

Polynomial Class using Chatgbt

and Co-pilot

**OOP-CS213-Assignment1\_Task2,3-2024\_s17**

Name: Hager Ahmed Mohamed

ID: 20231197

Email:hagerah5162@gmail.com

# Introduction

In this project, I set out to develop a comprehensive **Polynomial class** in C++ using a combination of two AI-powered tools: **ChatGPT** and **Co-pilot**. These tools offered distinct strengths in helping me tackle the various challenges involved in building the class.

The collaboration between these two AI tools enabled me to efficiently develop a functional, well-structured Polynomial class, balancing conceptual understanding with rapid code generation. Throughout the process, I gained valuable experience in leveraging AI to enhance my coding workflow.

This also enabled me to compare between both AIs to find out which one is better in making the most correct,clean and effiecient code. As well as the one that was easier to deal with and reach the requirements through.

# Co-pilot

## Conversation

First I asked co-pilot to make a polynomial class divided into a header file and implementation file and it did it perfectly and it implemented all the functions with clean and very good code.

Overall I see that the conversation was very good and it was able to do everything once it understood the requirements. It would have been better if it understood the requirements by itself without me telling it.

## Time Complexity

1. **Default constructor Polynomial()**
   * **Time complexity:** O(1)
2. **Parameterized constructor Polynomial(const vector<double>& coefficients)**
   * **Time complexity:** O(n), where n is the size of the input vector coefficients.
3. **Copy constructor Polynomial(const Polynomial& other)**
   * **Time complexity:** O(n), where n is the number of coefficients in other.
4. **Assignment operator Polynomial& operator=(const Polynomial& other)**
   * **Time complexity:** O(n), where n is the number of coefficients in other.
5. **Addition operator Polynomial Polynomial::operator+(const Polynomial& other) const**
   * **Time complexity:** O(max(n, m)), where n and m are the sizes of the two polynomials' coefficient vectors.
6. **Subtraction operator Polynomial Polynomial::operator-(const Polynomial& other) const**
   * **Time complexity:** O(max(n, m)), similar to the addition operator.
7. **Multiplication operator Polynomial Polynomial::operator\*(const Polynomial& other) const**
   * **Time complexity:** O(n \* m), where n and m are the sizes of the coefficient vectors of the two polynomials.
8. **Equality operator bool Polynomial::operator==(const Polynomial& other) const**
   * **Time complexity:** O(n), where n is the size of the coefficient vectors (assuming both polynomials are the same size).
9. **Output operator ostream& operator<<(ostream& out, const Polynomial& poly)**
   * **Time complexity:** O(n), where n is the size of the polynomial.
10. **Degree function int Polynomial::degree() const**

* **Time complexity:** O(1)

1. **Evaluate function double Polynomial::evaluate(double x) const**

* **Time complexity:** O(n), where n is the number of terms in the polynomial.

1. **Composition function Polynomial Polynomial::compose(const Polynomial& q) const**

* **Time complexity:** O(n^2), where n is the degree of the original polynomial.

1. **Derivative function Polynomial Polynomial::derivative() const**

* **Time complexity:** O(n), where n is the degree of the polynomial.

1. **Integral function Polynomial Polynomial::integral() const**

* **Time complexity:** O(n), where n is the number of terms.

1. **Definite integral function double Polynomial::integral(double x1, double x2) const**

* **Time complexity:** O(n), where n is the number of terms.

1. **Root finding function double Polynomial::getRoot(double guess, double tolerance, int maxIter)**

* **Time complexity:** O(k \* n), where k is the number of iterations, and n is the time complexity of evaluating both the polynomial and its derivative.

1. **Set coefficients function void Polynomial::setCoefficients(const vector<double>& coefficients)**

* **Time complexity:** O(n), where n is the size of the input coefficient vector.

1. **Get coefficient function double Polynomial::getCoefficient(int degree) const**

* **Time complexity:** O(1)

Overall, most functions are linear O(n), except for multiplication and composition, which involve quadratic complexity.

## Suggestions and Rating of Time Complexity

**Multiplication (operator\*) - O(n \* m):**

* This is quadratic in the worst case, which can become slow for large polynomials.
* **Improvement Suggestion:**
  + Implement a **faster multiplication algorithm** like the **Karatsuba algorithm** (O(n^log3)) or use the **Fast Fourier Transform (FFT)** for polynomial multiplication (O(n log n)). These methods can significantly reduce the time complexity for large polynomials.

**Composition (compose) - O(n^2):**

* Polynomial composition is inherently expensive due to repeated multiplications and additions.
* **Improvement Suggestion:**
  + You might explore **dynamic programming** approaches to cache intermediate compositions if the composition is used frequently in certain scenarios. This won't reduce the worst-case complexity but could help in cases where you repeatedly compose with the same polynomial.

**Root Finding (Newton's Method) - O(k \* n):**

* The current complexity depends heavily on the number of iterations (k) needed for convergence. While this is typical for Newton's method, it can still be slow for polynomials with many terms.
* **Improvement Suggestion:**
  + Consider **hybrid methods** such as combining **bisection** and **Newton's method** to provide more reliable convergence. This could reduce the number of iterations in cases where Newton's method struggles near roots with flat tangents.

**Integral with Bounds (integral(x1, x2)):**

* This is O(n), which is expected. However, if integration is used frequently over large intervals, optimizations could be useful.
* **Improvement Suggestion:**
  + If the same bounds x1 and x2 are used repeatedly, **cache results** of the integral function to avoid recalculating.

Rating: 77.8/100 (for having 4 out of 18 functions not good with TC)

## Cleanliness

 **Code Formatting: 9.5/10**

* The code is well-formatted with proper indentation and spacing.
* The use of whitespace between methods and operators is consistent.

 **Naming Conventions: 9.5/10**

* Function and variable names are descriptive and follow standard naming conventions.
* There’s no ambiguity in variable or method names, which makes it easy to follow the logic.

 **Extraneous Code/Clutter: 8.5/10**

* The code is free from unnecessary comments or clutter.

**Total Cleanliness: 9/10**

## Clarity

 **Functionality Clarity: 8.5/10**

* Each function clearly does what is expected from its name and its implementation.
* The polynomial operations are straightforward and easily understandable.

 **Output Clarity: 8/10**

* The operator<< works well for displaying polynomials, but printing zero coefficients and the leading + might make the output less clear.

 **Error Handling and Edge Cases: 5/10**

* Error handling in methods like getRoot isn’t done well.

**Total Clarity: 7/10**

## Simplicity

 **Function Simplicity: 8.5/10**

* Most functions are simple and efficient, focusing on the core operations without unnecessary complexity like the polynomial addition, subtraction, and multiplication are handled in a straightforward manner.

 **Code Efficiency: 7.5/10**

* The use of most functions is effiecient.

 **Avoidance of Redundancy: 8/10**

* No redundant code or logic, and methods are not overly repetitive.

**Total Simplicity: 8/10**

**Overall Rating: 80/100**

This program is well-written, but there is room for improvement in cleanliness (avoiding repetition), clarity and simplicity (breaking down large functions).

## Suggestions/Improvements

 Refactor repetitive code for selecting polynomials by creating a helper function to reduce redundancy.

 Centralize error handling to avoid scattered try-catch blocks and group related error handling logic.

 Add more inline comments to explain non-obvious code sections, improving overall readability.

 Break down the main function into smaller, modular functions for handling specific operations, simplifying the structure.

 Make the polynomial vector dynamic, allowing users to add or delete polynomials during runtime instead of hardcoding a fixed size.

 Add file input/output support to allow saving and loading polynomials, making the program more practical.

 Add options to dynamically add or remove polynomials during runtime for more flexibility.

 Consider creating a more structured menu system or submenus for operations, improving usability.

# ChatGPT

## Conversation

I told GPT that I needed to develop a Polynomial class in C++. I provided a header file and asked for the implementation to be split into three files: a header file, an implementation file, and a main file.

What GPT Did

Class Definition: GPT created the Polynomial class based on the provided header file. It included necessary member functions such as constructors, destructor, arithmetic operators, equality operator, and utility functions like degree, evaluate, compose, derivative, and integral.

Issues I Encountered

Missing Composition Function: Initially, the compose function was not included, which caused errors when I tried to use it. I pointed this out, and GPT added the function to correctly combine polynomials.

Operator Overloading Errors: During the build process, I faced compilation errors related to operator overloading, especially when trying to multiply a polynomial with a coefficient. GPT helped correct this by adjusting the operator functions to ensure they worked with both polynomial objects and scalar values.

Incorrect Function Calls: I encountered errors when trying to use coefficients directly as polynomials without converting them first. GPT clarified the need to convert coefficients into polynomial objects before using them in operations.

Build Failures: The final build failed several times due to type mismatches and issues with function calls. I worked with GPT to revisit the logic in the compose function and refine the arithmetic operations.

Final Outcome

After resolving the errors and improving the implementation, GPT successfully built the program.

## Time Complexity

 Default Constructor: O(1)

 Constructor with Coefficients: O(n)

 Copy Constructor: O(n)

 Destructor: O(1)

 Assignment Operator: O(n)

 Addition Operator: O(n)

 Subtraction Operator: O(n)

 Multiplication Operator: O(n \* m)

 Equality Operator: O(n)

 Output Operator: O(n)

 Degree Function: O(1)

 Evaluate Function: O(n)

 Compose Function: O(n^2 \* m)

 Derivative Function: O(n)

 Integral Function: O(n)

 Definite Integral: O(n)

 Root Finding (Newton's Method): O(n \* maxIter)

 Set Coefficients: O(n)

 Get Coefficient: O(1)

## Suggestions for Improvement

**1. Multiplication Operator: O(n \* m)**

* **Suggestion**: Implement **Karatsuba multiplication** or **Fast Fourier Transform (FFT)** for polynomial multiplication. These methods can reduce the time complexity to approximately O(n log n) for large polynomials.

**2. Compose Function: O(n² \* m)**

* **Suggestion**: Instead of evaluating the composition in a naive manner, you can:
  + Use **Horner's method** to evaluate polynomials efficiently.
  + Consider caching previously computed results for polynomial compositions to avoid redundant calculations.

