula quadràtica del revés (començant des de dins de l'equació), primer calculem el discriminant (que és el que hi ha dins de l'arrel), gràcies a la discriminant podíem saber la quantitat de solucions que té una equació de segon grau, si és negatiu l'equació no té solucions, si és zero té 1 solució i si és positiu té 2 solucions.

- EQ3 (Equacions de 3r + grau): Ho hem plantejat de 2 maneres, el cas 1 (només hi ha un terme amb part literal) i el cas 2 (n'hi ha més d'1 terme amb part literal). Per al cas 2, la solució era fàcil, simplement era aplicar l'algoritme de l'arrel anteriorment creat, cosa que no ha causat cap problema.
  Per al cas 1 sí que he hagut de trencar-me el cap, ja que Python no té cap mètode inclòs que et faci arrels cubiques o de més. Amb una mica d'investigació, m'adono que en mates ens van explicar que una arrel quadrada és el mateix que aquest número, elevat a una fracció (on la base és com està elevat el número, i el denominador és l'índex de l'arrel). Com en Python sí que té un

mètode per elevar, apliqui aquesta teoria i va funcionar.

6 Anàlisi

PyPil és una llibreria de Python dedicada als polinomis i a la resolució d'equacions. Es va desenvolupar durant una setmana amb unes 28h de treball i pretén donar eines als desenvolupadors de programes i facilitar als programadors principiants l'ús d'operacions polinòmiques.

L'objectiu de Pypil és ser una llibreria la qual ajudi els desenvolupadors, i després de tota la setmana creiem fermament que compleix amb la seva funció, després de tot el desenvolupament i la seva finalització a més de la petita demo que hem desenvolupat hem pogut comprovar que soluciona el problema que teníem en ment i no tan sols això, sinó que em agafat molts coneixements sobre diferents àmbits de aquet mon. Encara que, algunes funcions no estan finalitzades, continuarem treballant en elles i desenvolupant-les per poder publicar-les en una versió oficial en el futur.

A més, a continuació, deixo el progrés del nostre treball i com hem anat progressant.

Activitats	Desembre 2023														
	1 Setmana							2 setmanes							
	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Organitzar el grup de treball															
Seleccioneu el tema del projecte															
Planificació de tasques en grup															
Definicio de objectius del projecte															
Esborrany de memòria															
Cerca informació															
Creació de biblioteques															
Probes de la llibreria															
Implementació de la biblioteca															
Redacció de l'informe final															
Prepara la presentació															
Presentació															

La nostra biblioteca té més de 700 línies de codi, i tot i que les operacions poden semblar senzilles, en l'àmbit de la programació pot arribar a ser molt

complicat crear funcions com aquestes. No es poden abordar des d'un punt de vista purament matemàtic, sinó que cal visualitzar-les des d'una perspectiva lògica i complexa.

Com a desenvolupadors creiem que PyPil pot ser una gran ajuda per a cert programadors i també pensem que PyPil pot ser una gran eina per a alguns desenvolupadors principiants per a que aprenguin certes bases o que bé treguin profit de la nostra llibreria.

D'altra banda, no creiem que aquest sigui la fi d'aquest projecte, sinó que seguirem treballant en el per seguir-lo millorant i poder portar en un futur actualitzacions d'aquesta llibreria amb moltes més funcionalitats fora de l'àmbit polinòmic, la idea de PyPil és seguir evolucionant i arribar a englobar totes les funcions matemàtiques bàsiques de tal manera que molts més desenvolupadors la facin servir per als seus codis i sigui molt més coneguda.

Hem estat treballant i seguirem fent-ho perquè PyPil sigui el més accessible possible per a tothom, i perquè cobreixi un buit dins de l'ecosistema de llibreries de Python per així millorar tot aquest mon que a poc a poc es va fent més gran.

En resum, creiem que PyPil ha sortit tal com teníem en ment i planejat, estem contents amb els nostres resultats i creiem que amb el temps podrà continuar evolucionant i millorant.

Castellano

1 Introducción

En el complejo ámbito de la programación, hay varias funciones en este mundo, nuestro programa profundizará en las librerías, grandes agrupaciones de distintas funciones, que tienen un uso dentro de este mundo y pueden ser muy complejas, las librerías se centran en reducir esta complejidad y en facilitar y acrecentar la velocidad de los desarrolladores.

