Task ‘Evaluating the Development Models of Two Programming Languages’

Unit 12 Part A

1. Chatbot Utility Development in Python and JavaScript

Introduction

This document describes the development of two chatbot utilities in two different programming languages including analysis of the differences in the following:

1. Development of two small utilities, in two different programming languages, Python and JavaScript.
2. Demonstrate the working applications with sample test cases.
3. Document the development process including inline comments, code structure, and test results.
4. Write a 1000-word comparative report analyzing the strengths and weaknesses of each language in terms of:
    - Performance.
    - Security.
    - Ease of development.
    - Real-world applicability.

(University of ********, 2025)**

A comparative report which analyzes the strengths and weaknesses of each language. selected are Python and JavaScript as the programming languages for implementing a simple Chatbot Utility, JavaScript being executed using the Node.js runtime environment. Microsoft Visual Studio Code is used as the integrated development environment (IDE). JavaScript considered to be a client-side scripting language for web browsers, and it does not natively provide modules such as readline, thus browsers do not support direct terminal interaction, Node.js is used to expand the capabilities of JavaScript for developing terminal-based applications.

2. Development Process Step-by-Step Process Documentation

Python

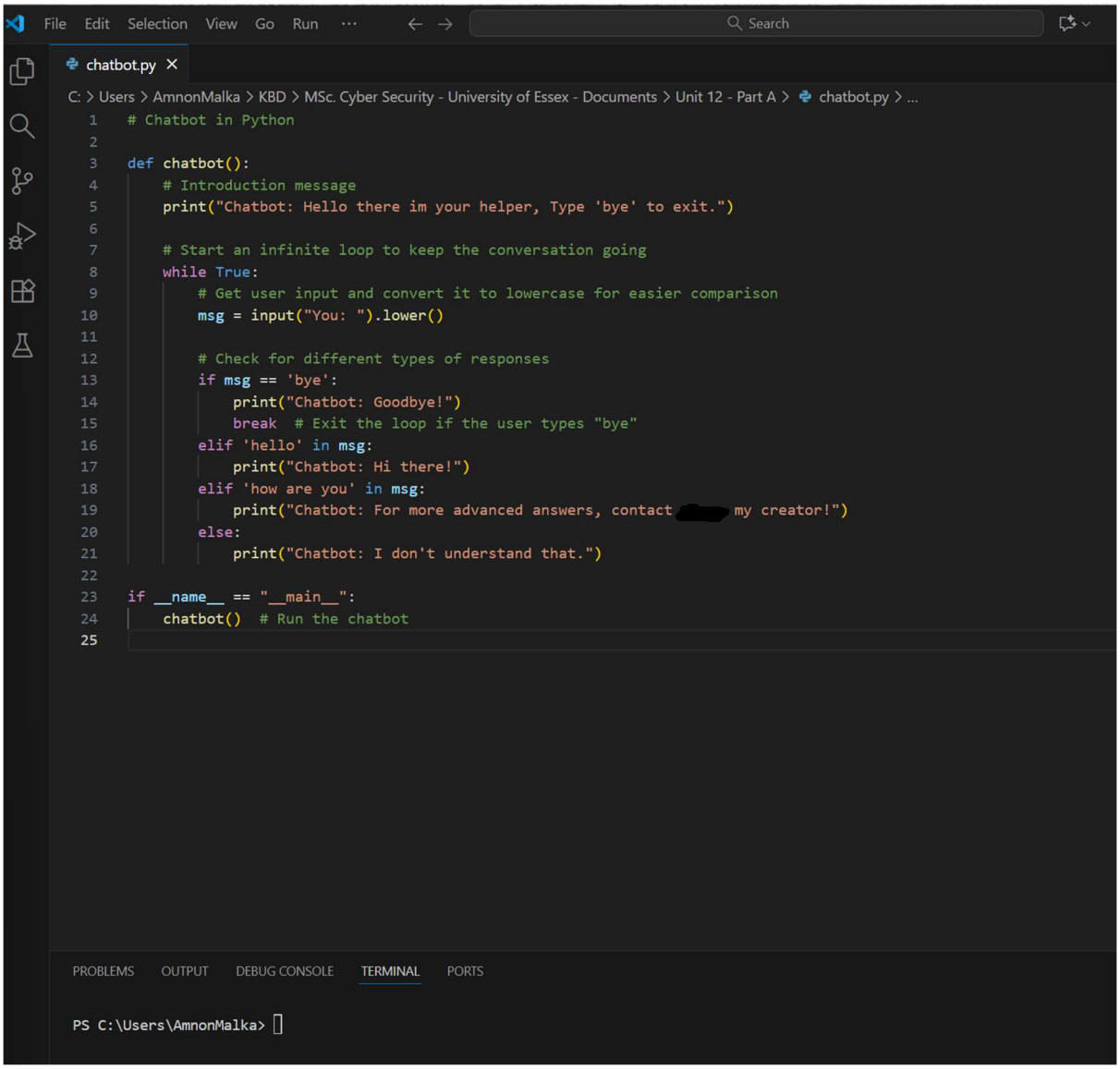
Setting Up the Environment

1. Python 3.13 was already pre-installed on my system. I use the built-in input() function for user input and a while loop to keep the conversation running.

Creating the Chatbot

1. I used simple if-elif-else statements, procedural style, to check the user's input and respond accordingly.
2. The chatbot responds based on a few key phrases (e.g., "hello", "how are you").
3. The conversation ends when the user types "bye", triggering the break statement to exit the loop.

4. The code for my chatbot incorporates fundamental programming elements, such as function definition, print statements and an infinite loop, as well as string methods like .lower() for case-insensitive comparison. Conditional statements (if, elif and else) determine the responses. The 'break' statement exits the loop when the user types 'bye'.



Testing the Chatbot

1. I evaluated the chatbot by typing various phrases like "hello", "how are you", and "bye".

2. The chatbot correctly responded to these inputs, and it ended when the "bye" input was entered.

JavaScript (Node.js)

Setting Up the Environment

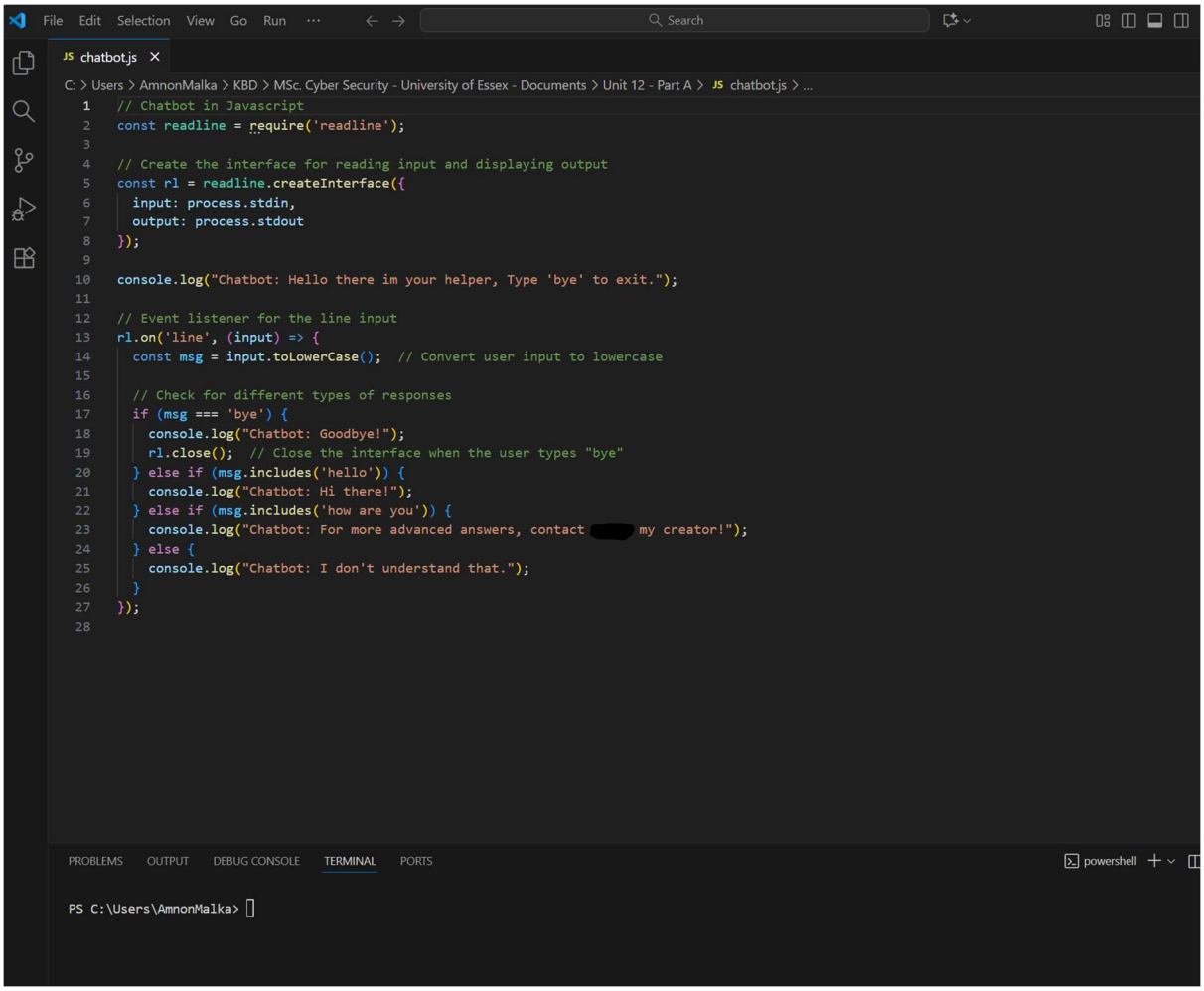
1. I installed the JavaScript (Node.js) extension on my system that is running using JavaScript in the terminal. readline module
2. I initially struggled with the readline module in Node.js but resolved it after reading the documentation. I used the built-in readline module to handle user input and output.

