Shaffan Aleem

shaffanaleem@hotmail.com • linkedin.com/in/shaffan-aleem-b7a852255/ • github.com/ibnaleem

EDUCATION

University of Liverpool | Liverpool, UK Graduating 2027

Degree: BSc (Honours) Computer Science with Software Development with a Year in Industry

Coursework: Data Structures & Algorithms, Database Development, Calculus, Computer Networks, Computer Systems, Cybersecurity, Discrete Mathematics, Intro to Artificial Intelligence, Intro to Data Science, Linear Algebra, Object Oriented Programming (Java), Principles of C & Memory Management, Programming Paradigms (Haskell), Software Engineering

SKILLS & TECHNICAL TOOLS

Languages: Python, Java, C, C++, C#, Haskell, JavaScript

Technologies: Git, RESTful APIs, JSON, Linux, Windows OS, Blockchains, Cryptography

PROJECTS

Blockchain | *Networking, Cryptography. Flask, Algorithms, URL Parsing, Dataclasses, JSON, Requests, Python*

- Created a blockchain implementation in Python.
- Defined a Blockchain class with several methods to manage the blockchain, transactions, nodes, and consensus algorithm.
- Implemented a Block class as a data structure within the blockchain, containing attributes like index, timestamp, transactions, proof, and previous block's hash.
- Defined a GenesisBlock class that inherits from Block class and represents the initial block of the blockchain.
- Utilised the dataclasses module to define the block and genesis block as data classes with specified attributes.
- Implemented methods to add new blocks, create new transactions, and calculate the proof of work for mining.
- Included a register node method to add new nodes to the network.
- Implemented a valid chain method to check the validity of the blockchain based on block hashes and proof of work.
- Created a resolve conflicts method to implement a consensus algorithm that replaces the local chain with the longest valid chain from the network.
- Utilised Python's requests module to make HTTP requests to interact with other nodes.
- Defined methods to calculate valid proofs and validate proofs based on the requirement of leading zeroes.
- Utilised time module for timestamping blocks and urlparse from urllib.parse to parse URLs.

InstaTracker | *Python, Instaloader, Rich*

- Developed an Instagram tracker script in Python using the Instaloader library for account monitoring, leveraging object-oriented programming principles for modular design and functionality.
- Implemented a robust InstaTracker class with methods to fetch and log Instagram metrics such as followers, following count, post count, and bio updates.
- Integrated the rich library for terminal styling, enhancing user interface with bold and coloured text for clearer information presentation.
- Managed script execution flow with a continuous monitoring loop that updates every 5 minutes, ensuring real-time tracking of account changes.
- Employed exception handling to address potential errors like login failures or network interruptions, ensuring script reliability and continuous operation.
- Enhanced usability by providing a command-line interface (CLI) powered by argparse, allowing users to specify the Instagram username they wish to track.
- Implemented automated login functionality to streamline authentication processes, ensuring seamless script operation without manual intervention.

- Incorporated logging functionality to record all account changes in a structured format within **{username}_logs.txt,** enabling comprehensive tracking and historical review.
- Utilised datetime module for timestamp management, capturing and displaying timestamps in UTC format for accurate timekeeping.
- Encouraged community contribution and collaboration by providing project details and links to the GitHub repository in the script's documentation footer.

