Appendix

Answers to the Test Your Knowledge Questions

This appendix has the answers to the questions in the **Test Your Knowledge** section at the end of each chapter.

Chapter 1 – Hello, C#! Welcome, .NET!

1. Is Visual Studio 2022 better than Visual Studio Code?

   Answer: No. Each is optimized for different tasks. Visual Studio 2022 for Windows is large, heavyweight, and can create applications with graphical user interfaces, for example, Windows Forms, WPF, UWP, and .NET MAUI apps, but it is only available on Windows. Visual Studio 2022 is an **Interactive Development Environment** (**IDE**) rather than a code editor. Visual Studio Code is smaller, lighter-weight, code-focused, supports many more languages, and is available cross-platform.

2. Is .NET 5 and later better than .NET Framework?

   Answer: For modern development, yes, but it depends on what you need. .NET 5 and later are modern, cross-platform, performance-oriented versions of the legacy, mature .NET Framework. Modern .NET is more frequently improved. .NET Framework has better support for legacy applications; however, .NET Framework 4.8 will be the last release apart from security and bug fixes. It will never support some language features of C# 8 and later.

3. What is .NET Standard and why is it still important?

   Answer: .NET Standard defines an API, aka contract, that a .NET platform can implement. The latest versions of .NET Framework, Xamarin, and modern .NET implement .NET Standard 2.0 to provide a single, standard API that developers can target for maximum reuse. .NET Core 3.0 or later implement .NET Standard 2.1, which has some new features not supported by .NET Framework. If you want to create a new class library that supports all .NET platforms you will need it to be .NET Standard 2.0-compatible.

4. Why can a programmer use different languages, for example, C# and F#, to write applications that run on .NET?

   Answer: Multiple languages are supported on .NET because each one has a compiler that translates the source code into **intermediate language** (**IL**) code. This IL code is then compiled to native CPU instructions at runtime by the **Common Language Runtime** (**CLR**).

5. What is a top-level program and how do you access any command-line arguments?

   Answer: A top-level program is a project that does not need to explicitly define a `Program` class with a `Main` method entry point, with a parameter named `args`, to access any command-line arguments. These are implicitly defined for you so that you can type statements without boilerplate code.

6. What is the name of the entry point method of a .NET console app and how should it be explicitly declared if you are not using the top-level program feature?

   Answer: The entry point of a .NET console app is the `Main` method. An optional `string` array for command-line arguments and a return type of `int` are recommended, but they are not required. They can be explicitly declared, as shown in the following code:

   ```
   public static void Main() // minimum
   public static int Main(string[] args) // recommended
   ```

   With .NET 6 and later, it is implicitly declared using the top-level program feature, as shown in the following code:

   ```
   public static int <Main>$(String[] args) // compiler-generated
   ```

7. What do you type at the command line to build and execute C# source code?

   Answer: In a folder with a `.csproj` file, you enter `dotnet run`.

8. What are some benefits of using .NET Interactive Notebooks to write C# code?

   Answer: The benefits of using .NET Interactive Notebooks to write C# code include mixing languages in the same document and mixing code with Markdown for rich text. This makes notebooks perfect for students learning languages, anyone exploring APIs, and data scientists analyzing and visualizing data.

9. Where would you look for help for a C# keyword?

   Answer: The Microsoft Docs website. Specifically, C# keywords are documented at the following link: `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/`.

10. Where would you look first for solutions to common programming problems?

    Answer: `https://stackoverflow.com/`.

Chapter 2 — Speaking C#

Exercise 2.1 — Test your knowledge

1. What statement can you type in a C# file to discover the compiler and language version?

   Answer: `#error version`

2. What are the two types of comments in C#?

   Answer: The two types of comments in C# are a single-line comment prefixed with `//` and a multi-line comment starting with `/*` and ending with `*/`. There are also XML comments, which are introduced in *Chapter 4*, *Writing, Debugging, and Testing Functions*, so I would not expect you to know about them yet.

3. What is the difference between a verbatim `string` and an interpolated `string`?

   Answer: A verbatim `string` is prefixed with the `@` symbol and each character (except ") is interpreted as itself; for example, a backslash \ is a backslash \. An interpolated `string` is prefixed with the `$` symbol and can include expressions surrounded with braces like this: `{expression}`.

4. Why should you be careful when using `float` and `double` values?

   Answer: You should be careful when using `float` and `double` values because they are not guaranteed to be accurate, especially when performing equality comparisons.

5. How can you determine how many bytes a type like `double` uses in memory?

   Answer: You can determine how many bytes a type like `double` uses in memory by using the `sizeof()` operator, for example, `sizeof(double)`.

6. When should you use the `var` keyword?

   Answer: You should only use the `var` keyword to declare local variables when you cannot specify a known type. It is easy to overuse `var` due to its convenience when initially writing code, but its use can make it harder to maintain code later.

7. What is the newest syntax to create an instance of a class like `XmlDocument`?

   Answer: The newest way to create an instance of a class like `XmlDocument` is to use a **target-typed new** expression, as shown in the following code:

   ```
   XmlDocument doc = new();
   ```

8. Why should you be careful when using the `dynamic` type?

   Answer: You should be careful when using the `dynamic` type because the type of object stored in it is not checked until runtime, which can mean runtime exceptions being thrown if you attempt to use a member that does not exist on the type.

9. How do you right-align a format `string`?

   Answer: To right-align a format `string`, after the index or expression, add a comma and an integer value to specify a column width within which to align the value. Positive integers mean right-aligned and negative integers mean left-aligned.

10. What character separates arguments for a console app?

    Answer: The space character separates arguments for a console app.

Exercise 2.2 – Test your knowledge of number types

What type would you choose for the following "numbers"?

1. A person's telephone number.

   Answer: `string`.

2. A person's height.

   Answer: `float` or `double`.

3. A person's age.

   Answer: `int` for best performance on most CPUs or `byte` (0 to 255) for the smallest size.

4. A person's salary.

   Answer: `decimal`.

5. A book's ISBN.

   Answer: `string`.

6. A book's price.

   Answer: `decimal`.

7. A book's shipping weight.

   Answer: `float` or `double`.

8. A country's population.

   Answer: `uint` (0 to about 4 billion).