Creating the Chatbot

1. I used readline.createInterface to set up the interaction.
2. The chatbot checks user input for certain phrases, just like the Python version, and responds with simple event-driven responses messages.
3. The conversation ends when the user types "bye", triggering the break statement to exit the loop.
4. This JavaScript chatbot uses the readline module to manage input/output using the console.log() function provides initial and responsive messages. An event listener (rl.on('line')) captures input. String methods .toLowerCase() and .includes() process input/determined responses using if-else conditions. The rl.close() ends the chat when the user types "bye".

(W3Schools.com, 2025)

Screenshot 2 chatbot.js in IDE



Testing the Chatbot

1. I tested the chatbot with the same phrases above used in Python testing and observed that the Node.js version worked correctly.

3. Demonstrating the Working Chatbot Applications

Python

How to Run: Python chatbot uses procedural logic and while loop.

1. Save the file chatbot.py to your computer.
2. Open a terminal or command prompt.
3. Navigate to the folder where chatbot.py is located.
4. Run the chatbot using the following command: chatbot.py

1. **Sample Test Cases:**
    1. **User**: "hello"

       Chatbot Response: "Hello there I’m your helper!"
    2. User: "how are you"

       Chatbot Response: "For more advanced answers, contact ***** my creator for more info!"
    3. **User**: "bye"

       Chatbot Response: "Goodbye!"
    4. **User**: "what's up?"

       Chatbot Response: "I don't understand that."

JavaScript (Node.js)

How to Run: Node.js chatbot uses event-driven readline module.

1. Save the file as chatbot.js.
2. Open a terminal or command prompt.
3. Navigate to the folder where chatbot.js is located.
4. Run the chatbot using the following command: chatbot.js

1. **Sample Test Cases:**
    1. **User**: "hello"

       Chatbot Response: "Hello there I'm your helper!"
    2. User: "how are you"

       Chatbot Response: "For more advanced answers, contact **** my creator for more info!"
    3. **User**: "bye"

       Chatbot Response: "Goodbye!"
    4. **User**: "what's up?"

       Chatbot Response: "I don't understand that."

Part 3: Test Results

Test Case	Python Chatbot Output	JavaScript Chatbot Output	Status
"Hello"	"Hello there, I'm your helper!"	"Hello there, I'm your helper!"	Pass
"How are you"	" For more advanced answers, contact, ****** my creator!"	"For more advanced answers, contact, ****** my creator!"	Pass
"bye"	"Goodbye!"	"Goodbye!"	Pass

"what's up?"	"I don't understand that."	"I don't understand that."	Pass

4. Comparative Analysis Report: Python vs JavaScript (Node.js)

Background

Python and JavaScript are two of the most widely used programming languages in the world, each with its own ecosystem, strengths, weaknesses and use cases. While Python dominates in areas like data science, scripting, and automation, JavaScript (especially via Node.js) powers most of the modern web. In this comparative report, we will analyze how both languages perform when building simple terminal utilities, using a Chatbot tool as a reference. This comparison will focus on four key dimensions: performance, security, ease of development, and real-world applicability. The aim is to learn and understand the pros and cons of each language in building command-line interfaces based chatbot utility.

