An overview of scripting languages

Alexander Kanavin
Lappeenranta University of Technology, Finland
Teachers: Barbara Miraftabi and Jan Voracek

1 Dec 2002

Abstract

Over the last few years the interest in scripting languages has dramatically increased. These languages have many important advantages over traditional programming languages, most notably, they eliminate the need for compilation, they manage memory automatically and they include high-level datatypes. In the future these languages are likely to become the core of most programming projects, because of their power to 'glue' existing components together into a working application system.

1 Introduction

Over the last few years the scripting programming languages made a giant leap ahead. About ten years ago they were viewed as an auxillaru tools, not really suitable for general programming per se. Now they generate a tremendous amount of interest both in academic circles and in the software industry.

The execution speed and memory sonsumption of scripting languages vs. the traditional languages is studied in [6]. Article [4] presents a historical background of the scripting languages. In [1] a practical case of using the scripting languages in a commercial environment is presented. Finally, [3] presents some trends for the future.

In this overview I first try to define what scripting languages are. Then a classification of the languages based on their application ares is presented. After that, the most popular of scripting languages are presented, and the peculiar features of each one are highlighted. The paper is concluded with the discussion on why scripting languages are important, and what their role is going to be in the future.

2 What are scripting languages?

The boundary between the scripting programming languages and the traditional ones is somewhat blurry. However, it is possible to highlight a few characteristics of scripting languages, that, when taken together, could serve as a definition:

- They are interpreted or bytecode-interpreted and never compiled to native code

- The memory handling is done by a garbage collector and not by a programmer
- They include high-level data types, such as lists, associative arrays and so on
- The execution environment can be integrated with the program being written
- The scripting programs (or simply, scripts) can access modules written in lower-level languages, such as C.

Not every scripting language has the whole set of these features. For example, shell scripts cannot access C modules. But it's a scripting language nevertheless.

The main idea behind the scripting languages is their dynamic nature, that allows to treat data as a program and vice versa.

The list of the scripting languages includes: shell, awk, Perl, TCL, Python, Java, Lisp and many others.

3 Application areas

The article [4] introduces four main usage areas for scripting languages:

- Command scripting languages
- Application scripting languages
- Markup languages
- Universal scripting languages

3.1 Command scripting languages

Command scripting languages are the oldest class of scripting languages. They appeared in 1960, when a need for programs and tasks control arised. The most known language from the first generation of such languages is JCL (Job Control Language), created for IBM OS/360 operating system.

Modern examples of such languages include shell language, described above, and also text-processing languages, such as sed and awk. These languages were one of the first to directly include support for regular expression matching - a feature that later was included into more general-purpose languages, such as Perl.

3.2 Application scripting languages

Application scripting languages were developed in 1980s, in the era of personal computers, when such important applications as spreadsheets and database clients were introduced, and interactive session in front of the PC became the norm.

One of the prime examples of these languages is Microsoft-created Visual Basic language, and especially it's subset named Visual Basic for Applications, designed explicitly for office applications programming. This language puts heavy

Name	Year of creation	Developers	Organization
Pilot	1962	-	IBM
JCL	1964	-	IBM
RPG	1965	-	IBM
MUMPS	1969	Okto Barnett+	Massachusetts General Hospital
sh	1971	Steve Bourne	AT&T Bell Labs
Awk	1977	Alfred Aho+	AT&T Bell Labs
csh	1978	-	UC Berkeley
Rexx	1979	Michael Qualishow	IBM UK Laboratories
AppleScript	1993	-	Apple Computer

Table 1: Command Scripting Languages

emphasis on user interface programming and component embedding (such as VBX, OCX, ActiveX). VBA replaced earlier languages, Word Basic and Excel Macro Language, as the universal single language for programming Microsoft Office suite. It influenced such later languages as VBScript (oriented towards creation of OLE components and for work within a browser), and also LotusScript (Lotus Notes programming).

Javascript language also belongs to this class. It is basically a de-facto standard for implementations of the client parts of web-programming projects. Javascript was introduced in Netscape Navigator 2.0 browser. It has a few dialects, such as JScript from Microsoft and ECMAScript (ECMA-262 standard).

Name	Year of creation	Developers	Organization
HyperTalk	1986	-	Apple Computer
Visual Basic	1990	-	Microsoft
JavaScript	1994	-	Netscape Communications
CorelScript	1995	-	Corel
LotusScript	1995	-	Lotus Development
VBScript	1995	-	Microsoft
Pnuts	2001	Toyokasu Tomatsu	Sun Microsystems

Table 2: Command Scripting Languages

3.3 Markup Languages

Markup languages are a special case in the sense that they are not a real programming languages, but rather a set of special command words called 'tags' used to mark up parts of text documents, that are later used by special programs called processors, to do all kinds of transformations to the text, such as displaying it in a browser, or converting it to some other data format. The basic idea of markup languages is the separation of contents and structure, and also including formatting commands and interactive objects into the documents.