Logo PyPil

Uno de los problemas de muchos programadores durante el desarrollo de un software o aplicación es la escasez de funciones que ciertos lenguajes de programación no te proporcionan, provocando así el uso de las librerías que te suplementan el código base cumpliendo tareas que, sin estas, serian mucho más monótonas y complejas.
Nuestro trabajo de Investigación trabajara con el lenguaje de programación Python, que pese a no ser el más eficaz ni el más veloz para ciertas aplicaciones es uno de los más usados hoy en día, teniendo más de 10 millones de usuarios y uno de los pocos lenguajes que proporciona un uso y una interfaz sencilla para el usuario, con esto nuestro objetivo es llegar a mucha más gente y dar mucha más accesibilidad a la programación.
Pypil es una librería que va dirigida principalmente a los polinomios y la gran mayoría de sus funciones están centradas en esto, aun así, tendrá otras funciones relacionadas con la matemática básica, como la ecuación, la cual permitirán al programador desarrollar distinto software, y de cierta manera mejorar la calidad y facilidad del desarrollo.

En resumen, debido a la escasez de bibliotecas polinómicas y algunas funciones matemáticas en Python, nos hemos propuesto elaborar una biblioteca que cubra este ámbito de manera sencilla.

2 Problema

Python tiene varias funcionalidades integradas en su base. También existen diversas bibliotecas desarrolladas por la comunidad que permiten ampliar su potencia. Algunas de estas herramientas las crean empresas o universidades dedicadas a la investigación, y otras son por la colaboración de programadores. Con frecuencia surgen para dar solución a necesidades puntuales o para proveer instrumentos más específicos en áreas como el aprendizaje automático, el análisis de datos o la visualización. La riqueza del ecosistema de Python se basa en la multitud de bibliotecas disponibles, los cuales cubren una amplia selección de aplicaciones. La elección de utilizar determinadas herramientas dependerá del propósito y de las ideas de cada programador el cual elegirá a gusto estas funcionalidades.

Dicho esto, el principal problema de muchos desarrolladores y por experiencia de muchos usuarios es la falta de ciertas funciones dentro de un código, por ejemplo; el código base de Python no te permite calcular raíces, por eso el Instituto de Algoritmos de Aprendizaje de Montreal (MILA) desarrolló la librería conocida como Math la cual te permite calcularlas, pero al igual que menciono esta hay otras muchas librerías que te permiten esto.

Uno de los problemas de Python es la falta de bibliotecas para ciertas operaciones matemáticas básicas, como los polinomios. Por lo tanto, nuestro objetivo es crear una biblioteca que abarque esta área.

3 Objetivos

El objetivo principal de nuestro trabajo de investigación es desarrollar una librería de Python la cual permitirá al usuario trabajar con distintas operaciones polinómicas y de matemática básica, además de facilitar y mejorar la vida del programador durante el desarrollo de software.

Por otro lado, queremos que nuestro código sea accesible para todo el mundo por eso hemos publicado nuestra librería en GitHub (https://github.com/), un repositorio online para gestionar proyectos y controlar el código, esta web es muy conocida entre desarrolladores, ya que tiene una gran base de datos de softwares que permite a los programadores

cooperar entre sí, o visualizar distintos códigos para usarlo o para conocerlo. El nombre de nuestro perfil es *LibPypil;* (*https://github.com/LibPypil/PyPil/tree/main*) y queremos que llegue a muchos desarrolladores principiantes y avanzados para mejorar y facilitar el desarrollo de aplicaciones concretas.

Durante la semana que hemos estado trabajando, nos hemos marcado los siguientes objetivos, en cuanto al desarrollo de la librería (Lo que veras a continuación son distintas funciones).

- ✓ PolyAddition (suma) - Finalizado el 11/12/23
- ✓ PolySubstraction (resta) – Finalizado el 12/12/23
- ✓ PolyMultiplication (multiplicación) – Finalizado el 12/12/23
- ✓ PolyDivision (división) – Finalizado el 12/12/23
- ✓ PolyRoot (raíz) - Finalizado el 13/12/23
- ✓ Ruffini - Finalizado el 12/12/23
- ✓ General Algorithm (algoritmo general) – Finalizado el 12/12/23
- ✓ PolyPow (Potencia) - Finalizado el 13/12/23
- ✓ EQ1 - Finalizado el 14/12/23
- ✓ EQ2 - Finalizado el 13/12/23
- ✓ EQ3+ - Finalizado 13/12/23
- ✓
- ✓ Readme.txt - Finalizado el 14/12/23

Y en cuanto a memoria y poster nos hemos marcado los siguientes objetivos.

