NAOGAON POLYTECHNIC INSTITUTE, NAOGAON.

Subject Name : Python Programming

Subject Code : 28521

Tchonology : Computer Science & Technology

Semester : 2nd

Lesson plan-1(Theory Classs 1)

1. BASICS OF PROGRAMMING

1.1 State Computer Programming.

1.2 Explain Programming Language and its Classification.

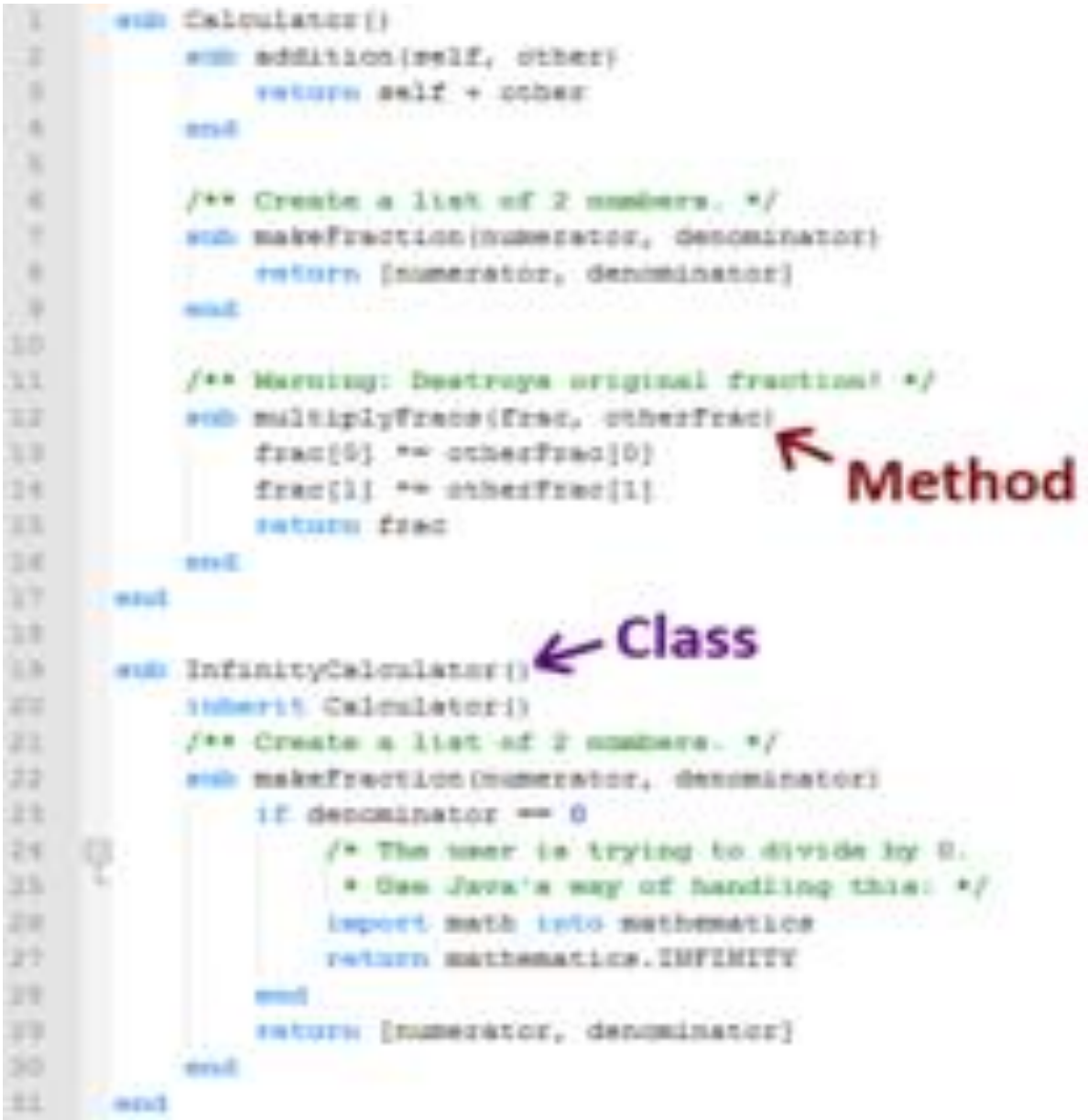
1.3 State Translator Programs.

