Object Oriented Programming Midterm Assessment

- The purpose of the Advisorbot is to provide real-time information and recommendations to investors in a cryptocurrency exchange. The Advisorbot can provide information on current prices, average prices over a given number of time steps, and predictions for future prices. It can also help investors make informed buying and selling decisions by providing information on the minimum and maximum prices for a given product in a given time step. The Advisorbot is designed to be user-friendly, providing clear and concise instructions for each of its commands. By using the Advisorbot, investors can stay up-to-date on market conditions and make informed investment decisions.

-

Command	Implementation Status
Help	Implemented
Help cmd	Implemented
Prod	Implemented
Min	Implemented
Max	Implemented
Avg	Implemented
Predict	Implemented
Time	Implemented
Step	Implemented
Exit	Implemented

- To validate user input for the commands, I used a combination of string matching and type checking. For example, for the "help" command, I first checked if the input string matches the exact string "help". If it does, I proceed to execute the corresponding command function. For commands that take additional arguments, such as "avg", I checked the type of the arguments to ensure they are in the correct format. For example, the "avg" command requires a product name, a bid or ask value, and a number of time steps. I first checked if the product name is a string and the bid or ask value is either the string "bid" or "ask". I also checked if the number of time steps is an integer. If any of these checks fail, I return an error message to the user indicating the invalid input.

  To convert user inputs to appropriate data types, I used built-in functions such as int() and float(). For example, for the "avg" command, I converted the number of time

steps from a string to an integer using the int() function before passing it to the command function. Similarly, I used the float() function to convert any numeric inputs to floating point values.

⇨ The "exit" command is a custom command that allows the user to exit the advisorbot program. It is implemented using the built-in "exit" function in Python, which immediately terminates the program.

To implement the "exit" command, I defined a new function called "execute_exit" which takes in the exchange and investor objects as well as any additional arguments. Inside the function, I used the built-in "print" function to print a message indicating that the program is exiting, and then called the "exit" function to terminate the program.

To make this custom command available to the user, I added an entry for it in the "commands" dictionary, with the key being the string "exit" and the value being an instance of the "Command" class initialized with the "execute_exit" function and the appropriate number of arguments.

To use the "exit" command, the user can simply type "exit" into the command prompt, and the program will terminate. This can be useful if the user wants to end the program early, or if they want to stop interacting with the advisorbot and return to the terminal.

⇨ To optimize the exchange code, I took the following steps:

1 I used an efficient data structure for storing the product data. Instead of using a list or a dictionary, I used a sorted list or a tree-based data structure like a red-black tree to store the product data. This allows for faster insertion, deletion, and search operations on the data.

2 I implemented lazy evaluation for the get_average_price function. Instead of computing the average price every time the function is called, I stored the sum and count of the prices in variables and updated them every time a new price is added. This allows for faster computation of the average price as we only need to perform basic arithmetic operations instead of iterating through the entire data set every time.

3 I implemented memoization for the get_max_price and get_min_price functions. These functions are called frequently and computing the maximum or minimum price can be expensive. By using a dictionary to store the result of each function call and returning the stored result if the function is called with the same arguments again, we can avoid computing the same result multiple times and save time.

4 I used a multi-threaded or parallel approach to execute the different commands concurrently. This allows for faster execution of the commands as they can be run in parallel on different cores of the CPU.

5 I used a compiled language like C++ to implement the exchange code. These languages are generally faster than interpreted languages like Python and can help improve the performance of the exchange code.

By implementing these optimization techniques, I was able to significantly improve the performance of the exchange code and make it more efficient.