Answer 1:

A person’s telephone number: string

A person’s height: float or double

A person’s age: int

A person’s gender (Male, Female, Prefer Not To Answer): enum or String

A person’s salary: decimal

A book’s ISBN: string

A book’s price: decimal

A book’s shipping weight: float or double

A country’s population: long

The number of stars in the universe: double

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

Answer 2:

| Feature | Value Type | Reference Type |
| --- | --- | --- |
| Storage location | Stored directly on the stack | Stored on the heap (reference stored on the stack) |
| Contains | The actual value | A reference (memory address) to the value |
| Examples | int, float, bool, char, struct, enum | string, class, array, interface, object |
| Copy behavior | Copying creates a new independent copy of the value | Copying creates a reference to the same memory location |
| Null allowed? | No (unless nullable, e.g., int?) | Yes |
| Performance | Faster (stack-based, auto-removed after scope) | Slower (heap-based, needs garbage collection) |

Boxing and Unboxing

* Boxing: Converting a value type into an object (reference type).  
  The CLR wraps the value inside a heap-allocated object.

int num = 10;

object obj = num; // Boxing

* Unboxing: Extracting the value type back from an object.

int num2 = (int)obj; // Unboxing

Answer 3:

In .NET, managed resources are the resources that are handled automatically by the Common Language Runtime (CLR), such as memory used by objects created in C# (like strings, arrays, and classes). The Garbage Collector takes care of allocating and releasing this memory.  
On the other hand, unmanaged resources are resources that the CLR does not control, such as file handles, database connections, or network sockets. These must be released manually using the Dispose() method or the using statement to avoid memory leaks and performance issues.

Answer 4:

to free up memory occupied by objects that are no longer in use, preventing memory leaks. It helps improve performance by reclaiming unused memory, compacting the heap, and optimizing memory allocation allowing developers to focus on application logic instead of manual memory management.

Answer 5:

class Program

{

static void Main()

{

Console.WriteLine("Welcome to the Hacker Name Generator!");

Console.Write("Enter your favorite color: ");

string color = Console.ReadLine();

Console.Write("Enter your astrology sign: ");

string sign = Console.ReadLine();

Console.Write("Enter your street address number: ");

string addressNumber = Console.ReadLine();

string hackerName = color + sign + addressNumber;

Console.WriteLine($"Your hacker name is {hackerName}!");

}

}

Answer 6:

using System;

class Program1

{

static void Main()

{

Console.Write("Enter number of centuries: ");

int centuries = int.Parse(Console.ReadLine());

int years = centuries \* 100;

int days = (int)(years \* 365.2422);

long hours = days \* 24L;

long minutes = hours \* 60L;

long seconds = minutes \* 60L;

long milliseconds = seconds \* 1000L;

long microseconds = milliseconds \* 1000L;

decimal nanoseconds = microseconds \* 1000m;

Console.WriteLine($"{centuries} centuries = {years} years = {days} days = " +

$"{hours} hours = {minutes} minutes = {seconds} seconds = " +

$"{milliseconds} milliseconds = {microseconds} microseconds = " +

$"{nanoseconds} nanoseconds");

}

}

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

It causes a runtime error — specifically a DivideByZeroException — because integer division by zero is undefined.

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

It does not throw an exception. Instead, it results in Infinity, -Infinity, or NaN (Not a Number), depending on the numerator.

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

If overflow checking is disabled (default), the value wraps around to the negative or positive end of the range.  
If overflow checking is enabled using the checked keyword, it throws an OverflowException.

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

* x = y++; → Post-increment: assign y to x, then increase y by 1.
* x = ++y; → Pre-increment: increase y by 1, then assign it to x.

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

* break → exits the current loop immediately.
* continue → skips the rest of the current iteration and moves to the next loop cycle.
* return → exits the entire method, not just the loop.

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

The three parts are:

1. Initialization (executed once)
2. Condition (checked before each iteration)
3. Iteration (executed after each iteration)  
   Only the condition is required; the others are optional.

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

* = → Assignment operator, assigns a value to a variable.
* == → Equality operator, compares two values for equality.

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

Yes, it compiles successfully.  
It creates an infinite empty loop that does nothing but runs indefinitely.

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

The underscore \_ acts as a default case or catch-all pattern that matches any value not handled by previous cases.

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

An object must implement the IEnumerable interface (or IEnumerable<T> for generic collections).

