The Language of Machines: Purpose, Use, and Application of Programming Languages

Anthony J. Coots

National University

TIM-8102: Principles of Computer Science

Dr. *******

May 4th, 2025

The Language of Machines: Purpose, Use, and Application of Programming Languages

Computer languages are central to everything computers do today, much like human languages are central to communication. Both computer and human languages share a primary goal in precise and effective communication. Computer languages provide instructions that a central processing unit can interpret and execute, making input the desired output. This paper explores the purpose, use, and applications of computer languages and how these languages resemble human languages. Additionally, it will analyze the relationship between syntax and semantics, how data is manipulated through input, processing, and output, and compare different categories of programming languages. Finally, it will address the rationale behind language selection by programmers and the factors that influence these decisions.

Purpose, Use, and Applications of Computer Languages

Computer languages connect human users to computer hardware. Their primary purpose is to translate human intentions into instructions that a computer can understand and execute. Computer languages allow people to communicate their goals to machines. For example, if a user wants to track inventory for a store, they would likely use a software program designed for that task. This program would be written in a high-level programming language, compiled into object code, and then translated into machine instructions that the central processing unit can process.

Across various domains, each computer language has its specific use cases. The languages are usually classified as high-level or low-level, depending on the level of abstraction from the hardware. The higher the language level, the more the language is abstracted away from

the machine hosting the language, whereas low-level languages require a complex understanding of the hardware but offer more direct control of the system.

Between high-level and low-level languages lies a trade-off between interpretability and efficiency. To take just one small step away from the hardware, assembly language is a language closer to the machine, sacrificing a small amount of efficiency for better interpretability than machine code, and is usually used for embedded systems. C, C++, and COBOL are considered Turing complete; they can perform any computation given enough time and memory. They have been computer languages used for decades because of their versatility, and are often used in systems programming and software that prioritizes performance and efficiency. Languages such as JavaScript and HTML have more specialized purposes. HTML for structuring web content and JavaScript for web interactivity. Python, R, or SPSS are huge for a variety of applications involving data analysis, machine learning, and statistics.

Computer Languages and Human Languages

Computer and human languages share important similarities, more than the average user might think. Human languages are often more variable and complex in vocabulary and grammar. Yet, both rely on two key rules, syntax and semantics, to convey information and communicate correctly.

In both cases, proper syntax is utilized to structure ideas or instructions so the intended receiver, human or machine, can understand. For example, how words are ordered in English to express something coherently (i.e., “The duck swam after the bread” vs “The bread swam after the duck”) matters in computer languages, so commands are executed in the proper order. A misplaced bracket, semicolon, or even a comma can lead to a significant failure of a rocket ship

launch Shirer (2024) details in their explanation of computer programming languages, like a misplaced word can change the meaning of a sentence.

Semantics also shares similarities between the two. Each sentence in a given human language conveys a logical flow of thought, like each line of code in a computer language contributes to the logic of a program. Both computer languages and human languages require consistency and structure to work appropriately.

Furthermore, another fundamental concept in computer science, abstraction, exists in both languages. In high-level programming languages, developers do not need to understand the underlying machine code to write programs. Similarly, there is no need to understand Latin roots or etymology in human languages to speak modern English well. Though just as a programmer could use assembly or machine language for greater control and efficiency, an interested human could study Latin to understand modern vocabulary better.

The distinction between high-level and low-level computer languages can loosely be compared to learning a language close to one's native tongue and a more distant or ancient language. For instance, a purely conceptual take would argue that someone fluent in English might find it easier to learn French than Russian, maybe or maybe not vice versa, just as a Python programmer might find Java easier to understand than assembly. Both human and computer languages serve the same purpose, to express ideas and accomplish goals through communication.

Semantics and Syntax in Programming

Semantics and syntax define how a computer writes and understands instructions. Syntax works with the structure or rules of how code should be written in a given language, similar to grammar in human languages; just as a sentence is grammatically incorrect, a line of code is syntactically incorrect. Semantics, conversely, pertains to the logic behind the structures; semantics is "how," where syntax is "what."

Shirer (2024) mentions that an expression such as "1 + pizza" might be valid syntactically in different languages, provided that 1 is an integer and pizza represents an integer variable; otherwise, it is semantically useless. This shares the sentiment of a human language (English) when saying "The banjo jumped over the river," perhaps a great passphrase that is grammatically correct, just semantically illogical. Shirer (2024) clarifies syntax as the valid combinations of words, symbols, and numbers, while semantics interprets such combinations, and programs must comply with both to function appropriately. Syntax errors can stop a program from running, and semantic errors will very well let a program run, but will be without apparent failures.