SnapscoreTracker | *Python, Pandas, Numpy, Rich*

- Created a Snapscore tracker in Python.
- Defined a SnapscoreTracker class with several methods to manage Snapscore logging, calculations, and reporting.
- Implemented methods to add new Snapscore logs, create directories for storing logs, and calculate the time difference between logs using **os.makedirs** for directory creation and **datetime.timedelta** for time calculations.
- Utilised the datetime module to handle timestamps and time calculations, specifically using **datetime.datetime.utcnow()** to capture the current date and time for each log entry.
- Employed the pandas module to manage and manipulate Snapscore data within CSV files. Used **pandas.read_csv** to read data from CSV files and **pandas.DataFrame.to_csv** to write data back to CSV files.
- Utilised **pandas.DataFrame.apply** function to apply custom functions across DataFrame rows, enabling complex calculations such as Snap rate per minute, hour, and day.
- Implemented a method to calculate Snap rate per minute, hour, and day, using apply to iterate over rows and perform calculations based on time differences and Snapscore increases.
- Utilised the rich module to display output messages in the console with formatting, leveraging **rich.console.Console** for styled and formatted console outputs.
- Used the numpy module to handle numerical operations, specifically using **numpy.nan** to initialise empty columns for Snap rate calculations.
- Implemented methods to calculate averages for specified columns within the logs, such as average Snapscore increase, using **pandas.DataFrame.mean** to compute average values.
- Used the argparse module to handle command-line arguments, allowing users to specify usernames, Snapscore, and custom times for logs. Defined arguments using **argparse.ArgumentParser** and **argparse.ArgumentParser.add_argument.**
- Ensured logs were maintained in CSV format for easy data manipulation and reporting, with proper headers and consistent formatting.
- Added error handling to manage directory creation and file writing operations, using try-except blocks to catch and handle exceptions such as **FileNotFoundError** and **IOError.**

Luhn Algorithm | *Data Structures and Algorithms, Python*

- Implemented a LuhnAlgorithm class in Python to validate credit card numbers using the Luhn algorithm.
- The class constructor takes either a single card number or a list of card numbers as input.
- Defined methods to double every other digit starting from the second-to-last digit and sum all digits in the card number according to the Luhn algorithm.
- Utilised Python's list comprehension and loop constructs to efficiently process card numbers.
- Included logic to handle cases where the length of the card number is odd or even.
- Implemented a validate method to apply the Luhn algorithm to each card number and determine if it is valid.
- The method returns a list of boolean values indicating the validity of each card number provided.
- Utilised object-oriented programming principles to encapsulate functionality within the class and promote code reusability.
- Implemented error handling to ensure that the input is of the correct type and format.

Queue | *Data Structures and Algorithms, Python*

- Developed a Queue class in Python to implement the functionality of a first-in-first-out (FIFO) data structure, formally known as a queue.
- Included custom exception classes QueueEmptyError, QueueFullError, and QueueError to handle specific error scenarios related to queue operations.
- Developed the Queue class constructor such that it initialises the queue with an optional list of items and a

maximum size parameter.
- Ensured that the queue does not contain nested lists to maintain consistency in the data structure.
- Implemented methods to enqueue, dequeue, peek, clear, merge, and calculate frequency of items in the queue.
- Utilised object-oriented programming principles to encapsulate functionality within the class and promote code reusability.
- Included methods to check if the queue is full, empty, and calculate available size.
- Implemented a method to sort the queue in ascending order while preserving its FIFO nature.

WORK EXPERIENCE

Healthcare/Clinical Assistant |InHealth Group April–October 2023

- Welcomed patients professionally, managing their journey from arrival to departure, ensuring a positive experience throughout their diagnostic pathway.
- Escorted patients to and from the clinical area, providing detailed explanations of procedures and addressing inquiries or concerns.
- Completed pre-scan data protection/consent forms and health and safety questionnaires.
- Assessed and monitored patients' conditions post-procedure, promptly reporting any changes to relevant staff.
- Maintained a patient and customer-focused approach to enhance the success of the Ultrasound department.
- Accurately entered patient data into the management system, ensuring data integrity.
- Managed administrative tasks, including handling inquiries, booking clinically validated appointments, and ensuring accurate and timely invoicing.
- Demonstrated an empathetic and caring approach, delivering the highest level of customer service.
- Applied understanding of health and safety and infection control principles.
- Worked independently and collaboratively within a multi-skilled team, adapting to flexible working patterns to meet site requirements.
- Exhibited excellent written and verbal communication skills, presenting information logically and efficiently.
- Utilised strong administration skills, quickly adapting to new systems.
- Attended courses to enhance knowledge and skills for the role.