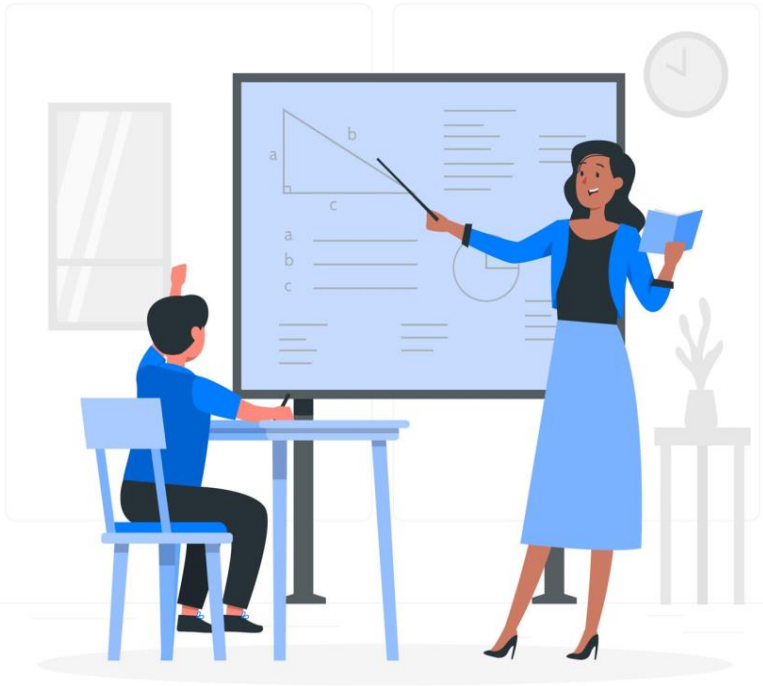
C#

# Array

Arrays are like organized shelves, helping us store and manage multiple items of the same type.

Syntax of defining an array

```
data_type[] array_name = new data_type[size];
```



# Initializing Arrays.

Once an array is declared, the next step is to initialize an array.

The initialization process of an array includes adding actual data to the array.

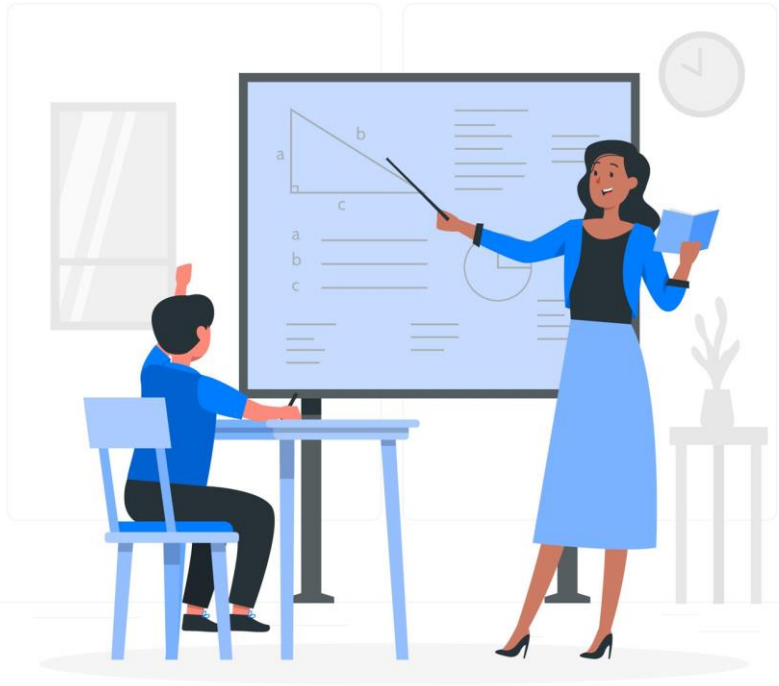
First:

```
// Initialize a fixed array
int[] FixedArray = new int[3] {1, 3, 5};


// Initialize a dynamic array items during declaration


string[] jobs= new string[] { "Manager", "Developer",
"Accountant", "Marketting};
```

# Array Types

Arrays can be divided into the following categories.

