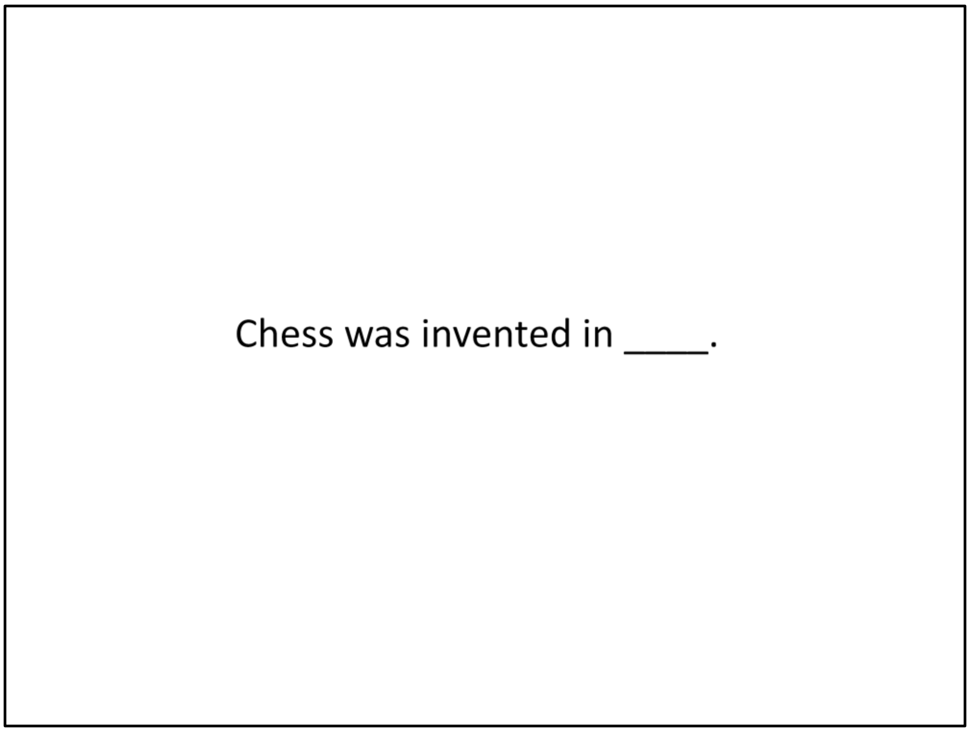
Introduction to WDDN
Dot Net Framework

Prepared for Vth semester DDU-CE students
2022-23 WAD

Apurva A Mehta

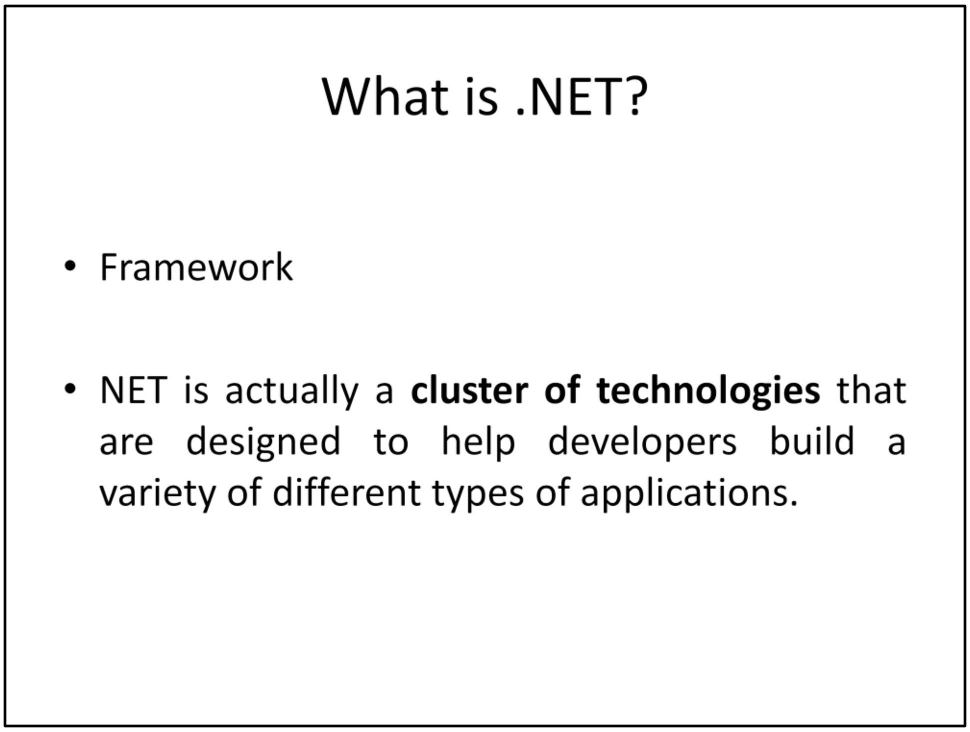


Chess was invented in _____.

Chess was invented in India.

A Big Bonanza...

WAD in 2022-23
.NET Framework and C# language
ASP.NET Web applications including Session, Database (ADO.NET) and Security management
.NET Core
.NET Core MVC
EF Core in .NET Core MVC
Security, TDD and Deployment in .NET Core MVC



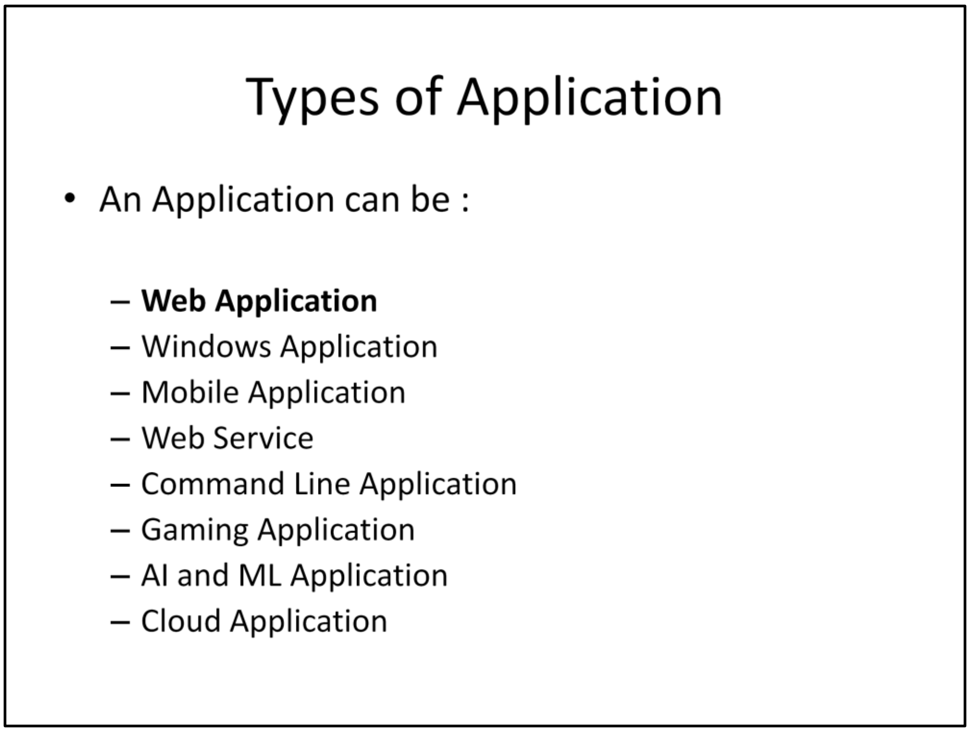
What is .NET?

- Framework
- NET is actually a **cluster of technologies** that are designed to help developers build a variety of different types of applications.

API
a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.

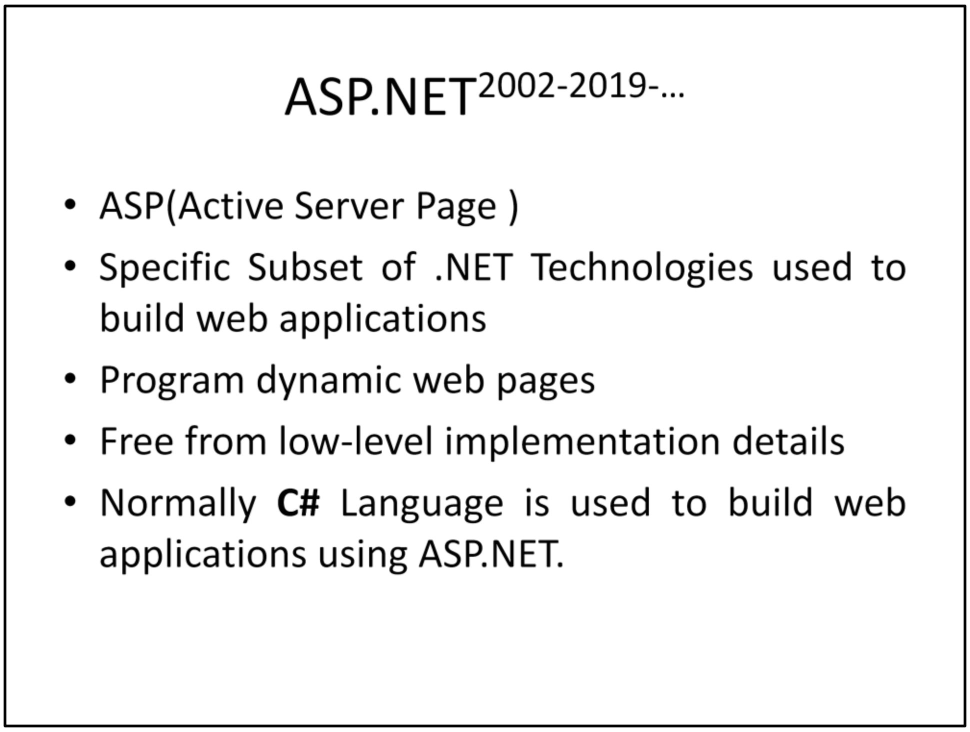
In computer programming, a software framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific software. A software framework provides a standard way to build and deploy applications.

With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, gaming, and IoT.



Types of Application

- An Application can be :
  - **Web Application**
  - Windows Application
  - Mobile Application
  - Web Service
  - Command Line Application
  - Gaming Application
  - AI and ML Application
  - Cloud Application

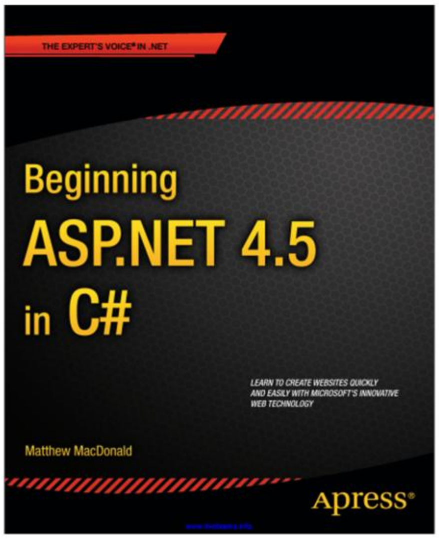


ASP.NET$^{2002-2019-\ldots}$

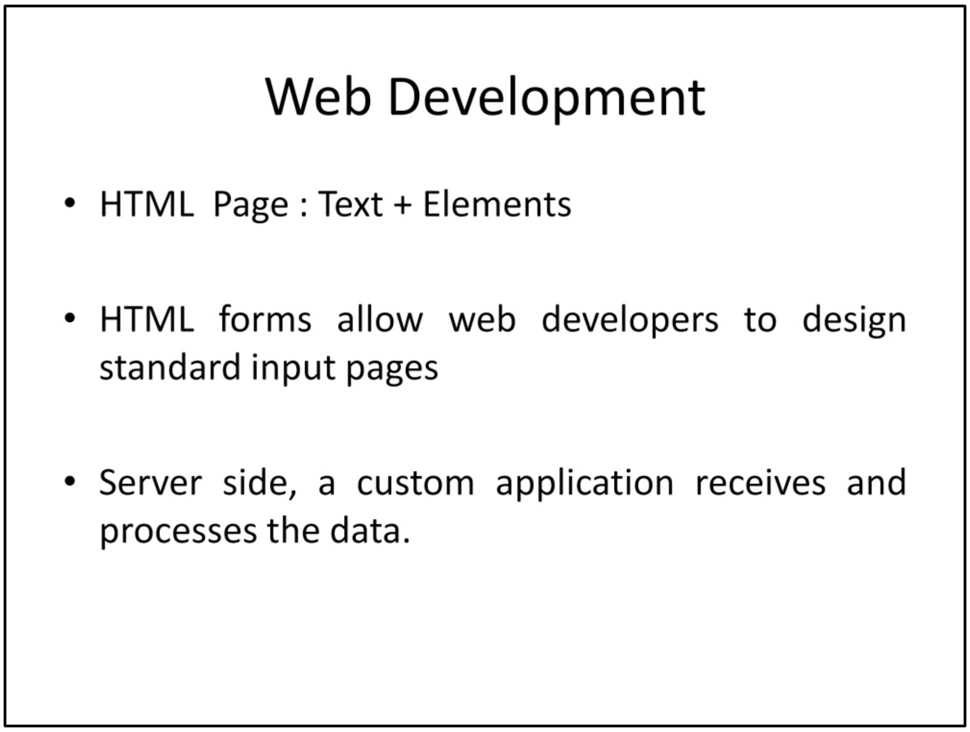
- ASP(Active Server Page)
- Specific Subset of .NET Technologies used to build web applications
- Program dynamic web pages
- Free from low-level implementation details
- Normally **C#** Language is used to build web applications using ASP.NET.

