Modern Programming Language Paradigms

- **Procedural Languages**. Based on grouping of statements into functions/procedures. Fast execution speed. Example languages: Fortran, C, Modula, Ada.
- **Object-Oriented Languages**. Based on the concepts of class and inheritance. Good for large-scale software development due to extensibility of code resulting from inheritance hierarchy and inheritance polymorphism (redefinition of function body code and its dynamic binding). Execution speed somewhat slower than procedural languages due to heavy use of reference pointers, dynamic binding of function code, and frequent dynamic allocation/deallocation of class objects. Example languages: Smalltalk, Eiffel, Java, C#
- **Functional Languages**. Programs are sets of pure function definitions. Program execution consists of evaluation of functional expressions. No assignment statements to update values of variables. No loop constructs. Iteration is done by recursive functions. Programming is easier and faster due to the absence of side effects by assignment statements and mathematically clear structures of programs. Execution speed fundamentally slower than procedural and object-oriented languages*. Example languages: ML, Miranda, Haskell.
- **Logic Languages**. Programs are sets of logical statements. Program execution consists of logical deduction process. No assignment statements to update values of variables. No loop constructs. Iteration is done by recursively defined logical statements. Programming is easier and faster due to the absence of side effects by assignment statements and mathematically clear structures of programs. Execution speed fundamentally slower than procedural and object-oriented languages*. Example language: Prolog.
- **Hybrid Languages**. Combine two or more of the above paradigms. Example languages: C++ = C (procedural) + object-oriented features. Lisp = functional + procedural. Programmers have options of staying with one paradigm or using mixed paradigms.
- **Script Languages**. Used for specialized purposes where execution speed is of little or no concern. Interpreter execution, no compilation. Suitable for Web applications, GUI/interactive applications, small-to-medium office business applications. Example languages: JavaScript, Python, HTML, PHP.

* The comparative inefficiency of functional/logic languages is with respect to execution on the conventional, sequential, uni-processor computer architecture. Possibilities of executing functional/logic languages on non-conventional computer architectures, such as massively parallel or data-flow architectures, will be touched on in the last part of the course.