FACULTY OF INFORMATION SCIENCES AND ENGINEERING

PROGRAMMING LANGUAGES
CCS21103
SESSION: SEPTEMBER 2024

ASSIGNMENT 1
30%

PREPARED BY:

Name	ID Number	Programme
Nur Fatwa Hidayah binti Mohd Nor Azizan	2082024090008	BCS
Nur Najiha Nabila binti Mohd Adira	2082024070012	BCS

PREPARED FOR: Dr. Jamal Abdullahi

TABLE OF CONTENT

1.0 APPLICATIONS: PYTHON

1.1 History

Python, created by Guido van Rossum and first released in 1991, was developed as a successor to the ABC programming language, with a vision to provide an accessible yet powerful tool for programming. Van Rossum aimed to design a language that prioritized code readability and simplicity, allowing developers to express complex ideas in a more concise and intuitive manner. This focus on clarity helped to lower the barriers for entry into programming, making Python particularly attractive to beginners. Over the years, Python has experienced significant evolution, spurred by a rapidly growing community that has contributed to its rich ecosystem of libraries and frameworks. This collaborative effort has enhanced Python's capabilities, especially in domains like scientific computing, data analysis, and machine learning. Libraries such as NumPy, SciPy, Pandas, and Matplotlib have established Python as a dominant force in the scientific realm, enabling researchers and data scientists to perform complex calculations, visualize data, and analyze large datasets with relative ease. The language's versatility has also led to its adoption in various other fields, from web development to automation and beyond.

Today, Python is not just a programming language but a vibrant ecosystem that continues to thrive, supported by a passionate community that drives its ongoing development and innovation. Its popularity reflects its status as a go-to choice for professionals and enthusiasts alike, further solidifying its role in shaping the future of programming and technology.

1.2 Characteristics

- High-level language
  - Python is classified as a high-level programming language, meaning it provides a strong level of abstraction from the underlying hardware and system architecture. This characteristic simplifies the programming process significantly.

- Interpreted language
  - Python is an interpreted language, which means that code is executed line-by-line rather than being compiled into machine code all at once.

- Extensive libraries
  - Python has a rich ecosystem of libraries, such as NumPy, SciPy, and Matplotlib, that support scientific computations and data visualization.

- Dynamic typing
  - Python employs dynamic typing, which means that variable types are determined at runtime rather than being explicitly defined at compile time.

1.3 Example of Applications

- Data Analysis and Manipulation with Pandas
  - Pandas is a powerful data manipulation and analysis library that provides data structures like Data Frames and Series, which are essential for handling structured data.

- Scientific Computing Using NumPy and SciPy
  - NumPy provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. SciPy builds on NumPy, offering additional functionality for optimization, integration, interpolation, eigenvalue problems, and other advanced mathematical tasks.

- Machine Learning with Libraries like TensorFlow and Scikit-learn

- Scikit-learn is a user-friendly library for machine learning that provides simple and efficient tools for data mining and analysis. TensorFlow, developed by Google, is a more complex framework for deep learning, enabling the building and training of neural networks.

- Visualization of Data with Matplotlib and Seaborn
  - Matplotlib is a versatile plotting library, while Seaborn builds on Matplotlib, providing a high-level interface for attractive statistical graphics.

1.4 Strengths

- Ease of Learning and Use
  - One of Python's most significant advantages is its straightforward and intuitive syntax, which makes it highly accessible to beginners.

- Community Support
  - Python boasts one of the largest and most active programming communities in the world, which is a tremendous asset for developers at all levels.

- Versatility
  - Python's versatility is one of its standout features, making it applicable across various domains and industries.

1.5 Limitations

- Global interpreter lock (GIL)
  - This can be a bottleneck in multi-threaded applications, limiting performance in certain scenarios.
- Performance
  - R can be slower than some other programming languages, especially when handling very large datasets, as it loads all data into memory.

- Steep Learning Curve
  - Although R is designed for statisticians, its syntax can be less intuitive for beginners, especially those without a background in statistics.

