1.2 DESCRIBING SYNTAX

Syntax: the form or structure of the expressions, statements, and program units.

In a programming language, **Syntax** defines the rules that govern the structure and arrangement of keywords, symbols, and other elements. Syntax doesn't have any relationship with the meaning of the statement; it is only associated with the grammar and structure of the programming language.

A line of code is syntactically valid and correct if it follows all the rules of syntax. Syntax does not have to do anything with the meaning of the statement. Syntax errors are encountered after the program has been executed.

Some examples of syntax errors include missing semicolons in C++, undeclared variables in Java, although such errors are easy to catch.

Programming languages are less accommodating to syntax errors. Syntax determines how we organize the elements in our code to make it legible to the computer. If there's a syntax error, the computer might not be able to read it.

Here are some examples of what programming syntax can determine:

- whether we use lower-case or upper-case characters
- how we notate code comments
- how we use whitespace
- how we indicate the relationships between statements (individual commands issued to the computer)

Syntax is important in programming because it would be impossible to write functioning code without it. Code is a set of instructions written in a language that a computer can read and act on. If there are syntax errors in the code, the program won't work.

Semantics: the meaning of the expressions, statements, and program units

Semantics refers to the meaning of the associated line of code and how they are executed in a programming language. Hence, semantics helps interpret what function the line of code/program is performing.

Semantic errors are encountered and handled during runtime of the program execution. If there is any semantic error and even when the statement has correct syntax, it wouldn't perform the function that was intended for it to do. Thus, such errors are difficult to catch.

Syntax and semantics provide a language's definition

Comparing syntax in programming languages

Syntax can vary quite a bit between languages. Python is clearly simpler than Java and C++. This is because Python was designed to read like a human language. It doesn't take much time to learn Python well enough that you can read a program and understand basically what it's doing. Here are a few specific ways that Python syntax is simpler than Java and C++.

Python doesn't require users to declare variable types, while Java and C++ do.

- Whitespace is an important part of Python syntax. The indentation of lines plays an important role in indicating the relationships between lines of code. In Java and C++, semicolons are used to indicate the end of a statement, and curly brackets are used to demarcate groups of statements.
- Code comments in Python are notated with #, while in Java and C++ they're notated with //.
- Python is unique among these three languages, and Java and C++ are quite similar. When languages have similar syntax, this often means that they're based on an earlier language. Java and C++ are similar because both are based on the C programming language. This is another way in which programming languages are similar to human languages. Similarities of vocabulary, grammar, and syntax generally indicate a common ancestor.

These similarities can be really helpful when trying to learn a new language. If you're proficient in Java and you're trying to learn C++, or vice versa, you'll encounter some familiar elements that will streamline the process.

Users of a language definition

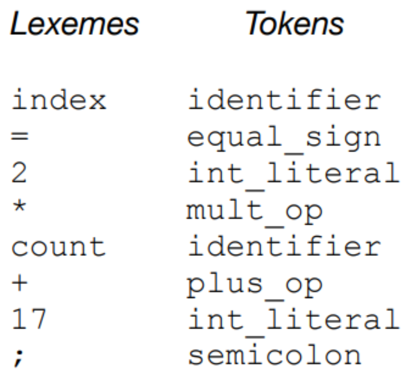
- Other language designers
- Implementers
- Programmers (the users of the language)

Terminologies

- A sentence is a string of characters over some alphabet
- A language is a set of sentences
- A lexeme is the lowest level syntactic unit of a language (e.g., * , sum, begin)
- A token is a category of lexemes (e.g., identifier)

Example: Lexemes and Tokens

index = 2 * count + 17



Lexemes	*Tokens*
index	identifier
=	equal_sign
2	int_literal
*	mult_op
count	identifier
+	plus_op
17	int_literal
;	semicolon

Formal approaches to describing syntax:

Recognizers

- A recognition device reads input strings over the alphabet of the language and decides whether the input strings belong to the language

Example: syntax analysis part of a compiler

Generators –

- A device that generates sentences of a language

- One can determine if the syntax of a particular sentence is syntactically correct by comparing it to the structure of the generator