To counter these problems, Microsoft created higher-level development platforms—first ASP and then
ASP.NET. These technologies allow developers to program dynamic web pages without worrying about the
low-level implementation details. Even better, ASP.NET is stuffed full of sophisticated features, including tools
for implementing security, managing data, storing user-specific information, and much more. And amazingly
enough, it's even possible to program an ASP.NET page without knowing anything about HTML (although a little
bit of HTML smarts will help you build your pages more quickly and effectively).

Text Book


THE EXPERT'S VOICE® IN .NET
Beginning
ASP.NET 4.5
in C#
LEARN TO CREATE WEBSITES QUICKLY AND EASILY WITH MICROSOFT'S INNOVATIVE WEB TECHNOLOGY
Matthew MacDonald
Apress®

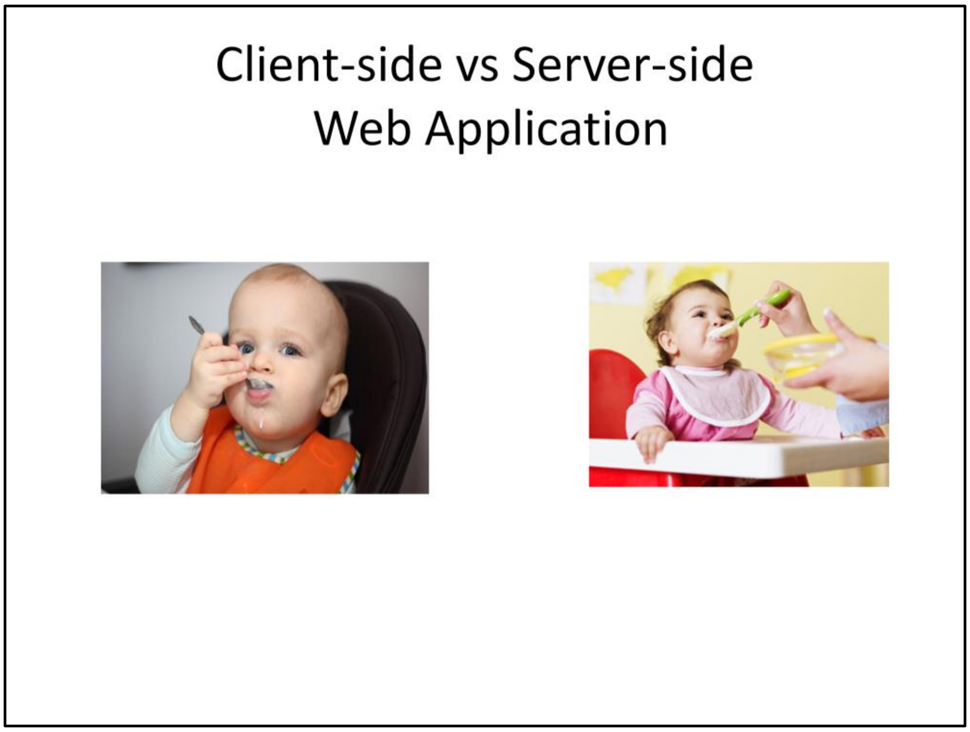
Author : Matthew MacDonald



Web Development

- HTML Page : Text + Elements
- HTML forms allow web developers to design standard input pages
- Server side, a custom application receives and processes the data.

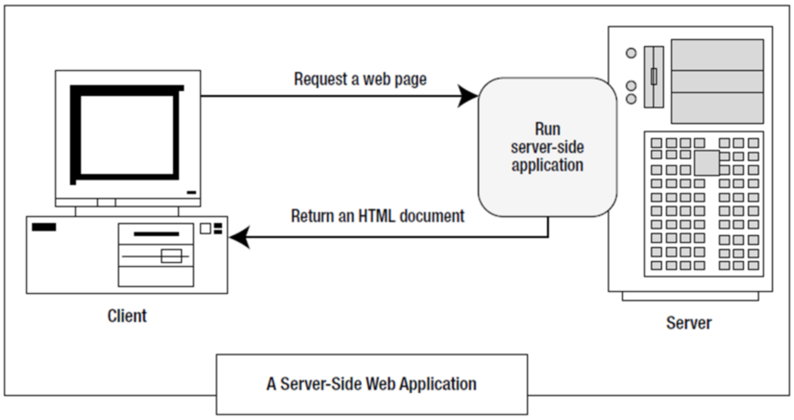
An HTML document has two types of content: the text and the elements (or tags) that tell the browser how to format it



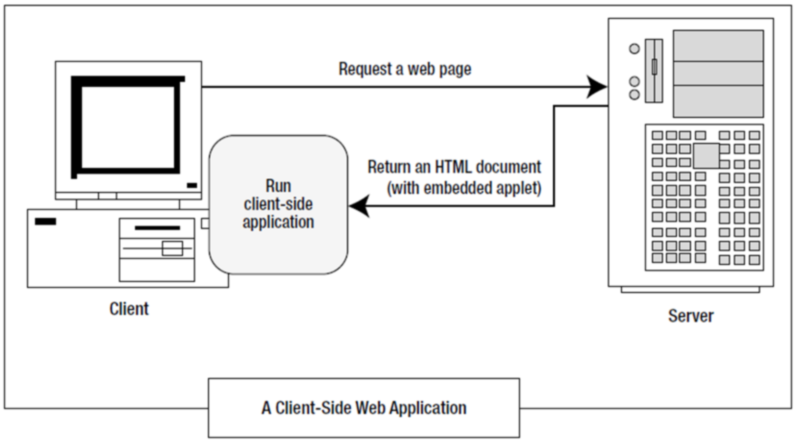
Client-side vs Server-side Web Application

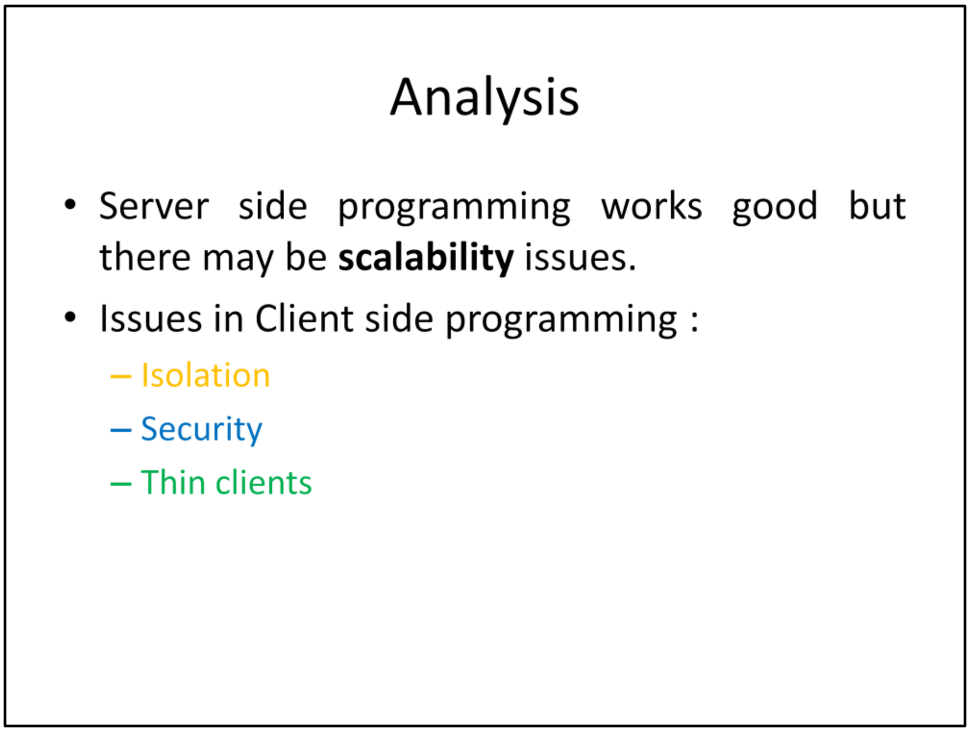
ASP.NET is designed first and foremost as a *server-side programming platform. That means that all ASP.NET*
code runs on the web server. When the ASP.NET code finishes running, the web server sends the user the final
result—an ordinary HTML page that can be viewed in any browser.
Server-side programming isn't the only way to make an interactive web page. Another option is *client-side*
programming, which asks the browser to download the code and execute it locally, on the client's computer. Just
as there are a variety of server-side programming platforms, there are also various ways to perform client-side
programming, from snippets of JavaScript code that can be embedded right inside the HTML of a web page, to
plug-ins such as Adobe Flash and Microsoft Silverlight.

Server-side web application



Client-side web application


Request a web page
Run client-side application
Return an HTML document (with embedded applet)
Client
Server
A Client-Side Web Application

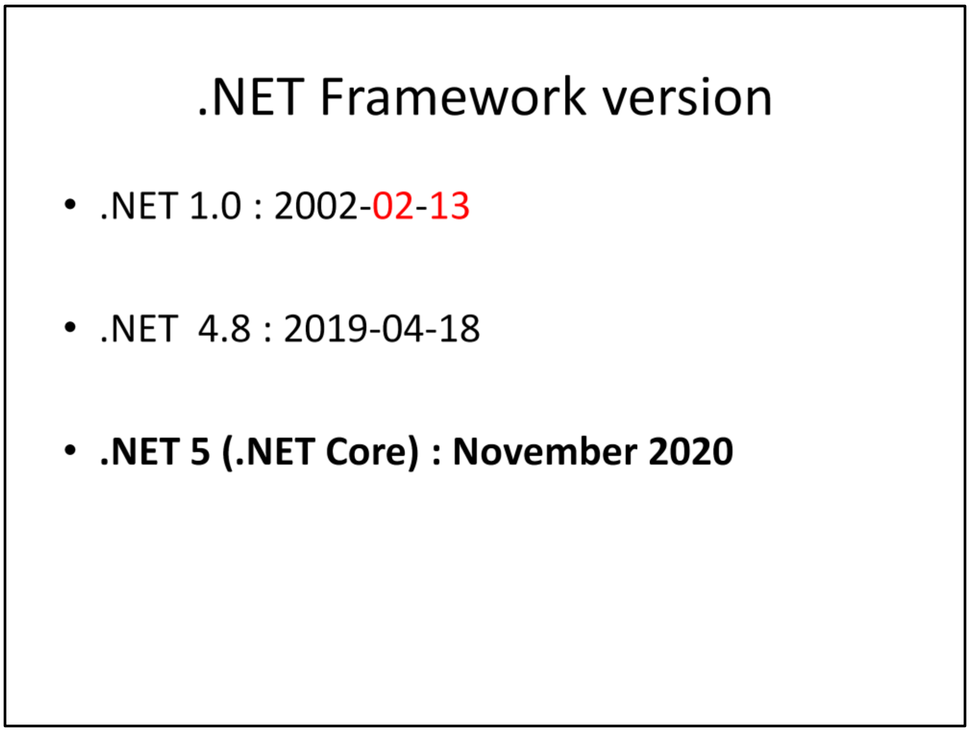


Analysis

- Server side programming works good but there may be **scalability** issues.
- Issues in Client side programming :
  - Isolation
  - Security
  - Thin clients

