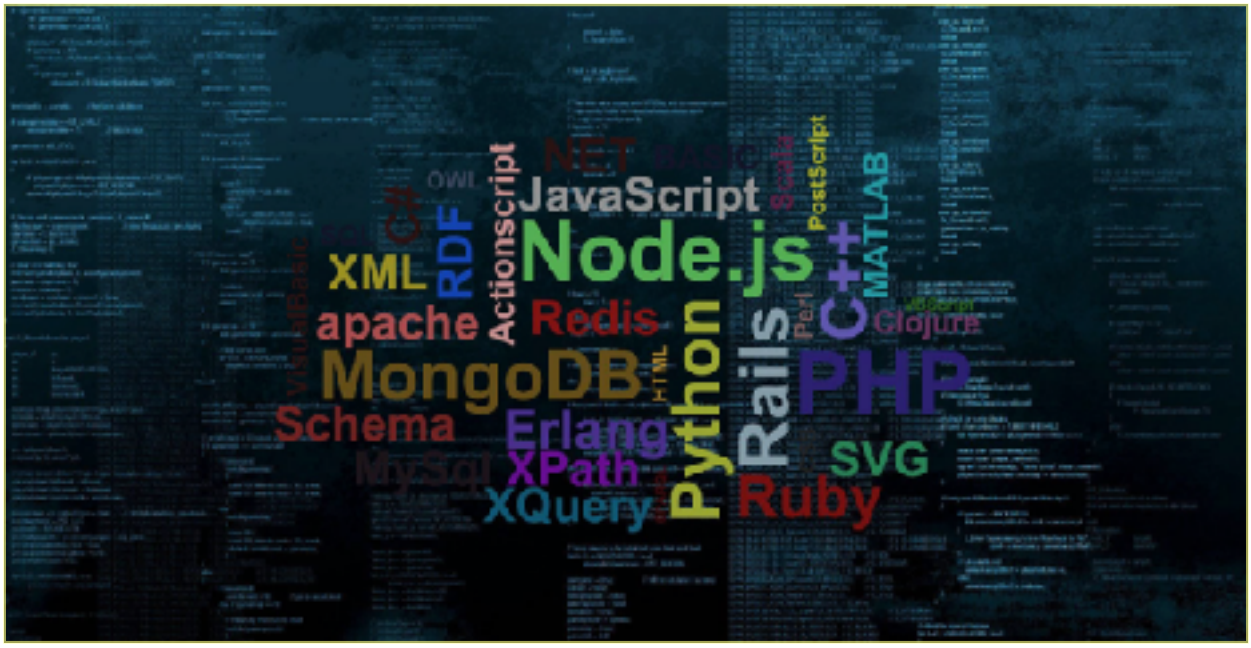
Advanced Learning

Tech Talks

Somewhere between wiring, motherboards, and microcode we start to climb out of the mire and hit a wall. Beyond the wall, we emerge through abstraction into the world of software. Words printed on a screen that are translated into something that seeps down through the wall into the hardware and make dreams come true. It's this gap I want to fill with this tech talk. I would like to make the topic of programming less mystic and more of a solid way to turn a box of switches into useful products and services that we have all come to rely on.



There are as many computer languages as dialects of spoken languages

In earlier talks, we discussed the importance of creating an assembly language that was defined by manufactures of our CPU chips and the like that allow those

above the wall to flesh out features and functions. I move beyond historical beginning and start when Bell Labs came up with the Unix operating system and the C-language. This is really the cornerstone of further software development. Unix laid out directory and file organization to include all the elements of who "root" and "users" were and how to move around with permissions to reach documents and data information. This is general enough that, what we know today as Linux distributions, was made possible. Unix was derived in 1969 and Linux became an open-source derivative in 1991. C morphed into C++ and has marched along as a stable and reliable programming language still in use today.

Much of the functionality of modern programming owes its workability to the shell scripting and command language first used with Unix. Unix was written in the C-language. Libraries and utilities were written that expanded programming capabilities. The use of a complier takes the programmers code along with the operating system functions and links in any required libraries an utilities needed resulting in what's called machine code that can pass though this theoretical wall I was talking about down into the hardware its self and exercise the computer functionality.

