 **Financial Concepts to Grasp**

* Understand call and put options, strike price, time to maturity, volatility, and interest rates.
* Familiarize with Black-Scholes, Binomial Tree, and Monte Carlo models, along with the Greeks (Delta, Gamma, etc.) and P&L analysis.

 **Project Planning and Design**

* Goals: Implement models in C++, develop a React UI, use Python Flask for the backend, and visualize with Streamlit on AWS.
* Architecture:
  + **Frontend**: React.js for user interaction.
  + **Backend**: Python Flask to handle C++ computations via bindings.
  + **Database**: AWS RDS (PostgreSQL) for storing user data.
  + **Hosting**: AWS EC2, S3, and CloudWatch for deployment and monitoring.

 **C++ Model Implementation**

* Implement Black-Scholes, Binomial Tree, and Monte Carlo models in C++.
* Ensure accuracy in option pricing and Greek calculations, with optimizations for efficiency.
* Use Python bindings (Boost.Python or pybind11) for integration with Flask.

 **Backend Development in Flask**

* Build APIs like /price and /pnl for computations.
* Integrate with C++ models and perform robust testing.

 **Frontend Development in React**

* Design forms for input, display results with dynamic updates, and visualize P&L using heatmaps.
* Use Axios/Fetch API to link with the backend.

 **Data Visualization with Streamlit**

* Develop interactive components to display P&L heatmaps and analytical charts.
* Embed Streamlit in Flask, ensuring responsive updates.

 **Database Integration**

* Design a schema on AWS RDS to store user inputs and results.
* Implement efficient CRUD operations in Flask for database interaction.

 **AWS Deployment and Hosting**

* Deploy on EC2 with load balancing and scaling.
* Use S3 for static assets and CloudWatch for monitoring.

 **Performance Optimization**

* Benchmark and optimize C++ code, considering multi-threading.
* Use auto-scaling and Lambda functions for intensive tasks as needed.

 **Full Development Lifecycle**

* Plan and document requirements, design, code, and testing strategies.
* Use CI/CD pipelines for deployment and ongoing maintenance.

 **Documentation and Presentation**

* Provide clear code and technical documentation, a user guide, and a final project presentation focusing on models, C++ implementation, and system architecture.