# Assignment Types of Programming languages

- ## Difference Between High-Level and Low-Level Languages

| Parameter | High-Level Language | Low-Level Language |
|---|---|---|
| Basic | These are programmer-friendly languages that are manageable, easy to understand, debug, and widely used in today's times | These are machine-friendly languages that are very difficult to understand by human beings but easy to interpret by machines. |
| Ease of Execution | These are very easy to execute. | These are very difficult to execute. |
| Process of Translation | High-level languages require the use of a compiler or an interpreter for their translation into the machine code. | Low-level language requires an assembler for directly translating the instructions of the machine language. |
| Efficiency of Memory | These languages have a very low memory efficiency. It means that they consume more memory than any low-level language. | These languages have a very high memory efficiency. It means that they consume less energy as compared to any high-level language. |
| Portability | These are portable from any one device to another. | A user cannot port these from one device to another. |
| Dependency on Machines | High-level languages do not depend on machines. | Low-level languages are machine-dependent and thus very difficult to understand by a normal user. |
| Debugging | It is very easy to debug these languages. | A programmer cannot easily debug these languages. |
| Usage | High-level languages are very common and widely used for programming in today's times. | Low-level languages are not very common nowadays for programming. |
| Speed of Execution | High-level languages take more time for execution as compared to low-level languages because these require a translation program. | The translation speed of low-level languages is very high. |
| Abstraction | High-level languages allow a higher abstraction. | Low-level languages allow very little abstraction or no abstraction at all. |
| Ease of Modification | The process of modifying programs is very difficult with high-level programs. It is because every single statement in it may execute a bunch of instructions. | The process of modifying programs is very easy in low-level programs. Here, it can directly map the statements to the processor instructions. |

| | | |
|---|---|---|
| **Examples** | **Some examples of high-level languages include Perl, BASIC, COBOL, Pascal, Ruby, etc.** | **Some examples of low-level languages include the Machine language and Assembly language.** |

- ## Difference between Compiled and Interpreted Language

| NO. | Compiled Language | Interpreted Language |
|---|---|---|
| 1 | Compiled language follows at least two levels to get from source code to execution. | Interpreted language follows one step to get from source code to execution. |
| 2 | A compiled language is converted into machine code so that the processor can execute it. | An interpreted language is a language in which the implementations execute instructions directly without earlier compiling a program into machine language. |
| 3 | The compiled programs run faster than interpreted programs. | The interpreted programs run slower than the compiled program. |
| 4 | In a compiled language, the code can be executed by the CPU. | In Interpreted languages, the program cannot be compiled, it is interpreted. |
| 5 | This language delivers better performance. | This language delivers slower performance. |
| 6 | Examples of pure compiled languages are C, C++, Haskell, Rust, and Go. | Examples of common interpreted languages are PHP, Ruby, Python, and JavaScript. |

- **Difference Between Scripting and Programming Languages**

| Parameter | Scripting Language | Programming Language |
|---|---|---|
| Language Type | The scripting languages are interpreter-based languages. | The programming languages are compiler-based languages. |
| Use | The scripting languages help in combining the existing components of an application. | The programming languages help in developing anything from scratch. |
| Running of Language | A user needs to run scripting languages inside an existing program. Thus, it's program-dependent. | Programming languages are program-independent. |
| Conversion | Scripting languages convert high-level instructions into machine language. | Programming languages help in converting the full program into the machine language (at once). |
| Compilation | You don't need to compile these languages. | These languages first need a compilation. |
| Design | These make the coding process simple and fast. | These provide full usage of the languages. |
| Support | These provide limited support to data types, user interface design, and graphic design. | These provide rich support for graphic design, data types, and user interface design. |
| File Type | Scripting languages don't create any file types. | Programming languages create .exe files. |
| Complexity | These are very easy to use and easy to write. | These are pretty complex in terms of writing and usage. |
| Type of Coding | Scripting languages help write a small piece of an entire code | Programming languages help write the full code concerning a program. |
| Developing Time | These take less time because they involve lesser code. | These take more time because a programmer must write the entire code. |
| Interpretation | We usually interpret a scripting language in another program. | The compile results of a programming language are stand-alone. No other program needs to interpret it. |

| | | |
|---|---|---|
| **Requirement of Host** | **Scripting languages require hosts for execution.** | **Programming languages are self-executable. They don't require any host.** |
| **Length of Codes** | **These involve very few and short coding lines.** | **These require numerous lines of coding for a single function.** |
| **Maintenance** | **These involve very low maintenance.** | **These involve high maintenance.** |
| **Cost** | **It is easier and cheaper to maintain a scripting language.** | **Maintaining a programming language is comparatively more expensive.** |
| **Example** | **VB Script, Perl, Ruby, PHP, JavaScript, etc.** | **C, C++, COBOL, Basic, VB, C#, Pascal, Java, etc.** |