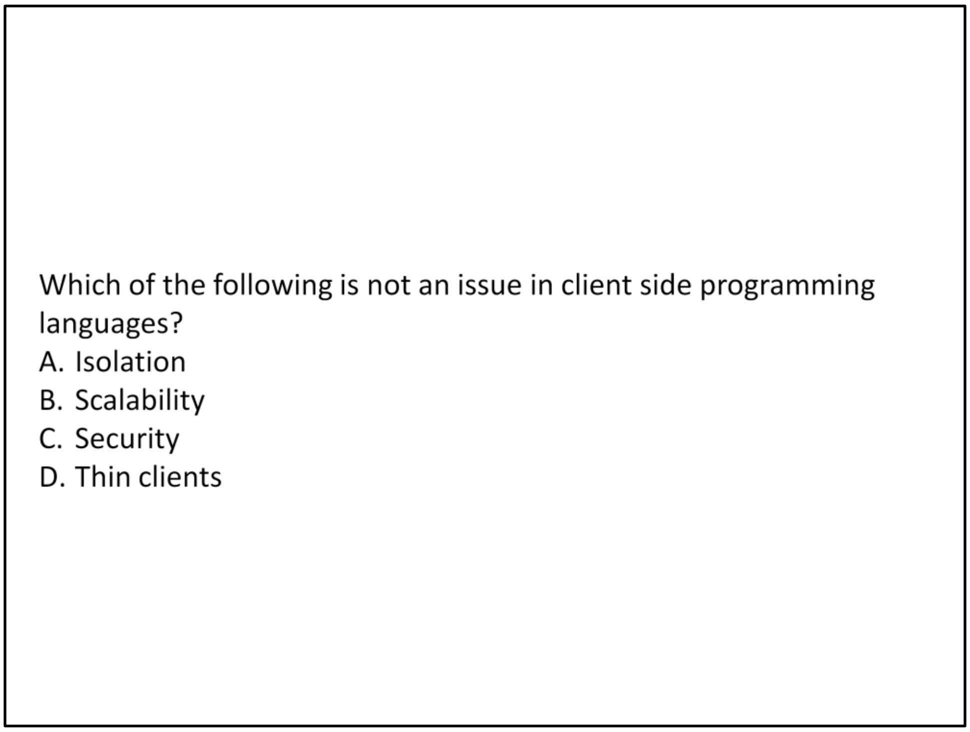
ASP.NET uses server-side programming to avoid several problems:
Isolation: Client-side code can't access server-side resources. For example, a clientside
application has no easy way to read a file or interact with a database on the server
(at least not without running into problems with security and browser compatibility).
Security: End users can view client-side code. And once malicious users understand
how an application works, they can often tamper with it.
Thin clients: In today's world, web-enabled devices such as tablets and smartphones
are everywhere. These devices usually have some sort of built-in web browsing
ability, but they may not support client-side programming platforms such as Flash or
Silverlight.

In recent years, there's been a renaissance in client programming, particularly with JavaScript. Nowadays
developers create client-side applications that communicate with a web server to fetch information and perform
tasks that wouldn't be possible if the applications were limited to the local computer. Fortunately, ASP.NET takes
advantage of this change in two ways:
JavaScript frills: In some cases, ASP.NET allows you to combine the best of client-side
programming with server-side programming. For example, the best ASP.NET controls
can "intelligently" detect the features of the client browser. If the browser supports
JavaScript, these controls will return a web page that incorporates JavaScript for a
richer, more responsive user interface. You'll see a good example of this technique
with validation in Chapter 9.
ASP.NET's Ajax features: Ajax is a set of JavaScript techniques used to create fast,
responsive pages with dynamic content. In Chapter 25, you'll learn how ASP.NET lets
you benefit from many of the advantages of Ajax with none of the complexity.
However, it's important to understand one fundamental fact. No matter what the capabilities of the browser,
the C# code that you write is always executed on the server. The client-side frills are just the icing on the cake.


.NET Framework version

- .NET 1.0 : 2002-02-13
- .NET 4.8 : 2019-04-18
- **.NET 5 (.NET Core) : November 2020**

https://en.wikipedia.org/wiki/.NET_Framework_version_history

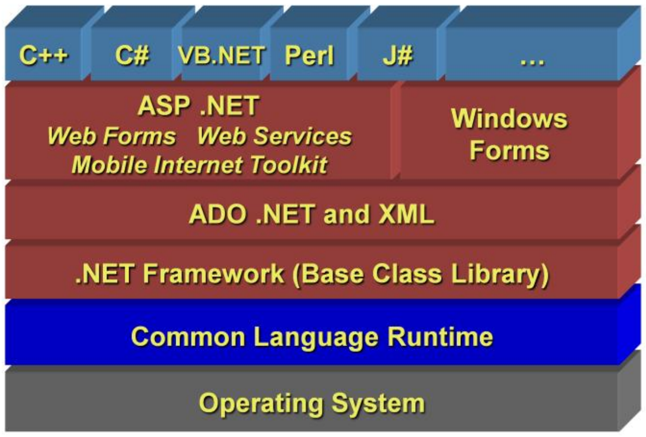


Which of the following is not an issue in client side programming languages?

A. Isolation
B. Scalability
C. Security
D. Thin clients

B

The .NET Framework



As you've already learned, the .NET Framework is really a cluster of several technologies:

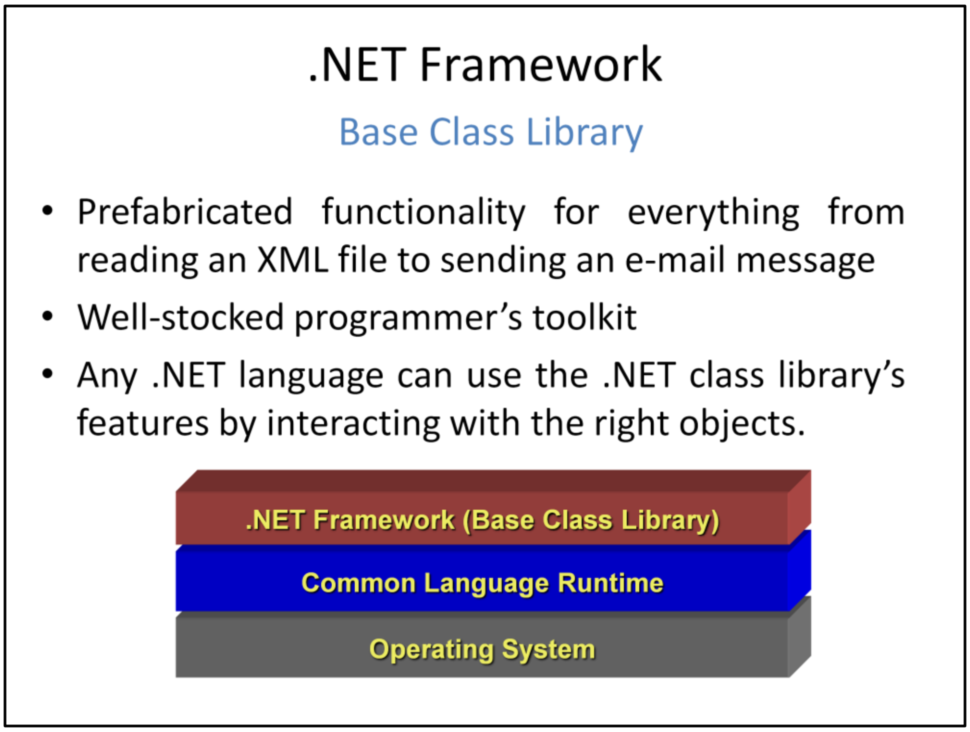
The .NET languages: These include Visual Basic, C#, F#, and C++, although third-party developers have created hundreds more.

The Common Language Runtime (CLR): This is the engine that executes all .NET programs and provides automatic services for these applications, such as security checking, memory management, and optimization.

The .NET Framework class library: The class library collects thousands of pieces of prebuilt functionality that you can "snap in" to your applications. These features are sometimes organized into technology sets, such as ADO.NET (the technology for creating database applications) and Windows Presentation Foundation (WPF, the technology for creating desktop user interfaces).

ASP.NET: This is the engine that hosts the web applications you create with .NET, and supports almost any feature from the .NET Framework class library. ASP.NET also includes a set of web-specific services, such as secure authentication and data storage.

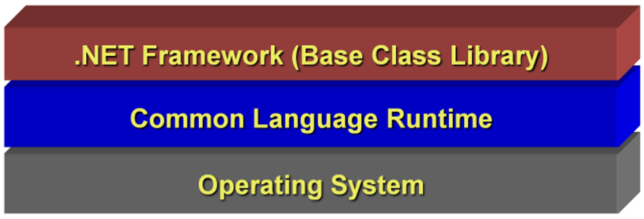
Visual Studio: This optional development tool contains a rich set of productivity and debugging features. Visual Studio includes the complete .NET Framework, so you won't need to download it separately.



.NET Framework

Base Class Library

- Prefabricated functionality for everything from reading an XML file to sending an e-mail message
- Well-stocked programmer’s toolkit
- Any .NET language can use the .NET class library’s features by interacting with the right objects.

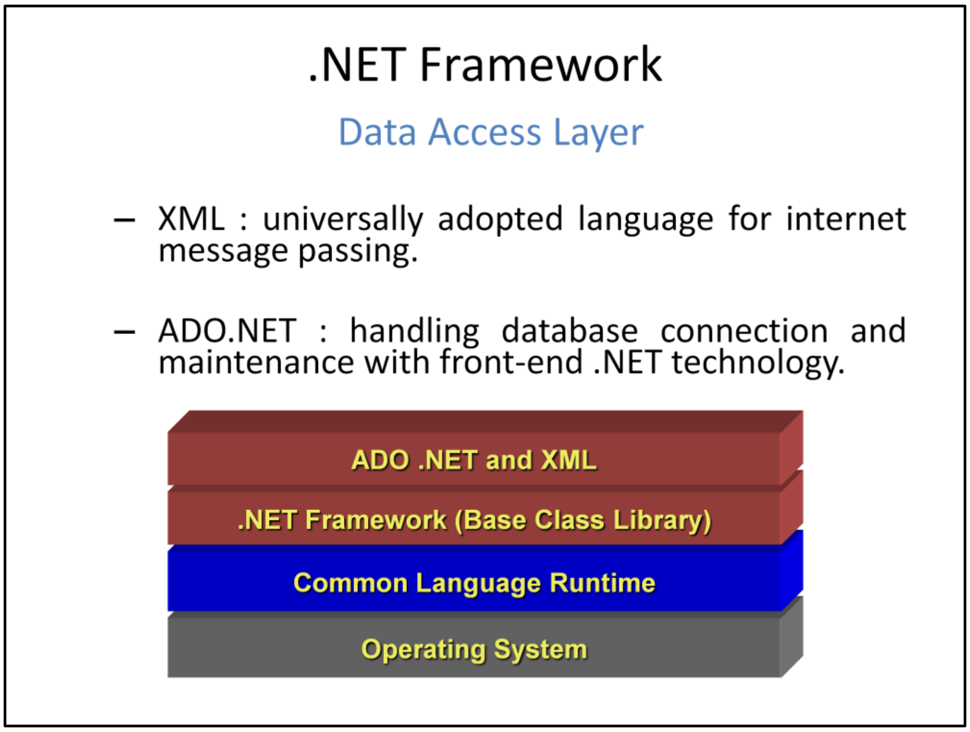


.NET supplies a library of base classes that we can use to implement applications quickly

String handling, IO operations, graphics, text handling, file operations and many more

The .NET class library is a giant repository of classes that provide prefabricated functionality for everything
from reading an XML file to sending an e-mail message. If you’ve had any exposure to Java, you may already be
familiar with the idea of a class library. However, the .NET class library is more ambitious and comprehensive
than just about any other programming framework. Any .NET language can use the .NET class library’s features
by interacting with the right objects. This helps encourage consistency among different .NET languages and
removes the need to install numerous components on your computer or web server.

