**Comparison between programming languages**

1. **Low level and high level**

* **high level**
* **One can easily interpret and combine these languages as compared to the low-level languages.**
* **They are very easy to understand.**
* **Such languages are programmer-friendly.**
* **Debugging is not very difficult.**
* **They come with easy maintenance and are thus simple and manageable.**
* **One can easily run them on different platforms.**
* **They require a compiler/interpreter for translation into a machine code.**
* **A user can port them from one location to another.**
* **Such languages have a low efficiency of memory. So it consumes more memory than the low-level languages.**
* **They are very widely used and popular in today’s times.**
* **Java, C, C++, Python, etc., are a few examples of high-level languages.**
* **low level**
* **They are also called machine-level languages.**
* **Machines can easily understand it.**
* **High-level languages are very machine-friendly.**
* **Debugging them is very difficult.**
* **They are not very easy to understand.**
* **All languages come with complex maintenance.**
* **They are not portable.**
* **These languages depend on machines. Thus, one can run it on various platforms.**
* **They always require assemblers for translating instructions.**
* **Low-level languages do not have a very wide application in today’s times.**

**2. interpreted and compiled**

* Interpreted

**In interpreted languages (e.g., Python, JavaScript, Ruby), the source code is executed line by line by an interpreter, allowing it to be run directly without compilation. These languages stand out in web applications or projects requiring rapid prototyping.**

**Advantages**

**• Platform Independence: In interpreted languages, source code is executed directly in the interpreter, allowing the same code to be run on different platforms. This is a considerable advantage, especially in web development. For example, JavaScript is an interpreted language and can be run in all modern browsers, making it an ideal solution for web application teams.**

**• Fast Development Cycle: Thanks to interpreters, code can be tested immediately after it is written, and changes are instantly reflected. This feature is particularly useful in rapid prototyping or minimum viable product (MVP) development processes. In a project developed with Python, developers can quickly experiment without wasting time in the compilation process and instantly see the results.**

**• Dynamic and Flexible: Interpreted languages allow changes to be made during runtime. This flexibility is vital in script-based projects, system automation, and data analysis. Languages like Python or Ruby are often preferred in data science projects due to their flexibility.**

**Disadvantages**

**• Low Performance: Interpreted languages generally run slower than compiled languages since the code is executed line by line. In projects that require high performance, choosing a compiled language over Python or Ruby might be more logical.**

**• Lack of Confidentiality: In interpreted languages, the source code is directly executed and presented to the user, making it difficult to keep the code confidential. This can be a disadvantage for commercial software.**

* **Compiled**

**Compiled languages (e.g., C, C++, Go) have a compiler that converts source code directly into machine code. This enables the application to communicate directly with hardware, making compiled languages ideal for high-performance and reliability-focused projects. Let’s look closely at the key features of these languages.**

**Advantages**

**• High Performance: During the compilation process, source code is transformed into machine code that the hardware can directly understand. Therefore, compiled languages offer high performance. For example, compiled languages are frequently used in game development and graphics processing software. Game engines developed with C++ are known for their speed and efficiency.**

**• Early Error Detection: Compilers perform extensive error checks during compilation, detecting errors early. This reduces the likelihood of runtime errors and contributes to developing a more robust application. As a result, compiled languages are preferred in sectors where reliability and accuracy are essential, such as finance and healthcare.**

**• Code Confidentiality: In compiled languages, the source code is not directly accessible to the user; it is converted into machine code, making it easier to protect the code. This feature is crucial in commercial software because it prevents the source code from falling into the hands of malicious individuals.**

**Disadvantages**

**• Platform Dependency: Compiled languages are generally platform-dependent. For instance, software compiled for Windows may not work on Linux or macOS. This requires additional effort and compilation when developing a cross-platform application. Compiling separately for each platform increases the workload, especially for cross-platform applications.**

**• Long Compilation Times: Compiling code can take a long time in large projects. This can slow down the development process and hinder frequent testing by developers. For example, if you are developing a large software project in C++, you may have to wait for long compilation times after every minor change.**

1. **programming and scripted**

**Every scripting language is basically a programming language. The only theoretical difference is that a scripting language does not include the compilation step- it is interpreted instead. For instance, one needs to first compile a C program before running it. On the other hand, one does not need to compile a scripting language such as PHP or JavaScript. There are various ways in which both languages vary. In this article, we will dive deeper into the difference between scripting and programming languages. But let us first take a brief look at both of them.**