- ## Difference between Open Source Software and Closed Source Software

| Parameters | Open Source Software | Not open Source Software (closed) |
|---|---|---|
| **Short-form** | **Also generally referred to as OSS.** | **Also generally referred to as CSS.** |
| **Basics** | **These refer to that computer software in which the source remains open. Thus, the general public is able to access it easily and use it.** | **These refer to that computer software in which the source code remains closed. Thus, the general public has no access to it.** |
| **Source Code** | **It is open source and public.** | **It is closed source and protected from all.** |
| **Modification of Software** | **Any user or organisation can easily change this code since it is available as an open-source for any person to take a look at.** | **Only the organisation or individual that created it has access to the code. Only they can modify it.** |
| **Price** | **It is comparatively cheaper and more cost-effective.** | **It is comparatively expensive and less cost-effective.** |
| **Restrictions** | **Users have no restrictions on the modification or usability of the software.** | **Users are not very restricted on the modification or usability of the software.** |
| **Programmer Recognition** | **Programmers need to compete against one another for recognition.** | **Programmers don't need to compete against each other for recognition.** |

| | | |
|---|---|---|
| **Programmer Feedback** | **All programmers are free to provide improvement strategies, and they get incorporated if they get accepted by the software developers.** | **The software organization/firm hires programmers to provide an improvement on their software.** |
| **Failure and Glitch** | **Open-source software is prone to failing faster as well as fixing faster.** | **There is no room for failure of closed source software.** |
| **Purchasing of Code** | **A user purchases it along with its source code.** | **You don't have to purchase it along with its source code.** |
| **Software Responsibility** | **No one is explicitly responsible for an OSS.** | **The vendor is responsible for anything that might happen to a CSS.** |
| **Installing on Computers** | **We can install an OSS on any computer device.** | **Having a license of a CSS is a prerequisite before you install it on any computer device, and this validity is also limited.** |
| **Examples** | **Firefox, Gimp, Open Office, Android, Alfresco, Thunderbird, Zimbra, Mailman, MySQL, TeX, Moodle, Perl, Samba, KDE, PHP, and many more.** | **Google Earth, Skype, Adobe Flash, Java, Adobe Reader, Virtual Box, WinRAR, Microsoft Windows, Microsoft Office, Adobe Flash Player, Mac OS, and many more.** |

- ## Difference between Supporting OOP and Not- Supporting OOP

| Parameter | Languages that Supporting OOP | Languages that Not-Supporting OOP |
|---|---|---|
| **Data Structure** | **Uses objects and classes** | **Uses primitive types and structures** |
| **Encapsulation** | **Supports encapsulation of data and methods** | **Data and functions are separate** |
| **Inheritance** | **Supports inheritance for code reuse** | **No inheritance; code is reused through functions** |
| **Polymorphism** | **Allows method overriding and overloading** | **No polymorphism; fixed function signatures** |

| | | |
|---|---|---|
| **Abstraction** | **Provides abstraction through interfaces and classes** | **limited abstraction capabilities** |
| **Code Reusability** | **High reusability through object composition and inheritance** | **Limited reusability; often requires duplication** |
| **State Management** | **Objects maintain their own state** | **State managed through global variables or passed parameters** |
| **Error Handling** | **Often uses exception handling** | **Typically relies on return codes** |
| **Development Approach** | **Encourages a modular and scalable approach** | **Encourages a modular and scalable approach** |
| **Initialization** | **Uses constructors for object initialization** | **No formal initialization for data types** |
| **Development Speed** | **Can speed up development for large projects** | **May speed up development for small scripts** |
| **Flexibility** | **More flexible for design changes** | **Less flexible; changes can require significant refactoring** |
| **Performance** | **May incur overhead from abstraction** | **Typically more efficient for low-level tasks** |
| **Testing and Maintenance** | **Easier unit testing and maintenance due to modularity** | **Testing can be less structured** |
| **User-defined Types** | **Supports user-defined types through classes** | **Primarily uses built-in data types** |
| **Examples of Languages** | **Java, C++, Python, C#, Ruby, Swift** | **C, Fortran, BASIC, Assembly, Pascal** |