Introduction to Software

- Module Introduction
- How software is built: Coding, scripting, and programming
- Common Scripting Solutions
- Scripting Languages
- Types of Software
- Supplemental Reading for Software Versioning
- Revisiting Abstracting
- Recipe for Computing
- Phelan: Learning IT in the Navy

Module Introduction

Hi. My name is Phelan Vendeville, and I'm a Systems Engineer in the Site Reliability Organization at Google. I'm really excited to be your instructor for the next few lessons. Before we jump in, I'll kick things off by telling you a little bit about myself. My passion for technology began in high school which was located in a geographically isolated part of California. This isolation meant that technology and the Internet played an important role in bringing the outside world to students and connecting them with ideas and opportunities by things like virtual field trips and remote learning. For example, I remember preparing for the SATs through digital classroom sessions, which would have been impossible to attend in person. After high school, I enlisted in the US Navy as an Information Systems Technician responsible for maintaining computer and network systems. I continued to witness the ways technology brings people together, whether that meant coordinating ship movements during training exercises or connecting loved ones on long deployments via video chat. Lots of people use technologies in various ways every day, but relatively few understand how it works. A career in IT can be challenging, as I can attest to personally. I can still remember the horror I felt after blowing up the power supply of a master chief's computer by using the wrong voltage switch. But a career in IT can be incredibly rewarding when you can do things like recover irreplaceable family photos from a failing hard-drive. As an IT support specialist, you'll be in a position to not only know how a given piece of technology functions, but also how to help fix it when it breaks. This means you'll have a direct impact on the flow of information going between people. Which is pretty cool. I'm excited to teach you about the third layer of computer architecture, known as software. Software is how we, as users directly interact with our computer. The operating system that we interact with is just software. The music programs, word processors, and more that we use every day are also software. But what exactly is it? If the hardware is the physical stuff that you can pick up and hold, software is the intangible instructions that tell the hardware what to do. In the next lesson, we're going to deep dive more into what software is, how we install it, and how it works.

How software is built: Coding, scripting, and programming

Video games, music players, and Internet browsers are all different types of software that have completely different functions. Think of the apps on your phone and your laptop. We spent a lot of time interacting with this type of software, but we may not know how it actually works or gets added to our systems. In the last few videos, we learned about networking in the internet. There are tons of applications out there that require the Internet to work. Think about it. Your social media apps, messaging apps, and others run off the Internet. This Internet integration isn't just magically added to your application, it's built-in to require it to function. Before we go too far into the world of software, I want to call out some common terms related to software that you might hear. Coding, scripting, and programming are all terms that might seem a little blurry. They generally refer to the same thing, but they each have small distinctions. Coding is basically translating one language to another. This can be coding from English to Spanish, English to Morse code, or even English to a computer language. When someone builds an application, we refer to it as coding in application. Scripting is coding in a scripting language. We'll talk about scripting languages in a later lesson, but scripts are mainly used to perform a single or limited-range task. There are languages we can use to build these. Programming is coding in a programming language. Programming languages are special languages that software developers use to write instructions for computers to execute. Larger applications like your web browser, text editors, and music players, are all usually written in programming languages. When we use the term software, it generally refers to something that was programmed. We use these terms interchangeably, so don't sweat the details. Now, onwards and upwards. What is software made of and who builds it? It's a great question. Anyone who knows a programming or scripting language can use it to write code. There's a huge demand for this skill set and it's becoming easier for someone to learn to code. If you're going to be working in IT, it's important that you understand how software works and how it gets installed on your systems. You might encounter software errors or just good old-fashioned failures, and you need to understand how to deal with them.

Common Scripting Solutions

In this reading, you will learn about a variety of scripting languages, their uses, and their risks. As an IT Support professional, you may need to automate routine tasks. For example, you might want to automate a backup of company data that runs every night. You might also need to automate high volume tasks, like changing security access settings on thousands of files. Scripting is a common tool used for automation. This tool can help IT Support staff save time and resources in a busy enterprise work environment.

Scripting languages

There are many scripting languages available to use for a variety of tasks in different operating system environments. Most scripts are written in command line environments.

Scripting languages for Windows environments:

- **PowerShell (.ps1)** - Windows PowerShell is among the most common command line scripting tools used in Windows environments. PowerShell is built on the .NET platform and employs many of the same elements that programming languages do. PowerShell scripts are used for building, testing, and deploying solutions, in addition to automating system management.