- Less General-Purpose
  - Unlike languages such as Python, R is primarily focused on statistics and data analysis, which can limit its application in broader programming contexts like web development or general software engineering.

2.0 BUSINESS APPLICATIONS: JAVA

2.1 History

Java was developed by James Gosling and his team at Sun Microsystems, with its official release occurring in 1995. The language was born out of a desire to create a versatile and platform-independent tool for software development that could seamlessly run on any device equipped with a Java Virtual Machine (JVM). This innovative approach embraced the "write once, run anywhere" philosophy, which revolutionized application development by enabling developers to create software that could be executed across diverse platforms without the need for modification or recompilation. As a result, Java quickly gained traction in various sectors, particularly within the enterprise environment, where its robust architecture and strong emphasis on security made it an ideal choice for building large-scale applications. The language's ability to handle complex tasks, combined with its rich set of APIs and extensive libraries, contributed to its reputation for reliability and scalability.

Furthermore, Java's strong community support and continuous evolution have allowed it to remain relevant in an ever-changing technological landscape. Over the years, it has become a foundational element in web development, mobile applications, and cloud computing, solidifying its status as a standard programming language that continues to power critical systems and applications worldwide. Today, Java remains a dominant force in the software industry, cherished by developers for its stability, extensive documentation, and the vibrant ecosystem that surrounds it, ensuring its place as a go-to language for a wide range of programming challenges.

2.2 Characteristics

- Object-Oriented
    - C is a fully object-oriented programming language, which encourages encapsulation, inheritance, and polymorphism, allowing for better organization and modularity of code.

- Type-Safe
    - C enforces strict type checking, reducing the chances of runtime errors and ensuring that operations are performed on compatible data types.

- Rich Standard Library
    - C benefits from the extensive NET framework, which provides a vast array of libraries and tools for tasks such as data access, web services, and user interface design.

- Cross-Platform Capabilities
    - With the introduction of .NET Core (now .NET 5 and later), C has become more versatile, allowing developers to build applications that run on multiple operating systems, including Windows, macOS, and Linux.

2.3 Example of Applications

- Enterprise Applications
    - C is widely used for developing large-scale enterprise applications, particularly within organizations that utilize Microsoft technologies, such as ERP and CRM systems.

- Web Applications
    - Frameworks like ASP.NET allow developers to build dynamic web applications and services, making C a popular choice for web development.

- Desktop Applications
    - C is often used to create Windows desktop applications, leveraging Windows Forms and WPF (Windows Presentation Foundation) for rich user interfaces.

- Game Development
  - With the Unity game engine, C is a leading language for developing video games across multiple platforms.

2.4 Strengths

➢ Strong Integration with Microsoft Ecosystem
  - C works seamlessly with other Microsoft technologies, such as Azure cloud services and SQL Server, making it an ideal choice for businesses already invested in the Microsoft stack.

➢ Robust Development Tools
  - The availability of powerful development environments, such as Visual Studio, enhances productivity through features like IntelliSense, debugging tools, and integrated testing frameworks.

➢ Active Community and Support
  - C has a large and active community, providing extensive documentation, tutorials, and resources, which can help developers solve problems and share knowledge.

➢ Modern Language Features:
  - C continues to evolve with features like async/await for asynchronous programming, LINQ for data querying, and pattern matching, making it a modern and efficient language for developers.

2.5 Limitations

➢ Platform Dependency (Historically)
- While C has become cross-platform with .NET Core, its historical association with the Windows ecosystem may deter developers who prefer open-source or non-Microsoft environments.

➢ Memory Management
- Although C provides automatic garbage collection, this can sometimes lead to performance overhead, particularly in resource-intensive applications.

➢ Learning Curve for Beginners
- C is user-friendly compared to some languages, its object-oriented nature and the breadth of features available can pose a challenge for absolute beginners.