TEACHER'S NAME : Md. Mahbubul Alam

DESIGNATION : Chief Instructor (Tech/Computer)

Computer program

A computer program is a sequence or set of instructions in a programming language for a computer to execute. It is one component of software, which also includes documentation and other intangible components. A computer program in its human-readable form is called source code.



Examples of a Computer Program:

Examples of a Computer Program · *Firmware* · Operating Systems · Applications · Mobile Apps · Cloud Services · Systems · Platforms · Servers.

MS Word, MS Excel, Adobe Photoshop, Internet Explorer, Chrome, etc., are examples of computer programs

A list of some of the most well-known computer software examples includes: Operating systems (such as Microsoft Windows, Linux, macOS) Productivity Software (for example, Microsoft Office Suite including Word, Excel, and PowerPoint) Internet Browsers (including Firefox, Chrome, and Safari)

Computer programming

Computer programming is the process that professionals use to write code that instructs how a computer, application or software program performs.

Computer programming or coding is the composition of sequences of instructions, called programs, that computers can follow to perform tasks.

Programming language

A *programming language* is a system of notation for writing computer programs.

A programming language is a way for programmers (developers) to communicate with computers. Programming languages consist of a set of rules that allows string values to be converted into various ways of generating machine code, or, in the case of visual programming languages, graphical elements.

Language types

Machine and assembly languages

A machine language consists of the numeric codes for the operations that a particular computer can execute directly. The codes are strings of 0s and 1s, or binary digits (“bits”), which are frequently converted both from and to hexadecimal (base 16) for human viewing and modification. Machine language instructions typically use some bits to represent operations, such as addition, and some to represent operands, or perhaps the location of the next instruction. Machine language is difficult to read and write, since it does not resemble conventional mathematical notation or human language, and its codes vary from computer to computer.

Assembly language is one level above machine language. It uses short mnemonic codes for instructions and allows the programmer to introduce names for blocks of memory that hold data. One might thus write “add pay, total” instead of “0110101100101000” for an instruction that adds two numbers.

Assembly language is designed to be easily translated into machine language. Although blocks of data may be referred to by name instead of by their machine addresses, assembly language does not provide more sophisticated means of organizing complex information. Like machine language, assembly language requires detailed knowledge of internal computer architecture. It is useful when such details are important, as in programming a computer to interact with peripheral devices (printers, scanners, storage devices, and so forth).

Algorithmic languages

Algorithmic languages are designed to express mathematical or symbolic computations. They can express algebraic operations in notation similar to mathematics and allow the use of subprograms that package commonly used operations for reuse. They were the first high-level languages.

FORTRAN

The first important algorithmic language was FORTRAN (*for*mula *tran*slation), designed in 1957 by an IBM team led by John Backus. It was intended for scientific computations with real numbers and collections of them organized as one- or multidimensional arrays. Its control structures included conditional IF statements, repetitive loops (so-called DO loops), and a GOTO statement that allowed nonsequential execution of program code. FORTRAN made it convenient to have subprograms for common mathematical operations, and built libraries of them.

FORTRAN was also designed to translate into efficient machine language. It was immediately successful and continues to evolve.

ALGOL

ALGOL (*algo*rithmic *l*anguage) was designed by a committee of American and European computer scientists during 1958–60 for publishing algorithms, as well as for doing computations. Like LISP (described in the next section), ALGOL had recursive subprograms—procedures that could invoke themselves to solve a problem by reducing it to a smaller problem of the same kind. ALGOL introduced block structure, in which a program is composed of blocks that might contain both data and instructions and have the same structure as an entire program. Block structure became a powerful tool for building large programs out of small components.

ALGOL contributed a notation for describing the structure of a programming language, Backus–Naur Form, which in some variation became the standard tool for stating the syntax (grammar) of programming languages. ALGOL was widely used in Europe, and for many years it remained the language in which computer algorithms were published. Many important languages, such as Pascal and Ada (both described later), are its descendants.

C

The C programming language was developed in 1972 by Dennis Ritchie and Brian Kernighan at the AT&T Corporation for programming computer operating systems. Its capacity to structure data and programs through the composition of smaller units is comparable to that of ALGOL. It uses a compact notation and provides the programmer with the ability to operate with the addresses of data as well as with their values. This ability is important in systems programming, and C shares with assembly language the power to exploit all the features of a computer's internal architecture. C, along with its descendant C++, remains one of the most common languages.

