**Qualities of Programs**

1. **Reliability (Works Properly)**
   1. Type Checking - Detect Type Errors (Either runtime or compile time)
   2. Exception Handling - Detect Errors at Runtime (Have means of processing errors)
   3. Aliasing - Allow for providing multiple names for the same physical object
   4. Readability + Writability - Provide natural way to implement an algorithm
2. **Cost (Total time/money/effort)**
   1. Training - Programmer Training
   2. Writing Programs - We want them to be fast and easy to write
   3. Runtime - Execution Speed
   4. Implementation - The compiler is correct.
   5. Reliability - Works as Intended.
   6. Maintainability - Easy to change programs after release
3. **Other** 
   1. Portability - Ease of moving programs to other systems.
   2. Generality - Wide ride of application
   3. Well-Definedness - Language specifications are precise
4. **Readability (How easy is it to read the program)**
   1. Simplicity - Small set of features/constructs
   2. Orthogonality - Constructs that are combined in arbitrary ways
   3. Control Structures - Provide wide range of control structures
   4. Data Types - Reasonable set of data types
   5. Syntax - Meaningful Keywords (Apply in certain contexts) + Reserved Words
5. **Writability (How easy to write)**
   1. Simplicity + Orthogonality - Prevents us from having to build every little thing
   2. Support for Abstraction - Allow details to be ignored
   3. Expressivity - Convenient set of constructs

**Important Dates for Language**

Foltran I - 1957

Algol - 1958

Lisp - 1959

COBOL- 1959

C - 1972

Scheme - Mid 1970s

Ada - 1975

C++ - 1980

Java - Early 1990s

Python - Early 1990s

C#- 2000

Fortran 15 - 2015

**Chapter 1**

1950s - Simple Applications

1960s - Readability and Better Control Structures dominate

1970s - Data Abstraction comes alive

1980s - Dawn of Object Oriented Programming

**Language Categories**

Imperative - Central features are variables, assignment statements, iteration

Includes scripting, visual languages. Anything that possesses OOP. **ex: C++**

Functional - Computations are done by applying functions to parameters **ex: Scheme**

Logic - Rule based **ex: Pro-log**

Markup/Programming hybrid - Support some programming **ex: JSTL**

**Zuse’s Plankulkülk**

A very high level programming language written about in a thesis dissertation by Zuse that was the godfather to array storage. It used a large memory table to present simple array addition.\

**Computer Architecture:**

Von Neumann architecture is what all **Imperative** languages are based after. They separate components into manageable chunks and prescribe specific areas to specific memory locations.

New **software development methodologies** lead to new programming languages.

**Von Neumann Bottleneck** - Programs can be run faster than the speed of the connection between CPU and memory which creates a bottleneck.

**Programming Domains**

1. **Scientific Application** 
   1. Large Number of floating point computations; use of arrays
   2. Fortran
2. **Business Application** 
   1. Produce reports with decimal numbers and characters
   2. COBOL
3. **Artificial Intelligence**
   1. Symbols rather than numbers manipulated; use linked list
   2. LISP
4. **Systems programming** 
   1. Need efficiency because of continuous use
   2. C
5. **Web Software**
6. Eclectic collection of languages, markup, scripting, general-purpose

**Implementation Methods**

1. **Compilation** 
   1. Programs are translated into machine language; including JIT systems.
   2. Large commercial applications
2. **Pure Interpretation**
   1. Programs are interpreted by another program known as an interpreter
   2. Small programs or when efficiency is not an issue
3. **Hybrid Implementation**
   1. A compromise between compilers and pure interpreters
   2. Small and medium systems when efficiency is not the first concern

**Just-in-Time Implementation Systems**

1. Initially translate programs to an intermediate language
2. Compile intermediate language into machine code when they are called.

**Lambda Expression**

Stand in for a general expression

ƛ(x) x \* x \* x

**Scheme**

**IMPORTANT: THE “ ` “ PREVENTS EXECUTION OF A STATEMENT**

**Car** - Gets the first element of a list

**Cdr** - Returns a set of everything except the first element

**Cdar -** Compound expressions have a C and an R at the start/end. Followed by **‘d’ for Cdr** and **‘a’ for car**

These statements are processed **right to left**

**Cons -** Adds an atom/list to the front of an existing list.

**Append -** Combines two lists into one

**List -** Makes a list of the following atoms/lists

**Define -** Asserts definition of a function

**NULL? -** Determines if the item is null or not

**LIST? -** Determines if the item is a list or not