How to Play:

This scientific calculator is designed to handle **basic and advanced mathematical operations**, including **trigonometric functions, logarithms, exponentiation, and graphing equations**.

- **Basic Operations:** Use the number pad and operators (+, -, ×, ÷) just like a standard calculator.
- **Advanced Functions:** Click the **2ND button** to access higher-level operations like logarithms, trigonometry, and factorials.
- **Radian/Degree Mode:** Toggle between **RAD** and **DEG** mode for trigonometric calculations.
- **Graphing Equations:** Go to Graph (there are 2 sections, one is Calculator, the other is Graph) Enter a function (e.g., `sin(x)`, `x^2 + 3x - 5`), and it will visualize on the coordinate plane. And Click on Add a function if you want to graph another function.
- **Calculation History:** The calculator keeps track of previous calculations so you can **review or reuse past inputs** without having to type them again.

View my Code: https://github.com/Patel112506/ScientificCalculator

Context:

This project was my biggest and most ambitious coding challenge yet—and easily my favorite. I wanted to create something practical and complex, so I set out to build a fully functional scientific calculator with advanced mathematical capabilities and graphing features.

I also used this project as my AP Computer Science Principles Performance Task, and I truly believe it was the reason I passed the AP Exam. The multiple-choice section wasn't as coding-focused as I expected, but my strong performance on this project carried my score.

The Coding Languages I used:

TypeScript (Primary Language):

- Used for all **React components** to structure the User Interface
- Helped with **type definitions and interfaces**, making the codebase more maintainable.
- Provided **type safety**, reducing potential runtime errors when dealing with mathematical computations.

JavaScript:

- Acted as the **underlying runtime** for executing operations.
- Used alongside **TypeScript** for handling **dynamic behavior** in the UI.

React (JSX for UI Components):

- Built the **interactive user interface** for the calculator.
- Allowed me to create reusable components for buttons, display screens, and graphing features.
- Helped manage **real-time state changes** when users input values or switch between modes.

HTML & CSS (Tailwind CSS for Styling):

- **HTML5:** Provided the structure of the application.
- **Tailwind CSS:** Allowed for clean, responsive styling while ensuring the calculator looked polished and professional.

math.js (Mathematical Computation Library):

- Used to **evaluate complex mathematical expressions** and ensure correct **order of operations**.
- Helped handle **floating-point precision issues**, preventing calculation errors.
- Allowed me to implement **trigonometric, logarithmic, and algebraic functions**.

plotly.js (Graphing Library):

- Integrated to create **interactive, real-time function graphs**.
- Allowed users to **zoom, pan, and analyze equations visually**.
- Required working with **coordinate transformations and function evaluations**.

Challenges I Faced:

- **TypeScript Complexities** – Since I was working with **advanced mathematical functions**, I had to ensure **proper type definitions** for **function inputs, return values, and complex state handling** which was more time consuming than anticipated..
- **Mathematical Challenges** – Implementing correct **order of operations**, handling **floating-point precision issues**, and making sure the calculator correctly switched between **radians and degrees** in trigonometric calculations were all hurdles I had to overcome and required a lot of trial and error. Additionally, for the graphing feature, I needed to **calculate function intersections and properly render equations which required** asking random CS strangers on reddit and or discord for assistance**.**
- **UI/UX Challenges** – Designing a **responsive, intuitive layout** was essential for a good user experience so that was one of the first things I decided to do. In addition, The calculator had different modes, like **2ND mode** for advanced functions and **RAD/DEG mode**, which meant I had to **visually and functionally manage these state changes** in the User Interface.

- **State Management** – Keeping track of **user input, calculation history, and multiple function inputs for graphing** made the logic more complex than a basic calculator. So, I had to ensure that **state updates were efficient and didn't cause unnecessary re-renders** in React.
- **Performance Considerations** – Graphing complex functions required **optimizing rendering speeds** to avoid lag was challenging. And, **Ensuring calculations were fast**, especially when handling **nested expressions or large-scale computations** was frustrating.

What I Learned: This project was by far my most rewarding and favorite coding experience. Not only did it challenge my ability to write clean and efficient TypeScript code, but it also deepened my understanding of math logic in regards to coding, as well as game state management, and User Interface and UX design.

But More than anything, this project solidified my passion for coding and problem-solving. It was the first time I felt like I was creating something both functional and meaningful, and it made me excited to keep pushing my skills further.