**3. Root Finding (Newton's Method): O(n \* maxIter)**

* **Suggestion**: Instead of evaluating the polynomial and its derivative at every iteration, consider using **memoization** to cache the results of previously computed values. This will speed up the evaluation during subsequent iterations.

**4.Precomputation**

* For functions like definite integrals or derivatives, consider precomputing values when possible, storing them, and returning the precomputed result for repeated queries

Overall Time Complexity Rating: 75 out of 100 (some complexities are too long)

## Cleanliness

 **Code Formatting: 9/10**

* The code is generally well-organized and neatly formatted.
* Proper indentation and spacing are used, but slight improvements could be made in consistently spacing out comments or logic-heavy sections (e.g., compose function).

 **Naming Conventions: 9.5/10**

* Variable and function names follow a consistent and clear naming convention.
* The use of terms like coeffs, degree(), and evaluate() are intuitive, making the purpose of the code easy to understand.

 **Extraneous Code/Clutter: 9/10**

* The code is free from unnecessary or redundant sections, and all parts serve a clear purpose.
* The destructor (~Polynomial()) is empty and could be omitted if no memory management is necessary.

**Total Cleanliness**: **9.1/10**

## Clarity

 **Functionality Clarity: 8.5/10**

* The functionality of the arithmetic operators, derivative, integral, and evaluation methods is clear.
* Some functions, such as compose and getRoot, could benefit from comments or explanations due to their complexity.

 **Output Clarity: 8/10**

* The operator<< overload prints the polynomial in a reasonably clear format but could be improved for better readability.
* The current output could get cluttered with a long polynomial, especially with positive terms being preceded by a +.

 **Error Handling and Edge Cases: 8.5/10**

* Error handling is reasonably well covered (e.g., checking for degree ranges in getCoefficient and ensuring valid iterations in getRoot).

**Total Clarity**: **8.3/10**

## Simplicity

 **Function Simplicity: 9/10**

* Most functions are simple and efficient in their structure, especially the basic arithmetic operators and getters/setters.
* The functions maintain a balance between simplicity and functionality.

 **Code Efficiency: 9/10**

* The use of std::vector is appropriate for managing polynomial coefficients, and the arithmetic operations are efficient.
* The evaluate method uses Horner’s method, which is a good optimization for polynomial evaluation.

 **Avoidance of Redundancy: 9/10**

* The code avoids redundancy, and no logic is repeated unnecessarily. Reuse of functions like evaluate in getRoot and derivative is effective.

**Total Simplicity**: **9/10**

**Overall Rating : 88/100**

The code is generally well-structured and easy to follow but could benefit from improvements in defensive programming and simplifying some of the operations to enhance both cleanliness and clarity.

## Suggestions/Improvements

**Refactor Common Logic**: Identify areas with repeated code and consolidate them into reusable functions. This will improve readability and reduce redundancy.

**Improve Variable Naming Consistency**: Ensure variable names follow a consistent and descriptive naming convention, making the code easier to understand at a glance.

**Centralize Error Handling**: Create a more uniform approach to error handling, so that it is clear where and how errors are managed. This will make debugging easier.

**Break Down Long Functions**: For functions like main(), consider breaking them down into smaller, more focused helper functions. This will make the code more modular and maintainable.

**Provide Detailed Feedback to Users**: Enhance clarity by giving users more specific feedback, such as acceptable input ranges or hints when errors occur, making the program more user-friendly.