Business-oriented languages

COBOL

COBOL (*co*mmon *b*usiness *o*riented *l*anguage) has been heavily used by businesses since its inception in 1959. A committee of computer manufacturers and users and U.S. government organizations established CODASYL (*Co*mmittee on *Da*ta *Sy*stems and *L*anguages) to develop and oversee the language standard in order to ensure its portability across diverse systems.

COBOL uses an English-like notation—novel when introduced. Business computations organize and manipulate large quantities of data, and COBOL introduced the record data structure for such tasks. A record clusters heterogeneous data—such as a name, an ID number, an age, and an address—into a single unit. This contrasts with scientific languages, in which homogeneous arrays of numbers are common. Records are an important example of "chunking" data into a single object, and they appear in nearly all modern languages.

SQL

SQL (structured query language) is a language for specifying the organization of databases (collections of records). Databases organized with SQL are called relational, because SQL provides the ability to query a database for information that falls in a given relation. For example, a query might be “find all records with both last name Smith and city New York.” Commercial database programs commonly use an SQL-like language for their queries.

Education-oriented languages

BASIC

BASIC (beginner’s all-purpose symbolic instruction code) was designed at Dartmouth College in the mid-1960s by John Kemeny and Thomas Kurtz. It was intended to be easy to learn by novices, particularly non-computer science majors, and to run well on a time-sharing computer with many users. It had simple data structures and notation and it was interpreted: a BASIC program was translated line-by-line and executed as it was translated, which made it easy to locate programming errors.

Its small size and simplicity also made BASIC a popular language for early personal computers. Its recent forms have adopted many of the data and control structures of other contemporary languages, which makes it more powerful but less convenient for beginners.

Pascal

About 1970 Niklaus Wirth of Switzerland designed Pascal to teach structured programming, which emphasized the orderly use of conditional and loop control structures without GOTO statements. Although Pascal resembled ALGOL in notation, it provided the ability to define data types with which to organize complex information, a feature beyond the capabilities of ALGOL as well as FORTRAN and COBOL. User-defined data types allowed the programmer to introduce names for complex data, which the language translator could then check for correct usage before running a program.

During the late 1970s and ’80s, Pascal was one of the most widely used languages for programming instruction. It was available on nearly all computers, and, because of its familiarity, clarity, and security, it was used for production software as well as for education.

Logo

Logo originated in the late 1960s as a simplified LISP dialect for education; Seymour Papert and others used it at MIT to teach mathematical thinking to schoolchildren. It had a more conventional syntax than LISP and featured “turtle graphics,” a simple method for generating computer graphics. (The name came from an early project to program a turtlelike robot.) Turtle graphics used body-centred instructions, in which an object was moved around a screen by commands, such as “left 90” and “forward,” that specified actions relative to the current position

and orientation of the object rather than in terms of a fixed framework. Together with recursive routines, this technique made it easy to program intricate and attractive patterns.

Hypertalk

Hypertalk was designed as "programming for the rest of us" by Bill Atkinson for Apple's Macintosh. Using a simple English-like syntax, Hypertalk enabled anyone to combine text, graphics, and audio quickly into "linked stacks" that could be navigated by clicking with a mouse on standard buttons supplied by the program. Hypertalk was particularly popular among educators in the 1980s and early '90s for classroom multimedia presentations. Although Hypertalk had many features of object-oriented languages (described in the next section), Apple did not develop it for other computer platforms and let it languish; as Apple's market share declined in the 1990s, a new cross-platform way of displaying multimedia left Hypertalk all but obsolete (*see* the section World Wide Web display languages).

Object-oriented languages

Object-oriented languages help to manage complexity in large programs. Objects package data and the operations on them so that only the operations are publicly accessible and internal details of the data structures are hidden. This information hiding made large-scale programming easier by allowing a programmer to think about each part of the program in isolation. In addition, objects may be derived from more general ones, "inheriting" their capabilities. Such an object hierarchy made it possible to define specialized objects without repeating all that is in the more general ones.

Object-oriented programming began with the Simula language (1967), which added information hiding to ALGOL. Another influential object-oriented language was Smalltalk (1980), in which a program was a set of objects that interacted by sending messages to one another.

C++

The C++ language, developed by Bjarne Stroustrup at AT&T in the mid-1980s, extended C by adding objects to it while preserving the efficiency of C programs. It has been one of the most important languages for both education and industrial programming. Large parts of many operating systems were written in C++. C++, along with Java, has become popular for developing commercial software packages that incorporate multiple interrelated applications. C++ is considered one of the fastest languages and is very close to low-level languages, thus allowing complete control over memory allocation and management. This very feature and its many other capabilities also make it one of the most difficult languages to learn and handle on a large scale.