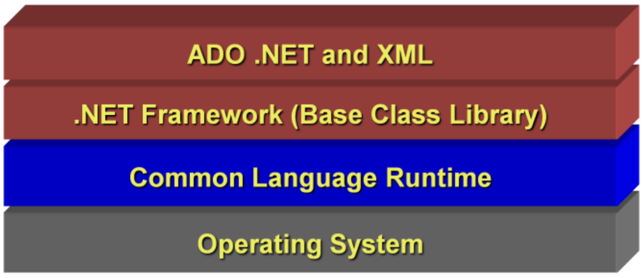
You can think of the class library as a well-stocked programmer’s toolkit. Microsoft’s philosophy is that it will
provide the tedious infrastructure so that application developers need only to write business-specific code. For
example, the .NET Framework deals with thorny issues such as database transactions and concurrency, making
sure that hundreds or thousands of simultaneous users can request the same web page at once. You just add the
logic needed for your specific application.



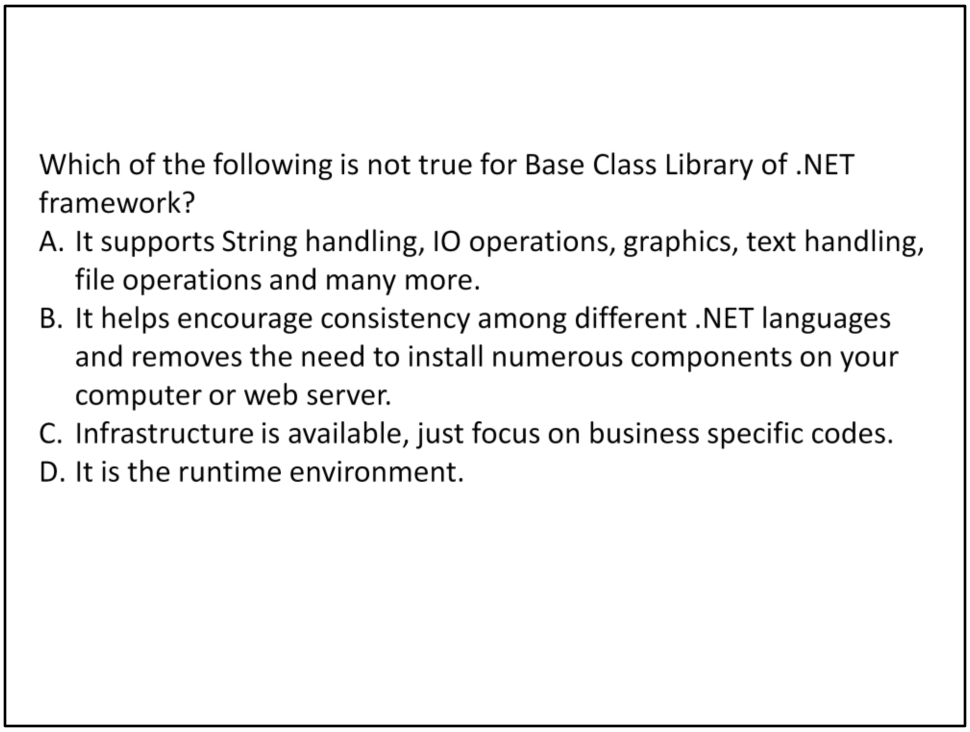
.NET Framework

Data Access Layer

- XML : universally adopted language for internet message passing.
- ADO.NET : handling database connection and maintenance with front-end .NET technology.



XML – Extended mark-up language, is a universally adopted language for internet message passing.
ADO.NET : provides class for handling database connection and maintenance with front-end .NET technology.



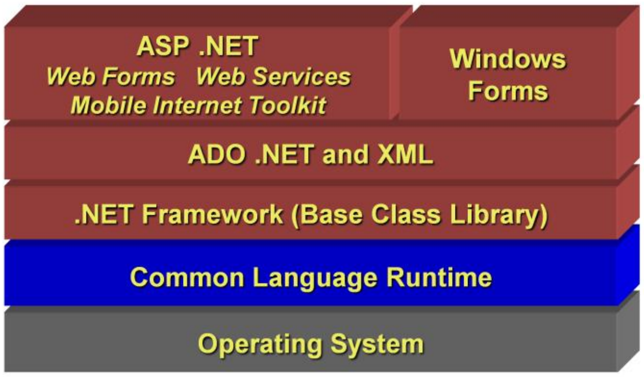
Which of the following is not true for Base Class Library of .NET framework?

A. It supports String handling, IO operations, graphics, text handling, file operations and many more.
B. It helps encourage consistency among different .NET languages and removes the need to install numerous components on your computer or web server.
C. Infrastructure is available, just focus on business specific codes.
D. It is the runtime environment.

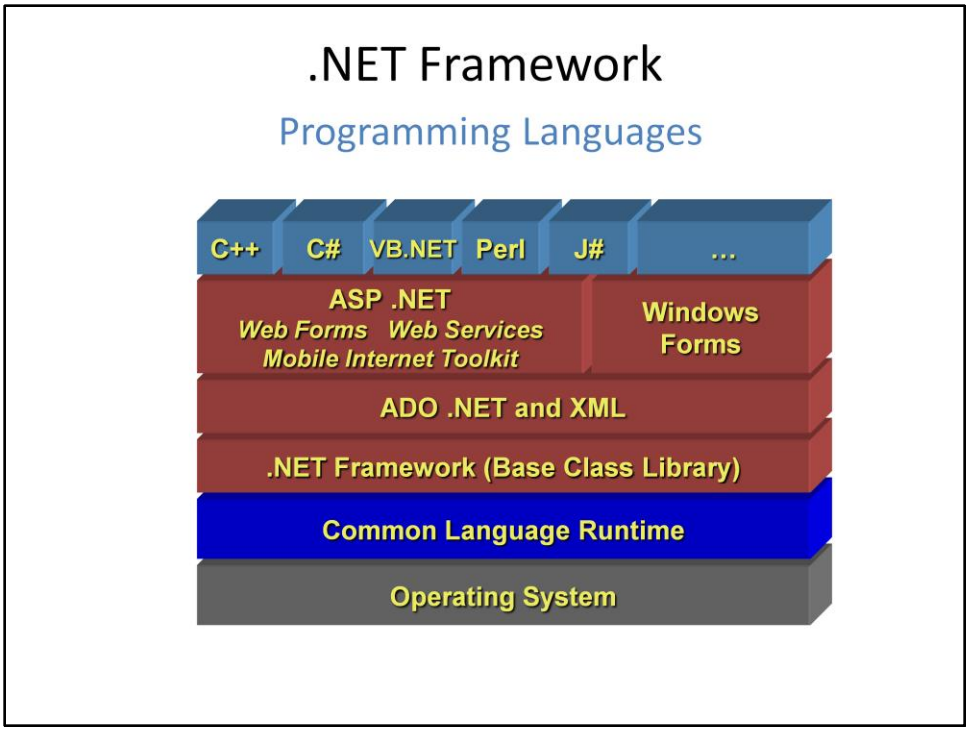
D

.NET Framework

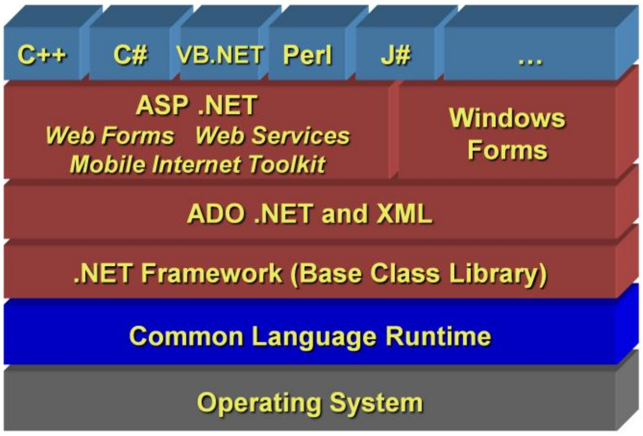
ASP.NET & Windows Forms



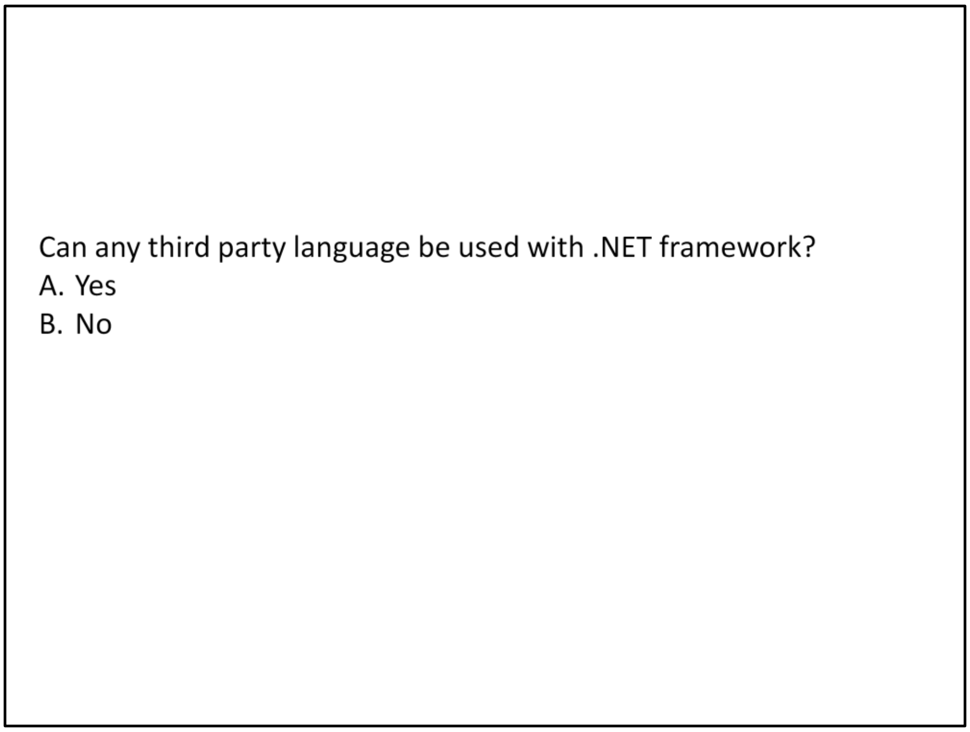
Create application’s front-end – Web-based user interface, Windows GUI, Web services, …


.NET Framework

Programming Languages



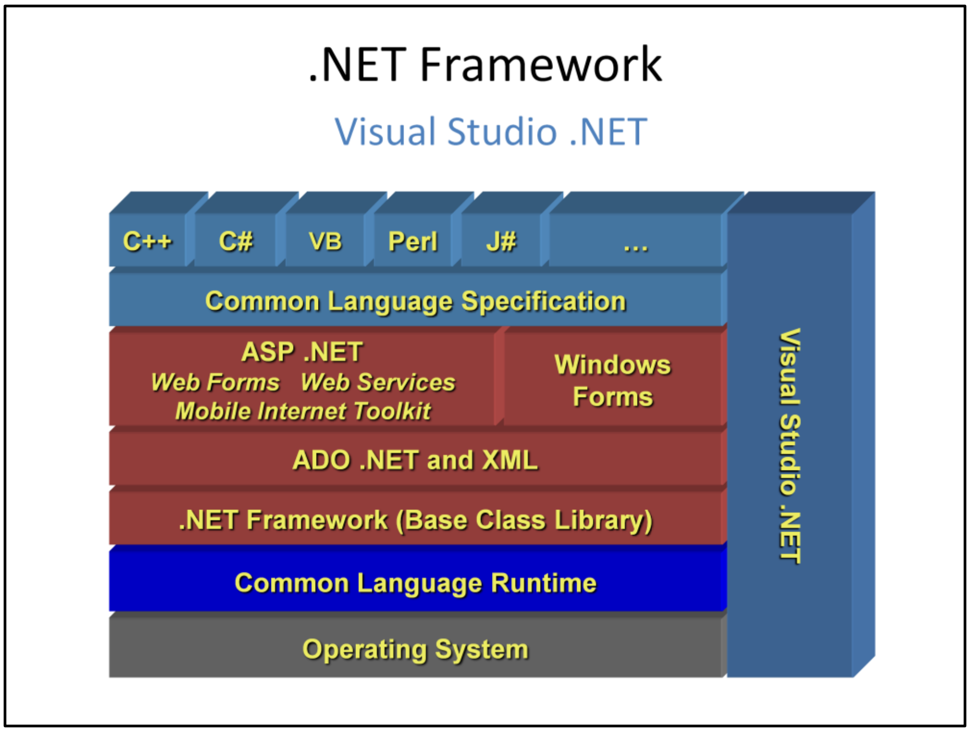
Use your favorite language



Can any third party language be used with .NET framework?