9. The number of stars in the universe.

   Answer: `ulong` (0 to about 18 quadrillion) or `System.Numerics.BigInteger` (allows an arbitrarily large integer).

10. The number of employees in each of the small or medium businesses in the United Kingdom (up to about 50,000 employees per business).

    Answer: Since there are hundreds of thousands of small or medium businesses, we need to take memory size as the determining factor, so choose `ushort`, because it only takes 2 bytes compared to an `int`, which takes 4 bytes.

Chapter 3 — Controlling Flow, Converting Types, and Handling Exceptions

Exercise 3.1 — Test your knowledge

1. What happens when you divide an `int` variable by `0`?

   Answer: `DivideByZeroException` is thrown when dividing an integer or decimal by `0`.

2. What happens when you divide a `double` variable by `0`?

   Answer: The `double` type contains a special value of infinity. Instances of floating-point numbers can have the special values of `NaN` (not a number), or in the case of dividing by `0`, either `PositiveInfinity` or `NegativeInfinity`.

3. What happens when you overflow an `int` variable, that is, set it to a value beyond its range?

   Answer: It will loop unless you wrap the statement in a `checked` block, in which case, `OverflowException` will be thrown.

4. What is the difference between `x = y++;` and `x = ++y;`?

   Answer: In the statement `x = y++;`, the current value of `y` will be assigned to `x` and then `y` will be incremented, and in the statement `x = ++y;`, the value of `y` will be incremented and then the result will be assigned to `x`.

5. What is the difference between `break`, `continue`, and `return` when used inside a loop statement?

   Answer: The `break` statement will end the whole loop and continue executing after the loop, the `continue` statement will end the current iteration of the loop and continue executing at the start of the loop block for the next iteration, and the `return` statement will end the current method call and continue executing after the method call.

6. What are the three parts of a `for` statement and which of them are required?

   Answer: The three parts of a `for` statement are the initializer, condition, and incrementer expressions. All three parts are optional. If the condition is missing, then it will loop forever unless you break out of the loop using `break`, `return`, or something else.

7. What is the difference between the `=` and `==` operators?

   Answer: The `=` operator is the assignment operator for assigning values to variables, while the `==` operator is the equality check operator that returns `true` or `false`.

8. Does the following statement compile? `for ( ; ; ) ;`

   Answer: Yes. The `for` statement only requires the semicolons to separate the initializer expression, condition expression, and incrementer expressions. All three expressions are optional so they can be missing. This `for` statement will execute the empty `;` statement after the closing brace, forever. It is an example of an infinite loop.

9. What does the underscore _ represent in a `switch` expression?

   Answer: The underscore _ represents the default return value. It is called a discard because it is used to represent a potential variable that is not needed. It can be used not just in `switch` expressions but other scenarios where a placeholder variable must be declared but its value is not needed, like parameters to lambda expressions and tuple deconstruction.

10. What interface must an object "implement" to be enumerated over using the `foreach` statement?

    Answer: An object must "implement" the `IEnumerable` interface. It must have the correct methods with the correct signatures even if the object does not actually implement the interface.

Exercise 3.2 – Explore loops and overflow

What will happen if this code executes?

```
int max = 500;
for (byte i = 0; i < max; i++)
{
  WriteLine(i);
}
```

Answer:

The code will loop forever because the value of `i` can only be between `0` and `255`. Once `i` gets incremented beyond `255`, it loops back to `0` and therefore will always be less than `max` (`500`).

To prevent the infinite loop, you can add a checked statement around the code. This would cause an exception to be thrown after `255` due to the overflow, as shown in the following output:

```
254
255
System.OverflowException says Arithmetic operation resulted in an overflow.
```

Exercise 3.5 – Test your knowledge of operators

1. What are the values of x and y after the following statements execute?

   ```
   x = 3;
   y = 2 + ++x;
   ```

   Answer: x is 4 and y is 6.

2. What are the values of x and y after the following statements execute?

   ```
   x = 3 << 2;
   y = 10 >> 1;
   ```

   Answer: x is 12 and y is 5.

3. What are the values of `x` and `y` after the following statements execute?

```
x = 10 & 8;
y = 10 | 7;
```

Answer: `x` is `8` and `y` is `15`.

Chapter 4 — Writing, Debugging, and Testing Functions

1. What does the C# keyword `void` mean?

   Answer: It indicates that a method has no return value.

2. What are some differences between imperative and functional programming styles?

   Answer: An imperative programming style means writing a sequence of statements that the runtime executes step by step like a recipe. Your code tells the runtime exactly how to perform the task. Do this. Now do that. It has variables meaning that the state can change at any time, including outside the current function. Imperative programming causes side effects, changing the value of some state somewhere in your program. Side effects are tricky to debug. A functional programming style describes what you want to achieve instead of how. It can also be described as declarative. But the most important point is that functional programming languages make all states immutable by default to avoid side effects.

3. In Visual Studio Code or Visual Studio, what is the difference between pressing *F5*, *Ctrl* or *Cmd* + *F5*, *Shift* + *F5*, and *Ctrl* or *Cmd* + *Shift* + *F5*?

   Answer: *F5* saves, compiles, runs, and attaches the debugger; *Ctrl* or *Cmd* + *F5* saves, compiles, and runs the application without the debugger attached; *Shift* + *F5* stops the debugger and running application; and *Ctrl* or *Cmd* + *Shift* + *F5* restarts the application without the debugger attached.

4. Where does the `Trace.WriteLine` method write its output to?

   Answer: `Trace.WriteLine` writes its output to any configured trace listeners. By default, this includes the terminal or the command line but can be configured to be a text file or any custom listener.

5. What are the five trace levels?

   Answer: 0 = Off, 1 = Error, 2 = Warning, 3 = Info, and 4 = Verbose.

6. What is the difference between the `Debug` and `Trace` classes?

   Answer: `Debug` is active only during development. `Trace` is active during development and after release into production.

7. When writing a unit test, what are the three "A"s?

   Answer: Arrange, Act, and Assert.

8. When writing a unit test using xUnit, what attribute must you decorate the test methods with?

   Answer: `[Fact]` or `[Theory]`.

9. What `dotnet` command executes xUnit tests?

   Answer: `dotnet test`.

10. What statement should you use to rethrow a caught exception named `ex` without losing the stack trace?

    Answer: Use `throw;`. Do not use `throw ex;` because this will lose stack trace information.