C#

C# (pronounced *C sharp* like the musical note) was developed by Anders Hejlsberg at Microsoft in 2000. C# has a syntax similar to that of C and C++ and is often used for developing games and applications for the Microsoft Windows operating system.

Ada

Ada was named for Augusta Ada King, countess of Lovelace, who was an assistant to the 19th-century English inventor Charles Babbage, and is sometimes called the first computer programmer. Ada, the language, was developed in the early 1980s for the U.S. Department of Defense for large-scale programming. It combined Pascal-like notation with the ability to package operations and data into independent modules. Its first form, Ada 83, was not fully object-oriented, but the subsequent Ada 95 provided objects and the ability to construct hierarchies of them. While no longer mandated for use in work for the Department of Defense, Ada remains an effective language for engineering large programs.

Java

In the early 1990s Java was designed by Sun Microsystems, Inc., as a programming language for the World Wide Web (WWW). Although it resembled C++ in appearance, it was object-oriented. In particular, Java dispensed with lower-level features, including the ability to manipulate data addresses, a capability that is neither desirable nor useful in programs for distributed systems. In order to be portable, Java programs are translated by a Java Virtual Machine specific to each computer platform, which then executes the Java program. In addition to adding interactive capabilities to the Internet through Web “applets,” Java has been widely used for programming small and portable devices, such as mobile telephones.

Visual Basic

Visual Basic was developed by Microsoft to extend the capabilities of BASIC by adding objects and “event-driven” programming: buttons, menus, and other elements of graphical user interfaces (GUIs). Visual Basic can also be used within other Microsoft software to program small routines. Visual Basic was succeeded in 2002 by Visual Basic .NET, a vastly different language based on C#, a language with similarities to C++.

Python

The open-source language Python was developed by Dutch programmer Guido van Rossum in 1991. It was designed as an easy-to-use language, with features such as using indentation instead of brackets to group statements. Python is also a very compact language, designed so that complex jobs can be executed with only a few statements. In the 2010s, Python became one of the most popular programming languages, along with Java and JavaScript.

Declarative languages

Declarative languages, also called nonprocedural or very high level, are programming languages in which (ideally) a program specifies what is to be done rather than how to do it. In such languages there is less difference between the specification of a program and its implementation than in the procedural languages described so far. The two common kinds of declarative languages are logic and functional languages.

Logic programming languages, of which PROLOG (*pro*gramming in *log*ic) is the best known, state a program as a set of logical relations (e.g., a grandparent is the parent of a parent of someone). Such languages are similar to the SQL database language. A program is executed by an "inference engine" that answers a query by searching these relations systematically to make inferences that will answer a query. PROLOG has been used extensively in natural language processing and other AI programs.

Functional languages have a mathematical style. A functional program is constructed by applying functions to arguments. Functional languages, such as LISP, ML, and Haskell, are used as research tools in language development, in automated mathematical theorem provers, and in some commercial projects.

Scripting languages

Scripting languages are sometimes called little languages. They are intended to solve relatively small programming problems that do not require the overhead of data declarations and other features needed to make large programs manageable. Scripting languages are used for writing operating system utilities, for special-purpose file-manipulation programs, and, because they are easy to learn, sometimes for considerably larger programs.

Perl was developed in the late 1980s, originally for use with the UNIX operating system. It was intended to have all the capabilities of earlier scripting languages. Perl provided many ways to state common operations and thereby allowed a programmer to adopt any convenient style. In the 1990s it became popular as a system-programming tool, both for small utility programs and for prototypes of larger ones. Together with other languages discussed below, it also became popular for programming computer Web "servers."

Document formatting languages

Document formatting languages specify the organization of printed text and graphics. They fall into several classes: text formatting notation that can serve the same functions as a word processing program, page description languages that are interpreted by a printing device, and, most generally, markup languages that describe the intended function of portions of a document.

TeX

TeX was developed during 1977–86 as a text formatting language by Donald Knuth, a Stanford University professor, to improve the quality of mathematical notation in his books. Text formatting systems, unlike WYSIWYG ("What You See Is What You Get") word processors, embed plain text formatting commands in a document, which are then interpreted by the language processor to produce a formatted document for display or printing. TeX marks italic text, for example, as {\it this is italicized}, which is then displayed as *this is italicized*.