- **Batch scripts (.bat)** - Batch scripts, also called batch files, have been around since the early days of MS DOS and OS/2. Batch files can execute simple tasks, like calling a set of programs to run when a computer boots up. This type of script could be useful in setting up employees' workspaces when they power on their computers.

- **Visual Basic Script (.vbs)** - Visual Basic Script is an older scripting language. It has reached its end of life for Microsoft support and has been replaced by PowerShell scripts. However, as an IT professional, you may encounter .vbs scripts on some legacy systems.

Scripting languages for Linux and Unix environments:

- **Shell script (.sh)** - Shell scripting languages, like Bash, are used in Unix or Linux environments. The scripts are often used to manipulate files, including changing file security settings, creating, copying, editing, renaming and deleting files. They can also be used to execute programs, print, navigate the operating system, and much more. The scripts run in command-line interpreter (CLI) shells, such as the Bourne shell, Bourne Again SHell (Bash), C shell, and Korn (KSH) shell.

Programming languages that can be used for scripting:

- **JavaScript (.js)** - JavaScript the most used programming language in the world. It is a lightweight language that is used for scripting in web development, mobile and web apps, games, and more. It can also be used to develop software and automate web server functions.

- **Python (.py)** - Python is a user-friendly programming language that can perform advanced tasks and import modules from libraries specially designed for automation scripts.

Scripting uses - finding the right tool for the job

- **Basic automation:** Python is an excellent script for automation. It’s one of the most commonly used, with many available automation libraries.

- **Restarting machines:** Many power users use PowerShell (.ps1) scripts to restart machines (Windows). For Linux machines, they can use .sh (shell) scripts.

- **Mapping network drives:** In the past, mapping network drives was accomplished with .bat or .vbs scripts. However, PowerShell scripts are most commonly used to map drives in Windows environments today. For Linux users, shell scripts can be used for this purpose.

- **Installing applications:** Batch files and shell scripts are often used for automated software installation.

- **Automated Backups:** Windows PowerShell and Linux/Unix shell scripts can automate backups.

- **Gathering of information and data:** Python is a popular choice for gathering data. Python has many available libraries to help with this task.

- **Initiating Updates:** Powershell and shell scripts can be used for initiating updates in Windows and Linux, respectively.

Security risks of using scripts

IT Support professionals need to be very careful when using scripts, especially with prewritten scripts copied or downloaded from the internet. Some of the security risks of using scripts could include:

- **Unintentionally introducing malware:** As an IT Support professional that is new to scripting, you may try to search the internet for assistance in writing scripts. In your search, you might find a script online for a task that you want to automate. It’s tempting to save time and effort by downloading the script and deploying it in your network environment. However, this is dangerous because scripts authored by an unverified source could potentially contain malware. Malicious scripts could have the power to delete files, corrupt data and software, steal confidential information, disable systems, and even bring down an entire network. Malicious scripts can create security weaknesses for the purpose of creating entry points for cybercriminals to penetrate networks. Scripts could also introduce ransomware attacks, which often works by encrypting file systems and then selling the decryption keys for ransom.

- **Inadvertently changing system settings:** Scripts are powerful tools for changing system settings. Using the wrong script can cause the user to inadvertently configure harmful settings. For example, one minor typo in a shell script that sets file permission security in Linux could make confidential files accessible to the world.

- **Browser or system crashes due to mishandling of resources:** Mishandling resources can lead to program crashes in the browser or cause the entire computer to crash. For example, directing too much memory to the browser can overload the computer system.

Key takeaways

A basic knowledge of scripting is an important tool for IT professionals. You may need to improve workflow efficiency by automating basic functions with a scripting language. Some common scripting languages include:

- Windows environments: batch scripts (.bat), Powershell (.ps1), Visual Basic Script (.vbs)

- Linux/Unix environments: shell scripts (.sh)
- Most OS environments: javascript (.js), Python (.py)

Scripts have multiple helpful uses, such as:

- Basic Automation
- Restarting Machines
- Remapping Network Drives
- Installing Applications
- Automating Backups
- Gathering of information/ data
- Initiating Updates

There are risks in using scripts, including:

- Unintentionally introducing malware
- Inadvertently changing system settings
- Browser or system crashes due to mishandling of resources

Resources for more information

For more information about scripting languages, please visit:

- https://library.naruzkurai.com/link/236#bkmrk-page-title

Scripting Languages

Scripting languages allow a coding professional to create scripts that execute tasks. Often, this is a useful method for automating tasks that don’t require human interaction or interpretation so that you can reduce the workload of your staff. If you're interested in a career as a coding professional, it can be beneficial to learn about common and practical scripting languages.

