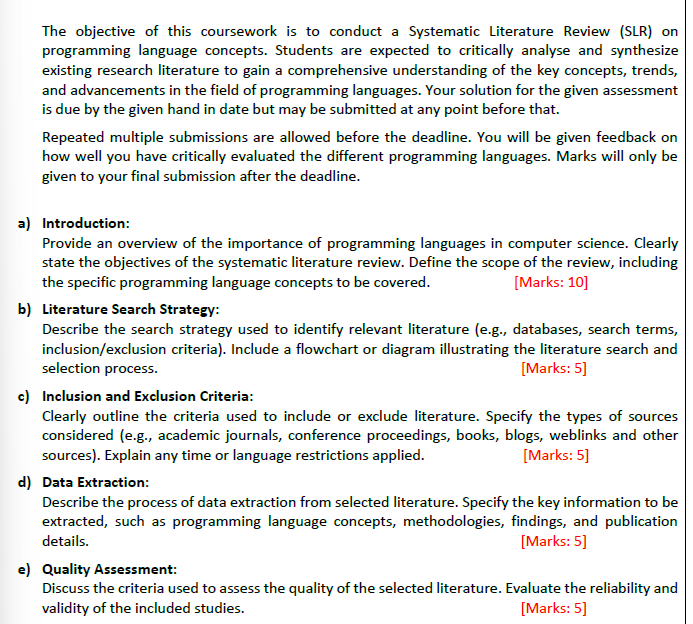
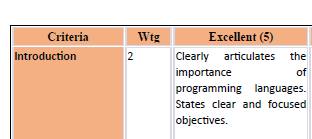


Student Declaration

I hereby declare that this coursework is written by me and is a result of my own work. I have not copied ideas/research and academic sources.





To write:

* Importance of programming languages in comp sci
* Objectives of systematic literature review
* Scope of review
  + Specific programming languages concepts to be covered

>> to avoid opposite of perpetual vagueness – have to be wary of choice of terms for new concepts – ‘name’, ‘address’, ‘value’ have meanings with complicated associations & overtones in ordinary usage or in mathematics – can misuse in terms of context/meaning

<https://link.springer.com/article/10.1023/A:1010000313106>

A programming language consists of a set of rules and steps meant for a computer’s understanding to execute a task.(<https://www.ijresm.com/Vol.3_2020/Vol3_Iss1_January20/IJRESM_V3_I1_134.pdf> ) Programming languages allow people such as developers to communicate with computers through machine, high level and assembly language, and thus allowing for more innovation and advancements in the world of technology and daily life. (Source: <https://www.ijresm.com/volume-3-issue-1-january-2020/> )

The objectives of this systematic literature review are to cross examine existing literature in the following aspects:

1. Identify and understand fundamental programming language concepts.
2. Analyse and interpret trends in programming languages.
3. Explore and assess advancements in programming languages.
4. Evaluate impact of programming languages on industry practices, education, and research.

The scope of the review involves the following language concepts:

1. Grammar (syntax and semantics)
2. Data types and structures
3. Control structures
4. Functions
5. Object-Oriented Programming (OOP)
6. Functional Programming
7. Concurrent Programming
8. Language Design and Implementation

In addition, trends regarding how programming languages develop according to industry demand will be identified and explored, providing a deeper understanding on the impact of programming languages in the field of computer science through the decades.

b) Literature Search Strategy

Databases: Google Scholar, Research Gate, IEEE Explore

Search terms: Programming languages, syntax, programming language concepts, data type, …

Search execution: Queried using boolean operators

c) Inclusion and Exclusion Criteria

Inclusion criteria: Peer-reviewed journal articles, conference papers, books; discuss programming language concepts, theories, trends, and/or advancements; written in English

Exclusion criteria: Non-peer-reviewed sources (blogs, forums); non-English publications unless translated version were available; papers not related to programming language concepts

d) Data Extraction

The process of extracting data from the chosen literature include choosing literature that only had information about programming language concepts such as syntax, data types, and concurrency, the methodology used such as case studies and literature reviews, key findings of programming language concepts and publication details such as title, author, publication year, and abstract.

e) Quality Assessment

Relevance to the topic matter. Credibility, only from peer-reviewed journals, reputable conference proceedings, books authored by experts. Acknowledgement of biases.

f) Synthesis and Analysis + Discussion

**Summarize the main programming language concepts identified in the literature. Analyse trends and patterns across different studies. Identify gaps, inconsistencies, and areas where further research is needed**

**Discussion: Interpret the findings in the context of the broader field of programming languages. Discuss the implications of the identified concepts for the development and evolution of programming languages. Highlight any challenges or controversies in the literature**

Main programming language concepts:

The first concept is syntax, where a language is defined by expression that are validated by a fixed set of rules for that language (<https://my.uopeople.edu/pluginfile.php/57436/mod_book/chapter/37622/understanding_programming_languages.pdf> ). Syntax can be identified via two types, which are concrete syntaxes that includes blank spaces, brackets and parentheses, and also abstract syntaxes which can be represented with trees and abstracted blank spaces and parentheses.

