### Language Evaluation Criteria

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Readability</th>
<th>Writability</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Orthogonality</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Data Types</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Syntax Design</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Support for Abstraction</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Expressivity</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Type Checking</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Exception Handling</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Restricted Accessing</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

The fourth primary criterion is 'cost', which is not included in the table because it is only slightly related to the other criteria and the characteristics that influence them.

- **Readability**: Ease with which programs can be read and understood.
  - Simplicity: Language should be as small as possible and redundant features are avoided.
  - Orthogonality: A relatively small set of primitive constructs can be combined in a relatively small number of ways to build the control and data structures of the language.

- **Writability**: How easy to create or write a program for given function.
  - Syntax Design: Special words and 'form and meaning' offset readability.
  - Expressivity: Primal concern ways for specifying computations.

- **Reliability**: Program performs to its specification under all conditions.
  - Type checking: Testing for type errors in a given program.

- **Robustness**: Ability to interrupt run-time errors.

- **Restricted aliasing**: Two or more distinct names that can be used to access the same memory cell.
2. Influences on language design -
   Factors that influence the form design of programming languages:
   (1) Language evaluation criteria
   (2) Computer architecture
   (3) Programming design methodologies
       **Computer Architecture** (Von Neumann Architecture) -
       ```
       \[
       \begin{array}{c|c}
       \text{ALU} & \text{CU} \\
       \end{array}
       \]
       ```
       \(	ext{CPU} \leftarrow \text{ALU} \leftrightarrow \text{CU} \rightarrow \text{I/O devices}
       \)
       \(\text{CU} \rightarrow \text{CONTROL UNIT}
       \)
       \(\text{AU} \rightarrow \text{Arithmetic and logic Unit}
       \)
       \(\text{CU} \rightarrow \text{Central Processing Unit}
       \)

   **Programming Design Methodologies**
   - 1960: structured programming (ABMS, ADA)
   - 1970: procedure-oriented programming (C, LISP, FORTRAN, PASCAL)
   - late 1970s: data-oriented programming (SIMULA, ADA)
   - early 1980s: object-oriented programming (C++, JAVA, SMALLTALK)

3. Language Categories -
   - Imperative, functional, logic and object-oriented
     - Imperative: use variables, assignments, statements & iteration
     - Functional: use functions, parameters & recursion
     - Logic: use facts and rules
     - Object-oriented: use data abstractions, inheritance & dynamic method binding

4. Programming Paradigms -
   It provides the view that the programmer has of the execution of the program. There are mainly four computational models that define most programming today -
   (1) Imperative or Procedural programming/languages
   (2) Functional or Applicative programming/languages
   (3) Algebraic or Rule-based programming/languages
   (4) Object-oriented programming/languages
Impure languages -

They are command driven, a statement oriented language. A program consists of a sequence of statements, and the execution of each statement causes the computer to change the value of one or more locations in its memory.

Syntax - Statement 1;
         Statement 2;
         ........
         Statement n;

eg - C, C++, FORTRAN, PASCAL, ADA

Functional language -

Here we look at the function that must be applied to initial machine state, by accessing the initial set of equalities and combine them in a way that leads to the problem solution.

Syntax - function (function2 (function1 (data)) ...)

eg - LISP, ML

Logical languages -

It checks for the presence of a certain enabling condition and when present, execute an appropriate action.

Syntax - enabling condition1 \rightarrow action1
         enabling condition2 \rightarrow action2
         ........
         enabling conditionn \rightarrow actionn

eg - Prolog

Object-oriented programming (OOP) -

It creates partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

eg - Smalltalk, C++, JAVA
Programming Language Implementation

An implementation of a programming language requires that programs in the language be analyzed, and then translated into a form that can be either:

1. Run by a computer (i.e., "Real Machine") called compiling
2. Run by an interpreter (i.e., "Virtual Machine") called interpreting

Compile-and-Run Process

Virtual Machines and Interpreters

Programming Environments

A programming environment is the collection of tools used in the development of software. This collection may consist of only a file system, a text editor, a linker, and a compiler, or it may include a large collection of integrated tools, each accessed through a uniform user interface.

Examples: UNIX (1970s), Borland TurboC, Microsoft Visual Studio, .NET (C#, Visual Basic.NET, TypeScript, F#, C++/CLI), NetBeans (Java, JavaScript, Ruby, PHP)
Issues in Language Translation -

Syntax -

It describes the structure of programs without any consideration of their meaning.

Key Concerns Regarding Syntax -
Readability, Usability, Verifiability, Translatability and lack of ambiguity.

Basic Syntax concepts -
Character set (alphabet), identifiers (thing of letters or digits usually begins with a letter), operator symbols (-, +, *), keywords, nonce words (optional words to improve readability), comments, blanks, delimiters (used to denote the beginning and the end of syntactic constructs), expressions, statements.

Overall Program - Subprogram Structure -

1. Separate subprogram definitions - separate compilation, linked at last line.
2. Separate data definitions - general approach in OOP.
3. Nested subprogram definitions.
4. Separate interface definitions - C/C++ Header files.
5. Data description separated from executable statements.
6. Unification subprogram definition.

Stages in Translation

Source Program Analyzed

Lexical Analyzer

Syntax Analyzer

Semantic Analyzer

Intermediate Code

Optimiser

Generator

Synthesis

Analyzing the Tokens - Keywords, identifiers, constants.

Syntax Analysis (Parsing) - Determining the structure of the program.

Semantic Analysis - Assigning meaning to the syntactic structures.

Basic Semantic tasks - Symbol-table maintenance, error detection, macro processing, insertion of implicit information.
Optimization - Removing redundant statements

Code Generation - Obtaining the object code of the program

Linking and Binding - Obtaining the executable code of the program

8. Content Free Grammar (CFG) -
   A content free grammar or CFG is represented by 4-tuple (V, T, P, S) where
   V → set of variables or non-terminals (phrase structure)
   T → set of terminals (words or token symbols)
   P → set of productions (produce the strings of language)
   S → Starting variable

9. Backus-Naur Form (BNF) - (equivalent to CFG)
   If CFG productions have used only the metasymbols "→" and "|". (Sometimes parentheses are allowed to group things together)
   It is used to describe another language.


11. Semantics -
    The meaning given to the various syntactic constructs.
    
    Operational semantics - how a computation is performed
    
    Denotational semantics - what is computed by giving a mathematical object
    
    Algebraic semantics - describe the meaning of the program by defining a algebra
    
    Translation semantics - how to translate a program into an other language.