**Chapter 1**

**Reasons to learn concept of programming languages.**

Increased ability to express ideas.

* You can’t express what you don’t describe.
* A language limits the structures you can use.
* You can implement structures even if they are not supported if you know your way around the language.

Improved background for choosing appropriate languages.

Increased ability to learn new programming languages.

* General language knowledge is helpful.

Better understanding of significance of implementation.

* Why languages work the way they do.
* Understanding low-level bugs.
* Understandi8nbg efficiency trade-offs.

Better use of languages that are already known.

* Programmers often don’t use all the features of a language.
* Knowing about the features lets you exploit them.

Overall advancement of computing.

* Language evolution is complex.
* Better understanding of the underlying issues of a language.

**Language Evaluation Criteria**

Readability: The ease at which programmers can read a program and understand what it does.

Writability: The ease at which a programmer can write a program.

Reliability: Whether a program runs under all edge case condition.

Cost: The cost of using the language.

**Readability**

* Overall simplicity
  + A manageable set of features and constructs.
  + Minimal feature multiplicity.
  + Minimal operator overloading.
  + Too much simplicity can be bad for readability.
* Orthogonality
  + Components of the language are independent of each other so changes in one component don’t effect another.
  + A relatively small set of primitive constructs can be combined in a relatively small number of ways
  + AND every possible combination is legal
  + Orthogonal language feature is context independent
  + Lack of orthogonality = exceptions
  + Too much orthogonality can be bad for readability (ALGOL 68)
* Data types
  + Data types must either be defined or able to be simulated.
* Syntax
  + Lack of naming restrictions improves readability

**Writability**

* Simplicity
  + Few constructs, small number of primitive operations.
* Abstraction
  + Allow subclasses and methods to handle data so that the details of how the data is handles can be ignored, only that the data coming back is acceptable.

**Reliability**

* Type checking
* Exception handling
* Aliasing
* Readability and Writability

**Computer Architecture**

* Languages are developed based on the current computer architecture, known as the von Neuman architecture.

**Programming Methodologies**

* New software development methodologies, OOP led to new programming paradigms and languages.

**Timeline**

* 1950 and 1960s
  + Simple applications, not much memory and therefor programs were focused around efficiency.
* Late 1960s
  + Top down design became more popular.
* Late 1970s
  + Data abstraction was starting to get implemented.
* 1980s
  + OOP was starting to become popular.

**Language categories**

Imperative

* Languages that support OOP
* Scripting languages
* Visual languages
* C, java, Perl, js, vb, .NET, c++

Functional

* Applying functions to parameters
* LISP, Scheme, ML, F#

Logic

* Rule-based
* Prolog

Markup/Programming hybrid

* Markup languages extended to support some programming
* Examples: JSTL, XSLT