what is hexadecimal/decimal/binary

🔢 1. Decimal (Base 10)

- **This is the number system we use every day.**
- It uses **10 digits**: 0 1 2 3 4 5 6 7 8 9
- Each digit is multiplied by a power of **10**.

🧠 Example:

$345 = 3\times10^2 + 4\times10^1 + 5\times10^0 = 300 + 40 + 5$

💻 2. Binary (Base 2)

- **Used by computers.**
- Only uses **2 digits**: 0 and 1
- Each digit is multiplied by a power of **2**.

🧠 Example:

Binary: 1011

$= 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$

$= 8 + 0 + 2 + 1 = 11$ (Decimal)

🧮 3. Hexadecimal (Base 16)

- Often used in **programming**, **memory addresses**, **colors**.
- Uses **16 symbols**: 0 1 2 3 4 5 6 7 8 9 A B C D E F
  (A=10, B=11, …, F=15)
- Each digit is multiplied by a power of **16**.

🧠 Example:

Hex: 2F

$= 2\times16^1 + F\times16^0$

$= 2\times16 + 15\times1$

= 32 + 15 = 47 (Decimal)

🔁 Conversion Summary

Decimal	Binary	Hex
10	1010	A
15	1111	F
16	10000	10
255	11111111	FF

🧑‍💻 Usage in Real Life

System	Where Used
Decimal	Human counting
Binary	Computer processors, logic
Hexadecimal	Memory, Colors (#FF5733), Assembly code

array of int / array of objects(strings)

✅ 1. Array of Integers (int[])

📌 Definition:

Stores integers, which are value types.

🔤 Example:

```
int[] numbers = { 10, 20, 30, 40 };
```

🔍 Accessing elements:

```
Console.WriteLine(numbers[0]);  // Output: 10
```

💡 Notes:

- Efficient in memory.
- All elements must be integers.
- Cannot store null (unless it's int? or object[]).

✅ 2. Array of Objects (e.g., Strings)

📌 Definition:

Can store reference types, like strings or even mixed object types.

🔤 Example with strings:

```
string[] names = { "Alice", "Bob", "Charlie" };
```

🔤 Example with objects:

```
object[] things = { 42, "Hello", true, 3.14 };
```

🔍 Accessing elements:

```
Console.WriteLine(names[1]);   // Output: Bob
Console.WriteLine(things[0]);  // Output: 42
```

💡 Notes:

- More flexible but uses more memory.
- Can store null values.
- If declared as object[], can mix types.

🔄 Comparison Table

Feature	int[]	string[] / object[]
Type	Value type (int)	Reference types (string or any type)
Memory	More efficient	Less efficient (due to references)
Null values	Not allowed (unless nullable)	Allowed
Usage	Fixed-type numeric data	Text or mixed-type data

✅ Example: Mixed Use

```
object[] mixed = { 1, "hello", 3.5, true };
foreach (object item in mixed)
{
  Console.WriteLine(item);
}
```

jagged array

🔷 What is a Jagged Array?

A jagged array is an array of arrays, where each inner array can have a different length.

🔁 Think of it like:

```
[
 [1, 2],
 [3, 4, 5],
 [6]
]
```

So unlike a 2D array (which is a grid with fixed dimensions), jagged arrays allow rows of different sizes.

✅ Declaration and Initialization

1. Declare a jagged array:

```
int[][] jagged = new int[3][];
```

This declares an array with 3 rows, but each row isn’t initialized yet.

---

2. Initialize each row separately:

```
jagged[0] = new int[] { 1, 2 };
jagged[1] = new int[] { 3, 4, 5 };
jagged[2] = new int[] { 6 };
```

🖨 Print all elements using nested loops:

```
for (int i = 0; i < jagged.Length; i++)
{
  for (int j = 0; j < jagged[i].Length; j++)
  {
    Console.Write(jagged[i][j] + " ");
  }
  Console.WriteLine();
}
```

🔍 Differences: Jagged Array vs 2D Array

Feature	Jagged Array (int[][])	2D Array (int[,])
Structure	Array of arrays	Single grid with fixed size
Flexible lengths	✅ Each row can have different length	❌ All rows have same length
Syntax to access	arr[i][j]	arr[i, j]

🧠 Use Cases of Jagged Arrays:

- Storing student grades for different subjects (some students take more subjects).
- Variable-length data like triangle matrices.
- Optimizing memory when rows differ in size.

✅ Full Example:

```
using System;
class Program
{
    static void Main()
    {
        int[][] jagged = new int[3][];
        jagged[0] = new int[] { 1, 2 };
        jagged[1] = new int[] { 3, 4, 5 };
        jagged[2] = new int[] { 6 };

        for (int i = 0; i < jagged.Length; i++)
        {
            Console.Write($"Row {i}: ");
            for (int j = 0; j < jagged[i].Length; j++)
            {
                Console.Write(jagged[i][j] + " ");
            }
            Console.WriteLine();
        }
    }
}
```