# Test Cases Co-pilot

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number/  Operation | Input | Output | Expected Output | Result |
| 1.Getdegree | Polynomial p1: 3x^0 -4x^1 +2x^2 | 2 | 2 | Success |
| 2.Getdegree | Polynomial p2: 1x^0 +5x^1 | 1 | 1 | Success |
| 3.Getdegree | Polynomial p3: 2x^0 -1x^1 +3x^2 | 2 | 2 | Success |
| 4.Getdegree | Polynomial p4: 0x^0 +0x^1 +0x^2 +0x^3 | 3 | 0/none | Fail(Failed to produce correct output) |
| 5.Getdegree | Polynomial p5: 10x^0 +0x^1 +8e+50x^2 | 2 | 2 | Success |
| 6.Getdegree | Polynomial p6: | -1 | 0/none | Fail(didn’t output correctly) |
| 7,8,9. Sum-Subtraction-Product | Polynomial p7: 10x^0 +20x^1 +5x^2 -3x^3 +5x^4  Polynomial p8: -8x^0 +7x^1 +9x^2 +1x^3  Polynomial p9: 0x^0 +2x^1 +17x^2 +23x^3 +62x^4 -5x^5 | p7 + p9: 10x^0 +22x^1 +22x^2 +20x^3 +67x^4 -5x^5  p8 - p7: -18x^0 -13x^1 +4x^2 +4x^3 -5x^4  p9 \* p8: 0x^0 -16x^1 -122x^2 -47x^3 -180x^4 +698x^5 +546x^6 +17x^7 -5x^8 | p7 + p9: 10x^0 +22x^1 +22x^2 +20x^3 +67x^4 -5x^5  p8 - p7: -18x^0 -13x^1 +4x^2 +4x^3 -5x^4  p9 \* p8: 0x^0 -16x^1 -122x^2 -47x^3 -180x^4 +698x^5 +546x^6 +17x^7 -5x^8 | Success  Success  Success |
| 10,11,12. Sum-Subtraction-Product | Polynomial p10: 11x^0 +1x^1 -14x^2 +5x^3 -5x^4 +13x^5 +7x^6 -11x^7 +15x^8 -13x^9  Polynomial p11: 6x^0 -5x^1 +12x^2 -7x^3 +4x^4 +8x^5 +11x^6 -1x^7 +13x^8 -4x^9 | p10 + p11: 17x^0 -4x^1 -2x^2 -2x^3 -1x^4 +21x^5 +18x^6 -12x^7 +28x^8 -17x^9  p10 - p11: 5x^0 +6x^1 -26x^2 +12x^3 -9x^4 +5x^5 -4x^6 -10x^7 +2x^8 -9x^9  p10 \* p11: 66x^0 -49x^1 +43x^2 +35x^3 -186x^4 +353x^5 -45x^6 -2x^7 +146x^8 -284x^9 +208x^10 +20x^11 +42x^12 +129x^13 +11  1x^14 -329x^15 +252x^16 -229x^17 +52x^18 | p10 + p11: 17x^0 -4x^1 -2x^2 -2x^3 -1x^4 +21x^5 +18x^6 -12x^7 +28x^8 -17x^9  p10 - p11: 5x^0 +6x^1 -26x^2 +12x^3 -9x^4 +5x^5 -4x^6 -10x^7 +2x^8 -9x^9  p10 \* p11: 66x^0 -49x^1 +43x^2 +35x^3 -186x^4 +353x^5 -45x^6 -2x^7 +146x^8 -284x^9 +208x^10 +20x^11 +42x^12 +129x^13 +11  1x^14 -329x^15 +252x^16 -229x^17 +52x^18 | Success  Success  Success |
| 13,14,15. Sum-Subtraction-Product | Polynomial p12: -1e+11x^0 +20x^1 +infx^2  Polynomial p13: 0x^0 -500x^1 | p12 + p13: -1e+11x^0 -480x^1 +infx^2  p12 - p13: -1e+11x^0 +520x^1 +infx^2  p12 \* p13: 0x^0 +5e+13x^1 nanx^2 -infx^3 | No inf present | Fail  Fail  Fail |
| 16,17,18-Equality check | Polynomial p14: 0x^0 +15x^1 +500x^2  Polynomial p15: 15x^0 -25x^1  Polynomial p16: 15x^0 -25x^1 | p14 == p15: 0  p15 == p16: 1  p14 == p16: 0 | 0  1  0 | Success  Success  Success |
| 19.Evaluate at x=3 | Polynomial p17: 2e+50x^0 +5e+50x^1 | 1.7e+51 | 1.7e+51 | Success |
| 20.Evaluate at x = 5e+200 | Polynomial p18: infx^0 +5e+50x^1 | inf | No inf | Fail(Failed to deal with big numbers) |
| 21.Derivative | Polynomial p19: 15x^0 +15x^1 +20x^2 | Derivative of p19: 15x^0 +40x^1 | Derivative of p19: 15x^0 +40x^1 | Success |
| 22.Derivative | Polynomial p20: 15x^0 +6x^1 -8x^2 +0x^3 -7x^4 | Derivative of p20: 6x^0 -16x^1 +0x^2 -28x^3 | Derivative of p20: 6x^0 -16x^1 +0x^2 -28x^3 | Fail(Term x^2 shouldn’t appear) |
| 23.Derivative | Polynomial p21: 0x^0 +infx^1 | Derivative of p21: infx^0 | Not inf | Fail(can’t deal with big numbers) |
| 24.Integration | Polynomial p22: 5x^0 -1x^1 +3x^2 -10x^3 +9x^4 +3x^5 -1x^6 -2x^7 -9x^8 +15x^9 +9x^10 | Integral of p22: 0x^0 +5x^1 -0.5x^2 +1x^3 -2.5x^4 +1.8x^5 +0.5x^6 -0.142857x^7 -0.25x^8 -1x^9 +1.5x^10 +0.818182x^11 | Integral of p22: 0x^0 +5x^1 -0.5x^2 +1x^3 -2.5x^4 +1.8x^5 +0.5x^6 -0.142857x^7 -0.25x^8 -1x^9 +1.5x^10 +0.818182x^11 | Success |
| 25.Integration | Polynomial p23: 0x^0 +0x^1 | Integral of p23: 0x^0 +0x^1 +0x^2 | 0 | Fail(should output 0) |
| 26.Integration | Polynomial p24: 0x^0 +2000x^1 | Integral of p24: 0x^0 +0x^1 +1000x^2 | Integral of p24: 0x^0 +0x^1 +1000x^2 | Success |
| 27.Definite integration(0->1) | Polynomial p25: -11x^0 +10x^1 -14x^2 -14x^3 +0x^4 -9x^5 -7x^6 +13x^7 | -15.0417 | -15.0417 | Success |
| 28.Definite integration(5->5) | Polynomial p26: 10x^0 +5x^1 +65x^2 | 0 | 0 | Success |
| 29.Definite integration(-500e200 -> 500e200) | Polynomial p27: 0x^0 +0x^1 +0x^2 +0x^3 | nan | 0 | Fail(failed to deal with big numbers) |
| 30. Definite integration(-1:10) | Polynomial p28: 10x^0 +50x^1 | 2585 | 2585 | Success |
| 31.Compose | Polynomial p29: 1x^0 +2x^1  Polynomial p30: -2x^0 +5x^1 +8x^2 | Composition of p29 with p30: -3x^0 +10x^1 +16x^2 +0x^3 | Composition of p29 with p30: -3x^0 +10x^1 +16x^2 +0x^3 | Success |
| 32.Compose | Polynomial p31: 2e+200x^0 +5000x^1  Polynomial p32: -2x^0 +5x^1 +8x^2 | Composition of p32 with p31: infx^0 +1.6e+205x^1 +2e+08x^2 | No inf | Fail(Failed to deal with big numbers) |
| 33.Compose | Polynomial p32: 0x^0 +1x^1 +0x^2  Polynomial p33: 1x^0 +0x^1 | Composition of p32 with p33: 1x^0 +0x^1 +0x^2 | Composition of p32 with p33: 1x^0 +0x^1 +0x^2 | Success |
| 34.Compose | Polynomial p34: 0x^0 +0x^1 +infx^2  Polynomial p35: 1x^0 +5x^1 | Composition of p34 with p35: infx^0 +infx^1 +infx^2 | No inf | Fail(Failed to deal with big numbers) |
| 35.GetRoot | Polynomial p36: -4x^0 +8x^1 +11x^2 | 0.340542 | 0.340542 | Success |
| 36.GetRoot | Polynomial p37: 1x^0 +0x^1 +1x^2 | Runtime error(code exit) | Imaginary roots | Fail |
| 37.GetRoot | Polynomial p38: 15x^0 | Runtime error(code exit) | No roots | Fail |
| 38.GetRoot | Polynomial p39: -1x^0 -2x^1 -5x^2 | Runtime error(code exit) | Imaginary roots | Fail |
| 39.GetRoot | Polynomial p40: 0x^0 +1x^1 | 0 | 0 | Success |
| 40.GetRoot | Polynomial p41: -14x^0 -5x^1 +8x^2 -5x^3 +0x^4 -7x^5 +6x^6 +4x^7 +1x^8 +15x^9 | 0.980736 | 0.980736 | Success |
| 41.Getroot | Polynomial p42: -14x^0 -5x^1 +8x^2 -5x^3 +0x^4 -7x^5 +6x^6 +infx^7 | Runtime error(code exit) | No inf | Fail(Couldn’t deal with big numbers and couldn’t get root) |
| 42.Getcoffiecient(x^4) | Polynomial p43: -14x^0 -5x^1 +8x^2 -5x^3 +0x^4 -7x^5 | 0 | 0 | Success |
| 43.GetCoffiecient(x^3) | Polynomial p44: -14x^0 | Runtime error(code exit) | 0 | Fail(Should print 0) |
| 44.Getcoffiecient(x^2) | Polynomial p45: 17x^0 -5x^1 -infx^2 | -inf | -25e2587 | Fail(didn’t print actual number) |
| 45.Getcoffiecient(x^1) | Polynomial p46: 150x^0 +280x^1 | 280 | 280 | Success |
| 46.Setcoffiecients-Getcoffiecients(x^2) | Polynomial p47: 150x^0 +280x^1  New p47: 4x^0 +1x^1 -5x^2 | 5 | 5 | Success |
| 47.Setcoffiecients-Getcoffiecients(x^5) | Polynomial p48: 10x^0 +170x^1  New p48: -2x^0 -infx^1 +50x^2 +8e+21x^3 +987x^4 +infx^5 | inf | No inf | Fail(failed to deal with big numbers) |
| 48.Setcoffiecients-Getcoffiecients(x^0) | Polynomial p49: 5x^0 +8x^1 +21x^2  New p49: -5x^0 | -5 | -5 | Success |
| 49.Setcoffiecients-Getcoffiecients(x^7) | Polynomial p50: 7x^0 +0x^1 -14x^2 +2x^3 +4x^4 -11x^5 -6x^6 -14x^7 +6x^8  New p50: 0x^0 +9x^1 +15x^2 +6x^3 +0x^4 +6x^5 -8x^6 +12x^7 +13x^8 | 12 | 12 | Success |
| 50.Setcoffiecients-Getcoffiecients(x^1) | Polynomial p51: 0x^0  New p51: 15x^0 +87x^1 +1x^2 +0x^3 | 87 | 87 | Success |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 51.degree | 0x^2 +0x^1 +0x^0 | 2 | 0 | Fail(There is no polynomial) |
| 52.degree | 0x^0 | -1 | 0 | Fail |
| 53.degree | 13x^7 -7x^6 -9x^5 +0x^4 -14x^3 -14x^2 +10x^1 -11x^0 | 7 | 7 | Success |
| 54.degree | -9x^6 -11x^5 +6x^4 +0x^3 -3x^2 +13x^1 -5x^0 | 6 | 6 | Success |
| 55.integral(-1->1) | 2x^2 -1x^1 +0x^0 | 1.33333 | 1.33333 | Success |
| 56.integral(17->27) | 18738x^9 +34532x^8 +1100x^7 +7x^6 +6x^5 +5x^4 +4x^3 +21x^2 +13x^1 -10x^0 | 4.10863e+17 | 4.10863e+17 | Success |
| 57.integral(3->5) | 1x^0 | 2 | 2 | Success |
| 58.integral(-4000->700) | 123x^17 +0x^16 +0x^15 +0x^14 +0x^13 +234x^12 +17x^11 +70x^10 +18738x^9 +34532x^8 +1100x^7 +7x^6 +6x^5 +5x^4 +4x^3 +21x^2  +13x^1 -10x^0 | -4.69583e+65 | -4.69583e+65 | Success |
| 59.Integration | 1x^2 +0x^1 +2x^0 | 0.333333x^3 +0x^2 +2x^1 +0x^0 | 0.333333x^3 2x^1 | Fail(It added the power of x to be x^2) |
| 60.Integration | 0x^2 +0x^1 +0x^0 | 0x^3 +0x^2 +0x^1 +0x^0 | 0 | Fail(It added powers of x even when the coefficient is zero) |
| 61.Integration | -9x^8 +2x^7 +15x^6 +13x^5 +12x^4 +1x^3 -12x^2 +14x^1 +2x^0 | -1x^9 +0.25x^8 +2.14286x^7 +2.16667x^6 +2.4x^5 +0.25x^4 -4x^3 +7x^2 +2x^1 +0x^0 | -1x^9 +0.25x^8 +2.14286x^7 +2.16667x^6 +2.4x^5 +0.25x^4 -4x^3 +7x^2 +2x^1 | Success |
| 62.Integration | 1x^0 | 1x^1 +0x^0 | 1x^1 | Success |
| 63.Printing empty polynomial | P13() | 1 | No output | Fail(There should be no output) |
| 64.Addition | 133526x^2 +1233x^1 -1.11111e+08x^0  23x^1 +1x^0 | 133526x^2 +1256x^1 -1.11111e+08x^0 | 133526x^2 +1256x^1 -1.11111e+08x^0 | Success |
| 65.Subtraction | 133526x^2 +1233x^1 -1.11111e+08x^0  23x^1 +1x^0 | -133526x^2 -1210x^1 +1.11111e+08x^0 | -133526x^2 -1210x^1 +1.11111e+08x^0 | Success |
| 66.Addition | 501233x^0  0x^0 | 501233x^0 | 501233x^0 | Success |
| 67.Subtraction | 501233x^0  0x^0 | -501233x^0 | -501233x^0 | Success |
| 68.Addition | 2x^3 +33x^2 +3x^1 +1x^0  -2x^3 -33x^2 -3x^1 -1x^0 | 0x^3 +0x^2 +0x^1 +0x^0 | 0x^3 +0x^2 +0x^1 +0x^0 | Success |
| 69.Subtraction | 2x^3 +33x^2 +3x^1 +1x^0  -2x^3 -33x^2 -3x^1 -1x^0 | 4x^3 +66x^2 +6x^1 +2x^0 | 4x^3 +66x^2 +6x^1 +2x^0 | Success |
| 70.Addition | -1e+308x^1 +1e+308x^0  1e+308x^1 +1e+308x^0 | 0x^1 +infx^0 | NO inf should be present | Fail(Failed to deal with big numbers) |
| 71.Subtraction | -1e+308x^1 +1e+308x^0  1e+308x^1 +1e+308x^0 | -infx^1 +0x^0 | No inf should be present | Fail(Failed to deal with big numbers) |
| 72.Multiplication | 0x^0  2x^1 +1x^0 | 0x^1 +0x^0 | 0x^1 +0x^0 | Success |
| 73.Multiplication | 1e+308x^1 +1e+308x^0  5.05606e+15x^1 +4.05045e+15x^0 | infx^2 +infx^1 +infx^0 | Not infinity | Fail(Failed to deal with larger numbers) |
| 74.Multiplication | -5x^1 -2x^0  5x^2 +0x^1 +1x^0 | -25x^3 -10x^2 -5x^1 -2x^0 | -25x^3 -10x^2 -5x^1 -2x^0 | Success |
| 75.Equality | 1e+308x^1 +1e+308x^0  5.05606e+15x^1 +4.05045e+15x^0 | No | No | Success |
| 76.Equality | 1e+308x^1 +1e+308x^0  1e+308x^1 +1e+308x^0 | YES | YES | Success |
| 77.Equaltiy | 0x^0  0x^1 +0x^0 | NO | NO | Success |
| 78.Evaluate at x=2 | 0x^0 | 0 | 0 | Success |
| 79.Evaluate at x=1e30 | 1e+307x^1 +1e+308x^0 | inf | Not inf | Fail(Failed to deal with big numbers) |
| 80.Evaluate at x=0 | 4221x^2 +132x^1 +8324x^0 | 8324 | 8324 | Success |
| 81.Composition | 1e+308x^1 +1e+308x^0  1e+308x^1 +1e+308x^0 | 0x^2 +infx^1 +infx^0 | No inf should be present | Fail(Failed to deal with big numbers) |
| 82.Compostion | 0x^1 +0x^0  0x^2 +0x^1 +0x^0 | 0x^4 +0x^3 +0x^2 +0x^1 +0x^0 | Should only output 0 | Fail |
| 83.Composition | 352x^3 +4725x^2 +927x^1 +1256x^0  213x^3 +735x^2 +589x^1 +425x^0 | 0x^12 +0x^11 +0x^10 +3.40159e+09x^9 +3.52136e+10x^8 +1.49731e+11x^7 +3.55092e+11x^6 +5.56047e+11x^5 +6.28068e+11x^4 +5.0  6085e+11x^3 +3.00484e+11x^2 +1.14712e+11x^1 +2.78753e+10x^0 | 0x^12 +0x^11 +0x^10 +3.40159e+09x^9 +3.52136e+10x^8 +1.49731e+11x^7 +3.55092e+11x^6 +5.56047e+11x^5 +6.28068e+11x^4 +5.0  6085e+11x^3 +3.00484e+11x^2 +1.14712e+11x^1 +2.78753e+10x^0 | Success |
| 84.Getcoffiecient(x^2) | 5x^1 +3x^0 | Out of range | 0 | Fail |
| 85.Getcoffiecient(x^0) | 5x^1 +3x^0 | 3 | 3 | Success |
| 86.Getcoffiecient(x^-1) | 5x^1 +3x^0 | Out of range | 0 | Fail |
| 87.Getcoffiecient(x^1) | 5x^1 +3x^0 | 5 | 5 | Success |
| 88.Getcoffiecient(x^-1e3) | 5x^1 +3x^0 | 0 | 0 | Fail |
| 89.Derivative | 0x^0 | No output | 0 | Fail |
| 90.Derivative | 1e+308x^0 | No output | 0 | Fail |
| 91.Derivative | 1e+308x^8 +0x^7 +0x^6 +0x^5 +0x^4 +0x^3 +0x^2 +0x^1 +1e+308x^0 | infx^7 +0x^6 +0x^5 +0x^4 +0x^3 +0x^2 +0x^1 +0x^0 | No inf should be there | Fail |
| 92.Derivative | 5e+06x^8 +0x^7 +0x^6 +0x^5 +0x^4 +0x^3 +0x^2 +0x^1 +1e+308x^0 | 4e+07x^7 +0x^6 +0x^5 +0x^4 +0x^3 +0x^2 +0x^1 +0x^0 | 4e+07x^7 +0x^6 +0x^5 +0x^4 +0x^3 +0x^2 +0x^1 +0x^0 | Success |
| 93.Derivative | 5e+06x^8 +0x^7 +0x^6 +20x^5 +5245x^4 +1063x^3 +5890x^2 +100x^1 +1e+308x^0 | 4e+07x^7 +0x^6 +0x^5 +100x^4 +20980x^3 +3189x^2 +11780x^1 +100x^0 | 4e+07x^7 +0x^6 +0x^5 +100x^4 +20980x^3 +3189x^2 +11780x^1 +100x^0 | Success |
| 94.Getroots | 1x^2 +3x^1 +1x^0 | -0.38196 | -0.38196 | Success |
| 95.Getroots | 1x^2 -1x^1 +1x^0 | Runtime error(exit code) | Imaginary roots | Fail(did not give correct answer) |
| 96.Getroots | 1x^2 +0x^1 -1x^0 | 1 | 1/-1 | Success |
| 97.Getroots | 1x^6 +0x^5 +0x^4 +0x^3 +0x^2 +0x^1 -8x^0 | 1.41421 | 1.41421 | Success |
| 98.Getroots | 1x^4 +0x^3 -9x^2 +0x^1 -5x^0 | -3.08625 | -3.08625 | Success |
| 99.Getroots | 1x^2 +3x^1 +100x^0 | 4.4964 | Imaginary roots | Fail |
| 100.Getroots | 1x^5 -8x^4 +0x^3 +9x^2 +0x^1 -66x^0 | 7.87195 | 7.87195 | Success |