The first markup language named GML (Generic Markup Language) was created in 1969 by IBM. In 1986, ISO created a standard called SGML, based on GML ideas.

Perhaps the most known achievements in markup Languages are TeX, HTML and XML. TeX was created in 1979 by Donald Knuth and was designed for precise description of how the documents look, no matter how complicated their structure is. It differs from Postscript created by Adobe in that it is targeted at users who not necessarily need to know how to program. TeX achieved enormous popularity in scientific community, where it can fulfil the need for high-quality rendering of complex formulas (no other language can do that still).

HTML really needs no introduction: it's the basic language of the world wide web. HTML is an SGML application in the sense, that the official HTML standard is defined in terms of SGML. (SGML itself is not a language, but a meta-language, a language for creation of languages).

XML could be briefly described as "SGML's younger brother". It is basically a simpler and streamlined version of SGML with an emphasis on transportation and storage of data, and exchange of data between systems of all kinds. It can be used for complex data transformations and for an unified approach to storing hierarchical data, such as components setup and programming. In 2001 HTML was redefined in terms of XML; this new revision was called XHTML.

Name	Year of creation	Developers	Organization
GML	1969	Charles Goldfarb+	IBM
TeX	1979	Donald Knuth	Stanford University
SGML	1986	-	ISO
HTML	1991	Tim Berners-Lee	CERN
CFML (Cold Fusion)	1995	-	Allaire
DHTML	1996	-	Microsoft
XML	1997 -	W3C	
XHTML	2001	-	W3C

Table 3: Markup Languages

3.4 Universal scripting languages

The languages that belong to this class are perhaps the most well-known. The very term "scripting languages" is accociated with them. Most of these languages were originally created for the Unix environment. The goals however were different.

The Perl programming language was made for report generation, which is even reflected in its name (Practical Extraction and Report Language). It is commonly said that the primary reason for it's enormous popularity is the ability to write simple and efficient CGI scripts for forming dynamic web pages with this language. Perl was there in the right place at the right time.

The Python language was originally made as a tool for accessing system services of the experimental operating system Amoeba. Later it became a universal object-oriented scripting language. Implementations exist for the Java Virtual Machine and also for Microsoft Intermediate Language used on Microsoft .NET platform.

Tcl language was created with the aim of string processing and close integration with Tk library. Tcl's mainly used as an application extension language.

Unlike Perl and Python, which make it easy to write completely standalone programs, Tcl relies heavily on C and C++ extension modules.

Most of the rest of the languages belong to the second wave, that appeared together with web services. The most known of them is the language named PHP, which combines HTML and traditional programing procedures with loops and functions and also provides extemely easy database access. We will look at

Name	Year of creation	Developers	Organization
Perl	1986	Larry Wall	-
Tcl	1990	John Ousterhaut	UC Berkeley
Python	1991	Guido Van Rossum	Stichting Mathematisch Centrum
Ruby	1993	Yukihiro Matsumoto	-
Euphoria	1993	R. Craig	Rapid Deployment Software
Luea	1994	U. Tseles+	PUC-Rio
PHP	1995	Rasmus Lerdordf	-
Mawl	1995	D. Ladd+	-
Pike	1996	Frederik Hubinett	InformationsVavarna
Curl	2000	Steve Ward+	MIT Lab for Computer Science

Table 4: Universal Sripting Languages

some of these languages closer in the next section.

4 Languages overview

4.1 Shell scripting language

The shell is an interactive command-line interpreter of the Unix operating system family. It also includes programming capabilities with all the standard procedural languages constructs: if-then statements, for and while loops, variables, functions and so on.

Simple shell programs are very easy and natural to write. As the program size gets larger however, they tend to become a sort of hodge-podge. Some parts of the syntax are also quite confusing (the quoting rules for example). These rules were introduced as an attempt to preserve shell's main role as an interactive interpreter.

Shell programs generally rely heavily on running and processing the output of external programs, such as Unix filters (sort, wc, grep and so on) and text-processing mini-languages, such as awk and sed.

The major advantage of shell is that almost all Unix variants come with it, you almost never have to install it. However, shell isn't really suitable for any programming tasks, but the simplest ones.

4.2 Perl

Perl was developed by Larry Wall in the late 80s. It is best described as "shell on steroids". It was designed specifically to replace awk and shell as the language for Unix script programming.

Perl is certainly a much more advanced language than shell: it includes stronger data types, including dynamic arrays, and a 'hash' type that allows

fast and convinient lookup of value by the name (in a phonebook fashion). Perl also suppports pattern-driven processing of textual data format; these facilities are built straight into the language and are not part of the library.