Chapter 5 — Building Your Own Types with Object-Oriented Programming

1. What are the six combinations of access modifier keywords and what do they do?

   Answer: The six combinations of access modifier keywords and their effects are described in the following list:

   - `private`: This modifier makes a member only visible inside the class.
   - `internal`: This modifier makes a member only visible inside the same assembly.
   - `protected`: This modifier makes a member only visible inside the class or derived classes.
   - `internal protected`: This modifier makes a member only visible inside the class, derived classes, or within the same assembly.
   - `private protected`: This modifier makes a member only visible inside the class or derived classes that are within the same assembly.
   - `public`: This modifier makes a member visible everywhere.

2. What is the difference between the `static`, `const`, and `readonly` keywords when applied to a type member?

   Answer: The difference between the `static`, `const`, and `readonly` keywords when applied to a type member is described in the following list:

   - `static`: This keyword makes the member shared by all instances and it must be accessed through the type, not an instance of the type.
   - `const`: This keyword makes the field a fixed literal value that must never change because during the compilation, assemblies that use the field copy the literal value at the time of compilation.
   - `readonly`: This keyword restricts the field so that it can only be assigned to using a constructor or field initializer at runtime.

3. What does a constructor do?

   Answer: A constructor allocates memory and initializes field values.

4. Why do you need to apply the `[Flags]` attribute to an `enum` type when you want to store combined values?

   Answer: If you don't apply the `[Flags]` attribute to an `enum` type when you want to store combined values, then a stored `enum` value that is a combination will be returned by a call to `ToString` as the stored integer value, instead of one or more of the comma-separated list of text values.

5. Why is the `partial` keyword useful?

   Answer: You can use the `partial` keyword to split the definition of a type over multiple files.

6. What is a tuple?

   Answer: A tuple is a data structure consisting of multiple parts. They are used when you want to store multiple values as a unit without defining a type for them.

7. What does the `record` keyword do?

   Answer: The `record` keyword defines a data structure that is immutable by default to enable a more functional programming style. Like a class, a record can have properties and methods, but the values of properties can only be set during initialization.

8. What does overloading mean?

   Answer: Overloading is when you define more than one method with the same method name and different input parameters.

9. What is the difference between a field and a property?

   Answer: A field is a data storage location that can be referenced. A property is one or a pair of methods that get and/or set a value. The value of a property is often stored in a private field.

10. How do you make a method parameter optional?

    Answer: You make a method parameter optional by assigning a default value to it in the method signature.

Chapter 6 – Implementing Interfaces and Inheriting Classes

1. What is a delegate?

   Answer: A delegate is a type-safe method reference. It can be used to execute any method with a matching signature.

2. What is an event?

   Answer: An event is a field that is a delegate having the `event` keyword applied. The keyword ensures that only `+=` and `-=` are used; this safely combines multiple delegates without replacing any existing event handlers.

3. How are a base class and a derived class related and how can the derived class access the base class?

   Answer: A derived class (or subclass) is a class that inherits from a base class (or superclass). Inside a derived class, you use the `base` keyword to access the class that the subclass inherits from.

4. What is the difference between `is` and `as` operators?

   Answer: The `is` operator returns `true` if an object can be cast to the type; otherwise, it returns `false`. The `as` operator returns a reference to the object if an object can be cast to the type; otherwise, it returns `null`.

5. Which keyword is used to prevent a class from being derived from or a method from being overridden?

   Answer: `sealed`.

6. Which keyword is used to prevent a class from being instantiated with the `new` keyword?

   Answer: `abstract`.

7. Which keyword is used to allow a member to be overridden?

   Answer: `virtual`.

8. What's the difference between a destructor and a deconstruct method?

   Answer: A destructor, also known as a finalizer, must be used to release resources owned by the object. A deconstruct method is a feature of C# 7 or later that allows a complex object to be broken down into smaller parts. It is especially useful when working with tuples.

9. What are the signatures of the constructors that all exceptions should have?

   Answer: The signatures of the three constructors that all exceptions should have are shown in the following list:

   - A constructor with no parameters.
   - A constructor with a `string` parameter, usually named `message`.
   - A constructor with a `string` parameter, usually named `message`, and an `Exception` parameter, usually named `innerException`.

10. What is an extension method, and how do you define one?

    Answer: An extension method is a compiler trick that makes a `static` method of a `static` class appear to be one of the members of another type. You define which type you want to extend by prefixing the first parameter of that type in the method with the `this` keyword.

Chapter 7 — Packaging and Distributing .NET Types

1. What is the difference between a namespace and an assembly?

   Answer: A namespace is the logical container of a type. An assembly is the physical container of a type. To use a type, the developer must reference its assembly. Optionally, the developer can import its namespace, or specify the namespace when specifying the type.

2. How do you reference another project in a `.csproj` file?

   Answer: You reference another project in a `.csproj` file by adding a `<ProjectReference>` element that sets its `Include` attribute to a path to the reference project file inside an `<ItemGroup>` element, as shown in the following markup:

   ```
   <ItemGroup>
     <ProjectReference Include="..\Calculator\Calculator.csproj" />
   </ItemGroup>
   ```

3. What is the benefit of a tool like ILSpy?

   Answer: A benefit of a tool like ILSpy is learning how to write code in C# for the .NET platform by seeing how other packages are written. You should never steal their intellectual property, of course. But it is especially useful to see how the Microsoft developers have implemented key components of the Base Class Libraries. Decompiling can also be useful when calling a third-party library that you need to better understand how it works to call it appropriately.

4. Which .NET type does the C# `float` alias represent?

   Answer: `System.Single`.

5. When porting an application from .NET Framework to modern .NET, what tool should you run before porting, and what tool could you run to perform much of the porting work?

   Answer: You should use the .NET Portability Analyzer before porting an application from .NET Framework to .NET 6. You could use the .NET Upgrade Assistant to perform much of the porting work.

6. What is the difference between framework-dependent and self-contained deployments of .NET applications?

   Answer: Framework-dependent modern .NET applications require .NET to be installed for an operating system to execute. Self-contained .NET applications include everything necessary to execute on their own.

7. What is a RID?

   Answer: **RID** is the acronym for **Runtime Identifier**. RID values are used to identify target platforms where a .NET application runs.