- ✓ Portada – Finalizado el 12/12/23

- ✓ Índex - Finalizado el 14/12/23
- ✓ Introducción - Finalizado el 11/12/23
- ✓ Planteamiento del problema – Finalizado el 11/12/23
- ✓ Objetivos – Finalizado el 12/12/23
- ✓ Marco teórico (documentación código) - Finalizado el 13/12/23
- ✓ Implementación/ Desenvolvimiento – Finalizado el 13/12/23
- ✓ Análisis - Finalizado el 14/12/23

A continuación, dejo adjunto el horario de trabajo del grupo.

Fecha	Tareas
11/12/2023	Organizar el calendario y trabajar en los métodos 5W's y SCQA
12/12/2023	Desarrollo de código, póster y memoria (6h)
13/12/2023	Desarrollo de código, póster y memoria (6h)
14/12/2023	Desarrollo de código, póster y memoria/Posible aplicación (5h)
15/12/2023	Acabar objetivos pendientes y preparar la presentación oral del lunes

18/12/2023	Presentación
19/12/2023	Entrega de Premios

4 Marco teórico

A continuación, podréis ver distintos conceptos los cuales hemos necesitado para desarrollar el trabajo de investigación o bien algunos conocimientos necesarios para poder entender correctamente la investigación.
Por otra parte, también se mostrará algunos conceptos de nuestra librería "PyPil" y de cómo funcionan y que hacen algunas de sus funciones.
Para comenzar hablare de lo que es un lenguaje de programación y de la función que tiene dentro de un software. Un lenguaje es un conjunto de palabras especializadas y de unas reglas gramaticales específicas que

ordenan a un dispositivo realizar actividades como mostrar por pantalla un texto o una imagen, estos lenguajes posibilitan aplicaciones, páginas web, IA (Inteligencia Artificial), redes neuronales e incluso aplicaciones de investigación y desarrollo científico.

Además de estas palabras específicas que tenemos como base en todo lenguaje de programación, hay ciertos lenguajes que te permiten el uso de librerías, una librería es un conjunto de funciones y métodos prescritos que alguien más ha creado para que tú los utilices en tu propio código... Otra manera de explicarlo si no tienes conocimiento en el mundo del desarrollo del software es imaginarte que estás construyendo una casa y necesitas herramientas para hacerlo. En este caso, la "librería" sería como un conjunto de herramientas especializadas que ya están hechas y listas para usar en tu proyecto de construcción.

La estructura de una librería de Python convencional se basa en:

- Carpeta principal del paquete
- Un archivo que indica que esa carpeta es un paquete de Python normalmente llamado init.py
- Un módulo el cual contiene todas las funciones que hay dentro de la librería
- Algunas pruebas para la librería
- Un archivo que configura toda la librería y otro que configura la instalación de esta
- Archivo que proporciona toda la información básica sobre la librería y su uso

Nuestra librería “PyPil” está especializada en el uso de polinomios y monomios y contiene las siguientes funciones:

- PolyAddition: Al detectar dos monomios de grado similar los suma.

- PolySubstraction: Al detectar dos monomios de grado similar los resta.

- PolyMultiplication: La multiplicación se divide en dos caminos. 1º el polinomio se multiplica por un término independiente. 2º el polinomio se multiplica por otro polinomio de la manera clásica.

- PolyDivision: Esta función se divide en dos maneras de dividir polinomios.

GeneralAlgorithm: Esta función imita el procedimiento que uno realiza al realizar un algoritmo general es decir es como una traducción, pero hecho de tal manera que la maquina entienda.

Ruffini: Esta función solo se ejecutará si en l división uno de los dos polinomios es de 1er grado. Lo que hace la función es coje el primer monomio y lo multiplica por el termino independiente del segundo polinomio; después lo suma con el siguiente monomio y se repite hasta el final.

- PolyRoot: Con un polinomio dado, esta función sacará factor común del polinomio (si es que se puede), encontrará los divisores del término independiente, e ira probando Ruffini para que cuando sea de grado 2, aplicar la formula cuadrática, devolviendo todas las raíces.

- PolyPow: En esta función un polinomio se multiplica por sí mismo “n” veces.

- EQ1 (ecuaciones de primer grado) - Una ecuación de primer grado consiste en una secuencia de números que pueden ir acompañados de una incógnita “x” la cual solo puede estar elevada a 1, dicho esto este algoritmo se divide en 3 subsecciones. 1 - Ecuaciones sin fracciones, 2 - Ecuaciones con una fracción y la tercera es - Mas de una fracción.