## Analysis

Through these test cases we can see that the getRoot function can’t deal at all with imaginary roots and gives runtime error. The code can’t deal with big numbers and empty polynomials and out of range cofficients in the polynomial.

## Correctness

62/100

# Test cases ChatGBT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number/  operation | Input | Ouput | Expected Output | Result |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Copilot code

## Co-pilot Header

**#ifndef POLYNOMIAL\_H  
#define POLYNOMIAL\_H  
  
#include <iostream>  
#include <vector>  
#include <cmath>  
#include <algorithm>  
  
using namespace std;  
  
class Polynomial {  
private:  
 vector<double> coeffs; *// Store coefficients of the polynomial*public:  
 *// Constructors* Polynomial();  
 Polynomial(const vector<double>& coefficients);  
 Polynomial(const Polynomial& other);  
  
 *// Destructor* ~Polynomial();  
  
 *// Assignment operator* Polynomial& operator=(const Polynomial& other);  
  
 *// Arithmetic operators* Polynomial operator+(const Polynomial& other) const;  
 Polynomial operator-(const Polynomial& other) const;  
 Polynomial operator\*(const Polynomial& other) const;  
  
 *// Equality operator* bool operator==(const Polynomial& other) const;  
  
 *// Output operator* friend ostream& operator<<(ostream& out, const Polynomial& poly);  
  
 *// Utility functions* int degree() const; *// Return the degree of the polynomial* double evaluate(double x) const; *// Evaluate the polynomial at x* Polynomial compose(const Polynomial& q) const; *// Composition* Polynomial derivative() const; *// Derivative of the polynomial* Polynomial integral() const; *// Return a polynomial of integration* double integral(double x1, double x2) const; *// Integrate from x1 to x2* double getRoot(double guess = 1, double tolerance = 1e-6, int maxIter = 100);  
 void setCoefficients(const vector<double>& coefficients); *// Set coefficients* double getCoefficient(int degree) const; *// Get coefficient of a specific degree*};  
  
#endif *// POLYNOMIAL\_H***

## Co-pilot Implementation file

**#include "Polynomial\_copilot.h"  
  
*// Default constructor*Polynomial::Polynomial() {}  
  
*// Parameterized constructor*Polynomial::Polynomial(const vector<double>& coefficients) : coeffs(coefficients) {}  
  
*// Copy constructor*Polynomial::Polynomial(const Polynomial& other) : coeffs(other.coeffs) {}  
  
*// Destructor*Polynomial::~Polynomial() {}  
  
*// Assignment operator*Polynomial& Polynomial::operator=(const Polynomial& other) {  
 if (this != &other) {  
 coeffs = other.coeffs;  
 }  
 return \*this;  
}  
  
*// Addition operator*Polynomial Polynomial::operator+(const Polynomial& other) const {  
 vector<double> result(max(coeffs.size(), other.coeffs.size()), 0);  
 for (size\_t i = 0; i < coeffs.size(); ++i) {  
 result[i] += coeffs[i];  
 }  
 for (size\_t i = 0; i < other.coeffs.size(); ++i) {  
 result[i] += other.coeffs[i];  
 }  
 return Polynomial(result);  
}  
  
*// Subtraction operator*Polynomial Polynomial::operator-(const Polynomial& other) const {  
 vector<double> result(max(coeffs.size(), other.coeffs.size()), 0);  
 for (size\_t i = 0; i < coeffs.size(); ++i) {  
 result[i] += coeffs[i];  
 }  
 for (size\_t i = 0; i < other.coeffs.size(); ++i) {  
 result[i] -= other.coeffs[i];  
 }  
 return Polynomial(result);  
}  
  
*// Multiplication operator*Polynomial Polynomial::operator\*(const Polynomial& other) const {  
 vector<double> result(coeffs.size() + other.coeffs.size() - 1, 0);  
 for (size\_t i = 0; i < coeffs.size(); ++i) {  
 for (size\_t j = 0; j < other.coeffs.size(); ++j) {  
 result[i + j] += coeffs[i] \* other.coeffs[j];  
 }  
 }  
 return Polynomial(result);  
}  
  
*// Equality operator*bool Polynomial::operator==(const Polynomial& other) const {  
 return coeffs == other.coeffs;  
}  
  
*// Output operator*ostream& operator<<(ostream& out, const Polynomial& poly) {  
 for (size\_t i = 0; i < poly.coeffs.size(); ++i) {  
 if (i > 0 && poly.coeffs[i] >= 0) {  
 out << "+";  
 }  
 out << poly.coeffs[i] << "x^" << i << " ";  
 }  
 return out;  
}  
  
*// Degree function*int Polynomial::degree() const {  
 return coeffs.size() - 1;  
}  
  
*// Evaluate function*double Polynomial::evaluate(double x) const {  
 double result = 0;  
 for (size\_t i = 0; i < coeffs.size(); ++i) {  
 result += coeffs[i] \* pow(x, i);  
 }  
 return result;  
}  
  
*// Composition function*Polynomial Polynomial::compose(const Polynomial& q) const {  
 Polynomial result;  
 for (int i = coeffs.size() - 1; i >= 0; --i) {  
 result = (result \* q) + Polynomial(vector<double>{coeffs[i]});  
 }  
 return result;  
}  
  
*// Derivative function*Polynomial Polynomial::derivative() const {  
 vector<double> result(coeffs.size() - 1);  
 for (size\_t i = 1; i < coeffs.size(); ++i) {  
 result[i - 1] = coeffs[i] \* i;  
 }  
 return Polynomial(result);  
}  
  
*// Integral function*Polynomial Polynomial::integral() const {  
 vector<double> result(coeffs.size() + 1, 0);  
 for (size\_t i = 0; i < coeffs.size(); ++i) {  
 result[i + 1] = coeffs[i] / (i + 1);  
 }  
 return Polynomial(result);  
}  
  
*// Definite integral function*double Polynomial::integral(double x1, double x2) const {  
 Polynomial integralPoly = integral();  
 return integralPoly.evaluate(x2) - integralPoly.evaluate(x1);  
}  
  
