Advantage and disadvantage :

Advantages (only when it is properly built)
- it can be user-friendly and speed up the user's work.
- it can be more attractive for non-technical people.
- in general, it looks more professional (but this does not mean it is always the best solution).

Disadvantages
- when it is not properly built, it can be very difficult to work with.
- it generally requires more memory resources than a non-graphical one.
- it might require the installation of additional software, e.g., the "runtime environment" in the case of java.
- depending on the programmer, it might require more time to be implemented.

WHAT IS IDE

An integrated development environment (IDE) or interactive development environment is a software application that provides comprehensive facilities to computer programmers for software development. An IDEnormally consists of a source code editor, build automation tools and a debugger.

PRINCIPLES OF USER INTERFACE DESIGN :

The **principles of user interface design** are intended to improve the quality of user interface design. According to Larry Constantine and Lucy Lockwood in their usage-centered design, these principles are:

- *The structure principle*: Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.

- *The simplicity principle*: The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

- *The visibility principle*: The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse with unneeded information.

- *The feedback principle*: The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

- *The tolerance principle*: The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

- *The reuse principle*: The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

According to Jef Raskin in his book *The Humane Interface*, there are two laws of user interface design, based on the fictional laws of robotics created by Isaac Asimov:

- *First Law*: A computer shall not harm our work or, through inactivity, allow our work to come to harm.
- *Second Law*: A computer shall not waste our time or require we to do more work than is strictly necessary.

Jef Raskin also mentions that "users should set the pace of an interaction," meaning that a user should not be kept waiting unnecessarily.

EVENT-DRIVEN PROGRAMMING :

In computer programming, **event-driven programming** is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads. Event-driven programming is the dominant paradigm used in graphical user interfaces and other applications (e.g. JavaScript web applications) that are centered on performing certain actions in response touser input.

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected. In embedded systems the same may be achieved using hardware interrupts instead of a constantly running main loop. Event-driven programs can be written in any programming language, although the task is easier in languages that provide high-level abstractions, such as closures.

ARRAY :

An array is a set of values that are logically related to each other, such as the number of students in each grade in a grammar school.
By using an array, we can refer to these related values by the same name, and use a number that's called an index or subscript to tell them apart. The individual values are called the elements of the array. They're contiguous from index 0 through the highest index value.
In contrast to an array, a variable that contain a single value is called a *scalar* variable.

The following example declares an array variable to hold the number of students in each grade in a grammar school.
VB

```
Dim students(6) As Integer
```

The array `students` in the preceding example contains seven elements. The indexes of the elements range from 0 through 6. Having this array is simpler than declaring seven variables.
The following illustration shows the array `students`. For each element of the array:

- The index of the element represents the grade (index 0 represents kindergarten).
- The value that's contained in the element represents the number of students in that grade.

We can define the size of an array several ways. We can supply the size when the array is declared, as the following example shows.

VB
```
Dim cargoWeights(10) As Double
Dim atmospherePressures(2, 2, 4, 10) As Short
Dim inquiriesByYearMonthDay(20)()() As Byte
```
We can also use a **New** clause to supply the size of an array when it's created, as the following example shows.
VB
```
cargoWeights = New Double(10) {}
atmospherePressures = New Short(2, 2, 4, 10) {}
inquiriesByYearMonthDay = New Byte(20)()() {}
```

If we have an existing array, we can redefine its size by using the **Redim** statement. We can specify that the **Redim** statement should keep the values that are in the array, or we can specify that it create an empty array. The following example shows different uses of the **Redim** statement to modify the size of an existing array.

Error Types (Visual Basic)

In Visual Basic, errors (also called *exceptions*) fall into one of three categories: syntax errors, run-time errors, and logic errors.

Syntax Errors

Syntax errors are those that appear while we write code. Visual Basic checks our code as we type it in the **Code Editor** window and alerts we if we make a mistake, such as misspelling a word or using a language element improperly. Syntax errors are the most common type of errors. We can fix them easily in the coding environment as soon as they occur.

Run-Time Errors

Run-time errors are those that appear only after we compile and run our code. These involve code that may appear to be correct in that it has no syntax errors, but that will not execute. For example, we might correctly write a line of code to open a file. But if the file is corrupted, the application cannot carry out the **Open** function, and it stops running. We can fix most run-time errors by rewriting the faulty code, and then recompiling and rerunning it.

Logic Errors

Logic errors are those that appear once the application is in use. They are most often unwanted or unexpected results in response to user actions. For example, a mistyped key or other outside influence might cause our application to stop working within expected parameters, or altogether. Logic errors are generally the hardest type to fix, since it is not always clear where they originate.