1- Esta es la sección sin fracciones, lo que hace la función primero es diferenciar entre valores con y sin “x”, después de esto los cambia de lado para aislar la “x” lo máximo posible, después de esto divide los dos lados de la ecuación y acabamos con el valor de "x”.

2- Esta es la sección con una fracción. Esta función empieza clasificando los tipos que hay dentro de la subsección (los cuales son 4), los 4 tipos son: donde la “x” es el nominador de una fracción, donde la “x” es denominador de una fracción y hay más de una “x”, donde la “x” es el denominador de una fracción y no hay más “x” y por último donde hay “x” tanto en nominador como en denominador.

2.1- Estas ecuaciones son las que la “x” está en el nominador. En estos ejemplos nuestro algoritmo multiplica toda la ecuación por el denominador de la fracción, después de esto la ecuación y transforma en una de tipo 1 y se soluciona siguiendo los mismos pasos.

2.2- Estas ecuaciones son las que la “x” están en el denominador y hay más de una “x”, estas se resuelven multiplicando toda la ecuación por “x” lo cual causa una ecuación de segundo grado la cual se soluciona con la ecuación cuadrática.

2.3- Estas ecuaciones son en las que la “x” está en el denominador de la fracción, pero no hay más “x”, estas se resuelven multiplicando toda la ecuación por “x” después de esto se aíslan todas las “x” y se iguala a 0.

2.4 - Estas fracciones son las que tienen x tanto en el numerador y en el denominador. Estas “x” se tacharán entre ellas y darán a lugar a 3 tipos distintos:

2.4.1 - Si es que la “x” es más grande en el numerador, significa que habrá que hacer el tipo 2.1

2.4.2 - Si la “x” es más grande en el denominador que en el numerador y no hay más “x”, haremos el tipo 2.3

2.4.3 - Pero si en el caso anterior sí que tenemos más “x”, haríamos el caso 2.2.

3 - Este tipo son las ecuaciones que tienen más de 1 fracción. Hay 3 casos.

3.1 - La x está en el numerador y hay un numero en el numerador. Para resolverla habrá que realizar el m.c.m de los denominadores y multiplicar los denominadores.

3.2 - Las x están en los denominadores. Se hace el m.c.m de los denominadores (el m.c.m es más complejo) y se multiplica por los numeradores.

3.3 - En unos hay x en el numerador, y en otros en el denominador. Se

combinan los métodos anteriores para resolverla.

3.4 - Este caso es por si hay “x” tanto en el numerador y en el denominador. En este caso se dividirán entre sí y dependiendo de quién sea más grande, se hará el tipo 3.1, el 3.2 o el 3.3.

- EQ2 (ecuaciones de segundo grado)- En esta función, tu introduces tres valores numéricos (a, b, c), con estos valores empieza la ecuación cuadrática, empieza por calcular el discriminante de la ecuación para ver los posibles resultados, después de esto se divide en dos caminos la ecuación, el de sumar y el de restar, después de esto se suma/resta la b y entonces se divide todo entre 2, esto dará los dos resultados deseados.

- EQ3 (ecuaciones de tercer grado o más) - En esta función, hemos cubierto 2 tipos de ecuaciones de grado más que 2. Las que tienen solo una x elevada a un número y un término independiente, y las que tienen más de una x con distintos elevados. Para las que tienen distintos elevados, hemos aplicado el algoritmo creado anteriormente de las raíces,
y de allí ya sacamos las respuestas. Y para las otras que quedan, se hace la raíz N al termino independiente.

La algebra computacional es un paquete de software que permite la manipulación de expresiones matemáticas de forma semejante a como si fuera a mano.

- Un monomio es una expresión matemática simple que consta de un coeficiente multiplicado por una variable (o más) elevada a un exponente no negativo, por otro lado, un polinomio es un conjunto de monomios que se suman o bien realizan una operación entre sí. A continuación, podremos ver cómo funcionan las operaciones polinómicas básicas:

La *suma de polinomios* implica combinar términos semejantes. Sumas los coeficientes de los términos que tienen la misma variable y exponente. Por ejemplo, en la suma (2x^2 + 3x + 5) y (4x^2 - 2x + 1), combinas los términos correspondientes: ((2x^2 + 4x^2) + (3x - 2x) + 5 + 1), resultando en (6x^2 + x + 6).

La *resta de polinomios* también implica combinar términos semejantes, pero en este caso, restas los coeficientes.

La *multiplicación de polinomios* implica distribuir cada término del primer polinomio sobre cada término del segundo polinomio y luego combinar
La *división de polinomios* implica dividir el polinomio divisor en el polinomio dividendo, de manera similar a la división numérica. El resultado es el cociente y, posiblemente, un residuo.