In this article, we discuss what a scripting language is, share the difference between scripting and programming languages, list 14 of the top scripting languages and offer tips for learning how to script.

What is a scripting language?

A scripting language is a coding language that offers a method of creating commands that do not require compilation. Instead, the scripting language runs through an interpreter that translates the script into actions. Scripting languages often facilitate automation within an organization to increase efficiency. Information that anyone enters in a scripting language executes sequentially from the top of the code to the bottom, following any protocols of the scripting language.

Scripting languages can fall into two broad categories:

- **Client-side:** Client-side scripting languages focus on the parts of a website or web application that the end user (or client) interacts with directly.

- **Server-side:** Server-side scripting languages interact directly with and run on the server. These scripts are typically invisible to the end user.

What's the difference between scripting and programming languages?

If your career involves coding, it's important to understand how scripting languages and programming languages differ. Although some use the two terms interchangeably, there are important differences between the two, including:

- **Compiled versus interpreted:** One key difference between scripting and programming languages is whether they're compiled or interpreted. Programming languages are compiled, allowing the machine that's accessing them to translate them directly with no intermediary interpreter.
- **Execution speed:** Programming languages often provide faster execution speeds for task completion than a scripting language. This occurs because of the benefits of executing a compiled program and not relying on an interpreter.
- **Purposes:** The purposes of the two types of languages can overlap, but there are also differences. Generally, professionals use programming languages for a variety of applications, while they use scripting languages mainly for web application development.
- **Simplicity:** Although scripting allows a coder to complete tasks in many situations, professionals often use it for simpler tasks. This can help make it easier to learn than the average programming language.

- **Amount of code:** The simplicity of scripting languages can be beneficial when seeking to create shorter pieces of code. Completed scripts are often shorter than completed programs, allowing a coder to complete them more quickly.
- **Portability:** Because scripting languages rely on an interpreter and do not need to be compiled prior to execution, they are often transferable across multiple operating systems. Programming languages, conversely, often require specific compilations for each operating system.

14 scripting languages (just learn python+c#+java)

If you're interested in a career in coding, learning about these languages may help:

1. JavaScript

JavaScript is one of the most common scripting languages. JavaScript is a high-level, text-based scripting language that can operate on either the client side or the server side. Web developers can use JavaScript, which works with HTML and CSS, to add interactive elements to websites.

2. PHP

PHP is another popular scripting language. PHP stands for Hypertext Preprocessor, which is a general-purpose, open-source scripting language. While PHP has become less prominent in web design as new developers have created new languages, many developers still use it because of its broad application and strong security.

3. Python

Another widely used scripting language is Python, which professionals can use for both scripting and programming. Because of its simple syntax, Python is known for being relatively easy to learn and understand, making it ideal for beginners. Some applications of Python include artificial intelligence, web development, mobile application development and operating systems.

4. Perl

Perl is a general-purpose, back-end scripting language that professionals frequently use to process text files, and they can also use it for web development and database management. Perl shares many structural similarities to programming in C languages. This can make it an excellent option for transitioning to scripting languages.

5. Ruby

Ruby is an object-oriented language known for its simple syntax. Developers can use Ruby to create web applications, and the language is also useful for data analysis and other purposes. It is a popular scripting language in the professional world, making it a valuable option for an aspiring code professional to learn.

6. Bash

Computer programmers can also create scripts with Bash, which is a Unix command language. One of the most common uses of Bash is accessing files and completing tasks through the command line. Because the syntax of Bash is simple and intuitive, this scripting language is fairly easy to learn.

7. R

R is a scripting language that's especially popular in statistics and data analysis. You can also use R to graph data, making this language popular with data scientists, statisticians and other professionals who work with data. The R language works in the R environment as well as in other development environments and platforms.

8. Lua

Lua is an embeddable scripting language. It has a diverse range of programming methods and applications, making it a versatile scripting language to learn. Lua is a popular scripting choice for use in the development of game engines, which allow video game developers to create systems in which they can design their games.

9. Emacs Lisp

Emacs Lisp is a scripting language originally designed for use with eMac computers. Although eMac computers are no longer in production, the language can still be a valuable learning aid because of its relationship with other scripting languages. It is particularly relevant to coders with an interest in working on Unix machines and with the command line.