(Castro, Bruneau, Sottet, and Torregrossa, 2023)

1. Performance

Python

Python is a high-level language interpreted known for its readability and developer-friendly syntax. However, when it comes to raw performance, Python does not

compete directly with lower-level languages or JavaScript's V8 engine. In terminal utilities, performance isn't usually a bottleneck, but a few key differences remain:

- Python's synchronous input model using input() is simple but blocks the thread, which can slow down programs when using more complex, I/O scenarios.
- It incurs some overhead from its interpreter, especially on startup.
- For a single-user CLI, Python's performance is more than adequate.

JavaScript (Node.js)

Node.js, which uses the so-called "V8 engine", is known for its high-speed execution, especially in I/O-heavy tasks. While simple input-output tasks may not stretch their capabilities, Node's non-blocking, event-driven architecture excels in scenarios with:

- Asynchronous input handling (via readline or streams).
- Low latency interaction.
- Fast startup and response, even in heavier tasks.

In this chatbot example, both languages execute almost instantaneously, however, for scaled to process large files or handle concurrent users (as in a server-side CLI tool), Node.js would likely outperform Python due to it's I/O optimization.

Performance Table

Language	Strengths	Limitations
Python	Adequate for CLI tools, readable	Slower startup and blocking I/O

JavaScript	Fast I/O, asynchronous, non-blocking design	More verbose for simple tasks

2. Security

Python

Python has a better and more secure standard library, and for most CLI-based tools where input is being used, it is secure. Python's input() function is straightforward and does not interpret code unless explicitly coded to do so. Security concerns could arise when:

- Executing arbitrary shell commands such as: OS. System and subprocess
- Accepting file uploads or untrusted data.

For small local utilities like this chatbot.py, risk is very insignificant. Furthermore, Python benefits from fewer "unfamiliar" dependencies compared to JavaScript's massive npm ecosystem, which can occasionally include malicious packages.

JavaScript (Node.js)

Node.js allows deep access to the system, file system, network, and child processes which makes it powerful but also riskier. Malicious packages on npm have historically been a security issue due to:

- Typo squatting (for instance, express)
- Malicious install scripts

- Unverified dependencies with postinstall hooks

While the Node.js core APIs (like readline) are safe, developers must be cautious when expanding CLI tools to include external libraries.

Also, event-driven code can introduce timing issues, race conditions, or insecure async logic if not carefully written. In general JavaScript with Node.js is superior in terms of scalability, performance, or high availability.

Security Table

Language	Strengths	Limitations
Python	Safer by default for scripts due to limited attack surface	Fewer malicious packages, easier auditing
JavaScript	Secure core APIs, powerful access	npm ecosystem poses higher risk

3. Ease of Development

Python

Python is known for its minimalism and readability and writing command-line tools are incredibly easy due to its simple input/output (input(), print()). It also has clear procedural flow and rich command-line libraries like argparse and click. Python lets developers build quick tools in fewer lines of code.

JavaScript (Node.js)

JavaScript is by nature asynchronous., writing simple command-line based applications requires setting up the Readline module, handling callbacks, or using async/await for better control. This adds verbosity and complexity. For instance, there are more boilerplates, such as createInterface and event listeners. It requires an understanding of event-driven architecture. Error handling is also more complex with try/catch in async. However, for developers already familiar with Node.js or building tools in the web ecosystem, JavaScript is still a good choice — especially when integrated into full-stack workflows.

Ease of Development Table

Language	Strengths	Limitations
Python	Easy to learn, fast to write, highly readable	Minimal setup
JavaScript	Powerful, flexible	More setup and async logic complexity

4. Real-world Applicability

Python

Python is the go-to scripting language for sysadmins, data scientists, automation engineers, and backend developers. Its applicability for CLI tools is vast:

- Common in tools like pip, ansible, AWS-cli and more.

- Supported by cross-platform packaging tools, for instance, pyinstaller.
- Integrated well with Unix/Linux shells.

Python is widely used to build command-line applications that process text, files, or automate system tasks.

(Lee, 2025)

JavaScript (Node.js)

Node.js is more commonly used in web development, but it is gaining popularity in building CLI tools — especially those tied to the JavaScript ecosystem. Examples:

- npm, ESlint, webpack.
- Easily bundled with pkg or nexe for cross-platform binaries.
- Useful for full-stack developers who want to use JavaScript everywhere.

