**CMSC330 Introductory overview (Day 1)**

**Fall 2019**

After this unit you should be able to:

I. List and explain at least four reasons for studying programming languages

A. Become a better programmer w/ wider set of tools, better understanding of them

B. Helps you learn new ones more quickly

C. Helps you choose the right language for an application

D. Supports you in designing a language, general purpose or specialized, when needed

E. Fascinating topic – fundamental to computing, rich and interesting

II. Explain why we consider all general purpose programming languages equivalent

A. Define what it means to be a "general purpose programming language", usually meant as

***Turing Complete***: can compute any function computable by Turing Machine

B. Explain the ***Church-Turing thesis***: If a fcn can be computed, it can be computed by a TM

C. Distinguish Turing Complete languages from non-general purpose (small) by the rough

rule that Turing Complete languages must have variables, loops/recursion, selection, integers

III. Understand that programming languages are artifacts of human design

A. Languages are created by individuals or teams to achieve particular goals

B. Explain why knowing design goals of a language can help understand it (and why so many!)

C. Changes in programming goals over time (1960s-now) means languages have evolved

i. 1950-60s – programmers cheap, machine expensive => efficiency key

ii. Now- programmers expensive, machines cheap => more emphasis on design

iii. Evolving and individual project goals: design quality, security, power efficiency, etc

D. Explain why knowing language goals can assist in choosing the right one for an application

IV. Define and explain selected primary attributes of programming languages

A. ***Paradigm***. A categorization of PLs on fundamental attributes such as data representation, available control structures, execution model, code grouping.

i. Explain and contrast the characteristics of *imperative* and *declarative* languages

ii. Explain the characteristics of *functional*, or *applicative*, languages

ii. Explain and apply the attributes *object oriented*, *dynamic*, *scripting*

iv. Explain that paradigms are not rigid as languages overlap in features, ideas

B. ***Syntax***. The keywords, formatting and grammar of the language – how a program looks

C. ***Semantics***. The meaning of a language – what a program computes

D. ***Pragmatics***. The conventions and good habits of a PL – how best to use it

V. List and define key terms in language implementation

A. ***Compilation***. Program translated to target code before run time(eg, machine or bytecode)

B. ***Interpretation***. Program translated on the fly, line by line, during execution

C. Diagram and explain the front end, back end architecture of compilers/interpreters

D. Compare and contrast the advantages and disadvantages of compilation vs. interpretation

E. Understand that in practice, implementations can be mix of compilation and interpretation

What you don't have to know after this unit:

All the specific languages mentioned – if you are responsible, the language will reappear

Paradigm terms not listed above – again, if important it will reappear

Details of Ruby, Ocaml, Rust and other languages – we'll do details later

The concepts of formal syntax and semantics – we'll have units on these

**Suggested readings:**

These are for motivation and background so are not required, but are suggested.

From our own UMD programming language expert, an advanced intro to PL research that you may not fully understand now, but still fun:

<https://www.cs.umd.edu/~mwh/talks/WhatisPL-PLMW19.pdf>

Other instructors on why and how to study PL:

<http://web.cse.ohio-state.edu/~rountev.1/6341/pdf/why.pdf>

<http://soft.vub.ac.be/~tvcutsem/whypls.html>

<https://cs.lmu.edu/~ray/notes/howtostudyprogramminglanguages/> A bit wild but a good list of PLs

And it can be interesting to read what you can find on the history of programming languages, from invention in the 1950s, standardization and formalization in the 1960s, and innovations ever since.