8. What is the difference between the `dotnet pack` and `dotnet publish` commands?

   Answer: The `dotnet pack` command creates a NuGet package that could then be uploaded to a NuGet feed like Microsoft's. The `dotnet publish` command puts the application and its dependencies into a folder for deployment to a hosting system.

9. What types of applications written for .NET Framework can be ported to modern .NET?

   Answer: Console, ASP.NET MVC, ASP.NET Web API, Windows Forms, and **Windows Presentation Foundation** (**WPF**) apps.

10. Can you use packages written for .NET Framework with modern .NET?

    Answer: Yes, if they only call APIs in .NET Standard 2.0.

Chapter 8 — Working with Common .NET Types

1. What is the maximum number of characters that can be stored in a `string` variable?

   Answer: The maximum size of a `string` variable is 2 GB or about 1 billion characters, because each char uses 2 bytes due to the internal use of Unicode (UTF-16) encoding for characters in a `string`.

2. When and why should you use a `SecureString` type?

   Answer: The string type leaves text data in the memory for too long and it's too visible. The `SecureString` type encrypts its text and ensures that the memory is released immediately. For example, in WPF, the `PasswordBox` control stores its password as a `SecureString` variable, and when starting a new process, the `Password` parameter must be a `SecureString` variable.

3. When is it appropriate to use a `StringBuilder` class?

   Answer: When concatenating more than about three `string` variables, you will use less memory and get improved performance using `StringBuilder` than using the `string.Concat` method or the + operator.

4. When should you use a `LinkedList<T>` class?

   Answer: Each item in a linked list has a reference to its previous and next siblings as well as the list itself. A linked list should be used when items need to be inserted and removed from positions in the list without moving the items in memory.

5. When should you use a `SortedDictionary<T>` class rather than a `SortedList<T>` class?

   Answer: The `SortedList<T>` class uses less memory than `SortedDictionary<T>`; `SortedDictionary<T>` has faster insertion and removal operations for unsorted data. If the list is populated all at once from sorted data, `SortedList<T>` is faster than `SortedDictionary<T>`.

6. In a regular expression, what does $ mean?

   Answer: In a regular expression, $ represents the end of the input.

7. In a regular expression, how can you represent digits?

   Answer: In a regular expression, you can represent digit characters using `\d` or `[0-9]`.

8. Why should you *not* use the official standard for email addresses to create a regular expression to validate a user's email address?

   Answer: The effort is not worth the pain for you or your users. Validating an email address using the official specification doesn't check whether that address exists or whether the person entering the address is its owner.

9. What characters are output when the following code runs?

   ```
   string city = "Aberdeen";
   ReadOnlySpan<char> citySpan = city.AsSpan()[^5..^0];
   WriteLine(citySpan.ToString());
   ```

   Answer: `rdeen`. `^5..` means the range is 5 characters long. `..^0` means the range ends zero characters in from the right end.

10. How could you check that a web service is available before calling it?

    Answer: Use the `Ping` class to call the web service and check the `Status` of the reply.

Chapter 9 — Working with Files, Streams, and Serialization

1. What is the difference between using the `File` class and the `FileInfo` class?

   Answer: The `File` class has `static` methods and it cannot be instantiated. It is best used for one-off tasks such as copying a file. The `FileInfo` class requires the instantiation of an object that represents a file. It is best used when you need to perform multiple operations on the same file.

2. What is the difference between the `ReadByte` method and the `Read` method of a stream?

   Answer: The `ReadByte` method returns a single byte each time it is called and the `Read` method fills a temporary array with bytes up to a specified length. It is generally best to use `Read` to process multiple bytes at once.

3. When would you use the `StringReader`, `TextReader`, and `StreamReader` classes?

   Answer:

   - `StringReader` is used for efficiently reading from a string stored in memory.
   - `TextReader` is an abstract class that `StringReader` and `StreamReader` both inherit from for their shared functionality.
   - `StreamReader` is used for reading strings from a stream that can be any type of text file, including XML and JSON.

4. What does the `DeflateStream` type do?

   Answer: `DeflateStream` implements the same compression algorithm as GZIP, but without a cyclical redundancy check; so, although it produces smaller compressed files, it cannot perform integrity checks when decompressing.

5. How many bytes per character does UTF-8 encoding use?

   Answer: The number of bytes per character used by the UTF-8 encoding depends on the character. Most Western alphabet characters are stored using one byte. Other characters may need two or more bytes.

6. What is an object graph?

   Answer: An object graph is any set of connected instances of classes that reference each other. For example, a `Customer` object may have a property named `Orders` that references a collection of `Order` instances.

7. What is the best serialization format to choose for minimizing space requirements?

   Answer: **JavaScript Object Notation** (**JSON**) has a good balance between space requirements and practical factors like human readability, but the **protocol buffers** (**Protobuf**) serialization format used by the gRPC standard is best for minimizing space requirements.

8. What is the best serialization format to choose for cross-platform compatibility?

   Answer: There is still an argument for **eXtensible Markup Language** (**XML**) if you need maximum compatibility, especially with legacy systems, although JSON is better if you need to integrate with web systems, or protocol buffers for best performance and minimum bandwidth use.

9. Why is it bad to use a `string` value like `"\Code\Chapter01"` to represent a path, and what should you do instead?

   Answer: It is bad to use a `string` value like `"\Code\Chapter01"` to represent a path because it assumes that backslashes are used as a folder separator on all operating systems. Instead, you should use the `Path.Combine` method and pass separate `string` values for each folder, or a `string` array, as shown in the following code:

   ```
   string path = Path.Combine(new[] { "Code", "Chapter01" });
   ```

10. Where can you find information about NuGet packages and their dependencies?

    Answer: You can find information about NuGet packages and their dependencies at the following link: `https://www.nuget.org/`.

Chapter 10 – Working with Data Using Entity Framework Core

1. What type would you use for the property that represents a table, for example, the `Products` property of a database context?

   Answer: `DbSet<T>`, where `T` is the entity type, for example, `Product`.

2. What type would you use for the property that represents a one-to-many relationship, for example, the `Products` property of a `Category` entity?

   Answer: `ICollection<T>`, where `T` is the entity type, for example, `Product`.

