1. Coding refers to the process of writing the source code for a program using a programming language.
2. It involves defining the program's logic, algorithms, data structures, and instructions that determine how the program should function.
3. During coding, developers write and edit lines of code using a text editor or an integrated development environment (IDE).

Compiling:

- Compiling is the process of translating the source code written in a high-level programming language into machine-executable code.
- It involves using a compiler, which analyzes the source code, checks for syntax errors, and converts it into a lower-level representation (e.g., machine code or bytecode) that the computer can understand.
- The compiler performs various optimizations and generates an output file, such as an executable file or bytecode, ready for execution.

Packaging:

- Packaging involves preparing the program and its dependencies for distribution or deployment.
- It includes bundling all the necessary files, libraries, and resources required for the program to run correctly.
- Packaging can involve creating installation packages, packaging the program as a container image, or creating distributable archives (e.g., ZIP files) that contain all the required components.

Deploying

- Deploying refers to the process of installing and configuring the packaged program on a target environment for execution.
- It involves setting up the necessary infrastructure, servers, and dependencies required to run the program in a production or live environment.
- Deployment includes tasks such as configuring databases, network settings, security measures, and ensuring the program is ready to be accessed by end-users.

Running:

- Running a program refers to the actual execution of the compiled or packaged code.
- Once the program is compiled and packaged, it can be executed on a computer or within a specific runtime environment.
- Running a program involves loading the executable or bytecode into memory, initializing the necessary resources, and executing the program's instructions.
- During runtime, the program interacts with the system, processes data, performs computations, and produces the desired output.

Compilers VS Interpreters A compiler and an interpreter are both tools used in software development, but they differ in how they execute programs and convert source code into machine code or bytecode.

Compiler A compiler is a software program that translates the entire source code of a program into an executable file or object code before its execution. Here's how it works:

- The compiler takes the entire source code as input.
- It analyzes and processes the code, checking for syntax and semantic errors.
- It translates the code into machine code or bytecode, specific to the target platform.
- The resulting output is a standalone executable file or object code that can be run independently.
- The compiled code is generally more optimized for performance, as the compiler performs various optimizations during the compilation process.

Advantages of Compilers

- Faster execution: Since the code is pre-translated, the compiled program generally executes faster.
- Portability: Compiled code can be distributed and executed on different platforms without requiring the presence of the compiler itself.

Interpreter An interpreter is a software program that reads and executes the source code line by line, without prior translation. Here's how it works:

- The interpreter reads the source code line by line.
- It analyzes and processes each line, checking for errors.
- It executes the instructions directly, translating them into machine code or intermediate representations on the fly.
- The output is generated and displayed or executed as each line is interpreted.

Advantages of Interpreters

- Easier debugging: Since the interpreter processes code line by line, it can provide detailed error messages and allow developers to debug the code more easily.
- Dynamic execution: Interpreted languages often have features that allow code to be modified or executed dynamically during runtime.

All in all, the key difference between a compiler and an interpreter lies in their approach to executing programs. A compiler translates the entire source code before execution, resulting in standalone executables, while an interpreter executes the source code line by line without prior translation. Each approach has its own strengths and weaknesses, and different programming languages and scenarios may benefit from one or the other.

- How is compilation process (programming languages that use compilers) different from interpretation process (programming languages that use a runtime to execute code line-by-line on the fly)?

- The compilation process and interpretation are two different approaches to executing or transforming source code. Here are the key differences between them:

Compilation:

- Ahead-of-time: Compilation is typically performed before the program is executed. The entire source code is translated into machine code or an intermediate representation that can be executed directly.
- Translation to machine code: The compiler analyzes the entire source code, performs various optimizations, and generates optimized machine code specific to the target platform or architecture.
- Separate execution: Once the compilation is complete, the resulting executable code can be executed multiple times without the need for recompilation, unless changes are made to the source code.
- Optimizations: Compilers can perform extensive optimizations during the compilation process, taking advantage of the global understanding of the code to improve performance.
- Static typing: Compiled languages often have static typing, meaning the types of variables are checked and resolved at compile time.

Interpretation:

- On-the-fly execution: Interpretation happens during the program's execution. The interpreter reads the source code line by line, translates it into machine code or performs other operations, and executes it immediately.
- No explicit translation to machine code: Instead of translating the entire source code upfront, interpretation typically involves parsing the code, building an internal representation, and executing it directly.
- Dynamic execution: Each line or statement is executed immediately after it is interpreted, and changes to the source code can take effect during runtime without the need for recompilation.
- Limited optimizations: Interpreters usually have less opportunity for global optimizations compared to compilers, as they analyze and optimize code on a smaller scale, statement by statement or block by block.
- Dynamic typing: Interpreted languages often support dynamic typing, allowing variables to change types during runtime.

Note: Compilers use compilation process, and Interpreters use interpretation method. Programs compiled using a compiler are generally standalone independent executables that don't need anything else to run on an operating system. For example a C++ program. Interpreter based languages use interpretation process and generally require a runtime environment to run the program. For example: Python. You cannot run a Python program without the python runtime environment. Some other languages like Java use part-compilation and part-interpretation. A Java program is compiled into object code, then that object code is given as an input to JVM (Java runtime) so that a Java program can be run. Whenever you run a Java program, first the JVM is loaded, and then your program's object code is supplied to the JVM (Java runtime).