Your Instructor

Ted Wetherbee

Office:	W217
Address:	FDLTCC
	2101 14 Street
	Cloquet, Minnesota 55720
Office Phone:	218-879-0840
Email:	ted@fdltcc.edu
Website:	cs.fdltcc.edu/wetherbee

Introduction to me, and programming notes

We will not necessarily meet in person during an online course, so Ill introduce myself a bit surrounding our area of focus which is programming computers in Python, plus a few gratuous notes and tips. If you are on campus, stop by to say hello sometime. I'm usually on campus in the mornings between 8-9am calc 1 and noon-1pm algebra & trig.

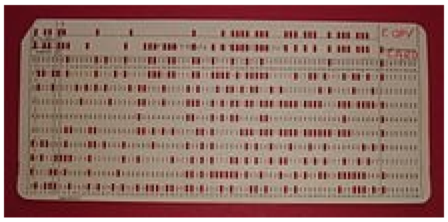
I've been teaching math and computer science courses at FDLTCC since 1990, and I've been using computers since summer 1972. To share where I'm coming from, I'll briefly describe some machines I've used along with operating systems and programming languages.

First Computer: IBM 1620

My first computer was an IBM 1620, an early transitorized mainframe from the early 1960s which my high school leased for math classes (there were no "Computer Science" courses). This machine had a room to itself along with two card-punching machines. Students used a card-punch machine to "write" programs line-by-line on 80 column cards, then the cards were ordered to form a source code deck.

The textbook was aptly titled, "Programming the IBM 1620". By any modern standard, this was a tough textbook using a "machine first" approach.

First, the machine design was outlined, and the first assignments were to write machine language programs which amounted to entering numbers on at most a few punched cards.



The next topic was using Symbolic Programming System (SPS), an assembly language. This was somewhat easier than using machine language, but still difficult.

The final topic was using FORTRAN IVd. FORTRAN used English-like statements such as

```
a = 1.3 + b
```

Machine language, SPS, or FORTRAN, programs were hand-coded on paper first, then punched into cards line-by-line. To run a FORTRAN program (the easiest), there was a specific sequence of steps required in order to prepare the machine, compile the program, then run the program.

> 0) Clear the machine by entering "490000000000" on the console typewriter. This clears all the iron-core memory and puts it in a state able to load a new program..
>
> 2) Load the FORTRAN pre-compiler deck, then load the program deck. The pre-compiler checked the program to make sure that there were no syntax errors.
>
> 3) Clear the machine. Load the compiler deck, then load the program deck. If all goes well, the compiled program will be created as a new deck of punched cards.
>
> 4) Clear the machine. Load the compiled program deck, then load the subroutine deck.
>
> 5) Interact with the program using the console--if the program is interactive.

This process could be completed in about half an hour if every step went well. More typically, a few days were needed to fix punchcard errors and then logic errors in the program. Further, the machine could only process one program at a time, so time was spent waiting for a turn on the machine.

Personal Computers

Before 1975, computers were expensive and rarely seen outside big businesses, larger goverment offices, and research laboratories. "Personal computers" (PCs) were typically defined as computers which were small enough to be picked up and cost less than a new car. PCs of all kinds (desktop, laptop, pad, ...) are cheaper now than they were in 1975, but thousands, millions, and billions of times more powerful in various catagories of performance.

> Altair, 1975. This is a build-your-own machine with 4k of memory. I have a floppy disk drive and an Altair disk operating system in my office for a later model.
>
> Tandy Color Computer (CoCo), 1979. This was a 32K memory model which included a keyboard, a cassette tape drive, some "software" in the form of plugin units, and a cable which hooked into a TV for a display. One could program one of these using BASIC which was built into the machine.
>
> Apple IIe, 1982. This was a wildly popular machine and comon for many years in K-12 schools, 64K of memory, a decent green-on-black display, and a good 5.25-inch floppy disk drive. It had built in BASIC (which almost every PC did in the 1970s-1990s), and it also could run a Pascal compiler. Pascal was the popular programming language in the 1980s for computer science courses.
>
> Apple Macintish: MacApple Mac machines were different and apart from the IBM PC-clone/Windows-DOS family. The Mac used Mac operating system and parts specific for the Mac. Some years ago they switched to Intel/AMD processors. Notably, Apple also started using OS-X which is essentially berkeley UNIX (BSD-UNIX). Thus, software written for UNIX and Linux machines could more easily be used on a Mac.
>
> "IBM PC Clone": 1985-current (Windows and Linux on Intel/AMD machines). IBM came out

with a PC in the early 1980s. An IBM or Macintosh was too expensive for most students at $4000-8000, but there were clones, machines which ran just about the same software and used the same operating system (Microsoft DOS), but could be built using $500-2000 of parts. These machines came with built-in BASIC, but there were good compilers available in C and Pascal, and these machines quickly made their way into offices everywhere. In particular, Borland came out with good C and Pascal compilers for around $60. These workstation/desktop full-size machines had somewhat interchangeable parts across many brands; one could buy parts and incrementally upgrade components and even motherboards (with the CPU) at a modest cost.