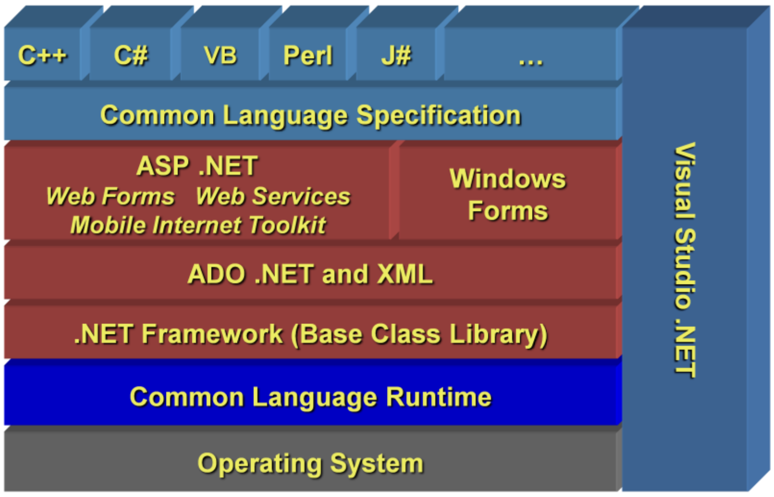
A. Yes
B. No

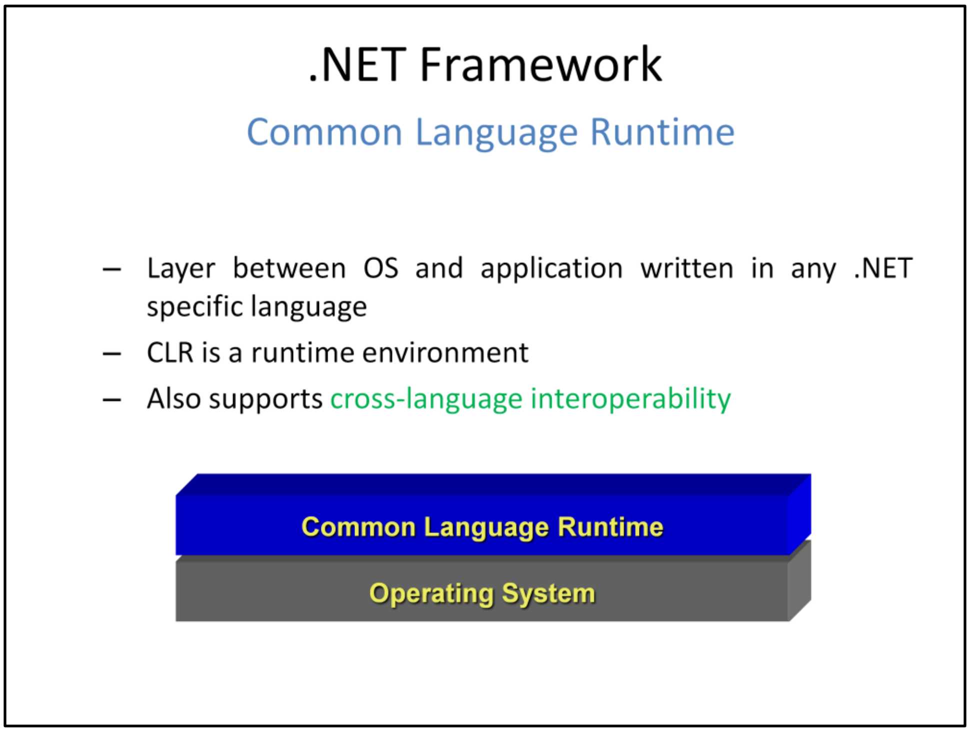
A
Yes, if language compiler targets CLR



.NET Framework

Visual Studio .NET





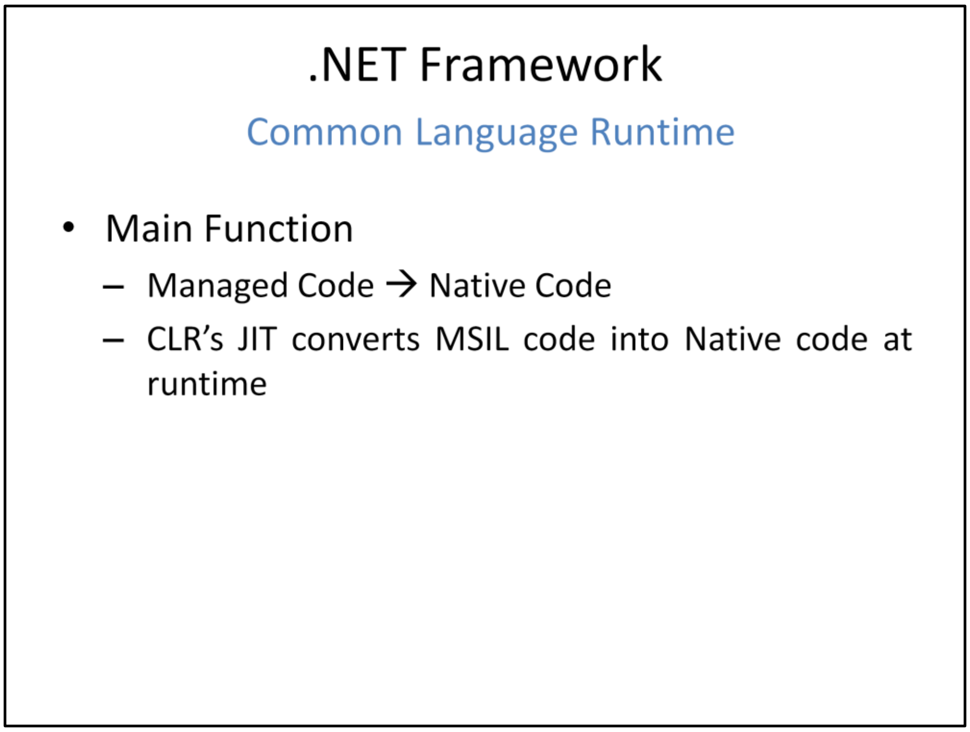
.NET Framework

Common Language Runtime

– Layer between OS and application written in any .NET specific language
– CLR is a runtime environment
– Also supports cross-language interoperability

Common Language Runtime
Operating System

CLR is the heart and soul of the .net framework.
CLR is a runtime environment in which programs written in C# and other .NET language are executed.
It also support s cross-language interoperability.



.NET Framework

Common Language Runtime

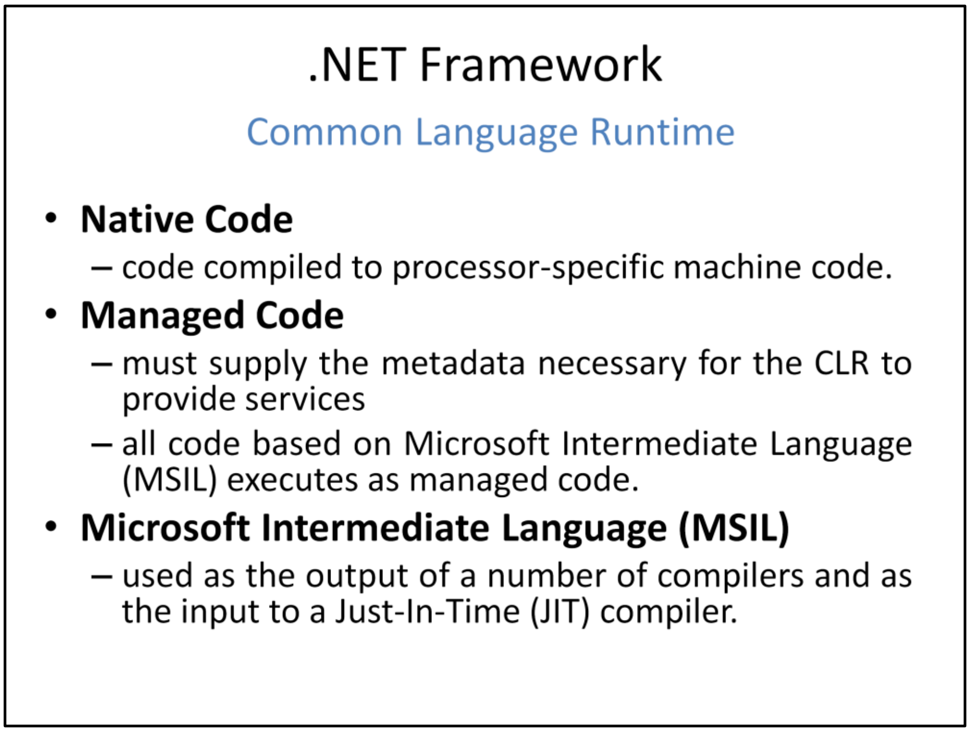
- Main Function
  - Managed Code → Native Code
  - CLR's JIT converts MSIL code into Native code at runtime

CLR is the heart and soul of the .net framework.
Managed code is a **code** whose execution is **managed** by Common Language Runtime. It gets the **managed code** and compiles it into machine **code**. After that, the **code** is executed. The runtime here i.e. CLR provides automatic memory management, type safety, etc

CLR is a runtime environment in which programs written in C# and other .NET language are executed.
It also support s cross-language interoperability.

MSIL generated through managed code

Native Code is code compiled to processor-specific machine code.
Managed Code runs under a "contract of cooperation" with the CLR and it must supply the metadata necessary for the CLR to provide services such as memory management, cross-language integration, Code Access Security and automatic lifetime control of objects. All code based on Microsoft Intermediate Language (MSIL) executes as managed code.
Microsoft Intermediate Language (MSIL) is used as the output of a number of compilers and as the input to a Just-In-Time (JIT) compiler. The CLR includes several JIT compilers for converting MSIL to native code.



.NET Framework

Common Language Runtime

- **Native Code**
  - code compiled to processor-specific machine code.
- **Managed Code**
  - must supply the metadata necessary for the CLR to provide services
  - all code based on Microsoft Intermediate Language (MSIL) executes as managed code.
- **Microsoft Intermediate Language (MSIL)**
  - used as the output of a number of compilers and as the input to a Just-In-Time (JIT) compiler.

CLR is the heart and soul of the .net framework.

CLR is a runtime environment in which programs written in C# and other .NET language are executed.