➢ Licensing Costs
- While the core .NET platform is open-source, some tools and enterprise-level features, particularly in the Microsoft ecosystem, may involve licensing costs, which can be a consideration for businesses.

3.0 ARTIFICIAL INTELLIGENCE

3.1 History

The history of artificial intelligence (AI) dates back to ancient times with early philosophers and mathematicians pondering the nature of human intelligence and the possibility of creating intelligent machines. The foundations for modern AI were laid in the mid-20th century when mathematician and logician Alan Turing proposed the idea of a "universal machine" capable of simulating any computational process now known as the Turing Machine. In 1950, Turing introduced the "Turing Test" a method for evaluating whether a machine could exhibit intelligent behavior indistinguishable from that of a human. This was a pivotal moment that sparked significant interest in the possibility of creating machines that could "think" like humans. Shortly after, in 1956 the term "artificial intelligence" was coined by John McCarthy during the Dartmouth Conference where AI was established as a legitimate field of study. Researchers were optimistic believing that machines capable of human-like intelligence would soon be a reality.

The early decades of AI research primarily from the 1950s to the 1970s which focused on symbolic AI or "good old-fashioned AI" (GOFAI). During this time, AI researchers aimed to create intelligent systems by hard-coding human knowledge and logical rules into programs. Early successes included programs that could solve algebra problems, play chess and prove mathematical theorems. However, as researchers attempted to scale these systems, they encountered significant challenges. Symbolic AI struggled with handling ambiguity, dealing with incomplete information and processing natural language. These limitations led to what is often referred to as the "AI winter" a period of reduced funding and interest in AI research during the late 1970s and early 1980s as the hype around the technology outpaced its capabilities.

The 21st century has seen exponential growth in AI which driven by advances in machine learning, deep learning and the availability of massive amounts of data, often referred to as "big data." Deep learning which utilizes large and multi-layered neural networks has revolutionized fields such as image recognition, natural language processing and autonomous systems. AI is now integrated into a wide range of industries from healthcare and finance to transportation and entertainment. Notable achievements such as Google's AlphaGo defeating the world champion Go player in 2016 have demonstrated the potential of AI to surpass human capabilities in specific tasks. At the same time, ethical concerns surrounding AI including issues related to bias, privacy and the future of work have prompted discussions on how to ensure that AI develops in a way that benefits humanity. Today, AI continues to evolve rapidly with ongoing research focused on achieving artificial general intelligence (AGI) a level of intelligence that matches or surpasses human cognitive abilities across a wide range of tasks.

3.2 Characteristics

➤ Automation

- AI systems are capable of automating complex and repetitive tasks with minimal human intervention, improving efficiency and productivity in various industries.

➤ Learning and Adaptation

- AI systems especially through machine learning can learn from data and experience. They improve over time by analyzing new inputs, adjusting algorithms and refining outputs.

➤ Problem-Solving

- AI systems can process vast amounts of information and apply logic to solve complex problems that may be difficult for humans to handle due to data volume or complexity.

➤ Natural Language Processing (NLP)

- AI can understand, interpret and generate human language making it possible for machines to communicate with humans through speech or text.

➤ Reasoning and Decision-Making

- AI can simulate human reasoning processes by analyzing situations and making decisions based on data-driven models, predictions and rule-based logic.

3.3 Example of Applications

- ➢ Healthcare
  - AI is used in medical diagnostics, drug discovery, personalized treatment plans and robotic surgery. For example, IBM's Watson assists doctors by analyzing medical literature and patient data to provide treatment recommendations.

- ➢ Finance
  - AI applications like fraud detection, automated trading systems and customer service chatbots are widely used in banking and finance. For instance, AI algorithms detect abnormal transactions to prevent fraud.

- ➢ Autonomous Vehicles
  - AI powers self-driving cars by processing real-time sensor data and making decisions about navigation, object detection and traffic conditions such as in Tesla's Autopilot.

- ➢ Retail
  - E-commerce platforms like Amazon use AI for personalized recommendations, customer behavior analysis and optimizing logistics in their supply chains.