However, for system-level tasks, data processing, or OS scripting, Node.js is not the first tool most developers reach for.

Real-World Use Table

Language	Strengths	Limitations
Python	Dominant in scripting and system tools	GUI requires additional libraries
JavaScript	Strong in full-stack and build tools	Less common for traditional CLI apps

Conclusion

Both Python and JavaScript are powerful tools when developing terminal-based utilities and the best choice depends on the specific context and existing ecosystem in addition developer expertise is particularly important. When looking for simple, script-like terminal utilities, Python is the more intuitive, secure, and a better choice. However, JavaScript/Node.js excels when it comes to integrating with web projects, managing asynchronous tasks, or building command-line interface tools within a heavy JavaScript stack. The right choice of language is key to achieving the project's goals and aligning with its workflow. This document helps to understand how state is maintained in a simple chatbot and how conditional logic can mimic conversation. Python's clean syntax and multi-paradigm support make it ideal for prototyping. It is also ideal for AI and learning. Node.js excels at real-time applications with event-driven concurrency Security awareness is essential in both ecosystems.

Summary Table

Criteria	**Python**	**JavaScript (Node.js)**
Performance	Good	Excellent (async I/O)
Security	Strong	Riskier due to npm
Ease of Development	Very easy	Moderate (async complexity)
Real-World Fit	Ideal for CLI tools	Better for web-related CLIs

5. Summary Comparison of Development Models

Aspect	Python	JavaScript (Node.js)
Development Models	Procedural, OOP, Functional — flexible	Event-driven, Functional, OOP — suited for real-time
Syntax & Readability	Highly readable, indentation-based	Flexible but can be inconsistent
Performance	Slower in I/O/concurrency	Fast for I/O, non-blocking
Scalability & Maintainability	Good readability, easy to maintain; concurrency harder	Naturally scalable; requires clean architecture to maintain
Security	Lower attack surface; less dependency risk	Larger ecosystem, more dependency risks, more network exposure

(Gascón, 2024; Kapoor, Sonu, 2025; OWASP, 2021)

References

Castro, O., Bruneau, P., Sottet, J.S. and Torregrossa, D., (2023). Landscape of high-performance Python to develop data science and machine learning applications. ACM Computing Surveys, 56(3), pp.1-30.

Gascón, Ulises. Node. Js for Beginners: *A Comprehensive Guide to Building Efficient, Full-Featured Web Applications with Node*. Js. First edition. Birmingham: Packt Publishing, 2024. Print. Available at:

https://essex.primo.exlibrisgroup.com/permalink/44UOES_INST/1vvl5tg/alma991008694640107346 [Accessed 25 September 2025].

Kapoor, Sonu. *Beginning JavaScript Syntax: Understanding Syntactical Rules and Structures for Better JavaScript Programming*. 1st ed. 2025. Berkeley, CA: Apress, (2025). Available at: Web. https://essex.primo.exlibrisgroup.com/permalink/44UOES_INST/1vvl5tg/alma991008826327507346 [Accessed 02 October 2025].

Lee, L., DevCademy Media Inc. REALPYTHON™ DBA Real Python (2025). *Using PyInstaller to Easily Distribute Python Applications.* Available at: https://realpython.com/pyinstaller-python/ [Accessed 30 September 2025].

OWASP, (2021). OWASP Node.js Security Cheat Sheet. OWASP Foundation. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html [Accessed 25 September 2025].

W3Schools.com. (2025). Node.js Readline Module: *Introduction to the Readline Module* Available at: https://www.w3schools.com/nodejs/nodejs_readline.asp [Accessed 30 September 2025].

W3Schools.com. (2025). Node.js Interface Reference: *Interface Object* Available at: https://www.w3schools.com/nodejs/ref_interface.asp [Accessed 30 September 2025].

University of **********, (2025). Launch into Computing July 2025 B: Unit 12 Part A: *Individual Programming Exercise – Comparing Programming Languages*. Available at: ********** [Accessed 15 September. 2025]

This document has been written solely for educational purposes. All references, names, and trademarks mentioned here remain the property of their respective owners and are used here strictly for the educational context. Grammarly was used exclusively for proofreading and enhancing the clarity and language of the text. ChatGPT was consulted for general research. All academic writing, analysis, argumentation, and conclusions are entirely the original work of the author.