Seguidamente se mostrará como se encuentra la raíz y factorizan los polinomios. Para encontrar las raíces de un polinomio, es decir, los valores de la variable que hacen que el polinomio sea igual a cero, puedes utilizar diferentes métodos. Uno de los métodos más comunes de utilizar es el de Ruffini y la formula cuadrática. Para poder factorizar un polinomio necesitamos el método Ruffini (que hemos realizado previamente) y la formula cuadrática. Con estos métodos, si el polinomio es mayor de grado 2, aplicaremos Ruffini, hasta que nos salga que el polinomio sea de grado 2, cuando pase, aplicaremos la formula cuadrática. Con todos estos resultados, los uniremos en una lista de polinomios y se lo devolveremos al usuario.
Para poder resolver ecuaciones de primer grado en un solo párrafo y de manera muy fácil se tienen que seguir los siguientes pasos. Toma una ecuación como *2x+5=112x*+5=11, por ejemplo. Primero, resta 5 de ambos lados para aislar el término con la variable: *2x*=6. Luego, divide ambos lados por 2 para encontrar el valor de la variable: x=*3*. Para poder verificar tu respuesta sustituye x=3 de nuevo en la ecuación original para asegurarte de que ambos lados sean iguales, en este caso, *2(3) +5=112*(3) +5=11. Si los dos lados son iguales has resuelto la ecuación.
Para resolver ecuaciones de segundo grado hay varios métodos, pero el principal y el más usado es la fórmula cuadrática:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

La fórmula cuadrática te da dos resultados distintos los cuales tienes que remplazar para verificar si es un resultado real, si al remplazar el resultado es 0 la solución es correcta.
Las ecuaciones de tercer grado, también conocidas como ecuaciones cúbicas, son expresiones algebraicas en las que la variable está elevada al exponente tres. Tienen la forma general ax^3 + bx^2 + cx + d = 0, donde (a), (b), (c), y (d) son coeficientes constantes, y (x) es la variable desconocida. Resolver una ecuación cúbica implica encontrar los valores de \(x\) que satisfacen la

igualdad. Aunque existen fórmulas específicas para resolver ecuaciones de tercer grado, también se puede recurrir a métodos numéricos o factorización. Estas ecuaciones pueden tener hasta tres soluciones reales o complejas, y su estudio es fundamental en álgebra y análisis matemático.
GitHub es una plataforma de desarrollo colaborativo que permite a los equipos de desarrolladores gestionar y compartir su código fuente de manera cómoda. Funciona utilizando el sistema de control de versiones distribuido Git. Los desarrolladores pueden crear repositorios para almacenar sus proyectos, y cada cambio realizado en el código se registra como un "commit". GitHub facilita la colaboración al permitir a múltiples personas trabajar simultáneamente en un proyecto, fusionar cambios de diferentes ramas y realizar un buen seguimiento del historial de versiones. Además, proporciona herramientas como solicitudes de extracción (pull requests) para facilitar la revisión y la integración de código, así como funciones para la gestión de problemas, seguimiento de proyectos y despliegue continuo. En resumen, GitHub simplifica el proceso de desarrollo de software, promoviendo la colaboración efectiva y la gestión eficiente del código fuente.

5 implementación y Desarrollo

Nuestra librería se puede llegar a utilizar para muchas cosas, ya que permite a la gente poder resolver las operaciones polinómicas gracias a que hemos implementado distintas funciones en nuestra librería, gracias a todas las funciones que hemos hecho se podrían hacer las siguientes cosas a partir de nuestras librería (teniendo en cuenta que solo formara parte, no que se puede crear a partir de lo nuestro): se puede crear una aplicación bastante parecida a la de Photomath que escanee la operación, y a base de lo escaneado, ordene la información y dependiendo de los datos obtenidos resuelva los polinomios de manera sencilla y rápida, otro ejemplo podría ser un software de algebra computacional, la algebra computacional permite al usuario realizar algebra de manera similar a como si fuera en papel y lápiz. El objetivo de la PyPil es mejorar la calidad y la efectividad de algunas aplicaciones como las mencionadas anteriormente, de tal manera que mejore así la velocidad de desarrollo de este tipo de aplicaciones y que mejore la disposición de los polinomios en el desarrollo de software con Python. Otro uso más común que se le podría dar seria para ayudar a los principiantes en el aprendizaje del lenguaje y ampliar las opciones de los novatos al crear aplicaciones.