Practice loops and operators:

Answer 1:

using System;

class Program

{

static void Main()

{

for (int i = 1; i <= 100; i++)

{

if (i % 3 == 0 && i % 5 == 0)

{

Console.WriteLine("fizzbuzz");

}

else if (i % 3 == 0)

{

Console.WriteLine("fizz");

}

else if (i % 5 == 0)

{

Console.WriteLine("buzz");

}

else

{

Console.WriteLine(i);

}

}

Console.WriteLine("\nGame Over!");

}

}

What Happens When This Code Runs

When you run this program:

* byte can only store values from 0 to 255.
* Once i reaches 255, the next increment (i++) causes an overflow, and it wraps back to 0.
* The condition i < max (i.e., i < 500) always remains true, so the loop never ends.

How to Warn About the Problem (Without Changing the Existing Code)

You cannot modify the existing code, but you can add a warning check inside the loop body:

int max = 500;

for (byte i = 0; i < max; i++)

{

Console.WriteLine(i);

if (i == 255)

{

Console.WriteLine("Byte value about to overflow!");

break; // stop before overflow happens

}

}

using System;

class Exercise03\_GuessNumber

{

static void Main()

{

Console.WriteLine("I'm thinking of a number between 1 and 3.");

int correctNumber = new Random().Next(3) + 1;

Console.Write("Enter your guess: ");

int guessedNumber = int.Parse(Console.ReadLine());

if (guessedNumber < 1 || guessedNumber > 3)

{

Console.WriteLine("❌ Out of range! Please enter a number between 1 and 3.");

}

else if (guessedNumber < correctNumber)

{

Console.WriteLine("Too low! Try again next time.");

}

else if (guessedNumber > correctNumber)

{

Console.WriteLine("Too high! Try again next time.");

}

else

{

Console.WriteLine("Correct! You guessed the number!");

}

Console.WriteLine($"\n(The correct number was: {correctNumber})");

}

}

Print Pyramid:

using System;

class Pattern

{

static void Main()

{

Console.WriteLine("Enter the number of rows: ");

int rows = int.Parse(Console.ReadLine());

for (int i = 1; i <= rows; i++)

{

for (int j = i; j < rows; j++)

Console.Write(" ");

for (int k = 1; k <= (2 \* i - 1); k++)

Console.Write("\*");

Console.WriteLine();

}

}

}

Answer 3:

using System;

class Exercise03\_GuessNumber

{

static void Main()

{

Console.WriteLine("I'm thinking of a number between 1 and 3.");

int correctNumber = new Random().Next(3) + 1;

Console.Write("Enter your guess: ");

int guessedNumber = int.Parse(Console.ReadLine());

if (guessedNumber < 1 || guessedNumber > 3)

{

Console.WriteLine("Out of range! Please enter a number between 1 and 3.");

}

else if (guessedNumber < correctNumber)

{

Console.WriteLine("Too low! Try again next time.");

}

else if (guessedNumber > correctNumber)

{

Console.WriteLine("Too high! Try again next time.");

}

else

{

Console.WriteLine("Correct! You guessed the number!");

}

Console.WriteLine($"\n(The correct number was: {correctNumber})");

}

}

Answer 4:

using System;

class Program4

{

static void Main()

{

DateTime birthDate = new DateTime(2000, 1, 1);

DateTime today = DateTime.Now;

TimeSpan age = today - birthDate;

int daysOld = (int)age.TotalDays;

Console.WriteLine($"You are {daysOld} days old.");

int daysToNextAnniversary = 10000 - (daysOld % 10000);

DateTime nextAnniversary = today.AddDays(daysToNextAnniversary);

Console.WriteLine($"Your next 10,000-day anniversary will be on {nextAnniversary:d}.");

}

}

Answer 5:

using System;

class Program5

{

static void Main()

{

DateTime currentTime = DateTime.Now;

int hour = currentTime.Hour;

Console.WriteLine($"Current time: {currentTime:T}");

if (hour >= 5 && hour < 12)

Console.WriteLine("Good Morning");

if (hour >= 12 && hour < 17)

Console.WriteLine("Good Afternoon");

if (hour >= 17 && hour < 21)

Console.WriteLine("Good Evening");

if (hour >= 21 || hour < 5)

Console.WriteLine("Good Night");

}

}

Answer 6:

using System;

class Program6

{

static void Main()

{

Console.WriteLine("Counting up to 24 with different increments:");

for (int outer = 1; outer <= 4; outer++)

{

for (int i = 0; i <= 24; i += outer)

{

if (i < 24)

Console.Write(i + ",");

else

Console.Write(i);

}

Console.WriteLine();

}

}

}