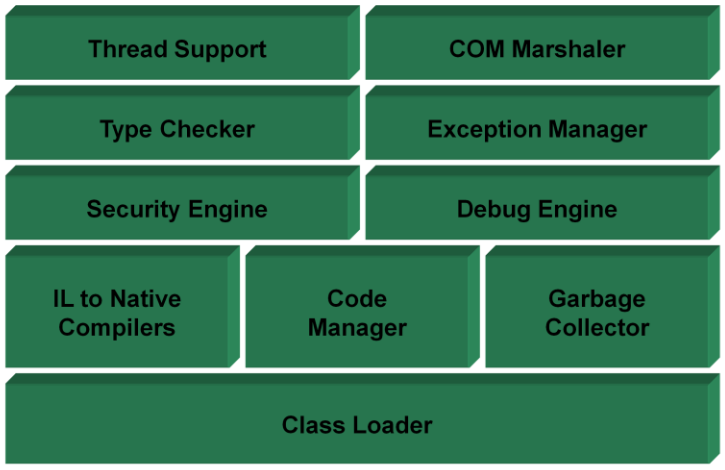
It also support s cross-language interoperability.

MSIL generated through managed code

Native Code is code compiled to processor-specific machine code.
Managed Code runs under a "contract of cooperation" with the CLR and it must supply the metadata necessary for the CLR to provide services such as memory management, cross-language integration, Code Access Security and automatic lifetime control of objects. All code based on Microsoft Intermediate Language (MSIL) executes as managed code.
Microsoft Intermediate Language (MSIL) is used as the output of a number of compilers and as the input to a Just-In-Time (JIT) compiler. The CLR includes several JIT compilers for converting MSIL to native code.

.NET Framework

Common Language Runtime



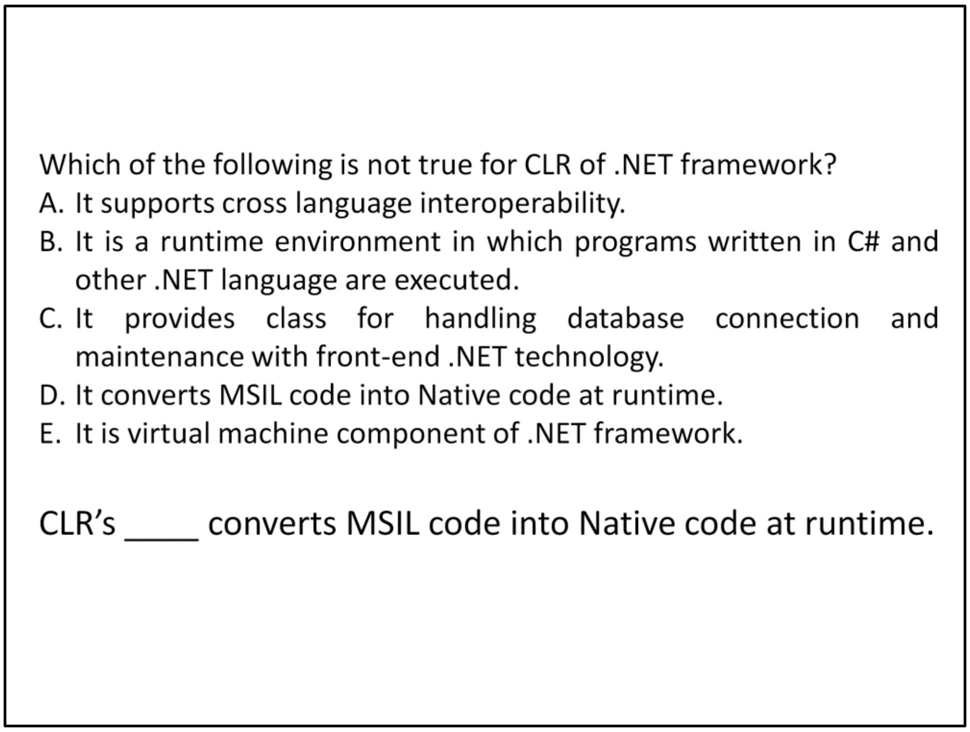
https://www.youtube.com/watch?v=KpkFaivsQ28
Marshalling (similar to serialization) is the process of transforming the memory representation of an object to a data format suitable for storage or transmission. It is typically used when data must be moved between different parts of a computer program or from one program to another

https://www.codeproject.com/Articles/1042196/Introduction-to-CLR-World
The Common Language Runtime
Features

The Common Language Runtime **(CLR)** is the execution engine for .NET Framework applications.
It provides a number of services including:
- Code management (loading and execution)
- Application memory isolation
- Verification of type safety
- Conversion of IL to native code
- Access to metadata (enhanced type information)
- Managing memory for managed objects
- Enforcement of Code Access Security
- Exception handling including cross-language exceptions
- Interoperation between managed code, COM objects and pre-existing DLLs (unmanaged code and data)
- Automation of object layout
- Support for developer services (profiling, debugging, etc.)

Native Code is code compiled to processor-specific machine code.
Managed Code runs under a "contract of cooperation" with the CLR and it must supply the metadata necessary for the CLR to provide services such as memory management, cross-language integration, Code Access Security and automatic lifetime control of objects. All code based on Microsoft Intermediate Language (MSIL) executes as managed code.
Microsoft Intermediate Language (MSIL) is used as the output of a number of compilers and as the input to a Just-In-Time (JIT) compiler. The CLR includes several JIT compilers for converting MSIL to native code.

Code Access Security (CAS), in the Microsoft .NET framework, is Microsoft's solution to prevent untrusted code from performing privileged actions .
When the CLR loads an assembly it will obtain evidence for the assembly and use this to identify the code group that the assembly belongs to.
A code group contains a permission set (one or more permission). Code that performs a privileged action will perform a code access demand.
Role Based Security
Enforcing security consists of two parts, Authentication and Authorization
Authentication : username & password/ anonymous access
Authorization : is after authentication, checking for grant or deny permission on particular set or portion of program.
Finds the security information associated with that user and based on this whether to grant or deny permission that is decided by application.

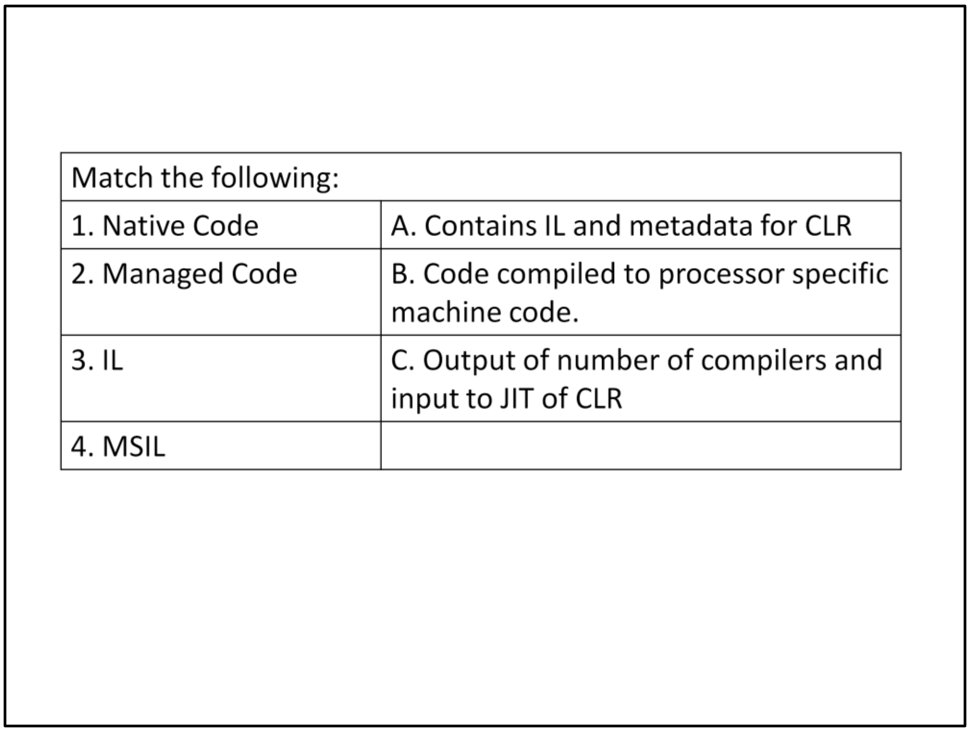


Which of the following is not true for CLR of .NET framework?

A. It supports cross language interoperability.
B. It is a runtime environment in which programs written in C# and other .NET language are executed.
C. It provides class for handling database connection and maintenance with front-end .NET technology.
D. It converts MSIL code into Native code at runtime.
E. It is virtual machine component of .NET framework.

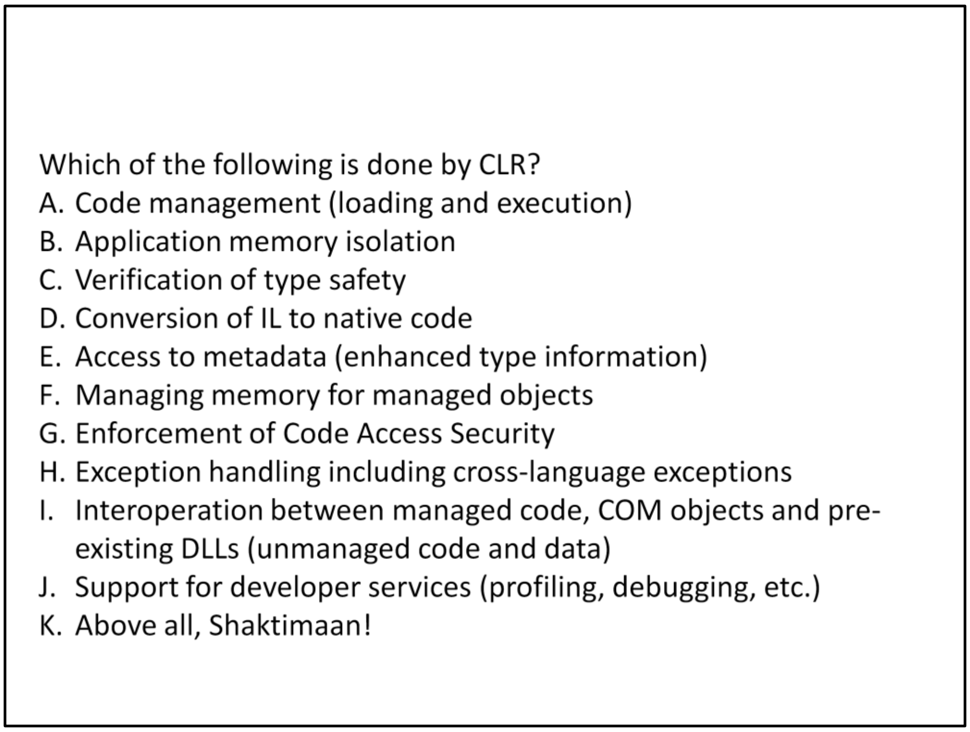
CLR’s ____ converts MSIL code into Native code at runtime.

C
JIT



Match the following:	
1. Native Code	A. Contains IL and metadata for CLR
2. Managed Code	B. Code compiled to processor specific machine code.
3. IL	C. Output of number of compilers and input to JIT of CLR
4. MSIL	

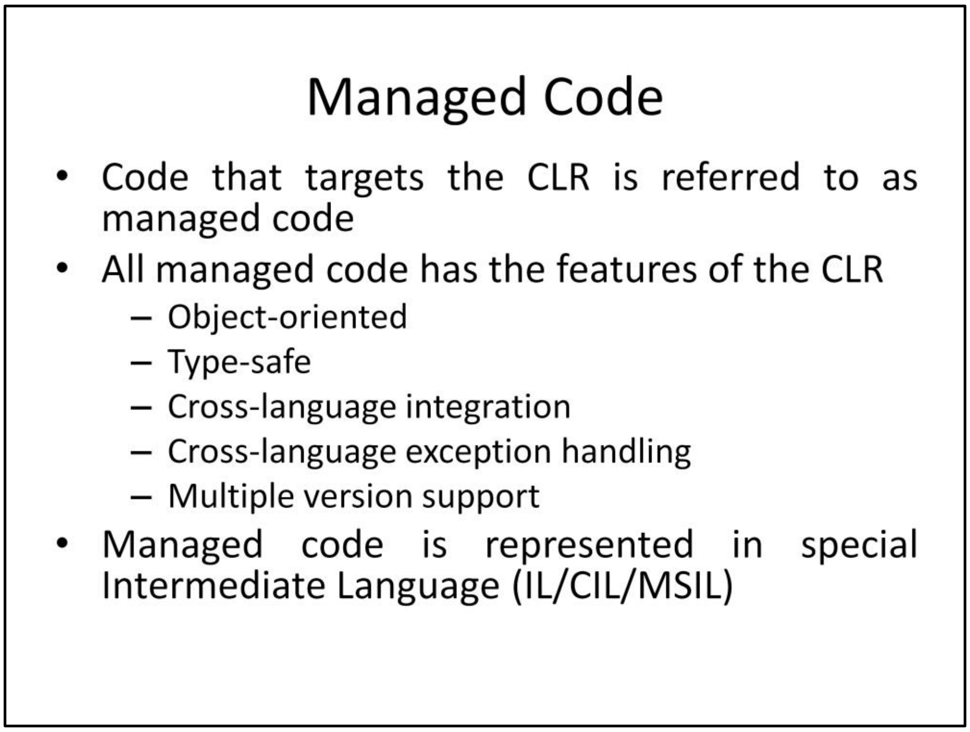
1 ←→B
2 ←→A
3, 4 ←→ C



Which of the following is done by CLR?