Nuestra librería esta publicada y abierta para todas las personas que requieran de su uso, la podrás encontrar en nuestro GitHub (https://github.com/LibPypil/PyPil/tree/main). Uno de nuestros objetivos era

llegar a las máximas personas posibles, y como desarrolladoras compartir nuestro trabajo con los demás por ello hemos implementado en nuestro trabajo “GitHub” una web de desarrolladores que nos permite llegar a las máximas personas posibles además de darle accesibilidad a todo que quiera utilizar consiguiendo así llevar nuestro proyecto fuera de las aulas y darle visibilidad.

A continuación, se mostrará el desarrollo que han tenido algunas funciones de nuestra librería “PyPil”, como su planteamiento y como ha ido siendo modificada según los inconvenientes que nos hemos encontrado durante el desarrollo de estas.

- *Addition (suma):* Nuestro planteamiento ha sido que al introducir los polinomios (con los grados bien puestos) se sumen los grados similares y el resultado sea imprimido por pantalla, con la suma no hemos tenido ningún problema y todo ha salido como teníamos planteado.

- *Substraction (restar)*: Nuestro planteamiento ha sido que al introducir los polinomios (con los grados bien puestos) se restan los grados similares y el resultado sea imprimido por pantalla, con la resta no habido ningún problema ya que el código es casi el mismo que el de la suma únicamente hemos modificado algunos operadores y todo ha funcionado como esperábamos.

- *Multiplication (multiplicación)*: Esta función está dividida en dos partes (polinomio x número, polinomio x polinomio), durante el desarrollo solo tuvimos problemas con la segunda (polinomio x polinomio), ya que teníamos bastantes inconvenientes con la multiplicación de los grados, uno de los errores más grande durante el desarrollo de esta función fue que en vez de multiplicar los grados, los sumaba. La solución a este problema fue cambiar unos operadores matemáticos al final de la función, tuvimos que cambiar un + por * (signo de multiplicación en Python). Un error bastante tonto que complico el desarrollo de esta función pero que pudimos solucionar a tiempo.

- *Division (división)*: Esta función se divide en dos partes.

Ruffini (Ruffini): Para plantear esta función, nosotros pensamos en dividirlas en funciones más simples, ya que el Ruffini es una mezcla de sumas y multiplicaciones complejas. Muchos de los errores que cometimos en el Ruffini fueron a causa del polinomio no se invertía como el programa necesitaba y, que el residuo de la operación aparecía como parte del polinomio resultante. Para solucionar el problema del residuo, creamos una variable que al final aislaba el residuo y lo imprime aparte.

General Algorithm (Algoritmo general): Este ha sido sin duda el algoritmo más difícil de programar, mi planteamiento inicial fue: primero hacer la primera iteración que hacemos en el papel, encontrar el múltiple, multiplicar lo que nos dé y restarlo con el polinomio inicial. Indenté (tabular) un bloque de código mal, cosa que retrasó el progreso un poco. Una vez que tenemos esto, lo siguiente fue crear un lo op que fuera repitiendo el mismo proceso hasta que el polinomio final sea más pequeño que el divisor (sería el resto). Una vez hecho esto, le devolvemos el coeficiente y el resto al usuario.

- *PolyPow (Exponenciación)*: Esta función solo tiene 1 configuración (polinomio x entero), en esta función no tuvimos ningún error ni grave ni leve, lo que hace es multiplicar un polinomio sobre sí mismo “n” veces. No hemos tenido ningún problema durante el desarrollo de este y todo ha salido como lo planteamos en un principio.

- *Polyroot (Raíces)*: Lo planteé igual que nosotros lo hacemos en el papel, primero extraer factor común (no fue un problema), hacer Ruffini (fue más complejo ya que había que probar los divisores, y el loop me dio bastantes problemas) y, por último, cuando el polinomio es de grado 2, apliqué la función de EQ2, que ya teníamos anteriormente hecha.

- EQ1 (Ecuaciones de 1r grado): La siguiente función se dividirá en 3 subsecciones.

1- Para estas ecuaciones de primer grado, lo que realizamos fue escribir una y analizarla muy lentamente, todo lo que íbamos realizando lo agregamos en una nota. Después de esto pensamos en como traducirlo al código (Python) y empezábamos a probar distintas maneras de realizarlo, hubo unos cuantos errores en el código que no nos permitía progresar como, por ejemplo: uno de nuestros errores fue que no cambiaba el símbolo a los números y no cambiaba de lado, otro error que tuvimos fue que al inicio de escribir el código no separaba los dos lados de la ecuación para trabajarlos por separado, luego arreglamos esto y ahora los separa, los calcula y finalmente los junta otra vez.