- ➢ Entertainment
  - AI-driven algorithms suggest content on platforms like Netflix and Spotify like curating personalized viewing or listening experiences based on user preferences and behavior.

3.4 Strengths

- ➢ Efficiency and Accuracy
  - AI can process large datasets quickly and accurately enabling businesses to make data-driven decisions with minimal errors.

- ➢ 24/7 Availability
  - Unlike humans, AI systems can operate continuously without fatigue, offering around-the-clock service in customer support, manufacturing and other sectors.

- ➢ Data-Driven Insights
  - AI excels in analyzing big data and identifying patterns or trends that may be missed by humans and providing valuable insights for strategic decisions.

- ➢ Cost Reduction
  - By automating repetitive tasks and improving operational efficiency, AI reduces labor costs and minimizes human errors in processes like manufacturing or logistics.

- ➢ Scalability
  - AI systems can handle increasing amounts of data or tasks with ease and making them highly scalable as business needs grow.

3.5 Limitations

- Lack of Human-Like Understanding
  - AI systems can analyze data and make decisions based on patterns but they lack true understanding of context, emotions and subjective reasoning.

- Bias in Data and Algorithms
  - AI systems can inherit biases from the data they are trained on which leads to unfair or discriminatory outcomes in areas such as hiring, lending or criminal justice.

- High Development Costs
  - Designing and implementing advanced AI systems can be expensive and require substantial investments in data, infrastructure and expertise.

- Limited Creativity
  - While AI can analyze data and make decisions based on predefined rules, it struggles with tasks that require creativity, abstract thinking and innovation.

- Dependency on Data
  - AI's effectiveness is reliant on the availability and quality of data. In cases of insufficient, outdated, or biased data, AI systems may deliver inaccurate or irrelevant results.

4.0 SYSTEMS PROGRAMMING

4.1 History

Systems programming is a branch of computer science focused on the development of software that interacts closely with the hardware and provides the foundation for application programs. The history of systems programming began in the early days of computing with the creation of machine languages in the 1940s and 1950s. The earliest systems programmers worked directly with the hardware through binary code, inputting data via punch cards and switches. Early computers such as ENIAC and UNIVAC, required meticulous programming in machine language which laid the foundation for assembly languages to provide a more manageable way to control hardware.

In the 1960s, systems programming advanced significantly with the development of operating systems beginning with mainframes. IBM's OS/360 which developed in the mid-1960s was one of the first operating systems to automate job scheduling and resource management and allowing computers to be more efficient. This era also saw the development of high-level programming languages like FORTRAN and COBOL for specific systems programming tasks. The concept of time-sharing was also introduced, where multiple users could interact with the same system simultaneously as demonstrated by MIT's CTSS (Compatible Time-Sharing System) and later UNIX developed in the 1970s at Bell Labs.

UNIX, particularly revolutionized systems programming by offering a multiuser and multitasking environment that could run on various hardware platforms. Its portability and modular design influenced future systems programming and led to the widespread adoption of C, the programming language in which UNIX was written. C became the language of choice for systems programming due to its balance between hardware control and high-level abstraction. This period also saw the rise of more sophisticated operating systems, such as MS-DOS in the 1980s which were designed for the burgeoning personal computer market further expanding the scope of systems programming.

The 1990s and beyond have seen the explosion of open-source systems, particularly Linux which was created by Linus Torvalds and modelled after UNIX. Linux democratized access to systems programming and became a foundation for numerous modern applications from servers to embedded systems. Advances in hardware such as multicore processors and the rise of cloud computing have continued to push the boundaries of systems programming as modern programmers now work on building scalable efficient systems that can run on a wide range of hardware and across distributed environments.

4.2 Characteristics

➢ Low-Level Interaction with Hardware

- Systems programming deals with software that directly interfaces with hardware components. This requires an understanding of how hardware operates as the software often controls memory, CPU and device input-output.