Mainframes and Supercomputers

Mainframes are less common these days because PCs have become powerful, and modern "supercomputers" are really just clusters of special PCs.

Control Data Corporation Cyber: 1983-5. The University of Minnesota-Duluth had a couple Cyber's for computer science classes. One could use "DEC-writers", an interactive typewriter, and editing was done one line at a time. There were a few VI display terminals where programs could be edited a page at a time. To print out a program, one sent a print job to the queue, and eventually the printout could be picked up at a window of a room where a line printer did nothing but print student jobs for the campus. Most programming classes used Pascal. Students had to wait for a 1-hour slot in order to use a terminal, and this could become long and difficult when assignments were close to being due.

DEC Vax: 1984-5. This was a mini-computer, larger than a PC, but smaller than a mainframe. There was a small set of VI terminals and a local printer for this one machine. It could be difficult to find a time slot to use this machine when assignments were close to being due.

Cray T3-D, SGI Origin: Minnesota Supercomputing Institute 1999-2002. These machines had custom processors, but their common operating system was a variation of UNIX.

Teragrid, XSEDE, and Blue waters: NSF-supported Supercomputers 2007-current. Most Teragrid and current XSEDE "supercomputers" are actually PCs (nodes) networked together so that a job can run on many nodes at the same time. Software development for these machines is not much different than what one can do on a $300 laptop or desktop, except when it comes to running in parallel on many nodes. The Blue Waters machine is a Cray with 22,640 Cray XE6 nodes which have 362,240 "bulldozer" CPU cores, but each node is an ordinary Linux machine similar to what I have on my office desk.

Operating Systems (OS)

The operating system is probably more descriptive these days than the manufacturer or CPU of a computer. There are three major operating systems used on computers. Python runs well on all of them. If you use the IDLE editor and Tkinter (as we will do), most programs should run identically even if there may be slight differences in appearance.

1. UNIX (Linux). UNIX was developed in the 1970s. Most modern operating systems were modeled after UNIX or derived from it. Linux was created in the early 1990s as an open source operating system, and it is by far the most popular OS for supercomputers as well as for scientific work. Python is often automatically installed by Linux distributions (Ubuntu, Fedora, Centos,...), but it is easy to install otherwise online using a simple command.

2. Windows by Microsoft. Microsoft created DOS for the IBM PC, but retained ownership of DOS for PCs. After various attempts to create a graphical user interface (Windows 1 and 2),

Microsoft came out with Windows 3.1, NT, and 95 versions which gained huge popularity. Current Windows 11 is an evolutionary refinement. There are various versions of Python to install on Windows. FDLTCC machines in room 208 and the student open lab have Python 2.7 installed under ESRI GIS software as required.

3. Macitosh OS X. Apple used to have their own operating system, MAC OS, but they moved to OS X which is really BSD UNIX. Python is required to be installed on a Mac with OS X, so you will always have it there, and it is currently version 2.7 on FDLTCC Macs.

My current favorite OS is Linux, and my favorite linux is currently Centos. Linux is free, an "open source project" which means that the source code is publicly available for anyone.

Programming Languages

There are two main categories: compiled and interpreted.

Compiled: Fortran, C, C++, Ada

Compiled languaged use a compiler which converts source code (English-like text) into binary code which runs directly on a machine. The compiled binary only runs on the specific machine it is compiled for, generally not portable.

Interpreted: Python, Perl, Javascript, BASIC, Postscript

An interpreter scans the source code (English-like text) and executes the program without producing a separate binary file. Interpreted programs run slower than compiled binaries, but they are highly portable and usually fast enough. Interpreted languages are easier to use.

Some are unique. Javascript is the practical choice for programming within a web page. Postscript is used by printers.

I've used these for significant programming projects at some time: FORTRAN IVd, Fortran, C, C++, Java, BASIC, Python, perl, bash, Postscript, Pascal, Scheme. I've recently use Fortran, Python, bash, Perl, and Javascript for projects. For an important note: Real programmers use multiple languages even if they might prefer one. For a heartening note: Learning one language like Python makes learning a different one much easier; so many features have to be the same even if the syntax (names and punctuation) are different; the logic and art of programming is something that stays the same.

Current Programming Projects

I'm member of the team working on "3-D SIMULATIONS OF I-PROCESS NUCLEOSYNTHESIS IN THE EARLY UNIVERSE" led by Paul Woodward.

https://bluewaters.ncsa.illinois.edu/documents/10157/162541/bwar16_woodward.pdf

My role on this team is writing visualization software which delivers imagery during running simulations. The current version of this software (Srend) is available online:

http://srend.nongnu.org/

This particular software arose from programming projects at FDLTCC to serve educational needs and within courses like intro. to programming Csci 1020.

Programming and the Future

Programming computers is one of the most useful and employable skills one can have in 2017, and this will likely become more so in the future. Machines have changed greatly in 45 years, yet core ideas needed for programing computers have not. My current programming on leading edge machines is not very much different from the exercises done 45 years ago, and the language is the same (Fortran) even if the code looks somewhat different. You can expect to learn Python and reuse these skills for many years. There are not many careers left in which computer programming is not a seriously valuable asset!