2- Hay 1 fracción:

2.1- Hay una x en el numerador, solo había que multiplicar el denominador por todos los monomios y listo.

2.2- Este fue más complejo, ya que había x en los denominadores, lo que pensamos es multiplicar toda la ecuación por x, y luego aplicar una ecuación de segundo grado (ya que, si hay más x, si o si habrá un x^2).

2.3- Este es como el anterior, pero más fácil, ya que no había más x, hicimos lo mismo, pero en vez de hacer una EQ2, hicimos una función recursiva con una ecuación normal.

2.4- Este caso se tratan de esas fracciones que tienen x tanto en el numerador y denominador, lo que pensamos aquí, fue realizar unas condiciones para que nos clasificara si el numerador es más grande que el denominador, si eso pasa, se hará el tipo 2.1. En el caso de que no se cumpla la condición, se comprobará si hay más x, si las hay, me hará la 2.2, si no, la 2.3.

3. Hay más de 1 fracción.

3.1- En este caso, pensamos que con simplemente hacer el m.c.m de los denominadores (que serán solo números) era la manera más fácil.

3.2- Este fue más difícil de programar ya que las x estaban en el denominador. El error que tuve es no pensar en hacer el m.c.m con x, ya que es distinto a los de siempre. Simplemente tuve que arreglar eso, implementar el módulo multiplicación para multiplicar polinomios y listo.

3.3. Este fue relativamente sencillo, simplemente había que usar los 2 últimos métodos para solucionar los trozos de ecuación, no fue ningún problema.

3.4. Este es parecido al 2.4, dependiendo de donde la x queda más grande, se habría que ejecutar la 3.3 que está también ejecutara la 3.1 y la 3.2.

- *EQ2 (Ecuaciones de 2o grado)*: En esta función, nosotros planteamos usar la formula cuadrática del revés (empezando desde dentro de la ecuación), primero calculamos el discriminante (que es lo que hay dentro de la raíz), gracias a la discriminante podíamos saber la cantidad de soluciones que tiene una ecuación de segundo grado, si es negativo la ecuación no tiene soluciones, si es cero tiene 1 solución y si es positivo tiene 2 soluciones.

- *EQ3p (Ecuaciones de 3r + grado)*: Lo hemos planteado de 2 maneras, el caso 1 (solo hay un término con parte literal) y el caso 2 (hay más de 1 termino con parte literal). Para el caso 2, la solución era fácil, simplemente era aplicar el algoritmo de la raíz anteriormente creado, cosa que no ha causado ningún problema.

Para el caso 1 sí que he tenido que romperme la cabeza, ya que Python no tiene ningún método incluido que te haga raíces cubicas o de más. Con un poco de investigación, me acordé de que en mates nos explicaron que una raíz cuadrada es lo mismo que ese número, elevado a una fracción (donde la base es cuanto está elevado el número, y el denominador es el índice de la raíz). Como en Python sí que tiene un método para elevar, aplique esa teoría y funcionó.

6 Análisis

PyPil es una librería de Python dedicada a los polinomios y a la resolución de ecuaciones. Se desarrolló durante una semana con unas 28h de trabajo y pretende dar herramientas a los desarrolladores de software y facilitar a los programadores novatos el uso de operaciones polinómicas.

El objetivo de Pypil es ser una librería la cual ayude a los desarrolladores, y tras toda la semana creemos firmemente que cumple con su función, tras todo el desarrollo y su finalización además de la pequeña demo que hemos desarrollado hemos podido comprobar que soluciona el problema que teníamos en mente y no tan solo eso, hemos sido capaces de desarrollar las ecuaciones de primer grado, aunque no están finalizadas aun, seguiremos trabajando en ellas y desarrollándolas para en un futuro poder publicarlas en la versión oficial.

Además, a continuación, dejo el progreso de nuestro trabajo y como hemos ido progresando.

Actividades	Diciembre de 2023														
	1 semana							2 semana							
	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Organizar el grupo de trabajo															
Seleccionar el tema del proyecto															
Planificación tareas grupo															
Definición objetivos proyecto															
REalizar borrador memoria															
Buscar información															
Creación librería															
Pruebas librería															
Implementación librería															
Redacción memoria final															
Preparar Presentacion															
Presentacion															

Nuestra librería dispone de más de 700 líneas de código y, aunque parezcan operaciones sencillas, en el ámbito de la programación es muy complicado crear funciones como estas ya que no se pueden ver matemáticamente, sino que se deben visualizar desde un punto de vista lógico y complejo.