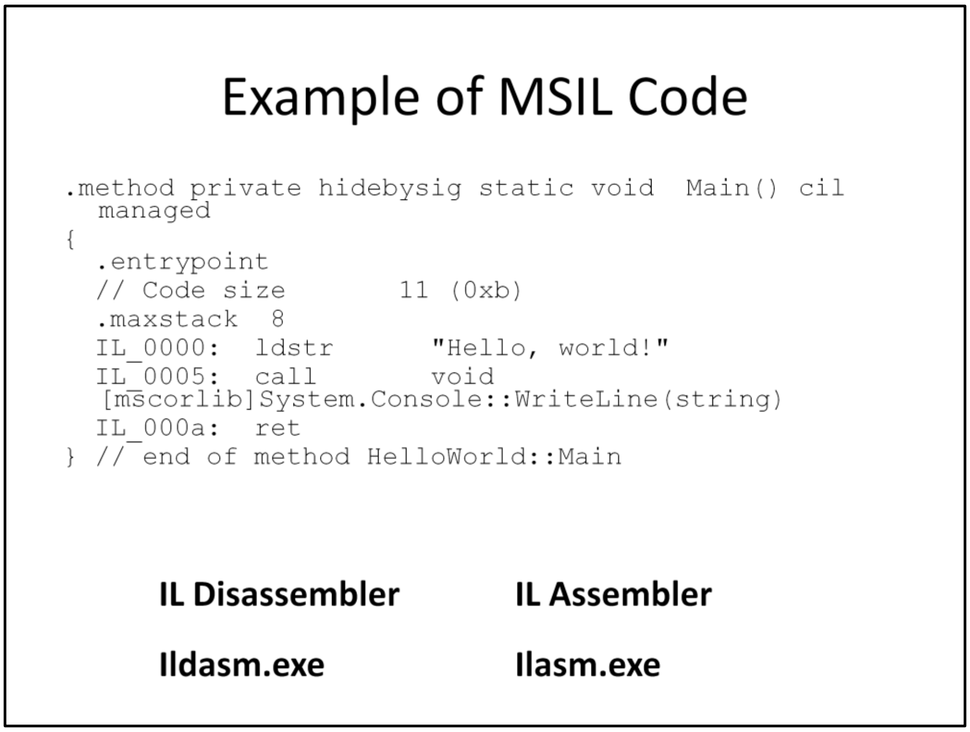
A. Code management (loading and execution)
B. Application memory isolation
C. Verification of type safety
D. Conversion of IL to native code
E. Access to metadata (enhanced type information)
F. Managing memory for managed objects
G. Enforcement of Code Access Security
H. Exception handling including cross-language exceptions
I. Interoperation between managed code, COM objects and pre-existing DLLs (unmanaged code and data)
J. Support for developer services (profiling, debugging, etc.)
K. Above all, Shaktimaan!

K



Managed Code

- Code that targets the CLR is referred to as managed code
- All managed code has the features of the CLR
  - Object-oriented
  - Type-safe
  - Cross-language integration
  - Cross-language exception handling
  - Multiple version support
- Managed code is represented in special Intermediate Language (IL/CIL/MSIL)



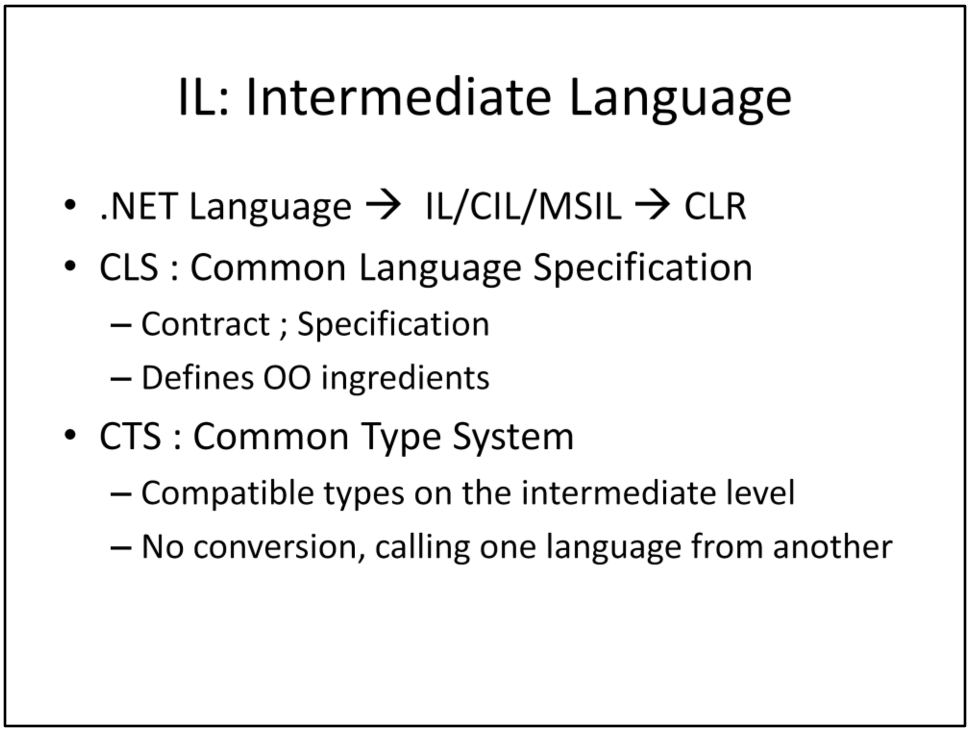
Example of MSIL Code

```
.method private hidebysig static void  Main() cil
  managed
{
  .entrypoint
  // Code size       11 (0xb)
  .maxstack  8
  IL_0000:  ldstr      "Hello, world!"
  IL_0005:  call       void
  [mscorlib]System.Console::WriteLine(string)
  IL_000a:  ret
} // end of method HelloWorld::Main
```

IL Disassembler **IL Assembler**

Ildasm.exe **Ilasm.exe**

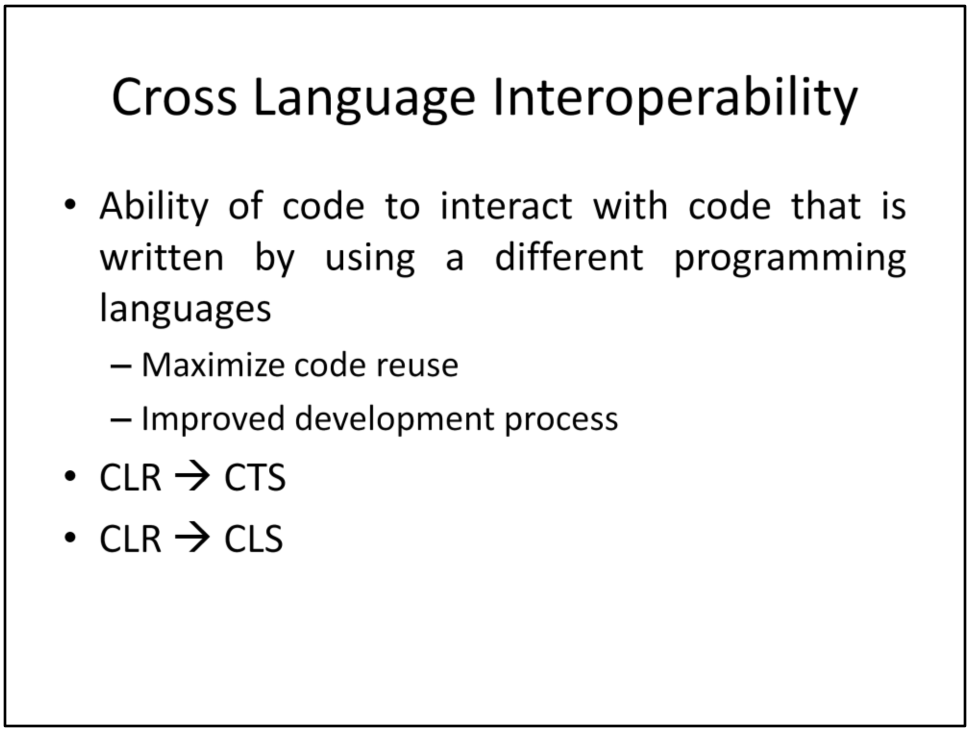
An assembler is a computer program that translates human-readable assembly language source code into machine language instructions that can be executed by the computer hardware. The input to an assembler is one or more assembly language source files. The output from an assembler is (typically) an object code file, which contains machine language instructions. The object file is then (typically) fed into a linker to produce an executable program file or library.
A disassembler is a computer program that translates machine language instructions into human-readable assembly language source code. Disassemblers are typically used for debugging or for reverse-engineering, when all you have available is machine language. Disassemblers can’t always distinguish between instructions and data, so they might convert data into a meaningless (and never executed) sequence of instructions. The assembly-language source code produced by a disassembler is typically less readable than assembly language written by a human, because it lacks meaningful symbols and comments that a human would normally use in their assembly language source code.
So, while a disassembler does essentially the opposite of an assembler, in practice the resulting source code from a disassembler is not as readable (and perhaps not as accurate) as human-written assembly language source code.



IL: Intermediate Language

- .NET Language → IL/CIL/MSIL → CLR
- CLS : Common Language Specification
  - Contract ; Specification
  - Defines OO ingredients
- CTS : Common Type System
  - Compatible types on the intermediate level
  - No conversion, calling one language from another

All the .NET languages are compiled into another lower-level language before the code is executed. This lower level
language is the *Common Intermediate Language (CIL, or just IL). The CLR, the engine of .NET, uses only* IL code. Because all .NET languages are based on IL, they all have profound similarities. This is the reason that
the VB and C# languages provide essentially the same features and performance. In fact, the languages are so
compatible that a web page written with C# can use a VB component in the same way it uses a C# component,
and vice versa.
The .NET Framework formalizes this compatibility with something called the *Common Language Specification (CLS). Essentially, the CLS is a contract that, if respected, guarantees that a component written in*
one .NET language can be used in all the others.
The specification defines an environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures.
The CLS also defines object-oriented ingredients such as classes, methods, events, and quite a bit more.
API designed with rules of CLS, can easily be used by all the .NET languages.
One part of the CLS is the *common type system (CTS), which*
defines the rules for data types such as strings, numbers, and arrays that are shared in all .NET languages.
CTS
All .NET languages have the same primitive data types. An *int* in C# is the same as an *int* in VB.NET. This is how .NET provide multiple language support.
When communicating between modules written in any .NET language, the types are guaranteed to be compatible on the Intermediate level.
Hence calling one language from another does not require type conversions.
The CTS also defines the rules that ensures that the data types of objects written in various languages are able to interact with each other.
Strings are a primitive data type now.