*// Root finding function (Newton's method)*double Polynomial::getRoot(double guess, double tolerance, int maxIter) {  
 Polynomial deriv = derivative();  
 for (int i = 0; i < maxIter; ++i) {  
 double f = evaluate(guess);  
 double fPrime = deriv.evaluate(guess);  
 if (fabs(fPrime) < tolerance) {  
 throw runtime\_error("Derivative too small");  
 }  
 double newGuess = guess - f / fPrime;  
 if (fabs(newGuess - guess) < tolerance) {  
 return newGuess;  
 }  
 guess = newGuess;  
 }  
 throw runtime\_error("Maximum iterations reached");  
}  
  
*// Set coefficients function*void Polynomial::setCoefficients(const vector<double>& coefficients) {  
 coeffs = coefficients;  
}  
  
*// Get coefficient function*double Polynomial::getCoefficient(int degree) const {  
 if (degree < 0 || degree >= coeffs.size()) {  
 throw out\_of\_range("Invalid degree");  
 }  
 return coeffs[degree];  
}**

# ChatGBT code

## ChatGBT Header

**#ifndef POLYNOMIAL\_H  
#define POLYNOMIAL\_H  
  
#include <iostream>  
#include <vector>  
#include <cmath>  
#include <algorithm>  
#include <iomanip>  
  
using namespace std;  
  
class Polynomial {  
private:  
 vector<double> coeffs; *// Store coefficients of the polynomial*public:  
 *// Constructors* Polynomial();  
 Polynomial(const vector<double>& coefficients);  
 Polynomial(const Polynomial& other);  
  
 *// Destructor* ~Polynomial();  
  
 *// Assignment operator* Polynomial& operator=(const Polynomial& other);  
  
 *// Arithmetic operators* Polynomial operator+(const Polynomial& other) const;  
 Polynomial operator-(const Polynomial& other) const;  
 Polynomial operator\*(const Polynomial& other) const;  
  
 *// Equality operator* bool operator==(const Polynomial& other) const;  
  
 *// Output operator* friend ostream& operator<<(ostream& out, const Polynomial& poly);  
  
 *// Utility functions* int degree() const; *// Return the degree of the polynomial* double evaluate(double x) const; *// Evaluate the polynomial at x* Polynomial compose(const Polynomial& other) const; *// Composition* Polynomial derivative() const; *// Derivative of the polynomial* Polynomial integral() const; *// Return a polynomial of integration* double integral(double x1, double x2) const; *// Integrate from x1 to x2* double getRoot(double guess = 1, double tolerance = 1e-6, int maxIter = 100);  
 void setCoefficients(const vector<double>& coefficients);  
 double getCoefficient(int degree) const; *// Get coefficient of a specific degree*private:  
 Polynomial power(int exponent) const; *// Helper to calculate the power of the polynomial*};  
  
#endif *// POLYNOMIAL\_H***

## ChatGBT Implementation file

**#include "Polynomial.h"  
#include <iostream>  
#include <vector>  
#include <cmath> *// Include this for pow and fabs*#include <algorithm>  
#include <iomanip>  
  
using namespace std;  
  
*// Default constructor*Polynomial::Polynomial() : coeffs(1, 0.0) {}  
  
*// Coefficient constructor*Polynomial::Polynomial(const vector<double>& coefficients) : coeffs(coefficients) {}  
  
*// Copy constructor*Polynomial::Polynomial(const Polynomial& other) : coeffs(other.coeffs) {}  
  
*// Destructor*Polynomial::~Polynomial() {}  
  
*// Assignment operator*Polynomial& Polynomial::operator=(const Polynomial& other) {  
 if (this != &other) {  
 coeffs = other.coeffs;  
 }  
 return \*this;  
}  
  
*// Addition operator*Polynomial Polynomial::operator+(const Polynomial& other) const {  
 size\_t maxSize = max(coeffs.size(), other.coeffs.size());  
 vector<double> result(maxSize, 0.0);  
  
 for (size\_t i = 0; i < maxSize; ++i) {  
 if (i < coeffs.size()) result[i] += coeffs[i];  
 if (i < other.coeffs.size()) result[i] += other.coeffs[i];  
 }  
 return Polynomial(result);  
}  
  
*// Subtraction operator*Polynomial Polynomial::operator-(const Polynomial& other) const {  
 size\_t maxSize = max(coeffs.size(), other.coeffs.size());  
 vector<double> result(maxSize, 0.0);  
  
 for (size\_t i = 0; i < maxSize; ++i) {  
 if (i < coeffs.size()) result[i] += coeffs[i];  
 if (i < other.coeffs.size()) result[i] -= other.coeffs[i];  
 }  
 return Polynomial(result);  
}  
  
*// Multiplication operator*Polynomial Polynomial::operator\*(const Polynomial& other) const {  
 vector<double> result(coeffs.size() + other.coeffs.size() - 1, 0.0);  
  
 for (size\_t i = 0; i < coeffs.size(); ++i) {  
 for (size\_t j = 0; j < other.coeffs.size(); ++j) {  
 result[i + j] += coeffs[i] \* other.coeffs[j];  
 }  
 }  
 return Polynomial(result);  
}  
  
*// Equality operator*bool Polynomial::operator==(const Polynomial& other) const {  
 return coeffs == other.coeffs;  
}  
  
*// Output operator*ostream& operator<<(ostream& out, const Polynomial& poly) {  
 for (int i = poly.coeffs.size() - 1; i >= 0; --i) {  
 out << poly.coeffs[i];  
 if (i > 0) out << "x^" << i << " + ";  
 }  
 return out;  
}  
  
*// Degree of polynomial*int Polynomial::degree() const {  
 return coeffs.size() - 1;  
}  
  
*// Evaluate polynomial*double Polynomial::evaluate(double x) const {  
 double result = 0.0;  
 for (int i = 0; i < coeffs.size(); ++i)  
 result += coeffs[i] \* pow(x, i); *// Use std::pow* return result;  
}  
  
*// Derivative*Polynomial Polynomial::derivative() const {  
 if (coeffs.size() <= 1) return Polynomial();  
  
 vector<double> derivedCoeffs(coeffs.size() - 1);  
 for (int i = 1; i < coeffs.size(); ++i) {  
 derivedCoeffs[i - 1] = i \* coeffs[i];  
 }  
 return Polynomial(derivedCoeffs);  
}  
  
*// Integral*Polynomial Polynomial::integral() const {  
 vector<double> integralCoeffs(coeffs.size() + 1);  
 integralCoeffs[0] = 0; *// constant of integration (C)* for (int i = 0; i < coeffs.size(); ++i) {  
 integralCoeffs[i + 1] = coeffs[i] / (i + 1);  
 }  
 return Polynomial(integralCoeffs);  
}  
  
*// Integrate over a range*double Polynomial::integral(double x1, double x2) const {  
 return integral().evaluate(x2) - integral().evaluate(x1);  
}  
  
*// Get root using Newton's method*double Polynomial::getRoot(double guess, double tolerance, int maxIter) {  
 for (int i = 0; i < maxIter; ++i) {  
 double eval = evaluate(guess);  
 double evalDeriv = derivative().evaluate(guess);  
 if (fabs(evalDeriv) < tolerance) break; *// Use std::fabs* double nextGuess = guess - eval / evalDeriv;  
 if (fabs(nextGuess - guess) < tolerance) return nextGuess; *// Use std::fabs* guess = nextGuess;  
 }  
 return guess; *// Return the last guess*}  
  
void Polynomial::setCoefficients(const vector<double>& coefficients) {  
 coeffs = coefficients;  
}  
  
double Polynomial::getCoefficient(int degree) const {  
 return degree < coeffs.size() ? coeffs[degree] : 0.0;  
}  
  
*// Composition of two polynomials*Polynomial Polynomial::compose(const Polynomial& other) const {  
 Polynomial result; *// Initialize to a zero polynomial using the default constructor* for (int i = degree(); i >= 0; --i) {  
 *// Create a polynomial for the coefficient* Polynomial termPoly({coeffs[i]}); *// Create polynomial with the coefficient* result = result + (termPoly \* other.power(i)); *// Multiply and add* }  
 return result;  
}  
  
*// Helper function to calculate powers of the polynomial*Polynomial Polynomial::power(int exponent) const {  
 Polynomial result({1.0}); *// Start with polynomial 1* Polynomial base = \*this; *// Current polynomial* while (exponent > 0) {  
 if (exponent % 2 == 1) {  
 result = result \* base; *// Multiply result by base when odd* }  
 base = base \* base; *// Square the base* exponent /= 2; *// Divide the exponent by 2* }  
  
 return result; *// Return final result*}**

# Main CPP Code

**#include <iostream>  
#include "Polynomial\_copilot.h"  
  
using namespace std;  
  
int main() {  
 *// test1 -success* Polynomial p1({3, -4, 2});  
 cout << "Polynomial p1: " << p1 << endl;  
 cout << "Degree of p1: " << p1.degree() << endl;  
  
 *//test2 // success* Polynomial p2({1, 5});  
 cout << "Polynomial p2: " << p2 << endl;  
 cout << "Degree of p2: " << p2.degree() << endl;  
  
 *//test3 // success* Polynomial p3({2, -1, 3});  
 cout << "Polynomial p3: " << p3 << endl;  
 cout << "Degree of p3: " << p3.degree() << endl;  
  
  
 *//test4 //Failed // No polynomial and degree must be zeros* Polynomial p4({0, 0, 0, 0});  
 cout << "Polynomial p4: " << p4 << endl;  
 cout << "Degree of p4: " << p4.degree() << endl;  
  
 *//test5 // Failed //the term 0x^2 must not found* Polynomial p5({10, 0, 8e50});  
 cout << "Polynomial p5: " << p5 << endl;  
 cout << "Degree of p5: " << p5.degree() << endl;  
  
 *//test6 //success* Polynomial p6;  
 cout << "Polynomial p6: " << p6 << endl;  
 cout << "Degree of p6: " << p6.degree() << endl;  
  
  
 Polynomial p7({10, 20, 5, -3,5});  
 cout << "Polynomial p7: " << p7 << endl;  
 Polynomial p8({-8, 7, 9, 1});  
 cout << "Polynomial p8: " << p8 << endl;  
 Polynomial p9({0, 2, 17, 23,62,-5});  
 cout << "Polynomial p9: " << p9 << endl;  
 *// Operations between polynomials* cout << "\n first Operations between polynomials:" << endl;  
  