➢ Efficiency and Performance

- Code written in systems programming must be highly optimized for performance. Efficient resource usage is critical since the software needs to maximize the hardware's potential, minimize latency and ensure minimal resource wastage.

➢ Portability

- While many systems programs are designed for specific hardware, the rise of languages like C and systems like UNIX have prioritized portability, enabling software to run across different platforms with minimal modification.

➢ Concurrency and Multitasking

- Systems programming often involves handling multiple processes or threads simultaneously. This requires careful management of shared resources, synchronization and inter-process communication.

➢ Resource Management

- A key aspect of systems programming is managing system resources like memory, CPU time and I/O devices. This includes managing memory allocation, and deallocation and preventing memory leaks or crashes.

4.3 Example of Applications

➢ Operating Systems

- Operating systems like Linux, Windows and macOS are prime examples of systems programming. They manage hardware resources, provide services to other software and ensure efficient execution of tasks.

➢ Device Drivers

- Device drivers control hardware peripherals like printers, graphics cards and network cards. These programs enable communication between the operating system and hardware components.

➢ Embedded Systems

- Systems programming is used in embedded systems such as automotive software, IoT devices and industrial machines. These systems must operate with strict performance and memory constraints.

➢ Database Management Systems (DBMS)

- While typically considered application software, DBMS like MySQL or PostgreSQL often involve systems programming, particularly in managing low-level data storage, concurrency and memory.

4.4 Strengths

- ➢ High Efficiency
  - Systems programs are designed to be efficient and fast also ensuring optimal use of system resources like CPU and memory.

- ➢ Close Control over Hardware
  - Systems programming allows direct access and control over hardware, giving developers the ability to optimize performance and tailor software to specific hardware configurations.

- ➢ Reliability
  - Systems programs form the backbone of computer operations (e.g., operating systems). They need to be stable and reliable as any faults could have wide-reaching effects on user applications.

- ➢ Customizability
  - By working at the system level, programmers can fine-tune performance and implement custom solutions for specific hardware and enhancing performance for specialized tasks.

4.5 Limitations

- Complexity
  - Systems programming requires an in-depth understanding of both software and hardware which makes it more complex and challenging than higher-level application development.

- Portability Issues
  - While languages like C enable portability, not all systems programs are easily transferable between platforms. Hardware dependencies can limit the ability of software to run on different systems.

- Debugging Difficulty
  - Debugging systems-level code is harder than debugging application code. Low-level interactions with hardware, memory management issues and concurrency errors can be difficult to track and fix.

- Time-Consuming Development
  - Writing efficient, reliable systems programs takes more time and effort compared to high-level programming due to the need for optimization and thorough testing.

5.0 SCRIPTING LANGUAGES

5.1 History

Scripting languages have a rich history rooted in the evolution of programming itself. Its beginning in the early days of computer science. The origins can be traced back to the 1960s when batch processing and automation in mainframe systems were critical to simplifying repetitive tasks. Early examples include IBM's Job Control Language (JCL) which was used to script the execution of batch jobs on mainframes. However, these early languages were limited in scope and focused on controlling the system rather than developing fully-fledged applications. The need for languages that allowed for more flexibility and ease of use led to the development of more sophisticated scripting tools.

In the 1970s and 1980s, the rise of UNIX brought about significant advancements in scripting languages with the introduction of shells like the Bourne Shell (sh) and C Shell (csh). These shells allowed users to write scripts to automate tasks on UNIX systems and enabling them to interact with the operating system more efficiently. Along with these shell scripts came powerful text processing utilities such as awk and sed which played a pivotal role in processing and manipulating data. During this era, scripting languages were mostly used by system administrators and engineers to perform tasks related to system maintenance, file manipulation and automation in marking a shift from exclusive system control to more general-purpose automation.