Another concept closely related to syntax is semantics, or the context of an expression written in a programming language. A semantic can describe the content within the program’s memory, and it provides the following instruction to be done.

Next are data types. This is a core concept for programming concepts, as they represent a value and a certain set of instructions attached to those values (<https://my.uopeople.edu/pluginfile.php/57436/mod_book/chapter/37622/understanding_programming_languages.pdf> ). Types include primitive data types built into programming languages, including int, char, bool, etc (<https://dl.acm.org/doi/epdf/10.1145/6041.6042> ). There are also abstract data types (ADTs), which are data specified using higher-order equations through constructors and operations (<https://www.sciencedirect.com/science/article/pii/S0304397596001612> ), where examples include stacks used for data structures like lists or arrays containing data such as integers (<https://dl.acm.org/doi/epdf/10.1145/6041.6042> ). There is a distinct trend in programming language types, as early as the years of FORTRAN, where integers and floating-points were identified by the first letter, but ALGOL 60 introduced identifier declarations. PL/I then followed with typed arrays and pointers but provided weak compile-time type checking, to which Pascal attempted to solve this problem but introduced another problem of type equivalence. Then, object-oriented languages were introduced, such as Simula, Modula-2 and Ada, providing polymorphism in operations. With data types, comes static and dynamic typing. The argument is static typing makes code more verbose, unsafe, less reusable and inexpressive compared to its counterpart, dynamic typing. Meijer and Drayton [date] note that in modern times, some prefer dynamic typing as it allows for seamless integration of data intensive applications, however they emphasize that focusing on the strengths of both types may be more ideal as context is crucial. They note that no concrete study has been done on the possibility of a hybrid type (<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e670c72afa489ffdfab77852a0c9781e082fc266> ).

Control structures are also a common concept in programming, where it allows for a defined order of execution of assignment statements. Well-structured control statements include choice and loop statements, which have if-statements and case-statements and for and while statements respectively. For if-statements, expression execution is determined by a Boolean value, and the implementation differs in programming languages. As C does not have a Boolean type, it uses integer values in its place, where true equates to non-zero values and zero value as false (<https://my.uopeople.edu/pluginfile.php/57436/mod_book/chapter/37622/understanding_programming_languages.pdf> ). Languages like Java and Python do have Boolean values. In contrast is the switch case statement, where the condition is an exact value, otherwise the default statement is executed. Another difference is the usage of ‘break’, that can be seen in C programming, explicitly halting the switch case statement if an exact value has been met. One disadvantage of the switch case is needing to know the values to be detected, otherwise the default statement will be executed, and run time checks can prove time consuming and inefficient should a desired output not be met (<https://my.uopeople.edu/pluginfile.php/57436/mod_book/chapter/37622/understanding_programming_languages.pdf> ).

Another concept in structured programming is the loop statement, where statements are executed repeatedly until it reaches a condition that can cause an exit or a break from the loop. It is challenging to program, being prone to errors due to incorrect boundaries and its inefficiency if declared wrongly. ( <https://dl.acm.org/doi/pdf/10.1145/954604.954615> ) Loop statements include for, while and do while loops. In terms of loop statements, there are doubts surrounding the usage of utility goto, and further research for higher level control structures are needed. In a paper by Ledgard and Marcotty, they state that programs using goto versus without can be more efficient and both provide the same clarity. On the other hand, a paper by Ishihata and Hikita propose a generalised loop exit statement which can better represent a postlude action for each exit to handle events. However, this proposal is not suitable to PASCAL as it cannot handle certain cases in exception handling.

Another concept are subprograms. A subprogram is a chunk of written code that can be called when needed within the bounds of the program. It includes the declaration of the subprogram, such as its name, parameter list if any apply, and the type to be returned. Within it also exists local declarations which cannot be accessed outside of the subprogram, and a number of statements to be executed. A subprogram that returns a value is called a function, while a subprogram that does not return a value are procedures. In C programming, a function that returns no type must write ‘void’ in its place. Subprograms are useful when statements need to be executed repeatedly at different stages of the program, which in turn enhances readability, memory saving, and efficient testing (<https://my.uopeople.edu/pluginfile.php/57436/mod_book/chapter/37622/understanding_programming_languages.pdf> ). In Fortran, there is a special syntax to call a procedure, while other languages such as C and Java only require the name of the procedure, and any parameters that need to be passed in. It is important that for a function, the result returned after statement execution is the same as the value declared to be returned. Because of this, some confusion may arise on the compiler’s side if the declared return type and actual return type are different, thus implicit type conversion is introduced such as in C, compared to Ada where the both types must be exactly the same.

Over the decades since mechanical calculating engine in the 17th century, to calculating engines by Charles Babbage, to electronic devices through the use of punch cards in the 19th century, computers have seen evolutions and developments, particularly during wartime.