* **Programming**
* **One needs to compile the programming languages to machine code so as to run them on the hardware of an underlying OS (operating system). A user needs to deploy a certain Integrated Development Environment (IDE) for using programming languages. A programmer needs to provide an instruction set for the computers for achieving certain goals. One can also implement certain algorithms by writing the programs.**
* **Out of all the programming languages present in the market, specific documentation dominates a majority of them. All the other languages comprise dominant implementation (treated as a reference). An example here is that the ISO standard associates with the C programming language. On the other hand, languages like Perl belong to the latter category.**
* **One can use a programming language for transforming data. It basically happens when creating those CPU instructions that jot down the input info into the output. An example here is using a set of conditions for solving an equation set. One can consider various programming languages such as C, C++, Scala, Java, etc., as general-purpose languages. These fall under the compiled programming languages. You must add some texts to write the score code, and then you can run them through a compiler. As a result, it would create various binary instructions.**
* **Scripted**
* **Scripting languages help in automating various software apps, web pages in a browser, shell usage of an OS (operating system), etc. The scripting languages like VBScript, Perl, Javascript, etc., do not require compilation, and they have less access to any computer’s native abilities. It is because these rather run on an original programming language’s subset. An example here could be that the Javascript won’t have the ability to access your file system.**
* **Generally, a scripting language is interpreted. It doesn’t primarily focus on building applications- but it can render behavior to an application that already exists. It basically helps in writing codes for targeting a software system. Thus, it can also automate a given operation on any software system. So basically, scripts act as a set of instructions that target any software system.**
* **The scripting languages have eventually evolved and become more powerful. They now no longer create minute scripts for automating a software system’s operations. One can also use scripting languages for building rich applications. These can customize, manipulate, and automate an existing system’s facilities. The scripting languages come with a mechanism that exposes functionality to the program control.**

1. **Open source and not open source**

**With closed source software (also known as proprietary software), the public is not given access to the source code, so they can’t see or modify it in any way.**

**But with open source software, the source code is publicly available to anyone who wants it, and programmers can read or change that code if they desire. Keep in mind that you don’t have to read or modify any code in order to use an open source product.** **The vast majority of apps, games, and other popular software is closed source. However, there are open source options for many types of programs. If you want an open source alternative to Microsoft Office, you could use LibreOffice. Instead of using Windows, you could try an open source Linux operating system. Other common open source examples include the Firefox web browser and WordPress blogging platform.**

**One of open source’s biggest advantages is that it’s usually free, although some features and technical support may cost extra. Also, because the code is available to anyone who wants it, public collaboration can fix bugs, add features, and improve performance within a relatively short amount of time.** **However, open source software isn’t perfect. It may not be as user friendly as closed source software, and if you run into trouble it may be difficult to find technical support, especially for less popular programs.**

**Closed source software is more likely to be a stable, focused product, and if you need support customer service is typically easier to access. However, closed source software often costs money, and if it has any bugs or missing features you’ll have to wait on the creator to address the problems.**

1. **Support OOP and not Supporting OOP**

**Object-oriented programming (OOP) is a programming paradigm that focuses on modeling software systems as collections of interacting objects, each with its own data and behavior. Languages that support OOP provide mechanisms to define objects, classes (blueprints for objects), and the interactions between them. On the other hand, non-object-oriented languages do not adhere strictly to the OOP paradigm and may use other programming paradigms such as procedural programming or functional programming. Here’s a comparison between object-oriented and non-object-oriented languages:**

* **OOP**

**Key Characteristics:**

**Objects and Classes:**

**Objects are instances of classes that encapsulate data (attributes) and behavior (methods or functions).**

**Classes define the blueprint or template from which objects are created, specifying the attributes and methods that objects of that class will have.**

**Encapsulation:**

**Encapsulation involves bundling data (attributes) and methods (functions or procedures) that operate on the data into a single unit (class).**

**Objects interact with each other through well-defined interfaces, hiding internal implementation details.**

**Inheritance:**

**Inheritance allows classes to inherit attributes and behaviors from parent classes (super classes).**

**It promotes code reuse and enables hierarchical relationships between classes.**

**Polymorphism:**

**Polymorphism allows objects of different classes to be treated as objects of a common superclass, enabling flexibility and extensibility in code design.**

**It supports method overriding (redefining a method in a subclass) and method overloading (defining multiple methods with the same name but different parameters).**

**Examples :**

**Java, C++ , python, C#**

* **Non OOP**

**Key Characteristics:**

**Procedural Programming:**

**Procedural languages focus on procedures (functions or subroutines) that operate on data.**

**Data and procedures are separate, and the emphasis is on step-by-step procedures for solving problems.**

**Functional Programming:**

**Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing state and mutable data.**

**Examples:**

**C ,Fortran ,Lisp**

**conclusion**

**The choice between using an object-oriented or non-object-oriented language depends on factors such as the nature of the problem domain, project requirements, performance considerations, and developer preference. Object-oriented languages provide powerful tools for modeling complex systems and promoting code reuse, while non-object-oriented languages offer alternative paradigms suited to specific types of applications and programming styles. Modern languages often support multiple paradigms, allowing developers to choose the most appropriate approach for each programming task.**