The 1990s saw an explosion in the popularity of scripting languages with the advent of the internet and the World Wide Web. As the web grew and so did the demand for languages that could facilitate web development and manage the interactivity of web pages. This led to the creation of languages like Perl which became a popular choice for web-based CGI scripting and PHP which allowed developers to build dynamic, server-side web applications. JavaScript introduced by Netscape in the mid-1990s, revolutionized client-side scripting by enabling developers to create interactive and dynamic web content that could run directly in a web browser. These languages were easier to learn and use compared to traditional programming languages like C or Java, fueling their widespread adoption among a new generation of web developers.

In the 21st century, scripting languages have continued to evolve, expanding their use cases far beyond just system administration and web development. Python, which was first developed in the late 1980s and gained popularity in the 2000s became a go-to language for scientific computing, data analysis and machine learning, due to its simplicity and rich ecosystem of libraries. Today, scripting languages are also critical in the DevOps and cloud computing ecosystems, with tools like Bash, PowerShell and Python automating infrastructure management, deployment and testing processes. They remain a crucial part of modern programming, bridging the gap between code and automation in a world where speed, efficiency and scalability are paramount.

5.2 Characteristics

- Interpreted Execution
  - Unlike compiled languages, scripting languages are typically interpreted which means the code is executed line-by-line without needing a separate compilation step. This makes development faster as changes can be tested immediately.

- Ease of Use
  - Scripting languages are often designed to be simple and easy to learn, with high-level and human-readable syntax. This makes them ideal for beginners and for tasks that require rapid development.

- Dynamically Typed
  - Most scripting languages are dynamically typed meaning variables do not need explicit type declarations. The language determines the data type at runtime and offers flexibility but sometimes at the expense of performance.

- Automation and Glue Code
  - Scripting languages excel at automating repetitive tasks and integrating different software components (glue code). They are commonly used to connect different systems, manage file operations or control software behaviors.

- Platform Independence
  - Many scripting languages such as Python and JavaScript are designed to be cross-platform and allow scripts to run on various operating systems without modification.

5.3 Example of Applications

- Web Development
  - Languages like JavaScript, PHP and Ruby are widely used in building dynamic websites and web applications. For example, JavaScript enables interactive client-side functionality while PHP and Ruby handle server-side processes.

- Automation and Task Management
  - Shell scripting (Bash, PowerShell) is often used to automate administrative tasks, such as file backups, system updates and batch processing of files.

- Data Analysis and Machine Learning
  - Python is heavily used in scientific computing, data analysis and machine learning. Tools like Jupyter Notebooks allow users to write scripts for data manipulation and visualization.

- Game Development
  - Scripting languages are used for game development especially for managing game logic, controlling characters or handling in-game events without recompiling the entire game.

5.4 Strengths

➢ Rapid Development
- The ease of writing and running scripts without needing compilation enables rapid prototyping, making scripting languages ideal for quick iterations and testing.

➢ Flexibility and Integration
- Scripting languages often provide built-in features for handling diverse tasks such as file manipulation, data handling and network communication. They are ideal for integrating various components of a system or managing workflows.

➢ Large Community and Libraries
- Languages like Python and JavaScript have extensive ecosystems offering a vast range of libraries and frameworks that simplify tasks like web development, data analysis and automation.

➢ Cross-Platform Support
- Most scripting languages are platform-independent allowing scripts to be executed on multiple operating systems without significant changes to the code.

5.5 Limitations

- ➢ Rapid Development
  - The ease of writing and running scripts without needing compilation enables rapid prototyping which makes scripting languages ideal for quick iterations and testing.

- ➢ Flexibility and Integration
  - Scripting languages often provide built-in features for handling diverse tasks such as file manipulation, data handling and network communication. They are ideal for integrating various components of a system or managing workflows.

- ➢ Large Community and Libraries
  - Languages like Python and JavaScript have extensive ecosystems offering a vast range of libraries and frameworks that simplify tasks like web development, data analysis and automation.

- ➢ Cross-Platform Support
  - Most scripting languages are platform-independent allowing scripts to be executed on multiple operating systems without significant changes to the code.