3. What is the EF Core convention for primary keys?

   Answer: The property named `ID` or `Id` or `ClassNameID` or `ClassNameId` is assumed to be the primary key. If the type of that property is any of the following, then the property is also marked as being an `IDENTITY` column: `tinyint`, `smallint`, `int`, `bigint`, and `guid`.

4. When might you use an annotation attribute in an entity class?

   Answer: You might use an annotation attribute in an entity class when the conventions cannot work out the correct mapping between the classes and tables, for example, if a class name does not match a table name or a property name does not match a column name. You might also define constraints, like a maximum length of characters in a text value or a range of numeric values, by decorating with validation attributes. These can be read by technologies like ASP.NET Core MVC and Blazor to provide automatic validation warnings to users.

5. Why might you choose Fluent API in preference to annotation attributes?

   Answer: You might choose Fluent API in preference to annotation attributes when you want to keep your entity classes free from extraneous code that is not needed in all scenarios. For example, when creating a .NET Standard 2.0 class library for entity classes, you might want to only use validation attributes so that the metadata can be read by Entity Framework Core and by technologies like ASP.NET Core model binding validation and .NET MAUI desktop and mobile apps. However, you might want to use Fluent API to define Entity Framework Core-specific functionality like mapping to a different table or column name.

6. What does a transaction isolation level of `Serializable` mean?

   Answer: Maximum locks are applied to ensure complete isolation from any other processes working with the affected data.

7. What does the `DbContext.SaveChanges` method return?

   Answer: An `int` value for the number of entities affected.

8. What is the difference between eager loading and explicit loading?

   Answer: Eager loading means related entities are included in the original query to the database so that they do not have to be loaded later. Explicit loading means related entities are not included in the original query to the database, and they must be explicitly loaded just before they are needed.

9. How should you define an EF Core entity class to match the following table?

```
CREATE TABLE Employees(
  EmpId INT IDENTITY,
  FirstName NVARCHAR(40) NOT NULL,
  Salary MONEY
)
```

   Answer: Use the following class:

```
public class Employee
{
  [Column("EmpId")]
  public int EmployeeId { get; set; }

  [Required]
  [StringLength(40)]
  public string FirstName { get; set; }

  [Column(TypeName = "money")]
  public decimal? Salary { get; set; }
}
```

10. What benefit do you get from declaring entity navigation properties as `virtual`?

    Answer: You can enable lazy loading if you declare entity navigation properties as `virtual`.

Chapter 11 — Querying and Manipulating Data Using LINQ

1. What are the two required parts of LINQ?

   Answer: A LINQ provider and the LINQ extension methods. You must import the `System.Linq` namespace to make the LINQ extension methods available and reference a LINQ provider assembly for the type of data that you want to work with.

2. Which LINQ extension method would you use to return a subset of properties from a type?

   Answer: The `Select` method allows projection (aka selection) of properties.

3. Which LINQ extension method would you use to filter a sequence?

   Answer: The `Where` method allows filtering by supplying a delegate (or lambda expression) that returns a Boolean to indicate whether the value should be included in the results.

4. List five LINQ extension methods that perform aggregation.

   Answer: Any five of the following: `Max`, `Min`, `Count`, `LongCount`, `Average`, `Sum`, and `Aggregate`.

5. What is the difference between the `Select` and `SelectMany` extension methods?

   Answer: `Select` returns exactly what you specify to return. `SelectMany` checks that the items you have selected are themselves `IEnumerable<T>` and then breaks them down into smaller parts. For example, if the type you select is a `string` value (which is `IEnumerable<char>`), `SelectMany` will break each `string` value returned into its individual `char` values and combine them into a single sequence.

6. What is the difference between `IEnumerable<T>` and `IQueryable<T>`? How do you switch between them?

   Answer: The `IEnumerable<T>` interface indicates a LINQ provider that will execute the query locally like LINQ to Objects. These providers have no limitations but can be less efficient. The `IQueryable<T>` interface indicates a LINQ provider that first builds an expression tree to represent the query and then converts it into another query syntax before executing it, like Entity Framework Core converts LINQ into SQL statements. These providers sometimes have limitations, like a lack of support for some expressions, and may throw exceptions. You can convert from an `IQueryable<T>` provider to an `IEnumerable<T>` provider by calling the `AsEnumerable` method.

7. What does the last type parameter `T` in generic `Func` delegates like `Func<T1, T2, T>` represent?

   Answer: The last type parameter `T` in generic `Func` delegates like `Func<T1, T2, T>` represents the type of the return value. For example, for `Func<string, int, bool>`, the delegate or lambda function used must return a Boolean value.

8. What is the benefit of a LINQ extension method that ends with `OrDefault`?

   Answer: The benefit of a LINQ extension method that ends with `OrDefault` is that it returns the default value instead of throwing an exception if it cannot return a value. For example, calling the `First` method on a sequence of `int` values would throw an exception if the collection is empty, but the `FirstOrDefault` method would return `0`.

9. Why is query comprehension syntax optional?

   Answer: Query comprehension syntax is optional because it is just syntactic sugar. It makes code easier for humans to read but it does not add any additional functionality except the `let` keyword.

You can learn more about the `let` keyword at the following link: `https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/let-clause`.

10. How can you create your own LINQ extension methods?

    Answer: Create a `static` class with a `static` method, with an `IEnumerable<T>` parameter prefixed with `this`, as shown in the following code:

```
namespace System.Linq
{
  public static class MyLinqExtensionMethods
  {
    public static IEnumerable<T> MyChainableExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
      // return something IEnumerable<T>
    }

    public static int? MyAggregateExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
      // return some int value
    }
  }
}
```

Chapter 12 – Introducing Web Development Using ASP.NET Core

Exercise 12.1 – Test your knowledge

1. What was the name of Microsoft's first dynamic server-side executed web page technology, and why is it still useful to know this history today?

   Answer: Active Server Pages (**ASP**). The name ASP.NET Core derives from ASP, and it is still used in new features like Tag Helpers, as shown in the following markup:

```
<a asp-controller="Home" asp-action="Index">Home</a>
```

2. What are the names of two Microsoft web servers?

   Answer: Kestrel (cross-platform) and **Internet Information Services** (**IIS**), which is Windows-only.

3. What are some differences between a microservice and a nanoservice?

   Answer: A microservice implements more than one function grouped into a small domain and is always running. A nanoservice implements a single function and is not always running, i.e., it can be "serverless."