 Polynomial sum = p7 + p9;*//test7 //success* cout << "p7 + p9: " << sum << endl;  
 Polynomial difference = p8 - p7;*//test8 //success* cout << "p8 - p7: " << difference << endl;  
 Polynomial product = p9 \* p8;*//test9 //success* cout << "p9 \* p8: " << product << endl;  
  
  
 Polynomial p10({11, 1, -14, 5, -5, 13, 7, -11, 15, -13});  
 cout << "Polynomial p10: " << p10 << endl;  
 Polynomial p11({6, -5, 12, -7, 4, 8, 11, -1, 13, -4});  
 cout << "Polynomial p11: " << p11 << endl;  
  
 *// Operations between polynomials* cout << "\n Second Operations between polynomials:" << endl;  
 Polynomial sum2 = p10 + p11;*//test10 //success* cout << "p10 + p11: " << sum2 << endl;  
 Polynomial difference2 = p10 - p11;*//test11 //success* cout << "p10 - p11: " << difference2 << endl;  
 Polynomial product2 = p10 \* p11;*//test12 //success* cout << "p10 \* p11: " << product2 << endl;  
  
 Polynomial p12({-10e10,20,20e3000});  
 cout << "Polynomial p12: " << p12 << endl;  
 Polynomial p13({0,-500});  
 cout << "Polynomial p13: " << p13 << endl;  
  
 *// Operations between polynomials* cout << "\n Third Operations between polynomials:" << endl;  
 Polynomial sum3 = p12 + p13;*//test13 //Failed //infinity on first term* cout << "p12 + p13: " << sum3 << endl;  
 Polynomial difference3 = p12 - p13;*//test14 //Failed //infinity in first term* cout << "p12 - p13: " << difference3 << endl;  
 Polynomial product3 = p12 \* p13;*//test15 //Failed //negative infinity in first tem & nan in the second term* cout << "p12 \* p13: " << product3 << endl;  
  
  
  
 *// Equality check* cout << "\n Equality check:" << endl;  
 Polynomial p14({0,15,500});  
 cout << "Polynomial p14: " << p14 << endl;  
 Polynomial p15({15,-25});  
 cout << "Polynomial p15: " << p15 << endl;  
 Polynomial p16({15,-25});  
 cout << "Polynomial p16: " << p16 << endl;  
 cout << "p14 == p15: " << (p14 == p15) << endl; *//test16 // success* cout << "p15 == p16: " << (p15 == p16) << endl; *//test17 // success* cout << "p14 == p16: " << (p14 == p16) << endl; *//test18 // success  
  
  
  
  
 // Evaluate polynomial* Polynomial p17({2e50,5e50});  
 cout << "Polynomial p17: " << p17 << endl;  
 double evaluationPoint = 3;  
 cout << "\np17 evaluated at x = " << evaluationPoint << ": " << p17.evaluate(evaluationPoint) << endl; *//test19 //success* Polynomial p18({2e500,5e50});  
 cout << "Polynomial p18: " << p18 << endl;  
 double evaluatPoint = 5e200;  
 cout << "\np18 evaluated at x = " << evaluatPoint << ": " << p18.evaluate(evaluatPoint) << endl;*//test20 //Failed //constant not infinity  
  
 // Derivative* Polynomial p19({15,15,20});  
 cout << "Polynomial p19: " << p19 << endl;  
 cout << "\nDerivative of p19: " << p19.derivative() << endl; *// test21 //success* Polynomial p20({15, 6, -8, 0, -7});  
 cout << "Polynomial p20: " << p20 << endl;  
 cout << "\nDerivative of p20: " << p20.derivative() << endl; *//test22 //Failed //term 0x^2 must not found* Polynomial p21({0,20e500});  
 cout << "Polynomial p21: " << p21 << endl;  
 cout << "\nDerivative of p21: " << p21.derivative() << endl;*//test23 //failed //should give constant not infinity  
  
 // Integral* Polynomial p22({5, -1, 3, -10, 9, 3, -1, -2, -9, 15, 9});  
 cout << "Polynomial p22: " << p22 << endl;  
 cout << "Integral of p22: " << p22.integral() << endl;*//test24 //success* Polynomial p23({0,0});  
 cout << "Polynomial p23: " << p23 << endl;  
 cout << "Integral of p23: " << p23.integral() << endl;*//test25 //failed //must equal zero* Polynomial p24({0,20e2});  
 cout << "Polynomial p24: " << p24 << endl;  
 cout << "Integral of p24: " << p24.integral() << endl;*//test26 //failed //term 0x^1 must not found  
  
  
 // Definite integral* Polynomial p25({-11, 10, -14, -14, 0, -9, -7, 13});  
 cout << "Polynomial p25: " << p25 << endl;  
 double integralResult = p25.integral(0, 1);  
 cout << "Definite integral of p25 from 0 to 1: " << integralResult << endl; *// test27 //success* Polynomial p26({10,5,65});  
 cout << "Polynomial p26: " << p26 << endl;  
 double integResult = p26.integral(5, 5);  
 cout << "Definite integral of p26 from 5 to 5: " << integResult << endl;*//test28 //success* Polynomial p27({0,0,0,0});  
 cout << "Polynomial p27: " << p27 << endl;  
 double integ\_Result = p27.integral(-500e200,500e200);  
 cout << "Definite integral of p27 from -500e200 to 500e200: " << integ\_Result << endl;*//test29 //failed //must be zero* Polynomial p28({10,50});  
 cout << "Polynomial p28: " << p28 << endl;  
 double int\_Result = p28.integral(-1,10);  
 cout << "Definite integral of p28 from -1 to 10: " << int\_Result << endl;*//test30 //success  
  
 // Composition of polynomials* Polynomial p29({1,2});  
 cout << "Polynomial p29: " << p29 << endl;  
 Polynomial p30({-2,5,8});  
 cout << "Polynomial p30: " << p30 << endl;  
  
 Polynomial comp = p29.compose(p30);  
 cout << "Composition of p29 with p30: " << comp << endl;*//test31 //success* Polynomial p31({2e200,5000});  
 cout << "Polynomial p31: " << p31 << endl;  
 Polynomial p32({-2,5,8});  
 cout << "Polynomial p32: " << p32 << endl;  
  
 Polynomial compo = p32.compose(p31);  
 cout << "Composition of p32 with p31: " << compo << endl;*//test32 //Failed //term 3 must be constant not infinity* Polynomial p\_32({0,1,0});  
 cout << "Polynomial p32: " << p\_32 << endl;  
 Polynomial p33({1,0});  
 cout << "Polynomial p33: " << p33 << endl;  
  
 Polynomial composition = p\_32.compose(p33);  
 cout << "Composition of p\_32 with p33: " << composition << endl;*//test33 //SUCCESS* Polynomial p34({0,0,2e2588});  
 cout << "Polynomial p34: " << p34 << endl;  
 Polynomial p35({1,5});  
 cout << "Polynomial p35: " << p35 << endl;  
  
 Polynomial composition\_2 = p34.compose(p35);  
 cout << "Composition of p34 with p35: " << composition\_2 << endl;*//test33 //Failed //big coefficient denoted as infinity  
  
 // Root finding* Polynomial p36({-4,8,11});  
 cout << "Polynomial p36: " << p36 << endl;  
 double root = p36.getRoot(1,1e-6,10000);  
 cout << "Root of p36 : " << root << endl;*//test34 //success* Polynomial p37({1,0,1});  
 cout << "Polynomial p37: " << p37 << endl;  
 double root2 = p37.getRoot(1,1e-6,10000);  
 cout << "Root of p37 : " << root2 << endl;*//test35 //failed //root must be i or -i* Polynomial p38({15});  
 cout << "Polynomial p38: " << p38 << endl;  
 double root3 = p38.getRoot(1,1e-6,10000);  
 cout << "Root of p38 : " << root3 << endl; *//test 36 //failed //no root* Polynomial p39({-1,-2,-5});  
 cout << "Polynomial p39: " << p39 << endl;  
 double root4 = p39.getRoot(1,1e-6,10000);  
 cout << "Root of p39 : " << root4 << endl;*//test37 //failed //it must be imaginary numbers not real* Polynomial p40({0,1});  
 cout << "Polynomial p40: " << p40 << endl;  
 double root5 = p40.getRoot(1,1e-6,10000);  
 cout << "Root of p40 : " << root5 << endl;*//test38 //success* Polynomial p41({-14, -5, 8, -5, 0, -7, 6, 4, 1, 15});  
 cout << "Polynomial p41: " << p41 << endl;  
 double root6 = p41.getRoot(1,1e-6,10000);  
 cout << "Root of p41 : " << root6 << endl;*//test39 //success* Polynomial p42({-14, -5, 8, -5, 0, -7, 6,2000e7000});  
 cout << "Polynomial p42: " << p42 << endl;  
 double root7 = p42.getRoot(1,1e-6,10000);  
 cout << "Root of p42 : " << root7 << endl;*//test40 //failed //nan because of big numbers  
  
 // Get coefficient* Polynomial p43({-14, -5, 8, -5, 0, -7});  
 cout << "Polynomial p43: " << p43 << endl;  
 double coefficient = p43.getCoefficient(4); *// Coefficient of x^4* cout << "Coefficient of x^4 in p43: " << coefficient << endl;*//test41 //success* Polynomial p44({-14});  
 cout << "Polynomial p44: " << p44 << endl;  
 double coefficient2 = p44.getCoefficient(3);  
 cout << "Coefficient of x^3 in p44: " << coefficient2 << endl;*//test42 //success* Polynomial p45({17,-5,-25e2587});  
 cout << "Polynomial p45: " << p45 << endl;  
 double coefficient3 = p45.getCoefficient(2);  
 cout << "Coefficient of x^2 in p45: " << coefficient3 << endl;*//test43 //failed //25e2587 not negative infinity* Polynomial p46({150,280});  
 cout << "Polynomial p46: " << p46 << endl;  
 double coefficient4 = p46.getCoefficient(1);  
 cout << "Coefficient of x^1 in p46: " << coefficient4 << endl;*//test44 //success  
  