@

Before proper programming languages were created, existed a ‘computer’ designed by Charles Babbage for an Analytic Engine, which could be used to program code with the help of Ada Lovelace, through punched cards (<https://www.sciencedirect.com/science/article/pii/009605519290019J> ). Following the discovery of assembly language, FORTRAN, COBOL, LISP and ALGOL were created, with FORTRAN being the first modern programming language by 1955, and C was released in 1969. According to Chowdhary <https://arxiv.org/pdf/2007.02699> <https://arxiv.org/abs/2007.02699> [source], as programming languages progressed and improved, some features were discarded, and some were continued via mutation of evolution theory. An example of this was C had the control feature from FORTRAN and not the goto statement. Programming languages such as AWK and Shell were left in favour of Perl and Python in terms of scripting languages. Chowdhary observed that new programming languages are created 11 per decade for the past 60 years. It is also to be noted that object-oriented programming (OOP) came about when in the era of FORTRAN and COBOL, there was a need to handle larger programs, showing the trend and needs of languages as programs grow in complexity and size. This paper also poses an interesting stance: Do we need more programming languages, with the abundance of languages already available in the modern day? And is there a need to reinvent the wheel? This author believes no one language is a one-size-fits-all, with the evolution of multi-cores, cloud computing and distributed systems. Thus, programming languages need to continue to develop and evolve, as the needs of programs are contextual in nature.

h) Conclusion

Further research is suggested on hybrid dynamic and static typing, as of current no language exists on a possible language that can provide for both. More research on loop statements is also encouraged.

Trends:

<https://ieeexplore.ieee.org/abstract/document/1438333?casa_token=aNDdhLr7_uAAAAAA:ohisHyfqwRz5UanhWqYvlKbbRcwXi_vZNFwsOLzzVMxpzIigZexPddeaHeLXngAfBIPuqviB>

<https://ieeexplore.ieee.org/abstract/document/1438333/references#references>

Programming concepts:

<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e94334412e47eb35ebb418037f7279a61021151a>

<https://books.google.com.my/books?hl=en&lr=&id=3EmjBQAAQBAJ&oi=fnd&pg=PP1&dq=related:F0RVg0yXOAgJ:scholar.google.com/&ots=D7DevtaxrZ&sig=-lD5uy-Bt32QIE2NzBz07Lcy04w&redir_esc=y#v=onepage&q&f=false> (military, birth of programming languages, AI,, grammar, …)

<https://www.sciencedirect.com/science/article/pii/009605519290019J> (history of computer programming languages – 4th gen languages, paradigms, environments)

<https://my.uopeople.edu/pluginfile.php/57436/mod_book/chapter/37622/understanding_programming_languages.pdf> (elements of programming languages, environments, data types, control structures, polymorphism, concurrency, OOP)

Types. Data abstraction, polymorphism

<https://dl.acm.org/doi/epdf/10.1145/6041.6042>

go to statements

<https://dl.acm.org/doi/epdf/10.1145/356635.356640>

More on the controversy between static and dynamic typing:

<https://link.springer.com/article/10.1007/s10664-013-9289-1>

Janhwi Goyal, A. S., Archee Gupta, Malvika. (2020). Why there are So Many Programming Languages? *International Journal of Research in Engineering, Science and Management*, *3*(1), 515-517. (Janhwi Goyal, 2020)

Ben-Ari, M. (1996). *Understanding programming languages*. Wiley Chichester.

Janhwi Goyal, A. S., Archee Gupta, Malvika. (2020). Why there are So Many Programming Languages? *International Journal of Research in Engineering, Science and Management*, *3*(1), 515-517.

Goyal, J., Singla, A., Gupta, A., Malvika. (2020). Why there are So Many Programming Languages? *International Journal of Research in Engineering, Science and Management*, *3*(1), 515-517.

Ben-Ari, M. (1996). *Understanding programming languages*. Wiley Chichester. (Ben-Ari, 1996)

Strachey, C. (2000). Fundamental Concepts in Programming Languages. *Higher-Order and Symbolic Computation*, *13*(1), 11-49. <https://doi.org/10.1023/A:1010000313106>

Cardelli, L., & Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *ACM Comput. Surv.*, *17*(4), 471–523. <https://doi.org/10.1145/6041.6042>

Jean-Pierre, J., & Mitsuhiro, O. (1997). Abstract data type systems. *Theoretical Computer Science*, *173*(2), 349-391. <https://doi.org/https://doi.org/10.1016/S0304-3975(96)00161-2>

Chowdhary, K. (2020). On the evolution of programming languages. *arXiv preprint arXiv:2007.02699*.

Linda Weiser, F. (1992). From Babbage to Babel and beyond: A brief history of programming languages. *Computer Languages*, *17*(1), 1-17. <https://doi.org/https://doi.org/10.1016/0096-0551(92)90019-J>

Meijer, E., & Drayton, P. (2004, 01). *Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages*

Yasumura, M. (1977). Evolution of loop statements. *SIGPLAN Not.*, *12*(9), 124–129. <https://doi.org/10.1145/954604.954615>