4. What is Blazor?

   Answer: Blazor is a .NET-based technology for implementing client-side web user interfaces. It is designed to be used instead of **single-page applications** (**SPAs**) like React, Angular, and Vue.

5. What was the first version of ASP.NET Core that cannot be hosted on .NET Framework?

   Answer: ASP.NET Core 3.0 requires .NET Standard 2.1, which is not supported by .NET Framework.

6. What is a user agent?

   Answer: A user agent is a client to a web server, for example, a web browser or a search engine web crawler.

7. What impact does the HTTP request-response communication model have on web developers?

   Answer: Website dynamic code resides on and executes on a web server. The server code cannot trigger communication. A web browser must make an HTTP request to trigger code on the server that can then generate an HTTP response. This makes updating a web page difficult because it is under the control of the client, not the server, when requests for more data are made.

8. Name and describe four components of a URL.

   Answer: The *scheme* determines if you are using HTTP or HTTPS. The *domain* is the unique address of the computer (or a web farm of servers acting as a single logical computer). The *port number* is the port on which the server(s) are listening (usually `80` for HTTP and `443` for HTTPS). The *path* is the relative path to a resource like a folder or file. The *query string* is for optional parameters passed along with the request. The *fragment* is an identified element within a resource like a web page.

9. What capabilities does Developer Tools give you?

   Answer: Developer Tools allows you to see every HTTP request and response, view client-side application data like cookies, sessions, and local storage, a console for logging, and so on.

10. What are the three main client-side web development technologies and what do they do?

    Answer: HTML5 (structure), CSS3 (styles), and JavaScript (execute actions).

Exercise 12.2 – Know your webbreviations

What do the following web abbreviations stand for and what do they do?

1. **URI (Uniform Resource Identifier)** is a unique sequence of characters that identifies a resource.
2. **URL (Uniform Resource Location)** is the address of a unique resource.

3. **WCF (Windows Communication Foundation)** is a framework for building service-oriented applications. It is part of .NET Framework and an open-source implementation named WCF Core is available for modern .NET.
4. **TLD (Top-Level Domain)** is one of the domains at the highest level in the hierarchical **DNS (Domain Name System)** of the internet, for example, `packt.com`.
5. **API (Application Programming Interface)** is a mechanism for a computer system to allow other computer systems to communicate with it. They should be well-documented.
6. **SPA (Single-Page Application)** is a web app that loads only a single web page and then updates the content dynamically via JavaScript APIs when needed. SPA frameworks include Angular, React, Vue, and Blazor.
7. **CMS (Content Management System)** is a software application that allows users to build and manage a website without having to write code.
8. **Wasm (WebAssembly)** is a binary instruction format and is designed as a compilation target for programming languages like C# for deployment on the web, on the client and server.
9. **SASS (Syntactically Awesome Style Sheets)** is a CSS preprocessor. Sass has features that don't exist in CSS yet like nesting, mixins, and inheritance that help you write maintainable CSS.
10. **REST (REpresentational State Transfer)** is an architectural style for distributed hypermedia systems. Roy Fielding presented it in 2000 in his famous dissertation.

In case you missed it (ICYMI): Acronyms and initialisms are types of abbreviation. Learn more at the following link: `https://www.rd.com/article/acronym-vs-abbreviation-whats-the-difference/`.

Chapter 13 – Building Websites Using ASP.NET Core Razor Pages

1. List six method names that can be specified in an HTTP request.

   Answer: `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, and `DELETE`. Others include `TRACE`, `OPTIONS`, and `CONNECT`.

2. List six status codes and their descriptions that can be returned in an HTTP response.

   Answer: `200` OK, `201` Created, `301` Moved Permanently, `400` Bad Request, `404` Not Found (missing resource), and `500` Internal Server Error. Others include `101` Switching Protocols (e.g. from HTTP to WebSocket), `202` Accepted, `204` No Content, `304` Not Modified, `401` Unauthorized, `403` Forbidden, `406` Not Acceptable (for example, requesting a response format that is not supported by a website), and `503` Service Unavailable.

3. In ASP.NET Core, what is the `Program` class used for?

   Answer: In ASP.NET Core, the `Program` class where you add and configure dependency services like Razor Pages, MVC, and Entity Framework Core data contexts. It is also where you configure middleware in the request and response pipeline. This might include error handling, security options, static files, default files, and endpoint routing.

4. What does the acronym HSTS stand for and what does it do?

   Answer: HTTP Strict Transport Security (**HSTS**) is an opt-in security enhancement. If a website specifies it and a browser supports it, then it forces all communication over HTTPS and prevents the visitor from using untrusted or invalid certificates.

5. How do you enable static HTML pages for a website?

   Answer: To enable static HTML pages for a website, you must add statements to use default files and then static files (this order is important!), as shown in the following code:

```
app.UseDefaultFiles(); // index.html, default.html, and so on
app.UseStaticFiles();
```

6. How do you mix C# code into the middle of HTML to create a dynamic page?

   Answer: To mix C# code into the middle of HTML to create a dynamic page, you can create a Razor file with the `.cshtml` file extension, and then prefix any C# expressions with the `@` symbol, and for C# statements, wrap them in braces or create a `@functions` section, as shown in the following markup:

```
@page
@functions
{
  public string[] DaysOfTheWeek
  {
    get => System.Threading.Thread.CurrentThread
      .CurrentCulture.DateTimeFormat.DayNames;
  }

  public string WhatDayIsIt
  {
    get => System.DateTime.Now.ToString("dddd");
  }
}
<html>
  <head>
    <title>Today is @WhatDayIsIt</title>
  </head>
  <body>
    <h1>Days of the week in your culture</h1>
    <ul>
    @{
      // to add a block of statements: use braces
      foreach (string dayName in DaysOfTheWeek)
      {
        <li>@dayName</li>
```

```
        }
      }
      </ul>
    </body>
  </html>
```

7. How can you define shared layouts for Razor Pages?

   Answer: To define shared layouts for Razor Pages, create at least two files: `_Layout.cshtml` will define the markup for the shared layout, and `_ViewStart.cshtml` will set the default layout, as shown in the following markup:

```
@{
  Layout = "_Layout";
}
```