10. Groovy

Groovy is a scripting language with a syntax similar to Java. Greedy makes use of a dot-separated notation and can carry out complex tasks when needed. Professionals often use Groovy for web development projects, and it can complete a range of tasks within a web design.

11. PowerShell

PowerShell is a command-line scripting language that can work with many platforms. Although you may use it for other tasks, PowerShell primarily serves to automate computer tasks. This frees up time for yourself or others in the organization to work on other projects.

12. VBA

VBA is a domain-specific scripting language and stands for Visual Basic for Applications. Professionals primarily use it as a scripting addition to Microsoft applications. For example, you may create a macro in Excel that completes multiple tasks on a single button press, with VBA providing the scripting to execute the actions.

13. GML

GML is a scripting language for the Game Maker Studios line of development software. Game Maker Studio is an engine for creating computer games that combine both visual layout and scripting. Although developers do not require scripting skills to use Game Maker Studio due to the inclusion of plug-and-play buttons, learning to script in GML significantly expands your options when creating a game in Game Maker Studios.

14. VBScript

VBScript is a scripting language based on the Visual Basic suite of computer programs. VBScript can create code for both online and client-side implementation. Although VBScript has decreased in prominence in recent years, it is a foundational scripting language for some developers.

Tips for learning scripting languages

If you’re interested in learning a scripting language, these tips can help you do so effectively:

- **Think about your goals.** Consider what your intentions are as a coding professional when choosing which scripting languages to learn. Adapting your language studies to match those which align with your objectives can make your study more effective.
- **Use online guides.** When learning to code, online resources and courses can be an excellent place to begin. Often, these courses are available for free and provide a comprehensive understanding of the language and potentially a certification or credentials as well.
- **Consider a formal education.** Although a degree in computer science is not a requirement when seeking to work as a coding professional, it may still be beneficial. Earning a degree may be useful when applying to coding positions, and it allows you to gain a high-quality education from a college or university.
- **Practice your scripting skills.** Practical learning is an effective way to learn a new scripting language for most coding professionals. By writing practice programs, you learn how to apply your scripting knowledge, and the process of troubleshooting can help you further strengthen your understanding of the language and how it works.
- **Search when you are stuck.** If you encounter a problem during your scripting and you do not know the solution, online resources may ?help. Most scripting languages have active communities with a simple search for your problem, providing you with links to multiple articles or forum posts discussing the issue which you may ?apply to fix your script.

Types of Software

When you write content, create a piece of art, or engineer something, your work is protected for your use and distribution. There's usually some other caveats depending on the laws in your country. But in general, copyright is used when creating original work. Software that's written is also protected by copyright. Software developers can choose what they do with their software. For commercial software, it's common to let someone else use their software if they pay for a license. For non-commercial software, a popular option is making it open source. This means that developers will let other developers share, modify, and distribute their software for free. Score, some amazing software efforts have been developed and advanced because of open source. One major example is the Linux kernel, which is used in the Android OS, and in enterprise and personal computers. Hundreds of millions of devices are running Linux at this very second. LibreOffice, GIMP, and Firefox are other examples of open-source software. Open-source projects are usually contributed by developers who work on the project for free in their free time. These massive software development efforts were essentially built by a community of volunteers. How great is that? In an IT environment, you'll have to pay special attention to the types of software you use. Some may require you to pay multiple licenses to use it, others might be free and open source. It's important to check the license agreement of any software before you install it. We've talked about some of the basics of software, but now let's shift to the two types of software you'll encounter categorized by function. Application software is any software created to fulfill a specific need, like a text editor, web browser, or graphic editor. System software is software used to keep our core system running, like operating system tools and utilities. There's also a type of system software that we haven't defined yet called firmware. Firmware is software that's permanently stored on a computer component. Can you think of a firmware that we've talked about already? If you thought of the BIOS, you're right. The BIOS helps start up the hardware on your computer, and also helps load up your operating system, so it's important that it's in a permanent location. I should also call out software versions. These are important because they tell us what features were added to a specific software iteration. You'll encounter lots of software versions while you work with software. Developers might sometimes use a different standard when distinguishing a version, but in general, the majority of versions follow a sequential numbering trend. You might see something like this, 1.2.5 or 1.3.4. Which of these do you think is the newer version? It's 1.3.4 because it's a larger number than 1.2.5. You can read more about software versioning in the supplemental reading. You'll have to work with all kinds of software. Fortunately, it basically all works the same way. Once you learn how one piece of software works, you'll understand how others might function.

