AIM: What are the differences between high level and low level program languages?

**High-level program language:**
These are computer programs such as Python, Java or Objective –C that have a more English like dialect so it can be easier to program the computer.

**Low-level program language:**
This type of language is the closest to the language of the machine. This is called Assembly language. Such languages are still in use today particularly when speed is essential or in cases when the programmer need to perform an operation that could not be done in any other form.

**High Level programs:**
These programs are designed to recode the programs instructions down to Machine language. Rather than dealing with registers, memory addresses and call stacks, high-level languages deal with variables, arrays, objects, complex arithmetic or Boolean(YES, NO) expressions, subroutines and functions, loops and other abstract computer science concepts, with a focus on usability over optimal program efficiency. Unlike low-level assembly languages, high-level languages have few, if any, language elements that translate directly into the machine's native codes. Other features, such as string handling routines, object-oriented language and file input/output, functionality can now be implemented more easily.

**Interpreters:**
An interpreter is a program that reads and executes code. Some of the more popular program languages that use interpreters include Python, and Ruby.

**Compiler:**
This type of translator translates code written in a high level language to a lower level language, object/machine code. The most common reason for translating source code is to create an executable program (converting from a high level language into machine language).

Interpreters and compilers are similar, since they both recognize and process source code. However, a compiler does not execute the code like an interpreter does. Instead, a compiler simply converts the source code into machine code, which can be run directly by the operating system as an executable program. Interpreters bypass the compilation process and execute the code directly.
Lisp:
(historically, **LISP**) Originally specified in 1958, Lisp is the second-oldest high-level programming language in widespread use today; only Fortran is older (by one year). Like Fortran, Lisp has changed a great deal since its early days, and a number of dialects have existed over its history. Today, the most widely known general-purpose Lisp dialects are Common Lisp and Scheme.

Lisp was originally created as a practical mathematical notation for computer programs, influenced by the notation of Alonzo Church’s lambda calculus. It quickly became the favored programming language for artificial intelligence (AI) research. As one of the earliest programming languages, Lisp pioneered many ideas in computer science, including tree data structures, automatic storage management, dynamic typing, conditionals, higher-order functions, recursion, and the self-hosting compiler.

Scheme:
Scheme is primarily a functional programming language. It shares many characteristics with other members of the Lisp programming language family. Scheme’s very simple syntax is based on s-expressions, parenthesized lists in which a prefix operator is followed by its arguments.