8. How can you separate the markup from the code-behind in a Razor Page?

   Answer: To separate the markup from the code-behind in a Razor Page, create two files: `MyPage.cshtml` contains the markup and `MyPage.cshtml.cs` contains a class that inherits from `PageModel`. In `MyPage.cshtml`, set the model to use the class, as shown in the following markup:

```
@page
@model MyProject.Pages.MyPageModel
```

9. How do you configure an Entity Framework Core data context for use with an ASP.NET Core website?

   Answer: To configure an Entity Framework Core data context for use with an ASP.NET Core website:

   - In the project file, reference the assembly that defines the data context class.
   - In `Program.cs` or the `Startup` class, import the namespaces for `Microsoft.EntityFrameworkCore` and the data context class.
   - In the `ConfigureServices` method or the section of `Program.cs` that configures services, add a statement that configures the data context with a database connection string for use with a specified database provider, like SQLite or SQL Server, as shown in the following code:

```
services.AddDbContext<MyDataContext>(options => // or UseSqlServer()
  options.UseSqlite("my database connection string"));
```

   - In the Razor Page model class or `@functions` section, declare a private field to store the data context and then set it in the constructor, as shown in the following code:

```
private MyDataContext db;

public SuppliersModel(MyDataContext injectedContext)
```

```
{
  db = injectedContext;
}
```

10. How can you reuse Razor Pages with ASP.NET Core 2.2 or later?

    Answer: To reuse Razor Pages with ASP.NET Core 2.2 or later, everything related to a Razor page can be compiled into a class library. To create one, enter the following command:

    ```
    dotnet new razorclasslib -s
    ```

Chapter 14 – Building Websites Using the Model-View-Controller Pattern

1. What do the files with the special names `_ViewStart` and `_ViewImports` do when created in the `Views` folder?

   Answer:

   - A `_ViewStart` file contains a block of statements that are executed when the `View` method is executed, when a controller action method passes a model to a view, for example, to set a default layout.
   - A `_ViewImports` file contains `@using` statements to import namespaces for all views, to avoid having to add the same import statements at the top of all views.

2. What are the names of the three segments defined in the default ASP.NET Core MVC route, what do they represent, and which are optional?

   Answer:

   - `{controller}`: For example, `/shippers` represents a controller class to instantiate, for example, `ShippersController`. It is optional because it can use the default value: `Home`.
   - `{action}`: For example, `/privacy` represents an action method to execute, for example, `Privacy`. It is optional because it can use the default value: `Index`.
   - `{id}`: For example, `/5` represents a parameter in the action method, for example, `int id`. It is optional because it is suffixed with `?`.

3. What does the default model binder do, and what data types can it handle?

   Answer: The default model binder sets parameters in the action method. It can handle the following data types:

   - Simple types like `int`, `string`, and `DateTime`, including nullable types.
   - Complex types like `Person` and `Product`.
   - Collection types like `ICollection<T>` or `IList<T>`.

4. In a shared layout file like `_Layout.cshtml`, how do you output the content of the current view?

   Answer: To output the content of the current view in a shared layout, call the `RenderBody` method, as shown in the following markup:

   ```
   @RenderBody()
   ```

5. In a shared layout file like `_Layout.cshtml`, how do you output a section that the current view can supply content for, and how does the view supply the contents for that section?

   Answer: To output the content of a section in a shared layout, call the `RenderSection` method, specifying a name for the section if it is required, as shown in the following markup:

   ```
   @RenderSection("Scripts", required: false)
   ```

   To define the contents of the section in the view, create a named section, as shown in the following markup:

   ```
   @section Scripts
   {
   <script>
     alert('Hello, Mr. Page!');
   </script>
   }
   ```

6. When calling the `View` method inside a controller's action method, what paths are searched for the view by convention?

   Answer: When calling the `View` method inside a controller's action method, three paths are searched for the view by default, based on combinations of the names of the controller and action method and a special `Shared` folder, as shown in the following example output:

   ```
   InvalidOperationException: The view 'Index' was not found. The following
   locations were searched:
   /Views/Home/Index.cshtml
   /Views/Shared/Index.cshtml
   /Pages/Shared/Index.cshtml
   ```

   This can be generalized as follows:

   - `/Views/[controller]/[action].cshtml`
   - `/Views/Shared/[action].cshtml`
   - `/Pages/Shared/[action].cshtml`

7. How can you instruct the visitor's browser to cache the response for 24 hours?

   Answer: To instruct the visitor's browser to cache the response for 24 hours, decorate the controller class or action method with the `[ResponseCache]` attribute, and set the `Duration` parameter to `86400` seconds and the `Location` parameter to `ResponseCacheLocation.Client`.

8. Why might you enable Razor Pages even if you are not creating any yourself?

Answer: If you have used features like ASP.NET Core Identity UI, then they requires Razor Pages.

9. How does ASP.NET Core MVC identify classes that can act as controllers?

   Answer: ASP.NET Core MVC identifies classes that can act as controllers by looking to see if the class (or a class that it derives from) is decorated with the `[Controller]` attribute.

10. In what ways does ASP.NET Core MVC make it easier to test a website?

    Answer: The **Model-View-Controller** (**MVC**) design pattern separates the technical concerns. These consist of:

    - The shape of the data (model).
    - The executable statements to process the incoming request and outgoing response (controller).
    - The generation of the response in a format requested by the user agent like HTML or JSON (view).

    This makes it easier to write unit tests. ASP.NET Core also makes it easy to implement the **Inversion-of-Control** (**IoC**) and **dependency injection** (**DI**) design patterns to remove dependencies when testing a component like a controller.

Chapter 15 — Building and Consuming Web Services

1. Which class should you inherit from to create a controller class for an ASP.NET Core Web API service?

   Answer: To create a controller class for an ASP.NET Core Web API service, you should inherit from `ControllerBase`. Do not inherit from `Controller` as you would in MVC, because this class includes methods like `View` that use Razor files to render HTML that is not needed for a web service.

2. When configuring an HTTP client, how do you specify the format of data that you prefer in the response from the web service?