6.0 PROVIDE THE PROGRAMMING LANGUAGES SYNTAX

CONCEPT	Scientific Applications	Business Applications	Artificial Intelligence	Systems Programming	Scripting Languages
Variable Declaration	`x = 10`	`int x = 10;`	`model = 'my_model'`	`int x = 10;`	`let x = 10;`
Input Statement	`x = input("Enter a value: ")`	`int x = Convert.ToInt 32(Console.Re adLine());`	`model_name = input("Enter model name: ")`	`printf("Ente r a value: "); scanf("%d", &x);`	`let x = prompt("Enter a value: ");`
Output Statement	`print("Value:", x)`	`Console.Write Line("Value: " + x);`	`print("Model name:", model_name)`	`printf("Valu e: %d\n", x);`	`console.log("Va lue:", x);`

Selection Control Structure	`IF (num > 0) THEN`
	`! do something`
	`ELSE`
	`! do something`
	`else`
	`END IF`
		`IF NUM > 0`
		`THEN`
		`DISPLAY`
		`"POSITIVE"`
		`ELSE`
		`DISPLAY`
		`"NEGATIVE"`
			`num > 0 -> true; false.`	`if (num > 0)`
				`{`
				`// do`
				`something`
				`} else {`
				`// do`
				`something`
				`else`
				`}`
					`if [ $num -gt 0 ]; then`
					`# do something`
					`else`
					`# do something`
					`else`

Repetition Control Structure	`DO i = 1, 10`
	`! loop body`
	`END DO`
		`PERFORM`
		`VARYING I`
		`FROM 1 BY 1`
		`UNTIL I > 10`
			`No direct loop structure, use recursion: repeat :- repeat.`	`for (int i = 0; i < 10; i++) {`
				`// loop body`
				`}`
					`for i in {1..10}; do`
					`# loop body`
					`done`

ASSESSMENT RUBRIC: ASSIGNMENT 1

Criteria	Weightage	(Excellent) 4	(Good) 3	(Fair) 2	(Poor) 1	Score
Introduction	5	The introduction completely and concisely summarizes the task in one paragraph.	The introduction summarizes the task.	The introduction does not completely summarize the task.	There is little to no evidence that an introduction is included in the report.	
Quality of Information (5 programming languages)	30	Complete contents as required by lecturer. The information included is accurate and completely addresses each component of the assigned task question. Extra information included.	20% - 30% of the required contents not included. The information included adequately addresses each component of the assigned task question.	40% - 70% of the required contents not included. The information included inadequately addresses the assigned task question. The information included is inaccurate.	80% of the required contents not included. There is little to no evidence of accurate content information. Content out of topic completely.	
Syntax Comparison table	20	Complete contents as required by lecturer. The information included is accurate and completely addresses each component of the assigned task question. Extra information included.	20% - 30% of the required contents not included. The information included adequately addresses each component of the assigned task question.	40% - 70% of the required contents not included. The information included inadequately addresses the assigned task question. The information included is inaccurate.	80% of the required contents not included. There is little to no evidence of accurate content information. Content out of topic completely.	

Organization	5	The report content has been organized using the appropriate method. The required information is easy to locate within the report.	The report content has been mostly organized using a logical sequence, but some flaws exist. The required information is generally easy to locate within the report.	The report content has been organized using a somewhat logical sequence. The presentation is sometimes confusing.	The report content is disorganized. The required information is difficult to locate within the report.	
Neatness	5	Tidy and organized. Complete format as required by lecturer.	Tidy and clean. Not follow 10% to 15% of the format as required by lecturer.	Tidy. Not follow 16% to 25% of the format as required by lecturer. Spelling mistakes	Irregular fonts. Not follow some of the format as required by lecturer. Confusing organization. No logical flow.	
Timeliness	5	Submit on time	Submit one day late	submit two days late	Submit more than two days late	

Team Member Evaluation (30%)

1) Task Completion
2) Quality of Work
3) Able to explain clearly
4) Problem-Solving &Technical skills
5) Explain things using programming language terms
6) Q&A

	Total Score	