Supplemental Reading for Software Versioning

For more information about software versioning, click here. (its wikipedia :D)

Heads up: A big part of being successful in an IT role is the ability to be a self-led learner -- someone who finds key resources and reads up on the latest tech trends and solutions. The supplemental readings we've provided have been designed to show you just some of the support materials available to you online; they're not meant to be considered a comprehensive list. Feel free to add to the conversation by posting other useful resources for learners to this forum thread.

Revisiting Abstracting

Earlier in this course, we talked about how programs are instructions that are given to a CPU. We can send binary code or bits to our CPU, then they'll use an instruction set to run those commands. But these CPUS might be from different manufacturers and may have different instructions. There might even be all kinds of different hardware components like video cards and hard drives that also have their own special interfaces. So how do we write a program that the hardware can understand? Well, one way would be to write a program for each possible combination of CPU and hardware using the native languages and interfaces of these components, but there are potentially millions of possible configurations of hardware. So how do we get anything to work with all this complex and diverse hardware? Well, thanks to the efforts of computer scientists and the principle of abstraction, we can now use programming languages to write instructions that can be run on any hardware.

Recipe for Computing

Remember that in the 1950s, computer scientists used punch cards to store programs. These punch cards represented bits that the CPU would read and then perform a series of instructions based on what the program was. The binary code could have looked like this, and the instructions will be translated to this, grab some input data from this location in memory. Using the input data, do some math, then put some output data into this location in memory. But storing programs on punch cards was a long and tedious task. The programs had to be kept on stacks and stacks of punch cards. Computer scientists needed a better way to send instructions to a machine, but how? Eventually a language was invented called assembly language, that allowed computer scientists to use human readable instructions assembled into code that the machines could understand. Instead of generating binary code, computer scientists could program using machine instructions like this. Take integer from register 1, take integer from register 2, add integer from register 1 and register 2 and output to register 4. This example makes it look like a human could read it, but don't be fooled. Let's take an example of saying something simple like, hello world in assembly language. It looks pretty robotic, don't get me wrong that's still an improvement over its binary code cousin. But assembly language was still a thin veil from machine code. It's still didn't let computer programmers use real human words to build a program, and a program that was written for a specific CPU could only be run on that CPU or family of CPUs. A program was needed that could run on many types of CPUs, enter compiled programming languages. A compiled programming language uses human readable instructions and sends them through a compiler. The compiler takes the human instructions and compiles them into machine instructions. Admiral Grace Hopper invented this to help make programming easier. Compilers are a key component to programming and helped pave the road that led us to today's modern computing. Thanks to compilers, we can now use something like this, and it would be the same thing as this. Computer scientists have developed hundreds of programming languages in the past couple of decades to try and abstract the different CPU instructions into simpler commands. Along the way, another type of language emerged that was interpreted rather than compiled, interpreted languages aren't compiled ahead-of-time. A file that has code written in one of these languages is usually called a script. The script is run by an interpreter which interprets the code into CPU instructions just in time to run them. You'll learn how to write code using a scripting language later in this program as an IT support specialist, scripting can help you by harnessing the power of a computer to perform tasks on your behalf, allowing you to solve a problem once and then move on to the next thing. Programming languages are used to create programs that can be run to perform a task or many tasks.

Phelan: Learning IT in the Navy

In high school, I wasn't really sure what I wanted to do yet, but when I joined the US Navy, one of the options for job was an information systems technician. Information technology and the Navy can be pretty exciting. You get to be very resourceful if you're out on a deployment in the desert. Perhaps, you have to use the tools that you have at hand to get the job done. So I remember being in a server room in a tent with the sand blowing in and we'd occasionally have to take out the servers and then reverse the vacuum and blow the dust and sand out of the server to make sure that they kept working. So obviously I can't go into too much specifics and details but in the Navy, one of my favorite technology moments was when the command came down that we needed to do this thing and I had to write a program to actually do it. And I've never written a program before. And I was like, okay, I mean, I'll try. And so I did the research, I did the learning, I figured it out, I wrote the program it ran, and it did the thing that I wanted it to do. And I was so satisfied. That was an amazing experience. I made this thing from nothing and it actually performed the action that I wanted it to, which was pretty cool.