What a good Programming Language can do for me and what I can't expect it to do?

- **What I Expect:**

  To provide a flexible, yet efficient type system. If I want to be able to multiply two `double` floating precision numbers, then I want the language to support that. By that, I mean that a language should provide a set of simple data types that directly corresponds to system resources (eg.: two doubles should take 16 bytes of memory) and a well defined set of operations between those types (+,−, ∗, /).

  On the other hand, simple types should be able to be used to build more complex types (eg.: a `Matrix` type) and facilities to supply operations for those complex types (eg.: a Matrix Inversion `Matrix^` operator should result in an inverted matrix).
  To summarize, I expect a type system and the corresponding language facilities to let the programmer to reach its goals, and represent ideas directly in code in an efficient and elegant manner (`performance, readability`).

  To provide language facilities which enables the programmer to implement abstract concepts and manifest them in some way. By this, general terms (eg. `Shape`) can be expressed as an abstract data type and specialized types (eg. `Polygon inherits from Shape`) can be implemented by inheriting from the given abstract type.
  This also implies support for the need of building type hierarchies, so introducing multileveled hierarchies where the root is an abstract class and the nth specialized class is connected by inheriting from its successors up to root (`inheritence chain support`).

  To provide a way for breaking down the code to easily reuseable fragments (eg. `functions`) and an efficient and safe way to handle resources (eg. `object instantiation and destruction`) without allowing the user to cause any resource leaks. Support the concept of data hiding (aka. `black box`) and provide an ability to expose the user interface only (eg. `public`, `private`).
  Allow the programmer to draw boundaries to be able to logically distinct parts of the codebase (eg. `namespaces`). Also a need for maintainability when it comes to large software is critical. Ensure modularization options for that (eg. `separate compilation`).

  To support programming mechanisms that enables building structured code such as `loops`, `conditionals`, `expressions` and `branches`. Enable scoping for blocks, but introduce constants, static data, dynamic data and memory pointers too.

Finally, provide a set of useful, default package built with the given programming language. In this way the basic building blocks (eg.: `I/O`, `containers`, `algorithms`) for the most common problems can be used enabling concentration and progress towards the actual goals.
So, a minimal support from the language needs to be provided through foundation libraries.

- **What I Can't Expect:**

  To maintain a rigorous set of connections between implemented compilers. A programming language should come up with its specifications first. Then, compilers can be made.

  I can't expect a good programming language to be specialized in one domain but not on another. A good programming language should have a general purpose in order to be applicable on many domains.
  Consequent reasoning by an implemented compiler (based on a good programming language) cannot be expected either in many cases to resolve issues or find errors before runtime.
  Overwhelming support for many domains in terms of foundation libraries cannot be expected: a good programming language doesn't have to come with libraries covering domains such as Web Development, 3D Graphics, Audio and so on.

  It also cannot be expected that a good programming language will help a given compiler to find and fix semantic errors. That's the job of the programmer to fix.
  In general, many things that is reasonably cannot be expected from a programming language is the responsibility either of the software architect, software engineer, software tester, and customer in terms of human resources and the compiler, libraries, frameworks and utilities in terms of technical resources.

  A good programming language doesn't have to provide much runtime support, but it must have the ambition to produce high performance machine code.

  Finally, I can't expect a programming language to ensure, that the code written in it can be ran safely without runtime errors or data loss.
  That's the concern of the programmer to avoid such situations.