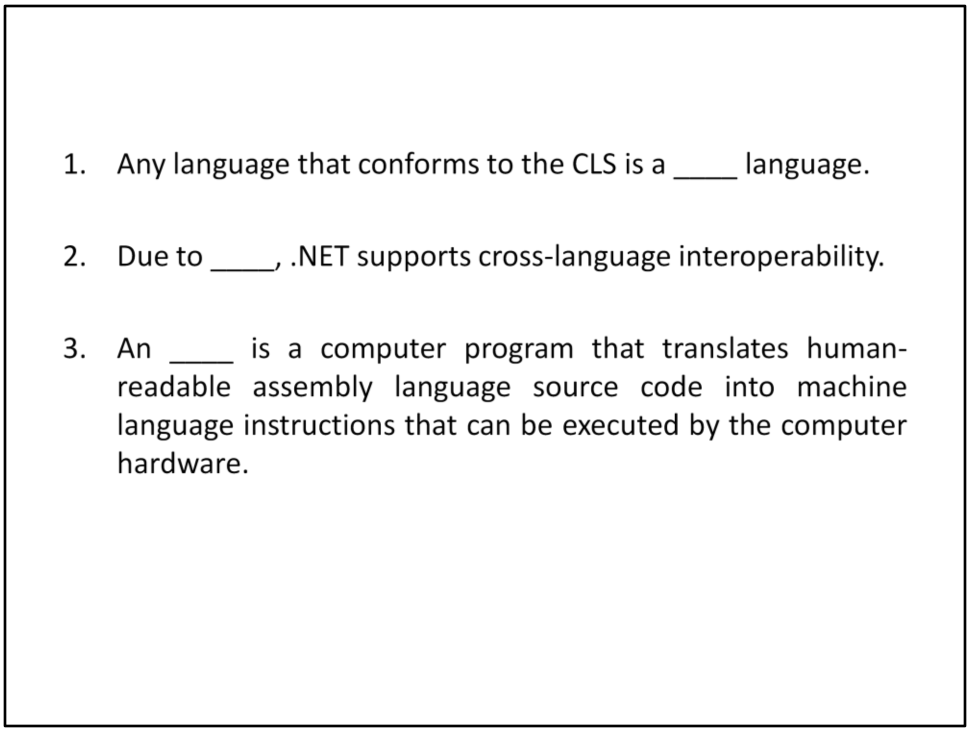
Cross Language Interoperability

- Ability of code to interact with code that is written by using a different programming languages
  - Maximize code reuse
  - Improved development process
- CLR $\rightarrow$ CTS
- CLR $\rightarrow$ CLS

Language interoperability is the ability of code to interact with code that is written by using a different programming language.

Language interoperability can help maximize code reuse and improve the efficiency of the development process.
The common language runtime provides the necessary foundation for language interoperability by specifying and enforcing a common type system.
Because all languages targeting the runtime follow the common type system (CTS) rules for defining and using types, the usage of types is consistent across languages.

To ensure that your managed code is accessible to developers who are using other programming languages, the .NET Framework provides the Common Language Specification(CLS)

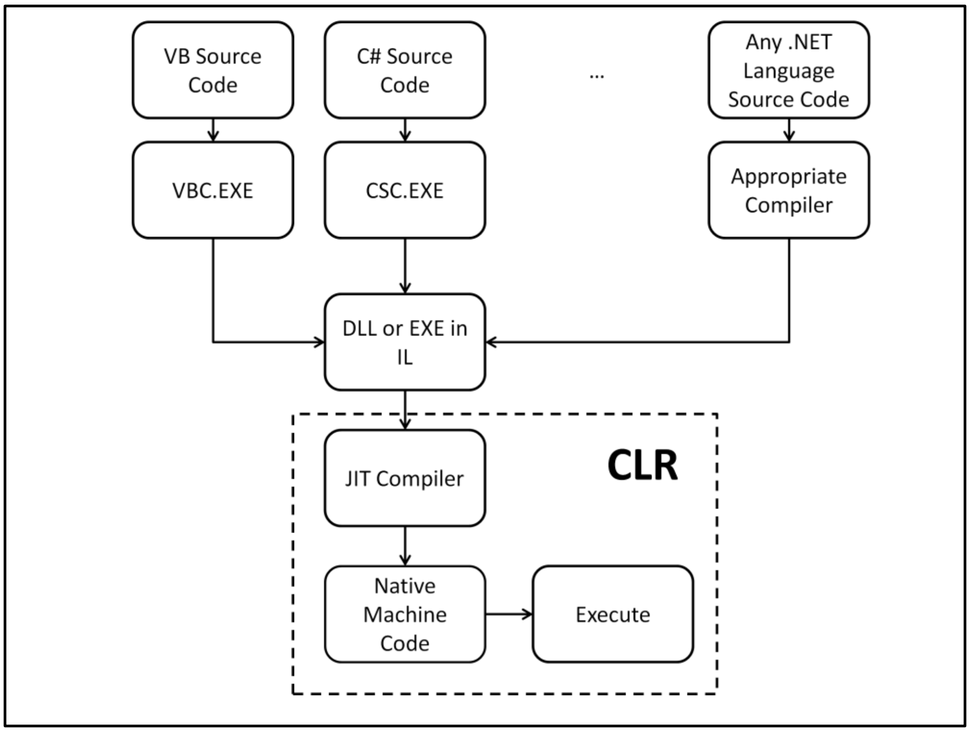


1. Any language that conforms to the CLS is a ____ language.
2. Due to ____, .NET supports cross-language interoperability.
3. An ____ is a computer program that translates human-readable assembly language source code into machine language instructions that can be executed by the computer hardware.

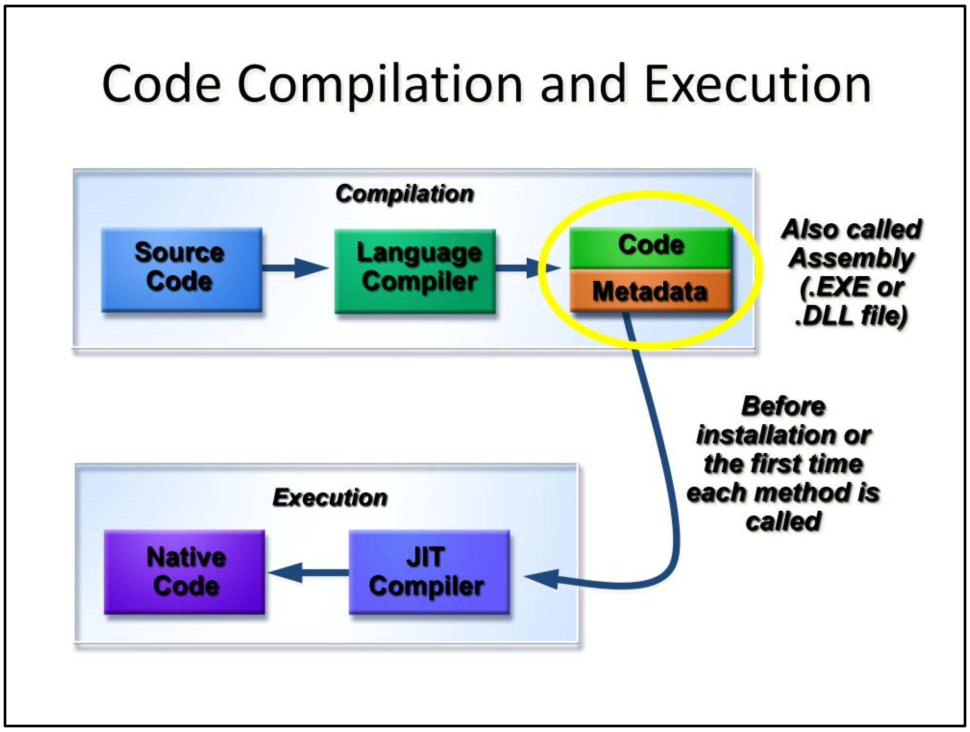
Any language that conforms to the CLS is a .NET language

Hence due to CLS/CLR, .NET supports cross-language interoperability.

An assembler is a computer program that translates human-readable assembly language source code into machine language instructions that can be executed by the computer hardware.



Language Compilation in .NET

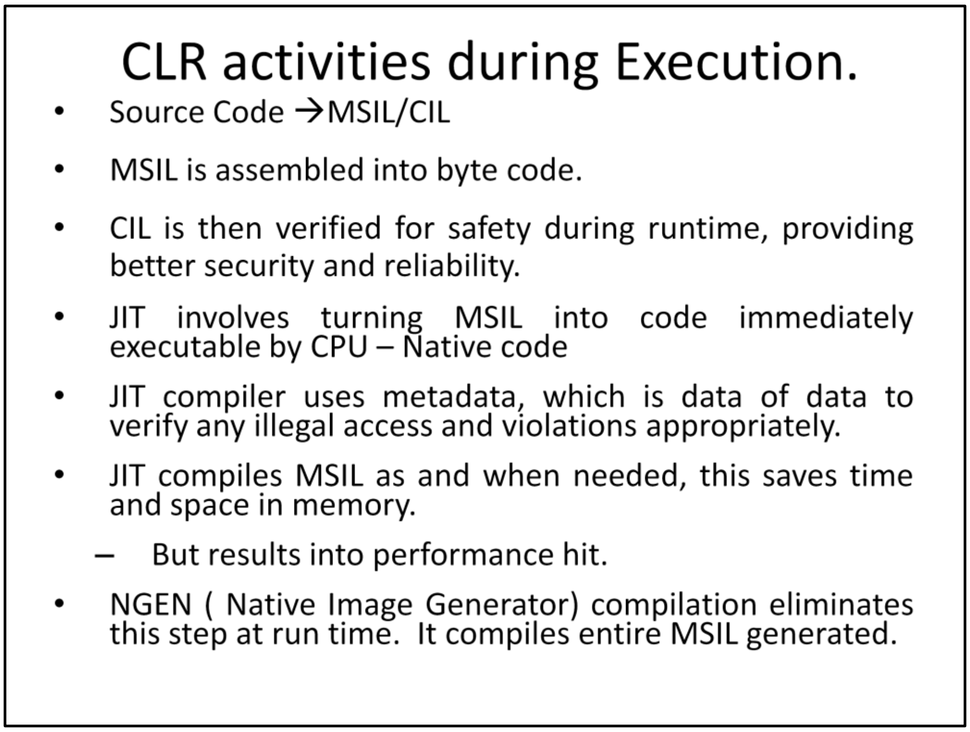

Code Compilation and Execution

Compilation
Source Code
Language Compiler
Code
Metadata
Also called Assembly (.EXE or .DLL file)
Before installation or the first time each method is called
Execution
Native Code
JIT Compiler

The Common Language Runtime
Compilation and Execution

The above diagram illustrates the process used to compile and execute managed code, or code that uses the CLR. Source code written in Visual C#, Visual Basic .NET or another language that targets the CLR is first transformed into MSIL by the appropriate language compiler. Before execution, this MSIL is compiled by the Just-in-Time (JIT) complier into native code for the processor on which the code will operate. The default is to JIT compile each method when it is first called, but it is also possible to “preJIT” MSIL code to native code at assembly install-time. With this option, all methods are compiled before the application is loaded so as to avoid the overhead of JIT compilation on each initial method call. You use the Native Image Generator (Ngen.exe) to create a native image from a managed assembly and install it into the native image cache. Once the code is compiled and cached, the CLR does not need to re-compile the MSIL code until the assembly is updated, the Just-In-Time compiled code is removed from the cache, or the machine is restarted.

All languages targeting the CLR *should* exhibit a similar performance. While some compilers may produce better MSIL code, large variations in execution speed are unlikely.


CLR activities during Execution.

- Source Code →MSIL/CIL
- MSIL is assembled into byte code.
- CIL is then verified for safety during runtime, providing better security and reliability.
- JIT involves turning MSIL into code immediately executable by CPU – Native code
- JIT compiler uses metadata, which is data of data to verify any illegal access and violations appropriately.
- JIT compiles MSIL as and when needed, this saves time and space in memory.
  - But results into performance hit.
- NGEN (Native Image Generator) compilation eliminates this step at run time. It compiles entire MSIL generated.

Your program source code is converted to MSIL or CIL with the use of language compiler.
This MSIL is platform independent instruction set executed by virtual machine (i.e CLR)