FUNCTION :

A **Function** procedure is a series of Visual Basic statements enclosed by the **Function** and **End Function** statements. The **Function** procedure performs a task and then returns control to the calling code. When it returns control, it also returns a value to the calling code.

Each time the procedure is called, its statements run, starting with the first executable statement after the **Function** statement and ending with the first **End Function**, **Exit Function**, or **Return**statement encountered.

A **Function** procedure in a module, class, or structure. It is **Public** by default, which means we can call it from anywhere in our application that has access to the module, class, or structure in which we defined it.
A **Function** procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

DATA TYPES :

The following table shows the Visual Basic data types, their supporting common language runtime types, their nominal storage allocation, and their value ranges.

Visual Basic type	Common language runtime type structure	Nominal storage allocation
Boolean	Boolean	Depends on implementing platform
Byte	Byte	1 byte
Char(single character)	Char	2 bytes
Date	DateTime	8 bytes
Decimal	Decimal	16 bytes
Double(double-precision floating-point)	Double	8 bytes
Integer	Int32	4 bytes
Long (long integer)	Int64	8 bytes
Object	Object(class)	4 bytes on 32-bit platform 8 bytes on 64-bit platform
SByte	SByte	1 byte
Short(short integer)	Int16	2 bytes
Single(single-precision floating-point)	Single	4 bytes
String(variable-length)	String(class)	Depends on implementing platform
UInteger	UInt32	4 bytes

ULong	UInt64	8 bytes
User-Defined(structure)	(inherits fromValueType)	Depends on implementing platform
UShort	UInt16	2 bytes

ARRAY :

An array is a set of values that are logically related to each other, such as the number of students in each grade in a grammar school.
By using an array, we can refer to these related values by the same name, and use a number that's called an index or subscript to tell them apart. The individual values are called the elements of the array. They're contiguous from index 0 through the highest index value.
In contrast to an array, a variable that contain a single value is called a *scalar* variable.

The following example declares an array variable to hold the number of students in each grade in a grammar school.
VB
```
Dim students(6) As Integer
```
The array `students` in the preceding example contains seven elements. The indexes of the elements range from 0 through 6. Having this array is simpler than declaring seven variables.
The following illustration shows the array `students`. For each element of the array:

- The index of the element represents the grade (index 0 represents kindergarten).
- The value that's contained in the element represents the number of students in that grade.

We can define the size of an array several ways. We can supply the size when the array is declared, as the following example shows.

VB
```
Dim cargoWeights(10) As Double
Dim atmospherePressures(2, 2, 4, 10) As Short
Dim inquiriesByYearMonthDay(20)()() As Byte
```
We can also use a **New** clause to supply the size of an array when it's created, as the following example shows.
VB
```
cargoWeights = New Double(10) {}
atmospherePressures = New Short(2, 2, 4, 10) {}
inquiriesByYearMonthDay = New Byte(20)()() {}
```

If we have an existing array, we can redefine its size by using the **Redim** statement. We can specify that the **Redim** statement should keep the values that are in the array, or we can specify that it create an empty array. The following example shows different uses of the **Redim** statement to modify the size of an existing array.

Introduction to Visual Basic Controls

The Visual Basic toolbox contains the tools we use to draw controls on our forms.

Figure 7.1 The Visual Basic toolbox

Control Categories

There are three broad categories of controls in Visual Basic:

- *Intrinsic controls*, such as the command button and frame controls. These controls are contained inside the Visual Basic .exe file. Intrinsic controls are always included in the toolbox, unlike ActiveX controls and insertable objects, which can be removed from or added to the toolbox.

- *ActiveX controls,* which exist as separate files with a .ocx file name extension. These include controls that are available in all editions of Visual Basic (DataCombo, DataList controls, and so on) and those that are available only in the Professional and Enterprise editions (such as Listview, Toolbar, Animation, and Tabbed Dialog). Many third-party ActiveX controls are also available.

  Note Controls with the .vbx file name extension use older technology and are found in applications written in earlier versions of Visual Basic. When Visual Basic opens a project containing a .vbx control, the default behavior is to replace the .vbx control with an .ocx control, but only if an .ocx version of the control is available. See "Updating Older Versions of Visual Basic Controls" later in this chapter for information on updating controls to the .ocx format.
- *Insertable Objects*, such as a Microsoft Excel Worksheet object containing a list of all our company's employees, or a Microsoft Project Calendar object containing the scheduling information for a project. Since these can be added to the toolbox, they can be considered controls. Some of these objects also support Automation (formerly called OLE Automation), which allows we to program another application's objects from within a Visual Basic application. See "Programming with Objects," for more information on Automation.