Speaking of the library, Perl includes a complete and well thought out binding of almost the entire Unix API. Thus it significantly reduces the need for C for simple tasks that do not require optimizing the memory usage or performance.

Another strong advantage of Perl is that it has a dedicated large community grown around it. Members of this community wrote hundreds of freely available modules that allow you to do virtually everything, from building graphical user interfaces to writing interactive dynamic websites. This collection of modules is known as CPAN.

Perl however has quite a number of drawbacks: as a language it's anything but pretty or elegant. Some parts of it are, quite simply, ugly. The syntax is generally considered to be one of the less readable, and there's even an annual contest to write the most obfuscated Perl program possible.

It is harder to get started in Perl than in shell. Perl also requires extraordinary effort to keep the program design simple and maintain modularity as the program size grows. Some of the more advanced language features look like an afterthought, something that was put there on top of existing things.

Like shell, Perl can be found on almost every Unix installation. Perl is also available and quite well documented on Microsoft platforms.

4.3 Tcl

Tcl (tool control language) is a small language interpreter designed to link with compiled C libraries, providing scripted control of C code.

Some facilities built on top of Tcl achieved widespread usage and perhaps even bigger popularity than the language itself. These include Tk toolkit, one of the most easy to use GUI toolkits that allows for rapid building of buttons, dialog boxes, menu trees and so on, and also Expect, a language that makes it easy to script fully interactive programs that were never intended as being controlled by a script. Tk toolkit is not limited to Tcl; it is often used with Perl and Python for example.

The main advantage of Tcl is that it is extremely flexible and simple. The syntax is somewhat weird (no difference between a function call and 'built-in' operators for example), but quite consistent.

The main drawback of Tcl is that the language has only weak facilities for namespace control and modularity, and thus makes it difficult to write large programs. Some of the oddities of syntax are a bit of a headache for newcomers.

Plain Tcl provides access only to a small part of the Unix API (just file handling, process-spawning, and sockets). Tcl extensions exist but are not guaranteed to be installed everywhere.

Tcl implementations exist not just for Unix, but also for the Windows family. Tcl/Tk scripts will run unchanged on any implementation.

4.4 Python

Python is a scripting and prototyping language that can be integrated with C. Like Tcl, it can both import data from and export data to dynamically loaded C libraries, and can be called as an embedded scripting language from C. Its

syntax is something in between C and the Modula family, but also has the unusual feature that block structure is actually controlled by indentation (there is no analogue of explicit begin/end or C curly brackets). This is certainly the most disputed over feature of the languages, but most people seem to like it, once they get used to it and consider it a real time-saver.

The Python language is a very clean and elegant design. It gives the program designers a choice to write in an object-oriented style, the traditional procedural style or a mix of those. It includes many high-level datatypes similar to those of Perl, including dynamic lists and hash arrays. It aslo supports a couple of functional languages features such as lamdas (a small in-place functions that can be used instead of real ones for the sake of simplicity). Python can use the Tk toolkit to easily build GUI interfaces.

The standard Python distribution includes client libraries for most of the important Internet protocols (SMTP, FTP, POP3, IMAP, HTTP) and generator classes for HTML. This makes Python very suitable for building protocol robots and network administration scripts. It is also suitable for writing dynamic CGI webpages and competes successfully with Perl at the high-complexity end of that application area (Google makes heavy use of Python for example).

Of all the languages in this overview, Python and Java are the only ones well suited for large and complex projects with many cooperating developers. Python is however simpler than Java, and is also friendlier to rapid prototyping. An implementation of Python for the Java Virtual Machine, intended for mixed use of these two languages, is available; it is called Jython.

As any other scripting language, Python is not as fast as C or C++. However it is common to rewrite critical parts of the program in C, and glue them together with Python. Python is not as good as Perl for small projects and scripts heavily dependent on regular expressions. It would also be overkill for tiny projects, to which shell or TCL might be better suited.

Like Perl, Python has a well-established development community and a central Web site carrying a lot of Python tools and extension modules.

Python implementations are available for Microsoft operating systems and for the Macintosh. Cross-platform GUI development is possible with either Tk or other toolkits, wxPython being the most obvious other choice.

4.5 Java

Java is an object-oriented language originally developed at Sun by James Gosling (it was then known by the name "Oak") with the intention of being the successor to C++. After the Internet exploded in 1993-1994, Java was transformed into a bytecode-interpreted language and became the focus of a huge advertising campaign by Sun, which marketed it as the new language of choice for distributed applications.

Java has a stronger and cleaner design than C++. Unlike C++, it does automatic memory management. Like Python and Perl, Java can be integrated with C code. It also has excellent documentation.