Extending the importance of semantics, Coblenz (2017) highlights how poor semantic definitions may lead to security flaws or unintended behavior. In their study on immutability in Java, they demonstrate that by extending the language with a tool called Glacier, programmers can enforce transitive immutability, resulting in fewer errors and more secure code. In blockchain environments, contracts must be exact and immutable; thus, semantic precision must be definite, as a single misinterpretation of logic may result in sizable financial loss.

As programming languages evolved from machine code to assembly and eventually to high-level languages like C, COBOL, and Python, syntax and semantics were abstracted to make programming more accessible. Bear in mind that the more accessible, interpretable programming languages, such as Python, will eventually be turned into machine code. However, this abstraction doesn't eliminate the requirement of semantic clarity. Doskas (2021) shows in a series of simple examples in C and Python, the same operations involving adding values to an accumulator, and how different this looks across languages, yet there exists a shared understanding of the meaning of the code. Python has looser syntax, making writing easier and faster, but places more burden on semantic clarity.

Brborich et al. (2020) explore how procedural and object-oriented programming paradigms rely on clear syntax and coherent semantics. Maintainable code isn't just well formatted; it must also mean something that developers in the future can understand and modify as needed. Their study finds differences in maintainability performance between paradigms, which critically change how syntax and semantics shape the code for machine execution, how humans comprehend the code, and how different paradigms require different understandings similar to computer languages.

As described in the studies, a significant interaction between syntax and semantics creates the groundwork for computer languages. Syntax defines how code can be compiled or interpreted through various combinations of keywords and characters, while semantics aims to align with a programmer's intentions. When syntax and semantics are well defined, they may contribute not only to optimal execution but also to proper security and maintainability for

operations related to data manipulation, including input, processing, creation of valuable operations, evaluation, intelligence, and machine learning.

Input, Processing, and Output

Most programs follow a universal structure of input, processing, and output. Input can come from users, sensors, files, or APIs. That input is then transformed through logic, conditions, or algorithms into useful output ranging from a simple response to complex predictions. For example, machine learning models take in massive datasets, process them through algorithms (e.g., neural networks), and output predictions or classifications. This requires handling of how the code is written, syntax, what the logic does, and semantics for useful output. Sun et al. (2025) demonstrate this in their integration of economic input-output and network analysis, where their Python models use data on resource use (input) that is processed through network mapping to produce output that reveals environmental impact patterns.

C-Based vs High-Level Comparison

	C-Based Languages (C++)	***High-Level Languages (Python)***
Syntax Complexity	High	Low
Memory Handling	Highly Complex	Abstracted
Performance	High	Moderate or Low
Learning	Harder	Easier
Use Case	Embedded systems	Data science

Rationale for Language Selection

Choosing a programming language is seldom arbitrary; technical need, project constraints, and human considerations decide which programming language to use in a project. Programmers must account for the project's domain, where web development may utilize JavaScript or HTML, while embedded systems may demand low-level languages such as C-based. Performance may also influence deciding when speed or memory efficiency is a must. Alternatively, if development speed is prioritized, then Python is currently the language of interest since its abstraction from the hardware level is its largest contributor to readability and versatility. Last, community support of a language also plays a role since active communities may have forums for troubleshooting.

Brborich et al. (2020) highlight that maintainability and paradigm choice influence language selection. While object-oriented programming promotes better modularity and maintenance, a common staple of the paradigm, procedural programming was favored for studying the pizzeria application. The programming paradigm affects not just development but also continuous support and scalability.

From an educational standpoint, Hromkovič and Komm (2024) demonstrate the importance of language simplicity and clarity for beginners. They refer to Niklaus Wirth and the language Pascal, arguing that the teaching value of a language must prioritize clarity and structure- in other words, syntax and semantics. Ultimately, a programmer's rationale for selecting a language must balance function, readability, performance, and context.

References

Brborich, W., Oscullo, B., Lascano, J. E., & Clyde, S. (2020). *An observational study on the maintainability characteristics of the procedural and object-oriented programming paradigms*. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)* (pp. 1–10). IEEE. https://doi.org/10.1109/CSEET49119.2020.9206213

Coblenz, M. (2017). *Principles of usable programming language design*. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)* (pp. 469–470). IEEE. https://doi.org/10.1109/ICSE-C.2017.24

Doskas, C. (2021). Python Programming: Object-Oriented Programming. *ISSA Journal*, *19*(2), 44–47.

Hromkovič, J., & Komm, D. (2024). Preface: Designing or Choosing Languages for Teaching Programming. *Informatics in Education*, *23*(4), 719–721.

Shirer, D. L. (2024). Computer programming languages. *Salem Press Encyclopedia of Science*.

Sun, W., Ma, F., Tzachor, A., Wang, Y., Gong, Y., Wang, C., Hu, X., & Wang, H. (2025). Bridging environmentally extended input-output models and complex network analysis: A bibliometric analysis of trends and opportunities. *Journal of Cleaner Production*, *486*. https://doi.org/10.1016/j.jclepro.2024.144427