**Reasons for Studying Concepts of Programming Languages**

* Increased ability to express ideas
* Improved background for choosing appropriate languages
* Fortran – Created for use in scientific research
* Increased ability to learn new languages
* Better understanding of significance of implication
* Better use of languages that are already known

**Programming Domains**

* Scientific applications
* Large numbers of floating point computations; use of arrays
* Fortran
* Business applications
* Produce reports, use decimal numbers and characters
* COBOL
* Artificial Intelligence
* Symbols rather than numbers manipulated; use of linked lists
* LISP
* Systems Programming
* Need efficiency because of continuous use
* C
* Web Software
* Eclectic collection of languages: markup (HTML), scripting (PHP), general purpose (Java)

**Language Evaluation Criteria**

* Readability: The ease with which programs can be read and understood
* Overall Simplicity
  + A manageable set of features and constructs
  + Minimal feature multiplicity
  + Minimal operator overloading
* Orthogonality
  + A relatively small set of primitive constructs can be combined in a relatively small number of ways
  + Every possibly combination is legal
* Data Types
  + Adequate predefined data types
* Syntax considerations
  + Identifier forms: flexible composition
  + Special words and methods of forming compound statements
* Writability: The ease with which a language can be used to create programs
* Simplicity and orthogonality
  + Few constructs, a small number of primitives, a small set of rules for combining them
* Support for abstraction
  + The ability to define and use complex structures or operations in ways that allow details to be ignored
* Expressivity
  + A set of relatively convenient ways of specifying operations
  + Strength and number of operators and predefined functions
* Reliability: Conformance to specifications (i.e. performs to its specifications)
* Type checking
  + Testing for type errors
* Exception handling
  + Intercept run-time errors and take corrective measures
* Aliasing
  + Presence of two or more distinct referencing methods for the same memory location
* Readability and writability
  + A language should support comfortable ways of expressing functions and actions
* Cost: The ultimate total cost (environment, employee skill, compiler/interpreter)
* Training programmers to use the language
* Writing programs (closeness to particular applications)
* Compiling programs
* Executing programs
* Language implication system: availability of free compilers
* Reliability: poor reliability leads to high costs
* Maintaining programs
* Others
* Portability
* Generality
* Well-definedness

**Influences on language design**

* Computer Architecture
* Languages are developed around the prevalent computer architecture, known as the *Von Neumann* architecture
* It is the many layers between the application, and it being understood and run by the physics of the computer
* Program Design Methodologies
* New software development methodologies (ex. object oriented software development) led to new programming paradigms and by extension, new programming languages

**Computer Architecture Influence**

* Well-known computer architecture: Von Neumann
* Imperative languages, most dominant, because of Von Neumann computers
* Data and programs stored in memory
* Memory is separate from CPU
* Instructions and data are piped from memory to CPU
* Basis for imperative languages
  + Variables model memory cells
  + Assignment statements model piping
  + Iteration is efficient

**The Von Neumann Architecture**

* Fetch-execute cycle (on a von neumann architecture computer)

Initialize the program counter

**Repeat** forever

Fetch the instruction pointed by the counter

Increment the counter

Decode the instruction

Execute the instruction

**End repeat**

**Von Neumann Bottleneck**

* Connection speed between a computer’s memory and the processor determines the speed of a computer (memory speed)
* Program instructions often can be executed much faster than the speed of the connection; thus, causing a *bottleneck* in the computer’s speed
* Memory speed is usually the primary limiting factor in the speed of computers

**Programming Methodologies Influences**

* 1950s and early 1960s: Simple applications; worry about machine efficiency
* Late 1960s: People efficiency became important; readability, better control structures
* Structured programming
* Top-down design and step-wise refinement
* Late 1970s: Procedure-oriented to data-oriented
* Data abstraction
* Middle 1980s: Object-oriented programming
* Data abstraction + inheritance + polymorphism
  + Polymorphism – A feature of a programming language that allows programs to use variables of different types at different times

**Language Categories**

* Imperative
* Central features are variables, assignment statements, and iteration
* Include languages that support object oriented programming, scripting language and visual languages
* Ex. C, Java, Perl, JavaScript
* Functional
* Main means of making computations is by applying functions to given parameters
* Ex. LISP, Scheme, ML, F#
* Logic
* Rule-based
* Ex .Prolog
* Markup/programming hybrid
* Markup languages extended to support some programming
* Ex. JSTL, XSLT

**Language Design Trade-Offs**

* Reliability vs cost of execution
* Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs, while C does not.
* Readability vs writability
* Example: APL provides many powerful operators (and many new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability
* Writability (flexibility) vs reliability
* Example: C++ pointers are powerful and very flexible but are unreliable

**Compilation**

* Translate high-level program (source language) into machine language
* Slow translation, fast execution
* Complication process has several phases:
* Lexical analysis
* Syntax analysis
* Semantics analysis
* Code generation

**Pure interpretation**

* No translation
* Easier implementation of programs (run-time errors can easily and immediately be displayed)
* Slower execution (10 to 100 times slower than compiled programs)

**Hybrid implementation systems**

* A compromise between compilers and pure interpreters
* A high-level language program is translated to an intermediate language that allows easy interpretation
* Faster than pure interpretation