TeX largely replaced earlier text formatting languages. Its powerful and flexible abilities gave an expert precise control over such things as the choice of fonts, layout of tables, mathematical notation, and the inclusion of graphics within a document. It is generally used with the aid of

“macro” packages that define simple commands for common operations, such as starting a new paragraph; LaTeX is a widely used package. TeX contains numerous standard “style sheets” for different types of documents, and these may be further adapted by each user. There are also related programs such as BibTeX, which manages bibliographies and has style sheets for all of the common bibliography styles, and versions of TeX for languages with various alphabets.

PostScript

PostScript is a page-description language developed in the early 1980s by Adobe Systems Incorporated on the basis of work at Xerox PARC (Palo Alto Research Center). Such languages describe documents in terms that can be interpreted by a personal computer to display the document on its screen or by a microprocessor in a printer or a typesetting device.

PostScript commands can, for example, precisely position text, in various fonts and sizes, draw images that are mathematically described, and specify colour or shading. PostScript uses postfix, also called reverse Polish notation, in which an operation name follows its arguments. Thus, “300 600 20 270 arc stroke” means: draw (“stroke”) a 270-degree arc with radius 20 at location (300, 600). Although PostScript can be read and written by a programmer, it is normally produced by text formatting programs, word processors, or graphic display tools.

The success of PostScript is due to its specification’s being in the public domain and to its being a good match for high-resolution laser printers. It has influenced the development of printing fonts, and manufacturers produce a large variety of PostScript fonts.

SGML

SGML (standard generalized markup language) is an international standard for the definition of markup languages; that is, it is a metalanguage. Markup consists of notations called tags that specify the function of a piece of text or how it is to be displayed. SGML emphasizes descriptive markup, in which a tag might be “<emphasis>.” Such a markup denotes the document function, and it could be interpreted as reverse video on a computer screen, underlining by a typewriter, or italics in typeset text.

SGML is used to specify DTDs (document type definitions). A DTD defines a kind of document, such as a report, by specifying what elements must appear in the document—e.g., <Title>—and giving rules for the use of document elements, such as that a paragraph may appear within a table entry but a table may not appear within a paragraph. A marked-up text may be analyzed by a parsing program to determine if it conforms to a DTD. Another program may read the markups to prepare an index or to translate the document into PostScript for printing. Yet another might generate large type or audio for readers with visual or hearing disabilities.

World Wide Web display languages

HTML

The World Wide Web is a system for displaying text, graphics, and audio retrieved over the Internet on a computer monitor. Each retrieval unit is known as a Web page, and such pages frequently contain “links” that allow related pages to be retrieved. HTML (*h*yper*t*ext *m*arkup *l*anguage) is the markup language for encoding Web pages. It was designed by Tim Berners-Lee at the CERN nuclear physics laboratory in Switzerland during the 1980s and is defined by an SGML DTD. HTML markup tags specify document elements such as headings, paragraphs, and tables. They mark up a document for display by a computer program known as a Web browser. The browser interprets the tags, displaying the headings, paragraphs, and tables in a layout that is adapted to the screen size and fonts available to it.

HTML documents also contain anchors, which are tags that specify links to other Web pages. An anchor has the form <A HREF= “http://www.britannica.com”> Encyclopædia Britannica</A>, where the quoted string is the URL (uniform resource locator) to which the link points (the Web “address”) and the text following it is what appears in a Web browser, underlined to show that it is a link to another page. What is displayed as a single page may also be formed from multiple URLs, some containing text and others graphics.

XML

HTML does not allow one to define new text elements; that is, it is not extensible. XML (extensible markup language) is a simplified form of SGML intended for documents that are published on the Web. Like SGML, XML uses DTDs to define document types and the meanings of tags used in them. XML adopts conventions that make it easy to parse, such as that document entities are marked by both a beginning and an ending tag, such as <BEGIN>…</BEGIN>. XML provides more kinds of hypertext links than HTML, such as bidirectional links and links relative to a document subsection.

Because an author may define new tags, an XML DTD must also contain rules that instruct a Web browser how to interpret them—how an entity is to be displayed or how it is to generate an action such as preparing an e-mail message.

