PROGRAMMING LANGUAGES

ISSUE 6, VOLUME 4
OFFICE OF TECHNOLOGY STRATEGIES (TS)

INTRODUCTION

You might wonder why we have programming languages for computers and why there are so many languages. Why not just use one universal language that exists to operate computers?

Programming languages are tools designed to bridge the gap between computer processing units (CPUs) which read machine language, and the natural language we speak. Programming language allows for clarity in expressing instructions to computers, which form the basis of all computer functions. The instructions must be precise and clear every time they run. This Tech Insight introduces programming language and paradigms, describes programming language attributes, and provides examples and uses of different types of programming languages, including usage at the Department of Veterans Affairs (VA).

OVERVIEW

A programming language is a language of code that describes instructions for a computer. There are thousands of programming languages, and more are created and revised as technology and ideas continue to evolve. Most programming languages use English words and have a mathematical basis. Languages are designed with syntax (form) and semantics (meaning) in mind, two factors that affect whether code is easy to fix or quickly executed. Programming languages have many purposes, including education, system programming, scripting languages, or compiling languages. Code is written in a variety of programming languages and is organized like a company. Each language supports different activities—compiling, executing, supporting, accessing, etc. Fewer lines of code lead to quicker production and more efficient programs. However, there is no "best" language; there are tradeoffs in using each language, expressed by programming language paradigms.

Programming paradigms are styles of building the structure and elements of computer programs. They describe the variety of dimensions that they support. Languages can be classified along multiple paradigms, including imperative, declarative, functional, object-oriented, procedural, or machine code. Programming languages often support multiple paradigms.

DECLARATIVE AND IMPERATIVE LANGUAGES

One paradigm allows people to write coded instructions to specify a desired result—the what; this is a *declarative paradigm*. For example, Structured Query Language (SQL) is a common declarative language used for storing, manipulating, and retrieving data in databases.

Another paradigm is written as a sequence of operations to perform—the how; this is an *imperative paradigm*. In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform.

LOW AND HIGH LEVEL LANGUAGES

Programming language can be divided into low and high level languages, ranging in complexity and flexibility. Absolute machine languages are the precursors to programming languages. The programs, in decimals, zeros, and ones, were read in from punched cards or magnetic tape. Absolute machine languages were named first-generation programming languages (1GL). Assembly languages are closely tied to the architecture of specific computing machines, but are a small step to natural language. Assembly languages are second-generation programming languages (2GL). The first high-level programming languages, or third-generation programming languages (3GL), were written in the 1950s.

Scripting languages are also high level languages. Programs written in scripting languages operate by following instructions (script) to automate the execution of tasks that would alternatively be executed one-by-one by a human operator. Scripting languages do not require translators to generate machine code from source code. Two examples include Ruby and Python. Ruby is a dynamic, reflective, object-oriented, general-purpose programming language. It is a multi-paradigm language, with functional, object-oriented, and imperative capabilities. Python is a high-level programming language. It uses a scripting or glue language to connect together disparate components of existing code. Python's simple, easy-to-learn-and-read syntax is relatively similar to natural language, reducing the cost of program maintenance.

COMPILING LANGUAGES

To bridge the gap between machine language and natural language, computer scientists developed “compilers.” A compiler is a computer program that transforms source code written in a programming language or computer language, into another computer language. Any program written in a high level programming language must be translated to object code, before it can be executed. Therefore, compilers are very important to programmers. Examples

of compiling languages include Java, C#, C++, and Common Business-Oriented Language (COBOL).

Grace Hopper wrote the first compiler program in 1952. Her work popularized the idea of machine-independent programming languages, which led to the development of COBOL, one of the first high-level programming languages. In 1997, Gartner approximated that 80% of business transactions around the world were supported by COBOL code.

Object Oriented Language

Object-oriented languages started in the 1970s and are still popular today. Recently, two new languages have become widely used in the micro services and development operations (DevOps) world – Go and Rust. Go is used in data analytics activities. While Rust is new, it is gaining a lot of traction in the startup world because it allows for rapid program development.

Choosing a Language

To create a unique, customized technology solution, programmers select from a buffet of languages which are assembled together to achieve required capabilities. There are many factors to consider when choosing programming languages. No single language is the best choice, but rather each language has pros and cons when used in a coding environment. Factors to consider when selecting a programming language include: the targeted platform; production deadlines; performance requirements; support and community; language interoperability; code speed; and security. Selecting the right combination of programming languages yields solutions that are concise, clear, interoperable, easy to document, and easy to fix.

VA Contribution to Medical Programming

Massachusetts General Hospital Utility Multi-Programming System (MUMPS) originated in 1966 for the healthcare system, and is still in use today in electronic health record systems. VA was one of the earliest major adopters of the MUMPS language. VA developments and contributions to the free MUMPS application codebase influenced medical programming around the world. In 1995, VA's Decentralized Hospital Computer Program (DHCP) was the recipient of the Computerworld Smithsonian Award for best use of Information Technology in Medicine. In July 2006, VA received the Innovations in American Government Award presented by Harvard University for its extension of DHCP into the Veterans Health Information Systems and Technology Architecture (VistA). MUMPS databases track clinical data in most of the VA hospital systems, the Indian Health Service, and the Department of Defense hospital system.

Programming languages are to the digital world as the laws of physics are to our physical environment. With programming languages, developers write new dimensions within which end users work and live. Programming languages are truly a fabric of life in the twenty-first century and will continue to grow in importance over time.

Read more technology topics in TS Tech Insights: Anatomy of a Computer and Development Operations; and Enterprise Design Patterns. If you have any questions about programming languages, don't hesitate to ask TS for assistance or more information.

TS TECH INSIGHT SERIES

The monthly Tech Insight series aims to help readers make better decisions and be more informed customers (of Office of Information & Technology's products and services) by providing them with high-level overviews of technology issues that impact or will impact VA's Information Technology (IT) environment. Tech Insights introduce topics in an easily digestible fashion by presenting background information on the topic, clearly explaining its importance within VA, and providing recommendations for success from TS. View all TS Tech Insights here.

DISCLAIMER: This document includes links to websites outside VA control and jurisdiction. VA is not responsible for the privacy practices or the content of non-VA websites. We encourage you to review the privacy policy or terms and conditions of those sites to fully understand what information is collected and how it is used.