Como desarrolladores creemos que PyPil puede ser una gran ayuda para cierto software y también pensamos que PyPil puede ser una gran herramienta para algunos desarrolladores novatos aprendan ciertas bases o que bien saquen provecho de nuestra librería.

Por otro lado, no creemos que este sea el fin de este proyecto, sino que seguiremos trabajando en el para seguirlo mejorando y poder traer en un futuro actualizaciones de esta librería con muchas más funcionalidades fuera del ámbito polinómico, la idea de PyPil es seguir evolucionando y llegar a englobar todas las funciones matemáticas básicas de tal manera que muchos más desarrolladores la usen para sus códigos y sea mucho más conocida.

Hemos estado trabajando y seguiremos haciéndolo para que PyPil sea lo más accesible posible para todo el mundo, y para que cubra un hueco dentro del ecosistema de librerías de Python para así mejorar todo este mundillo que poco a poco se va haciendo más grande.
En resumen, creemos que PyPil ha salido tal y como teníamos en mente y planeado, estamos contentos con nuestros resultados y creemos que con el tiempo podrá seguir evolucionando y mejorando.

Readme

"Attached below is the readme of our library. If you want to view it correctly, please access our GitHub to see it."

Lib PyPil

• A simple Python library that allows you to work with equations and polynomials.

• Contact: LibPypil@gmail.com

• GitHub: Pypil

Data Structure:

• Polynomials:

• A polynomial can be represented as an array, where every index is the degree of the monomial: [1, 1.2, -2] > 1, 1.2x, -2x^2

• Equations:

• 1rst degree:

• Represented as an array. Items in the array can be a monomial or a distinctive character, where the monomials without literal part; are integers or floats, and monomials with literal part; are strings.

• Special Characters:

- =
- (
-)
- /

• The coefficient must be only a number, without operators:

• ["1x+1x", "3(3x + 3)"] <- This is wrong.

• ["2x", "9x", 9] <- This is perfect.

• 2nd degree:

• It must be an array that its length must be max 3 (it will use the monomial rules). If the discriminant is negative, it will return a NaN (Not a Number). If 0, will return an array of one float, if < 0, will return an array of 2.

• 3rd and more degrees:

• The input must be an array, following the instructions for a polynomial. The method will return an array of all the possibles answers. Methods:

• Polynomials:

• PolyAddition

• Parameters: polynomial1, polynomial2 (they must be lists, with int and float items).

• Returns: Returns a polynomial

• Description: The function adds 2 polynomials.

• PolySubtraction

• Parameters: *polynomial1, polynomial2 (They must be lists, with int and float items).

• Returns: Returns a polynomial.

• Description: The function subtracts 2 polynomials (the order is important!).

• PolyMultiplication

• Parameters: polynomial1, polynomial2. They can be both lists (with int and float items), or one of them a list and the other an int / float, but they cannot be only two numbers.

• Returns: Returns a polynomial.

• Description: Multiplicates 2 polynomials, or multiplicates a polynomial with a number.

• PolyDivision

• Parameters: polynomial1, polynomial2. They can be both lists (with int and float items), or one of them a list and the other an int / float, but they cannot be only two numbers. (Order is important!) (polynomial2 cannot be 0).

• Returns: Returns a polynomial.

• Description: Divides 2 polynomials or divides a polynomial with a number.

- PolyRoot
    - Parameters: polynomial. It must be a list, with int and float items.
    - Returns: Returns an array.
    - Description: It computes the roots of a given polynomial.
- PolyRuffini
    - Parameters: polynomial, independentTerm. One must be a list and the other one must be an int or float item.
    - Returns: Returns a tuple with a polynomial (0) and rest (1).
    - Description: It computes the Ruffini rule.
- PolyPow
    - Parameters: polynomial, n. One must be a list and the other one must be an int or float item.
    - Returns: Returns a polynomial.
    - Description: It powers the polynomial to "n".

- Equations:
    - 1rst Degree
        - Parameters: **
        - Returns: **
        - Description: **
    - 2nd Degree

• Parameters: Polynomial. It must be a list (with only int / float items).

• Returns: Returns a tuple with the two results.

• Description: It computes the quadratic formula and returns x1 (+ result) and x2 (- result).

• 3rt Degree and more

• Parameters: Polynomial. It must be a list (with only int / float items)

• Returns: It returns an array with all the possible results.

• Description: It computes it using ruffini and using custom index roots.