The ABC++ of Programming

C++ is really the first high-level, general purpose programming language that had all the features and functionality to work across various platforms, that is, different manufactured computer products. The plus-plus business comes from an operator that increments a variable value by 1 by just adding the ++ to the syntax. So C++ is an increment above C. The most relevant feature augmentation is allowing the programmer to create their own data structures that contains not only the variables needed but the functions used by those variables all in one package call a Class. Classes can be extended and varied which enhances the ability to pass functionality on to other programs without having to rewrite this program piece. The programmer has control over memory allocation and interface functionality with peripherals or internal components like hard drives and monitors. Nowadays, that functionality is so ubiquitous that the languages automatically control memory areas and computer assets within the language definition.

To show that C++ is still a hot ticket, C++23 was released in March of 2023. C++ has always been a language that uses a compiler to generate code that the platform can use. This is a great approach since the compliers can be adjusted for each variation on a platform so that programs will run on many platform types and at top efficiency.

There are many branches of computer languages that have been developed for specific purposes too many to mention here. I want to point out concepts, approaches, and systems that make up our computing environment that are in use today. I'll start with computers talking to computers. That sounds handy.

Client-Server Connectivity

It's great we have developed programs/applications that are designed to work on one computer like word processors, spreadsheets, data bases and print the results out, but as computer usage expanded, it was clear there was a strong desire to use computer resources that were only available at other locations, both across campus or even across nations. Banking for one stands out. Data collections and analysis is another. Just the need to share massive amounts of data that goes way beyond a phone call but is required by on-going process like weather forecasting or national defense needs. Here's the basics:

We need to start a glossary of terms to capture the jargon used within computer technologies. The first might be - what do we call the computer we are working with? It can be described as the host, the client, or maybe the end user. The term host is general and can refer to any computer no matter where it is in a system because the connotation is it hosts applications. A Client machine generally refers to the machine a user is currently running an application on. This can also refer to the program or application its self. If we are talking about a communications network, the client is the machine requesting information and the server is the computer at the other end that's suppling the information. So client-server connectivity infers a networked communication link between a client host and a server host. What's important here is the link and not the name of the computers. Now we have a situation because there are a lot of ways to connect two computers and make them talk. Like any negotiated communication, a set of rules-of-engagement need to be formalized especially considering computers that have to be precisely timed and no loss of bits that would cause disruptions. The term protocol is used to capture the guidelines and strict requirements to define a communications link. A whole bunch of links would constitute a network. There can be local networks that have their own protocols. This arrangement can be classified as an infra-network or a Local Area Network(LAN).

You can already see there are general terms and specific terms for the same thing. We have to stay aware of the focus of the discussion. Many links are described by the technology being used like Ethernet, twisted-pairs, fiber, coax cable, and many other systems of communication that imply there is an application at either end. We have front-end and back-end applications referring

to software used on the client machine to display, for instance, a web page or a back-end process that query a database. Programmers are classified as to a front-end or back-end developers or if do both, a full-stack developer. What's a stack?

For an application to communicate with another application running at a distance, several steps must be attached to the request that accommodate security, transmission media type, transferring the request from one address on the network to the other address on the network, considerations for dropped data, etc. Each of these steps are a handoff of the application's request to another process on the stack until the message is appropriately packaged for delivery. There are delivery boundaries that have their own protocols like going from a local area network to a wide area network(WAN). These boundaries are controlled by network switches and routers both that can contain their own processing especially considering routers. Routers on a WAN are the backbone (no pun intended) of the system. The Internet with a capital "I" is a network of networks all interconnected via routers.

I've gone through this rain-dance to show that most any programming effort that produces a useful product will involve a communications network external to the host computer. The software developed has to be capable of including these features. This means the programmer has to be savvy to how it all works. This sounds like a lot of abstraction coming towards the would-be programmer and that's for sure. Let's focus on some important particulars and not get bogged down in the details but get a general sense of what's needed to understand the details.

