**1. What type would you choose for the following “numbers”?**

**A person’s telephone number: String**

**A person’s height: double**

**A person’s age: int /int 16**

**A person’s gender (Male, Female, Prefer Not To Answer): Enum**

**A person’s salary: decimal**

**A book’s ISBN: string**

**A book’s price: decimal**

**A book’s shipping weight: double/ float**

**A country’s population: int**

**The number of stars in the universe: long / ulong**

**The number of employees in each of the small or medium businesses in the**

**United Kingdom (up to about 50,000 employees per business): ushort**

**2. What are the difference between value type and reference type variables? What is**

**boxing and unboxing?**

**Value type:** are created using structure or enum or char, bool, float, int

Cannot accept null

Removed automatically once not being used

Stored in stack memory

**Reference type:** are class or interface or arrays or strings

Stores memory location

Accept null

Removed by garbage collector

Stored in heap

Boxing: process of converting a value type to the reference type object

* It creates a new object on the heap
* Copies the value into that object
* Can impact performance if done frequently

Unboxing: extracting the value type from a boxed object

* Process of extracting the value type from a boxed object
* Requires explicit type casting
* Can throw InvalidCastException if types don’t match
* And has a performance overhead

**3. What is meant by the terms managed resource and unmanaged resource in .NET**

**Managed resource:**

Directly controlled by the .NET runtime

Memory is automatically managed through garbage collection

No explicit cleanup code required

EX: classes, collection, managed database connection, dataset obj, and memory stream

**Unmanaged resource:**

Resource outside CLR’s control

Must be explicitly released/ cleaned up

Should use IDisposal pattern

Wrapped in a Safehandle or similar classes   
EX: File handle, network socket, database connections, windows handles.

**4. What’s the purpose of Garbage Collector in .NET?**

Automatic memory management:  
automatically frees memory used by unused objects

Handles memory allocation and release

Helps prevent memory leaks

Manages heap fragmentation

**Controlling Flow and Converting Types:  
1. What happens when you divide an int variable by 0?**

Divide by zero exception and doesn’t run and at runtime, so runtime error not compiled.

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

Doesn’t throw an error when you divide a double by 0, but return special values

Will return ∞ when divided by a positive number

Will return -∞ when divided by a negative number

Will return NAN if 0/0

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

**range?**

It will throw overflow exception, ex:   
int max = int.MaxValue; // 2147483647

max = max+1; // -2147483648

checked{ // using checked block will throw overflowexeption

int x = int.MaxValue;

x = x+2;

}

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

**Int y =5;**

X = y++ (assign then increment) x = ++y (increment then assing)

// 5 = 6 6 = 6

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

**statement?**

Break : exit the loop

Continue: skip the rest of the current iteration

Return: Exist the entire loop / method

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

for (initializer; condition; iterator)

**i=0. i < 5 i++**

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

= is an assignment operator

== is a equality operator, compares values and return a Boolean   
**8. Does the following statement compile? for ( ; true; ) ;**

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

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

Must be IEnumerable to be used in a foreach

**What will happen if this code executes?  
int max = 500;  
for (byte i = 0; i < max; i++)  
{  
WriteLine(i);  
}  
Create a console application and enter the preceding code. Run the console application  
and view the output. What happens?  
What code could you add (don’t change any of the preceding code) to warn us about the problem?**

This code will create an infinite loop because byte holds values 0 to 255

When it reaches 255 the counter will overflow to 0 creating an infinite loop

To warn us we could add:

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

**}**

02 Array and strings

**When to use String vs. StringBuilder in C#**

String is immutable but string builder is mutable

We use string when operations are minimal and memory efficiency isn’t critical because every time we change or append to the string, we are creating entirely new strings and leave the old one to the garbage collector.

On the other hand, stringbuilder we change the value and not the memory location

Ex:

string s = ""; for(int i = 0; i < 10000; i++) { s += i; *// Creates 10000 strings* }

StringBuilder sb = new StringBuilder();

for(int i = 0; i < 10000; i++) {

sb.Append(i); *// Uses same memory buffer* }

**What is the base class for all arrays in C#?**

The base class of all arrays is **System.Array**

int[] numbers = {1, 2, 3};

Array arrayBase = numbers; *// Implicit conversion to base class* Console.WriteLine(arrayBase.Length); *// Outputs: 3*

**How do you sort an array in C#?**

We use sort:

int[] numbers = {1, 2, 3};

Array.**Sort**(numbers); // Modifies original array

Create a new array:

Var sorted = numbers.OrderBy(x=>x).ToArray();

**What property of an array object can be used to get the total number of elements in an array?**

.Length

int[] numbers = { 1, 2, 3, 4, 5 };

Console.WriteLine(numbers.Length); *// Outputs: 5*

**Can you store multiple data types in System.Array?**

No, a single array can only store elements of the same type.

**What’s the difference between the System.Array.CopyTo() and System.Array.Clone()?**

System.Array.CopyTo():

Copies to an existing array

Can specify start index

int[] source = {1, 2, 3};

int[] dest = new int[3];

source.CopyTo(dest, 0);

System.Array.Clone():

Create a new array instance

Creates a shallow copy

Return object (require casting)

int[] original = {1, 2, 3};

int[] cloned = (int[])original.Clone();