Web scripting

Web pages marked up with HTML or XML are largely static documents. Web scripting can add information to a page as a reader uses it or let the reader enter information that may, for example, be passed on to the order department of an online business. CGI (common gateway interface) provides one mechanism; it transmits requests and responses between the reader’s Web browser and the Web server that provides the page. The CGI component on the server contains small programs called scripts that take information from the browser system or provide it for display. A simple script might ask the reader’s name, determine the Internet address of the system that the reader uses, and print a greeting. Scripts may be written in any programming language, but, because they are generally simple text-processing routines, scripting languages like PERL are particularly appropriate.

Another approach is to use a language designed for Web scripts to be executed by the browser. JavaScript is one such language, designed by the Netscape Communications Corp., which may be used with both Netscape's and Microsoft's browsers. JavaScript is a simple language, quite different from Java. A JavaScript program may be embedded in a Web page with the HTML tag <script language="JavaScript">. JavaScript instructions following that tag will be executed by the browser when the page is selected. In order to speed up display of dynamic (interactive) pages, JavaScript is often combined with XML or some other language for exchanging information between the server and the client's browser. In particular, the XMLHttpRequest command enables asynchronous data requests from the server without requiring the server to resend the entire Web page. This approach, or "philosophy," of programming is called Ajax (asynchronous JavaScript and XML).

VB Script is a subset of Visual Basic. Originally developed for Microsoft's Office suite of programs, it was later used for Web scripting as well. Its capabilities are similar to those of JavaScript, and it may be embedded in HTML in the same fashion.

Behind the use of such scripting languages for Web programming lies the idea of component programming, in which programs are constructed by combining independent previously written components without any further language processing. JavaScript and VB Script programs were designed as components that may be attached to Web browsers to control how they display information.

Computer Language Translator and its Types

A translator is a computer program that translates a program written in a given programming language into a functionally equivalent program in a different language.

Depending on the translator, this may mean changing or simplifying the flow of the program without changing its core. This makes a program that works the same as the original.

Types of Language Translators

There are mainly three types of translators that are used to translate different programming languages into machine-equivalent code:

1. Assembler
2. Compiler
3. Interpreter

Assembler

An assembler translates assembly language into machine code.

Assembly language consists of mnemonics for machine op-codes, so assemblers perform a 1:1 translation from mnemonic to direct instruction. For example, LDA #4 converts to 0001001000100100.

Conversely, one instruction in a high-level language will translate to one or more instructions at the machine level.

The Benefits of Using Assembler

Here is a list of the advantages of using assembler:

- As a 1 to 1 relationship, assembly language to machine code translation is very fast.
- Assembly code is often very efficient (and therefore fast) because it is a low-level language.
- Assembly code is fairly easy to understand due to the use of English, like in mnemonics.

The Drawbacks of Using Assembler

Assembly language is written for a certain instruction set and/or processor.

Assembly tends to be optimized for the hardware it is designed for, meaning it is often incompatible with different hardware.

Lots of assembly code is needed to do a relatively simple task, and complex programs require lots of programming time.

Compiler

A compiler is a computer program that translates code written in a high-level language into a low-level language, machine code.

The most common reason for translating source code is to create an executable program (converting from high-level language into machine language).

Advantages of using a compiler

Below is a list of the advantages of using a compiler:

- Source code is not included; therefore, compiled code is more secure than interpreted code.
- tends to produce faster code and is better at interpreting source code.
- Because the program generates an executable file, it can be run without the need for the source code.

Disadvantages of using a compiler

Below is a list of the disadvantages of using a compiler:

- Before a final executable file can be created, object code must be generated; this can be a time-consuming process.
- The source code must be 100% correct for the executable file to be produced.

Interpreter

An interpreter program executes other programs directly, running through the program code and executing it line-by-line. As it analyses every line, an interpreter is slower than running compiled code, but it can take less time to interpret program code than to compile and then run it. This is very useful when prototyping and testing code.

Interpreters are written for multiple platforms; this means code written once can be immediately run on different systems without having to recompile for each. Examples of this include flash-based web programs that will run on your PC, Mac, gaming console, and mobile phone.

Advantages of using an interpreter

Here is a list of some of the main advantages of using an interpreter:

- easier to debug (check errors) than a compiler.
- It is easier to create multi-platform code, as each different platform would have an interpreter to run the same code.
- useful for prototyping software and testing basic program logic.

Disadvantages of using an interpreter

And here is the list of some of the main disadvantages of using an interpreter:

- Source code is required for the program to be executed, and this source code can be read, making it insecure.
- Due to the on-line translation method, interpreters are generally slower than compiled programs.