Data Dumps

What are the languages a programmer would likely use as a team member or indeed a full-stack type? We mentioned C++ is still a contender. Python is a leading language, especially because of its association with Artificial Intelligence(AI). So much of our presentation of information involves Browser apps we need to realize that inside this app is the JavaScript language interpreter. What's an interpreter? That is a complier for your code but on a run-time basis. This means you can write your code and immediately run it. As it runs, the interpreter converts the code into something it can handoff to the OS and accomplish the task which is mainly going out on a link and getting information from a server or local storage and displaying it on a screen. JavaScript is the mover and shaker for HTML/CSS layouts that display information on screens. In fact, the "screen" is called the viewport and is an area of pixels on the monitor

that's under the control of the Browser. It's all an inside job. HyperText Markup Language(HTML) is a text-based coding the Browser understands to layout what's seen on the viewport. The term hypertext refers to text that has the ability to send a request for data via the Internet and that's done using HTTP or hypertext transfer protocol procedures. The Cascading Style Sheets is a peculiar term that refers to the styling/formatting that the programmer can apply to the structured layout that the HTML provides. The cascading idea is priority styling applied at specific locations in the code. CSS can be applied in a separate file, or the head area of the HTML page, or right at an element. There is a priority assigned that gives the code listed closest to the where it is to be applied the highest priority.

All of this obfuscates the fundamental concept here, so in layman's terms, Browsers have the ability to make network requests using HTTP to go onto the Internet and reach a server by using the address of that server which is linked by using a hypertext call. This request is packaged into data packets that contains the IP address of both sender and destination and is in a format the server can understand. The reply from the server is a collection of files that are essentially HTML CSS and JavaScript files that also include images or other supporting files that allows the Browser to construct a hierarchal layout, styled by the CSS rules, and interactively controlled by code in a JS file. Sounds complex. It actually is rather easy to learn and do. One thing is missing.

Media Types

This is not trolls on social media but understanding that the format used to send and receive data has to be decided. For a long time XML was used - yuck. This kind-da gummed up the works. XML is a bastardize version of HTML that is best forgotten about. JavaScript is used inside the Browser to get everything in and onto the screen. Why not use that? Well the Google engineers separated out a version of JS that could run outside the browser. It was picked up by independent developers into what is now called Node.js. This code can be run as a stand-along app or at the server (backend process) to facilitate communication because JS uses something called JSON or JavaScript Object Notation to create objects within the language that is a way to hold together a bunch of related data and move it around. Nowadays, JSON is the preferred data format used on the Internet for client-server communications.

One of the things to realize here is a Browser screen is a great way to display anything and everything. Even if your jam is applications that do physics

problems. You still need a way to show the results to people even at a distance. So to learn how to build web pages implies learning JavaScript.

What Do I Learn

Here's the best-in-show for 2024:

1. Javascript
2. Python
3. Go
4. Java
5. Kotlin
6. PHP
7. C#
8. Swift
9. R
10. Ruby
11. C and C++
12. Matlab
13. TypeScript
14. Scala
15. SQL
16. HTML
17. CSS
18. NoSQL
19. Rust
20. Perl

That's it. Just get a grip on all the above and your set! Clearly, this implies that specialists in various areas are the norm. Can you learn all the languages in this list. Maybe, but several are highly integrated like JS implies HTML/CSS. Some are undercarriages of others like Python is written in C++ as well as many other "higher level" languages. If you are a scientist, Matlab, Python, and a database would be a must. If you are a web developer, HTML/CSS/JS is clear but also a backend code implies Node.js and that's not even on the list! Swift and Android are specific for mobile apps. I would say the tools or editor programs are just as important. The leader there is VS Code. Any programmer worth their salt has to know how to use GIT and the Github website. Databases are another whole area. One has to know at least one to get much done. MySQL and Postgres are classics but MongoDB makes a lot of sense. And what about whole self-contained systems like WordPress or Django? They count as well. There is also an important interface for the server with all your precious webpages and that's the proxy server and firewall to keep the bad guys from your hard work. Apache/PHP was the workhorse. Now it NGINX and Node.js.

Summary

It's a far cry from the lowly shift register to ChatGPT, but you have to start somewhere. Learning to build a webpage is fun and useful. Learning how Internet security is set up is a whole other discipline. IT work or data transmission are worlds of their own as well. For the hobbyist, microcontrollers are a great way to start. You can learn all the basics and create fun projects. Maker groups are wonderful.

If you can get along without knowing something about computers and their various apps, let me know. I don't thinks it's possible. This implies, even if you don't plan to go into computers as a vocation, you should take the time to learn how to become a wizard with the one and zero.