 // Set new coefficients* Polynomial p47({150,280});  
 cout << "Polynomial p47: " << p47 << endl;  
 p47.setCoefficients({4, 1, -5});  
 cout << "New p47: " << p47 << endl;  
 *// Get new coefficient* double get\_coefficient = p47.getCoefficient(2);  
 cout << "Coefficient of x^2 in new p47: " << get\_coefficient << endl;*//test 45 //success* Polynomial p48({10,170});  
 cout << "Polynomial p48: " << p48 << endl;  
 p48.setCoefficients({-2,-8000e400,50,80e20,987,6548e8527});  
 cout << "New p48: " << p48 << endl;  
 *// Get new coefficient* double get\_coefficient2 = p48.getCoefficient(5);  
 cout << "Coefficient of x^5 in new p48: " << get\_coefficient2 << endl;*//test46 //failed //coefficient 6548e8527 not infinity* Polynomial p49({5,8,21});  
 cout << "Polynomial p49: " << p49 << endl;  
 p49.setCoefficients({-5});  
 cout << "New p49: " << p49 << endl;  
 *// Get new coefficient* double get\_coefficient3 = p49.getCoefficient(0);  
 cout << "Coefficient of x^0 in new p49: " << get\_coefficient3 << endl;*//test47 //success* Polynomial p50({7, 0, -14, 2, 4, -11, -6, -14, 6});  
 cout << "Polynomial p50: " << p50 << endl;  
 p50.setCoefficients({0, 9, 15, 6, 0, 6, -8, 12, 13});  
 cout << "New p50: " << p50 << endl;  
 *// Get new coefficient* double get\_coefficient4 = p50.getCoefficient(7);  
 cout << "Coefficient of x^7 in new p50: " << get\_coefficient4 << endl;*//test48 //success* Polynomial p51({0});  
 cout << "Polynomial p51: " << p51 << endl;  
 p51.setCoefficients({15,87,1,0,});  
 cout << "New p51: " << p51 << endl;  
 *// Get new coefficient* double get\_coefficient5 = p51.getCoefficient(1);  
 cout << "Coefficient of x^1 in new p51: " << get\_coefficient5 << endl;*//test50 //success  
   
 //Second set 0f 50 cases  
   
 //Test 1 //fail // there is no polynomial* Polynomial p\_1({0,0,0});  
 cout<<p\_1<<endl;  
 cout<<p\_1.degree()<<endl;  
  
 *//Test 2 //success* Polynomial p\_2;  
 cout<<p\_2<<endl;  
 cout<<p\_2.degree()<<endl;  
  
 *//Test 3 //success* Polynomial p\_3({-11 ,10 ,-14 ,-14 ,0 ,-9 ,-7 ,13});  
 cout<<p\_3<<endl;  
 cout<<p\_3.degree()<<endl;  
  
 *//Test 4 //success* Polynomial p\_4({-5, 13, -3, 0, 6, -11, -9});  
 cout<<p\_4<<endl;  
 cout<<p\_4.degree()<<endl;  
  
 *//Test 5 Success* Polynomial p\_5({0,-1,2});  
 cout<<p\_5<<endl;  
 cout<< p\_5.integral(-1,1)<<endl;  
  
 *//Test 6 Success* Polynomial p\_6({-10,13,21,4,5,6,7,1100,34532,18738});  
 cout<<p\_6<<endl;  
 cout<<p\_6.integral(17,27)<<endl;  
  
 *//Test 7 Success* Polynomial p\_7({1});  
 cout<<p\_7<<endl;  
 cout<<p\_7.integral(3,5)<<endl;  
  
 *//Test 8 Success* Polynomial p\_8({-10,13,21,4,5,6,7,1100,34532,18738,70,17,234,0,0,0,0,123});  
 cout<<p\_8<<endl;  
 cout<<p\_8.integral(-4000,700)<<endl;  
  
 *//Test 9 Fail* Polynomial p\_9({2,0,1});  
 cout<<p\_9<<endl;  
 cout<<p\_9.integral()<<endl;  
  
 *//Test 10 Fail* Polynomial p\_10({0,0,0});  
 cout<<p\_10<<endl;  
 cout<<p\_10.integral()<<endl;  
  
 *//Test 11 Success* Polynomial p\_11({2,14,-12,1,12,13,15,2,-9});  
 cout<<p\_11<<endl;  
 cout<<p\_11.integral()<<endl;  
  
 *//Test 12 Success* Polynomial p\_12({1});  
 cout<<p\_12<<endl;  
 cout<<p\_12.integral()<<endl;  
  
 *//Test 13 Fail* Polynomial p\_13();  
 cout<<p\_13<<endl;  
  
 *//Test 14* Polynomial p\_14({-111111111.1221,1233,133526});  
 Polynomial p\_15({1,23});  
 cout<<p\_14<<endl<<p\_15<<endl;  
 cout<<p\_14+p\_15<<endl;  
 *//Test 15* cout<<p\_15-p\_14<<endl;  
  
 *//Test 16 success* Polynomial p\_16({501233.2132});  
 Polynomial p\_17({0});  
 cout<<p\_16<<endl<<p\_17<<endl;  
 cout<<p\_16+p\_17<<endl;  
 *//Test 17 success* cout<<p\_17-p\_16<<endl;  
  
 *//Test 18 success* Polynomial p\_18({1,3,33,2});  
 Polynomial p\_19({-1,-3,-33,-2});  
 cout<<p\_18<<endl<<p\_19<<endl;  
 cout<<p\_18+p\_19<<endl;  
 *//Test 19 success* cout<<p\_18-p\_19<<endl;  
  
 *//Test 20 success* Polynomial p\_20({1e308,-1e308});  
 Polynomial p\_21({1e308,1e308});  
 cout<<p\_20<<endl<<p\_21<<endl;  
 cout<<p\_20+p\_21<<endl;  
 *//Test 21 success* cout<<p\_20-p\_21<<endl;  
  
 *//Test 22* Polynomial p\_22({0});  
 Polynomial p\_23({1,2});  
 cout<<p\_22<<endl<<p\_23<<endl;  
 cout<<p\_22\*p\_23<<endl;  
  
 *//Test 23* Polynomial p\_24({1e308,1e308});  
 Polynomial p\_25({4050452153055045,5056058630600552});  
 cout<<p\_24<<endl<<p\_25<<endl;  
 cout<<p\_24\*p\_25<<endl;  
  
 *//Test 24* Polynomial p\_26({-2,-5});  
 Polynomial p\_27({1,0,5});  
 cout<<p\_26<<endl<<p\_27<<endl;  
 cout<<p\_26\*p\_27<<endl;  
  
 *//Test 25* Polynomial p\_28({1e308,1e308});  
 Polynomial p\_29({4050452153055045,5056058630600552});  
 cout<<p\_28<<endl<<p\_29<<endl;  
 if(p\_28==p\_29)cout<<"Yes"<<endl;  
 else cout<<"No"<<endl;  
 *//Test 26* Polynomial p\_30({1e308,1e308});  
 Polynomial p\_31({1e308,1e308});  
 cout<<p\_30<<endl<<p\_31<<endl;  
 if(p\_30==p\_31)cout<<"Yes"<<endl;  
 else cout<<"No"<<endl;  
 *//Test 27* Polynomial p\_\_32({0});  
 Polynomial p\_33({0,0});  
 cout<<p\_\_32<<endl<<p\_33<<endl;  
 if(p\_\_32==p\_33)cout<<"Yes"<<endl;  
 else cout<<"No"<<endl;  
 *//Test 28* Polynomial p\_34({0});  
 cout<<p\_34<<endl<<p\_34.evaluate(2)<<endl;  
 *//Test 29* Polynomial p\_35({1e308,1e307});  
 cout<<p\_35<<endl<<p\_35.evaluate(1e30)<<endl;  
 *//Test 30* Polynomial p\_36({8324,132,4221});  
 cout<<p\_36<<endl<<p\_36.evaluate(0)<<endl;  
  
 *//Test 31* Polynomial p\_37({1e308,1e308});  
 Polynomial p\_38({1e308,1e308});  
 cout<<p\_37<<endl<<p\_38<<endl;  
 cout<<p\_37.compose(p\_38)<<endl;  
 *//Test 32* Polynomial p\_39({3,5});  
 Polynomial p\_40({0,0,0});  
 cout<<p\_39<<endl<<p\_40<<endl;  
 cout<<p\_39.compose(p\_40)<<endl;  
 *//Test 33* Polynomial p\_41({1256,927,4725,352});  
 Polynomial p\_42({425,589,735,213});  
 cout<<p\_41<<endl<<p\_42<<endl;  
 cout<<p\_41.compose(p\_42)<<endl;  
 *//Test 34* Polynomial p422({3,5});  
 cout<<p422<<endl<<p422.getCoefficient(2)<<endl;  
 *//Test 35* cout<<p422.getCoefficient(0)<<endl;  
 *//Test 36* cout<<p422.getCoefficient(-1)<<endl;  
 *//Test 37* cout<<p422.getCoefficient(1)<<endl;  
 *//Test 38* cout<<p422.getCoefficient(-1e3)<<endl;  
  
 *//Test 39* Polynomial p\_43({0});  
 cout<<p\_43<<endl<<p\_43.derivative()<<endl;  
 *//Test 40* Polynomial p\_44({1e308});  
 cout<<p\_44<<endl<<p\_44.derivative()<<endl;  
 *//Test 41* Polynomial p\_45({1e308,0,0,0,0,0,0,0,1e308});  
 cout<<p\_45<<endl<<p\_45.derivative()<<endl;  
 *//Test 42* Polynomial p\_46({1e308,0,0,0,0,0,0,0,5e6});  
 cout<<p\_46<<endl<<p\_46.derivative()<<endl;  
 *//Test 43* Polynomial p\_47({1e308,100,5890,1063,5245,20,0,0,5e6});  
 cout<<p\_47<<endl<<p\_47.derivative()<<endl;  
 *//Test 44* Polynomial p\_48({1,3,1});  
 cout<<p\_48<<endl<<p\_48.getRoot(1,1e-6,10000)<<endl;  
 *//Test 45* Polynomial p\_49({1,-1,1});  
 cout<<p\_49<<endl<<p\_49.getRoot(1,1e-6,10000)<<endl;  
 *//Test 46* Polynomial p\_50({-1,0,1});  
 cout<<p\_50<<endl<<p\_50.getRoot(1,1e-6,10000)<<endl;  
 *//Test 47* Polynomial p\_51({-8,0,0,0,0,0,1});  
 cout<<p\_51<<endl<<p\_51.getRoot(1,1e-6,10000)<<endl;  
 *//Test 48* Polynomial p\_52({-5,0,-9,0,1});  
 cout<<p\_52<<endl<<p\_52.getRoot(1,1e-6,10000)<<endl;  
 *//Test 49* Polynomial p\_53({100,3,1});  
 cout<<p\_53<<endl<<p\_53.getRoot(1,1e-6,10000)<<endl;  
 *//Test 50* Polynomial p\_54({-66,0,9,0,-8,1});  
 cout<<p\_54<<endl<<p\_54.getRoot(1,1e-6,10000)<<endl;  
  
  
 return 0;  
}**

# Technology Training Opportunities

## 

## Introduction

In an era where technology is advancing rapidly, acquiring new skills has become essential for personal and professional growth. This report will explore various technology training opportunities, focusing on **freeCodeCamp**, **AI Bootcamps**, and **Coursera**.