For More Information See "Using ActiveX Controls" in the *Component Tools Guide* for more information about the ActiveX controls available in the Professional and Enterprise versions of Visual Basic.

Basic control :

	Introduction to Visual Basic Controls Introduces we to the standard controls available in Visual Basic.
	Validating Control Data By Restricting Focus Explains how to use the Validate event with the CausesValidation property.
	Working with Control Arrays Describes control arrays and how they are used in Visual Basic.
	Using the ADO Data Control Use the ADO Data Control to create simple database applications without writing any code.
	Using the Check Box Control Use check box controls in groups to display multiple choices from which the user can select one or more.
	Using the Combo Box Control Use the combined features of a text box and a list box to allow the user to select an item either by typing text into the combo box, or by selecting it from the list.
	Using the Command Button Control Use the command button control to begin, interrupt, or end a process. When clicked, it invokes the command written into its Click event procedure.
	Using the Common Dialog Control Use the common dialog control to provide a standard set of dialog boxes for operations such as opening and saving files, setting print options, and selecting colors and fonts.
	Using the Data Control Use the intrinsic Data control to connect to desktop databases, such as Access.

	Using the DataCombo and DataList Controls Use the DataCombo and DataList controls with the ADO Data Control to display information from a database.
	Using the DataGrid Control Use the DataGrid control to create an application that allows the end user to read and write to most databases.
	Using the File-System Controls (Directory List Box, Drive List Box, and File List Box) Use the drive list box, the directory list box, and the file list box controls to allow users of our applications to explore the file system.
	Using the Frame Control Use the frame control to provide an identifiable grouping for other controls.
	Using the Microsoft Hierarchical FlexGrid Control Use the hierarchical flexgrid control for displaying data in a grid.
	Using the Horizontal and Vertical Scroll Bar Controls Use the horizontal and vertical scroll bar controls to provide scrolling to applications or controls when they are not provided automatically.
	Using the Image Control Use the image control to display graphics in the following formats: bitmap, icon, metafile, enhanced metafile, JPEG, or GIF files.
	Using the Label Control Use the label control to display text that cannot be edited by the user or to provide a description of what a certain control will do if clicked.
	Using the Line Control Use the line control to create simple line segments on a form, a frame, or in a picture box.
	Using the List Box Control Use the list box control to display a list of items from which the user can select one or more.

	Using the OLE Container Control Use the OLE container control to display and manipulate data from other Windows–based applications, such as Microsoft Excel and Microsoft Word for Windows.
	Using the Option Button Control Use the option button control to display options in option button groups from which the user can choose one.
	Using the Picture Box Control Use the picture box control to display graphics, to act as a container for other controls, and to display output from graphics methods or text using the Print method.
	Using the RemoteData Control Use the RemoteData Control to connect to large scale databases.
	Using the Shape Control Use the Shape control to draw predefined shapes on forms, frames, or picture boxes.
	Using the Text Box Control Use the text box control to display information entered by the user at run time, or assigned to the Text property of the control at design or run time.
	Using the Timer Control Use the timer control to execute timer events at specified time intervals.

Error Types (Visual Basic)

In Visual Basic, errors (also called *exceptions*) fall into one of three categories: syntax errors, run-time errors, and logic errors.

Syntax Errors

Syntax errors are those that appear while we write code. Visual Basic checks our code as we type it in the **Code Editor** window and alerts we if we make a mistake, such as misspelling a word or using a language element improperly. Syntax errors are the most common type of errors. We can fix them easily in the coding environment as soon as they occur.

Run-Time Errors

Run-time errors are those that appear only after we compile and run our code. These involve code that may appear to be correct in that it has no syntax errors, but that will not execute. For example, we might correctly write a line of code to open a file. But if the file is corrupted, the application cannot carry out the **Open** function, and it stops running. We can fix most run-time errors by rewriting the faulty code, and then recompiling and rerunning it.

Logic Errors

Logic errors are those that appear once the application is in use. They are most often unwanted or unexpected results in response to user actions. For example, a mistyped key or other outside influence might cause our application to stop working within expected parameters, or altogether. Logic errors are generally the hardest type to fix, since it is not always clear where they originate.

FUNCTION :

A **Function** procedure is a series of Visual Basic statements enclosed by the **Function** and **End Function** statements. The **Function** procedure performs a task and then returns control to the calling code. When it returns control, it also returns a value to the calling code.
Each time the procedure is called, its statements run, starting with the first executable statement after the **Function** statement and ending with the first **End Function**, **Exit Function**, or **Return**statement encountered.

A **Function** procedure in a module, class, or structure. It is **Public** by default, which means we can call it from anywhere in our application that has access to the module, class, or structure in which we defined it.
A **Function** procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.