   Answer: When configuring an HTTP client, you specify the format of the response that you prefer by adding an `Accept` header to the HTTP request that specifies the document format you prefer, and if you specify multiple and want to give them different weights, then set quality values between 0.0 and 1.0, as shown highlighted in the following code:

```
builder.Services.AddHttpClient(name: "Northwind.WebApi",
  configureClient: options =>
  {
    options.BaseAddress = new Uri("https://localhost:5002/");
    options.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue(
      mediaType: "application/json", quality: 1.0));
  });
```

3. What must you do to specify which controller action method will be executed in response to an HTTP request?

   Answer: To specify which controller action method will be executed in response to a request, you must decorate the action method with an attribute. For example, to respond to an HTTP `POST` request, decorate the action method with `[HttpPost]`.

4. What must you do to specify what responses should be expected when calling an action method?

   Answer: To specify what responses should be expected when calling an action method, decorate the action method with the `[ProducesResponseType]` attribute, as shown in the following code:

```
// GET: api/customers/[id]
[HttpGet("{id}", Name = nameof(Get))] // named route
[ProducesResponseType(200, Type = typeof(Customer))]
[ProducesResponseType(404)]
public IActionResult Get(string id)
{
```

5. List three methods that can be called to return responses with different status codes.

   Answer: Three methods that can be called to return responses with different status codes include:

   - `Ok`: This returns the `200` status code and the object passed to this method in the body.
   - `CreatedAtRoute`: This returns the `201` status code and the object passed to this method in the body.
   - `NoContentResult`: This returns the `204` status code and an empty body.
   - `BadRequest`: This returns the `400` status code and an optional error message.
   - `NotFound`: This returns the `404` status code and an optional error message.

6. List four ways that you can test a web service.

   Answer: Four ways that you can test a web service include:

   - Using a browser to test simple HTTP `GET` requests.
   - Installing the REST Client extension for Visual Studio Code.
   - Installing the Swagger NuGet package in your web service project, enabling Swagger, and then using the Swagger testing user interface.
   - Installing the Postman tool from the following link: `https://www.postman.com`.

I listed the four above because they are techniques that I covered or mentioned in the chapter. There are, of course, many other ways to test a web service, for example, Curl. Give yourself an extra point for any valid additional methods beyond the ones I have listed.

7. Why should you not wrap your use of `HttpClient` in a `using` statement to dispose of it when you are finished even though it implements the `IDisposable` interface, and what should you use instead?

   Answer: `HttpClient` is shared, reentrant, and partially thread-safe, so it is tricky to use correctly in many scenarios. You should use `HttpClientFactory`, which was introduced in .NET Core 2.1.

8. What are the benefits of HTTP/2 and HTTP/3 compared to HTTP/1.1?

   Answer: HTTP/2 benefits from full multiplexing that reduces latency, support for request prioritization, and minimization of overhead in the protocol using header compression. HTTP/3 benefits from being based on UDP rather than TCP, so any packet loss does not block all streams. HTTP/3 also has 0-RTT support, which means subsequent connections do not repeat the TLS acknowledgment, so the client can start requesting data faster.

9. How can you enable clients to detect if your web service is healthy with ASP.NET Core 2.2 and later?

   Answer: To enable clients to detect if your web service is healthy, you can install health check APIs including database health checks for Entity Framework Core data contexts. Health checks can be extended to report detailed information back to the client.

10. What benefits does endpoint routing provide?

    Answer: Endpoint routing provides improved performance of routing and action method selection, and a link generation service.

Chapter 16 — Building User Interfaces Using Blazor

1. What are the two primary hosting models for Blazor, and how are they different?

   Answer: The two primary hosting models for Blazor are Server and WebAssembly:

   - Blazor Server executes code on the server side, which means the code has full and easy access to server-side resources like databases. This can simplify implementing functionality. UI updates are made using SignalR, which means a permanent connection is needed between the browser and server, which limits scalability.
   - Blazor WebAssembly executes code on the client side, which means the code only has access to resources within the browser. This can complicate the implementation because a callback to the server must be made whenever new data is required.

2. In a Blazor Server website project, compared to an ASP.NET Core MVC website project, what extra configuration is required?

   Answer: In the section that configures services you must call `AddServerSideBlazor`, and in the section that configures the HTTP pipeline you must call `MapBlazorHub` and `MapFallbackToPage` when setting up endpoints.

3. One of the benefits of Blazor is being able to implement client-side components using C# and .NET instead of JavaScript. Does a Blazor component need any JavaScript?

   Answer: Yes, Blazor components need some minimal JavaScript. For Blazor Server this is provided by the file `_framework/blazor.server.js`. For Blazor WebAssembly this is provided by the file `_framework/blazor.webassembly.js`. Blazor WebAssembly with PWA also uses a JavaScript service worker file, `service-worker.js`. JavaScript is also needed to invoke browser and other client-side APIs, like getting the current geolocation or interacting with browser storage and alert dialog boxes.

4. In a Blazor website project, what does the `App.razor` file do?

   Answer: The `App.razor` file configures a `Router` used by all Blazor components in the current assembly. For example, it sets a default shared layout for components that match a route and a view to use when no match is found.

5. What is the main benefit of using the `<NavLink>` component?

   Answer: The main benefit of using the `<NavLink>` component is that it integrates with the Blazor routing system, so it can automatically apply a current style to visually indicate when the current route matches the `<NavLink>` component.

6. How can you pass a value into a component?

   Answer: You can pass a value into a component by decorating a public property in the component with the `[Parameter]` attribute and then setting the attribute in the component when using it, as shown in the following code:

```
// defining the component
@code {
  [Parameter]
  public string ButtonText { get; set; };
}

// using the component
<CustomerDetail ButtonText="Create Customer" />
```

7. What is the main benefit of using the `<EditForm>` component?

   Answer: The main benefit of using the `<EditForm>` component is automatic validation messages.

8. How can you execute some statements when parameters are set?

   Answer: You can execute some statements when parameters are set by defining an `OnParametersSetAsync` method to handle that event.

9. How can you execute some statements when a component appears?

   Answer: You can execute some statements when a component appears by defining an `OnInitializedAsync` method to handle that event.

10. What are two key differences during initialization between a Blazor Server and Blazor WebAssembly project?

    Answer: Two key differences during initialization between a Blazor Server and Blazor WebAssembly project are: (1) the use of `WebAssemblyHostBuilder` instead of `Host.CreateDefaultBuilder`, and (2) the registering of an `HttpClient` with a base address of the host environment.