However Java also has lots of drawbacks including uneven support on different Web browser platforms, performance problems, and some painful problems with some of the standard toolkits (Abstract Window Toolkit in particular). The language itself is also not without problems, for example class visibility rules are unnecessarily complex. The multiple inheritance problems of C++ are

avoided by introducing the interface facility, which is only slightly less difficult to understand and use.

Java has already been accepted by academic circles, where it has more or less replaced Pascal as the preferred tool for teaching the basics of programming to the next generation of IT specialists. It is also used a lot for 'servlets' that run inside web application servers and for in-house development. However, whether or not Java will be able to replace C++ as a general-purpose programming language remains unclear.

Java implementations are available for all Unixes and for Microsoft operating systems and support cross-platform portability of all pure-Java programs (including GUI capabilities).

4.6 Lisp

Lisp differs from all the languages above in that it is a functional language and not an imperative one. It is based on the ideas of variable-length lists and trees as fundamental data types, and the interpretation of code as data and vice-versa. Lisp was the language where such now commonplace ideas as interpretation, garbage-collection, operator overloading and so on, were introduced.

Lisp, being a functional language, manages memory automatically, and is far more elegant and powerful than most conventional programming languages. By modern standards (heavily influenced by C and C++) it has an odd syntax however with its use of prefix notation and parenthesis.

The main problem with Lisp is that it is in fact not a single language, but a family of languages, and thus the Lisp community is deeply fragmented nowadays. For this reason Lisp was never able to deliver what it promised, and perhaps the only area where it's widely used today is for programming the behavior of the Emacs text editor.

As such, it is very suitable for tasks involving interactively processing a special file format or text database. It is also suitable for building applications that have to be closely integrated with a text editor, or which function primarily as text browsers with some editing capability. User agents for email and USENET news fall in this category, and also certain kinds of database front ends.

5 Why are scripting languages important?

The most dominant application programming languages for almost a decade have been C and C++. However, now they clearly outlived themeselves for almost all tasks but system programs and time-critical kernels of applications. Most other tasks (such as application prototyping and gluing together other applications and tools) are better served by scripting languages.

The central problem of C and C++ is that they require programmers to handle the memory by hand and not let them leave it to the execution environment. It is needed to declare variables, explicitly manage pointer-chained lists and dimension buffers, detect or prevent buffer overruns and to allocate and deallocate dynamic storage.

This is the cause of an enormous amount of complication and error. Buffer overruns are a common cause of crashes and security holes. Some bugs are especially hard-to-find, such as memory leaks.

Scripting languages avoid manual memory management by having a memory manager in their runtime part, typically a program interpreter. Several program can also share one interpreter, reducing the memory usage.

That's one side of the story. The other one is that it makes sense to write today's programs in several languages, selecting the best tool for each subtask. For example, the time-critical parts could be done in C, the data access routines in SQL and the glue that brings it all together, and adds a user interface on top of that could be written in Tcl or Python. Global complexity of the system implemented in this style would be lower than if it had been coded as one big monolith in a general-purpose language.

For example, when building a scientific application, C/C++ programmers can implement efficient numerical algorithms, while scientists on the same project can write scripts that test and use those algorithms. The scientist doesn't have to learn a low-level programming language, and the C/C++ programmer doesn't need to understand the science involved.

Examples of successful use of scripting languages are numerous, but I will mention just two of them: [1] and [2].

6 The future

In the article [3], a trend is presented, which I wholeheartedly agree with. Basically, as the decade progresses, we will see a decline in use of statically-typed languages (the ones that require variables declaration before their usage), such as C++, Java, and Ada and an increasing use of dynamically typed languages, such as Python, Ruby, and Perl. It is these languages that will become mainstream in the coming years. The primary reasons for that is that they have almost zero compile time (unlike C++ where the compilation sometimes has to be done overnight on extremely powerful machines), and the fact that type-safety is no longer needed which greatly simplifies development and testing. So, it seems to be a good idea to keep an eye on languages like Python and Ruby. They are likely to become extremely important.

References

[1] Case Study: Python in a Commercial Environment, by Greg Stein, Microsoft, in Proceedings of the 6th International Python Conference, http://www.python.org/workshops/1997-10/proceedings/

[2] Alice Virtual Reality project at Carnegie Mellon University, http://alice.cs.cmu.edu

[3] Robert Martin. Are scripting languages the wave of the future? http://www.itworld.com/AppDev/1262/itw-0314-rcmappdevint/

[4] Ruslan Bogatyrov. The nature and evolution of scripting languages. http://www.osp.ru/pcworld/2001/11/144.htm

[5] Eric Raymond. To C or not to C. http://tuxedo.org/ esr/writings/taoup/chapter03.html

[6] Prechelt, Lutz; An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program. http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=ira/2000/5