We will delve into the offerings of these platforms, including their learning tracks, application conditions, and the importance of the AI track. By highlighting these resources, this report aims to guide learners in choosing the right path to enhance their skills in the ever-evolving tech landscape.

### (freeCodeCamp)

## Introduction

This report focuses on different technology training opportunities, with a particular emphasis on freeCodeCamp, a leading platform in the field of online learning. The purpose of this report is to provide comprehensive details about freeCodeCamp, including its learning tracks, application conditions, and a detailed look into the AI track. In today's rapidly evolving technological landscape, acquiring new skills in programming and technology is essential for career development and personal growth.

**1. Who offers it and where it is located**

**freeCodeCamp.org** is a global, nonprofit platform that offers free programming and technology lessons online. With its open-source resources, learners from anywhere in the world can access it.

**Additional Details**

* **Location**: As an online platform, freeCodeCamp is accessible from anywhere with an internet connection, making it convenient for learners across different geographical locations.
* **Global Community**: The platform also fosters a vibrant community of learners, where participants can collaborate, share knowledge, and support one another through forums and study groups.

**2. What it offers**

freeCodeCamp provides multiple learning paths, including web development, data science, machine learning, and AI. Each path is designed to equip learners with the necessary skills to thrive in various technology fields.

**Additional Details**

* **Hands-on Projects**: The curriculum includes numerous hands-on projects, which allow learners to apply their knowledge to real-world scenarios. These projects are crucial for building a strong portfolio that can be showcased to potential employers.
* **Certificates**: Upon completion of each track, learners receive certificates that validate their skills and knowledge, adding value to their professional profiles.

**3. When it opens, duration, and mode**

Courses on freeCodeCamp are accessible anytime online. Learners progress at their own pace, with course durations varying from weeks to months.

**Additional Details**

* **Self-paced Learning**: The self-paced nature of the courses allows learners to balance their studies with other commitments, such as work or family responsibilities.
* **Continuous Updates**: freeCodeCamp frequently updates its content to keep up with the latest industry trends and technologies, ensuring that learners have access to the most relevant information.

**4. Conditions for application, acceptance, and fees**

There are no strict requirements for applying to freeCodeCamp. It is open to everyone, and all resources are completely free. Learners receive certificates upon completion without any fees.

**Additional Details**

* **Inclusivity**: The platform is designed to be inclusive, catering to individuals from diverse backgrounds, including students, professionals, and those looking to switch careers.
* **No Financial Barriers**: By eliminating fees, freeCodeCamp removes financial barriers to education, making technology training accessible to all.

**5. Different learning tracks**

freeCodeCamp provides different learning paths, each focusing on specific skills and technologies:

* **Full Stack Web Development**: Focuses on HTML, CSS, JavaScript, and React, teaching learners how to build complete web applications.
* **Data Analysis**: Teaches Python, Pandas, and data manipulation, preparing learners for careers in data science and analytics.
* **Machine Learning**: Involves building models with Python and TensorFlow, providing foundational knowledge for those interested in AI and ML.
* **Information Security**: Covers cybersecurity fundamentals, equipping learners with skills to protect digital information.
* **Artificial Intelligence (AI)**: Focuses on developing AI applications and deep learning, addressing the growing demand for AI professionals.

**Additional Details**

* **Career Opportunities**: Each learning track is designed to meet industry needs, ensuring that learners acquire skills that are in high demand in the job market.
* **Project-Based Learning**: All tracks emphasize project-based learning, where students build real-world applications to solidify their understanding.

**6. Track Selection and Reasons: AI**

The AI track is selected for its relevance to the future of technology, as AI is impacting industries like healthcare, finance, and more. freeCodeCamp's AI track covers deep learning, neural networks, and natural language processing (NLP), making it a perfect choice for those interested in cutting-edge innovation.

**Importance of the AI Track**

* **Future-Proof Skills**: As AI continues to evolve and integrate into various sectors, skills in AI are becoming increasingly valuable. Learning AI can open doors to various career paths, from machine learning engineer to data scientist.
* **Real-World Applications**: The AI track at freeCodeCamp prepares learners to work on real-world AI projects, such as developing chatbots, predictive models, and recommendation systems.
* **Community Support**: By choosing the AI track, learners join a community of like-minded individuals, which fosters collaboration and support as they navigate their learning journey.

### AI Bootcamps

**Overview of AI Bootcamps:**

AI bootcamps are intensive, short-term training programs designed to equip participants with the skills needed to succeed in the field of artificial intelligence and machine learning. These bootcamps focus on practical, hands-on learning and often culminate in projects that participants can showcase in their portfolios.

**Key Features**

1. **Curriculum:**
   * Bootcamps typically cover a range of topics, including:
     + Introduction to AI and machine learning concepts
     + Data preprocessing and analysis
     + Machine learning algorithms (supervised and unsupervised learning)
     + Deep learning and neural networks
     + Practical applications in areas like computer vision and natural language processing
2. **Learning Format:**
   * Classes are often held in-person, allowing for direct interaction with instructors and fellow participants.
   * Participants engage in hands-on exercises and group projects, reinforcing theoretical knowledge through practical application.
3. **Duration:**
   * AI bootcamps can vary in length, typically ranging from 8 to 24 weeks.
   * Some bootcamps offer full-time immersive programs, while others provide part-time options to accommodate working professionals.
4. **Networking Opportunities:**
   * Bootcamps often foster a collaborative environment where participants can network with instructors, industry professionals, and peers.
   * Many bootcamps host guest speakers from the tech industry, providing insights into current trends and best practices in AI.
5. **Career Support:**
   * Many bootcamps offer career services, including resume building, interview preparation, and job placement assistance.
   * Participants often have access to job boards and networking events, increasing their chances of securing employment in the field.

**General Assembly’s Data Science Immersive Bootcamp**

* **Location:** Multiple cities and online options available.
* **Curriculum Highlights:**
  + Data analysis with Python and SQL
  + Machine learning with Scikit-learn and TensorFlow
  + Building predictive models and conducting data visualizations
* **Duration:** 12 weeks (full-time) or 24 weeks (part-time).
* **Career Support:** Offers networking opportunities, mentorship, and access to job placement services.

**Conclusion**

AI bootcamps represent a valuable opportunity for individuals looking to break into the field of artificial intelligence. With a focus on practical skills, networking, and career support, these programs provide a comprehensive pathway to building a career in AI. Discussing the benefits and features of such bootcamps can help others understand the potential of offline training in advancing their careers in technology.

### Coursera

**Introduction**

Coursera is a leading online learning platform that provides access to a diverse range of courses, particularly in technology. Founded in 2012 by Stanford professors Andrew Ng and Daphne Koller, Coursera partners with over 200 renowned universities and organizations globally. This report examines Coursera’s offerings, application requirements, and highlights a selected learning track in Artificial Intelligence (AI).

**Overview of Coursera**

* Founder: Andrew Ng and Daphne Koller
* Founded: 2012
* Location: Online platform accessible globally
* Partnerships: Collaborates with prestigious institutions like Stanford, Yale, and Google.

**Course Offerings**

**Coursera provides various educational resources tailored to learners' needs:**

* Individual Courses: Covering specific subjects such as machine learning and data science.
* Specializations: Series of courses designed to deepen expertise in a particular area.
* Professional Certificates: Industry-focused programs to prepare learners for job roles.

Key Features

* Hands-on Projects: Practical applications of skills learned in courses.

Certificates: Verified certificates available for completion, enhancing resumes.

**Access and Duration**

**Courses on Coursera are flexible and available year-round:**

* Availability: Courses can be started anytime.
* Duration: Ranges from a few weeks to several months.

**Learning Format**

Self-Paced: Many courses allow learners to progress at their own speed.

Regular Updates: Courses are frequently refreshed to stay current with industry trends.

**Application Process and Fees**

Coursera courses are designed to be accessible:

Free Access: Most courses can be audited for free; fees apply for certificates.

Application Requirements: Open to all; no strict prerequisites.

Financial Aid

Options Available: Financial aid is provided for those unable to pay course fees, ensuring inclusivity.

**Learning Tracks**

Coursera offers numerous learning tracks in technology, including:

1. Data Science

2. Artificial Intelligence (AI)

3. Web Development

4. Cloud Computing

**Track Highlights**

Industry-Relevant Skills: Courses are designed to meet the demands of today’s job market.

Project-Based Learning: Emphasizes real-world applications of knowledge.

**Selected Track: Artificial Intelligence (AI)**

* The Artificial Intelligence (AI) track is chosen for its significance and growing relevance in various sectors.
* Importance of the AI Track
* High Demand: Skills in AI are increasingly sought after as businesses adopt AI technologies.
* Innovative Projects: Prepares learners to engage in cutting-edge applications, including smart systems and AI-driven analytics.
* Community Engagement: Connects learners with a network of professionals and peers passionate about AI.