- ✓Single-dimensional arrays
- ✓Multidimensional arrays or rectangular arrays
- ✓Jagged arrays



# Single-dimensional arrays

Single-dimensional array is a collection of elements of the same type, arranged in a sequential manner. It is the most basic form of an array and can be declared and used as follows:

```
// Example: Creating an integer array with 4 elements
string[] roles= new string[4];

// Assigning values to array elements
roles[0] = "CompanyMember";
roles[1] = "JobSeeker";
roles[2] = "JobProvider";
roles[3] = "Admin";

// Iterating over array elements using a loop
for (int i = 0; i < roles.Length; i++)
{
    Console.WriteLine(roles[i]);
}
Console.ReadLine();
```

```
CompanyMember
JobSeeker
JobProvider
Admin
```

# Multi-dimensional arrays

The multidimensional array is also known as rectangular arrays in C#. It can be two dimensional or three dimensional. The data is stored in tabular form (row * column) which is also known as matrix.

```
//declaration of 2D array
string[,] roles = new string[2, 2];

roles[0, 0] = "JobProvider";              //initialization
roles[0, 1] = "Admin";
roles[1, 0] = "JobSeeker";
roles[1, 1] = "CompanyMember";

//traversal
for (int i = 0; i < 2; i++)
{
   for (int j = 0; j <2; j++)
   {
     Console.Write(roles[i,j] + " ");
    }
    Console.WriteLine();          //new line at each row
}
Console.ReadLine();
```

```
JobProvider Admin
JobSeeker CompanyMember
```

# Jagged Arrays

In C#, jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different

```
// Declare the array
string[][] roles = new string[2][];

// Initialize the array
roles[0] = new string[] {"JobProvider" };
roles[1] = new string[] { "JobSeeker",
                "JobProvider","CompanyMember" };

// Traverse array elements
for (int i = 0; i < roles.Length; i++)
{
    for (int j = 0; j < roles[i].Length; j++)
    {
       System.Console.Write(roles[i][j] + " ");
    }
    Console.WriteLine();
 }
Console.ReadLine();
```

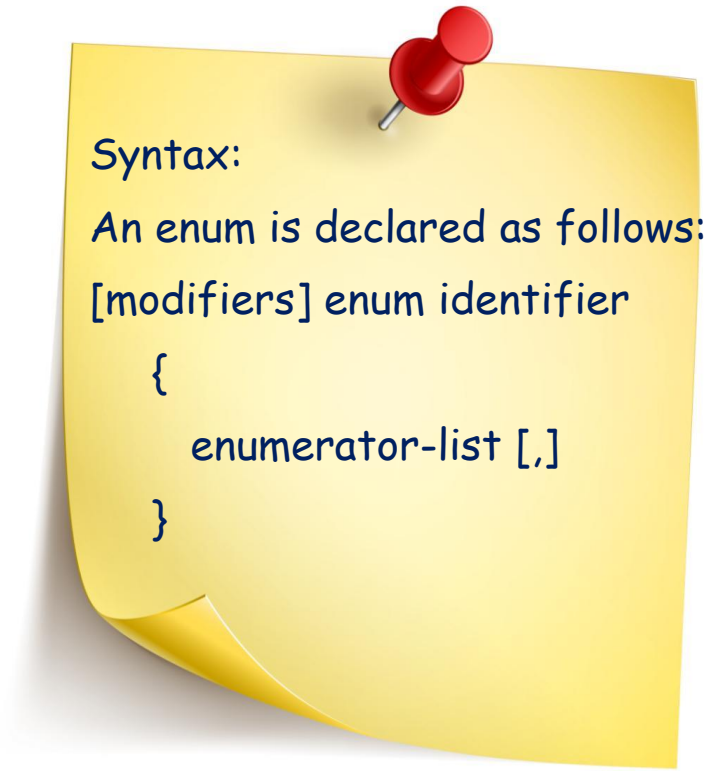
```
JobProvider
JobSeeker JobProvider CompanyMember
```

# Enum

- An enum type is a distinct value type with a set of named constants.
- The enum keyword is used to declare an enumeration.
- Enums are strongly typed constants.
- All member of enum are of enum type.
- Enums type can be integer (float, int, byte, double etc.).
- The default underlying type of the enumeration element is int.

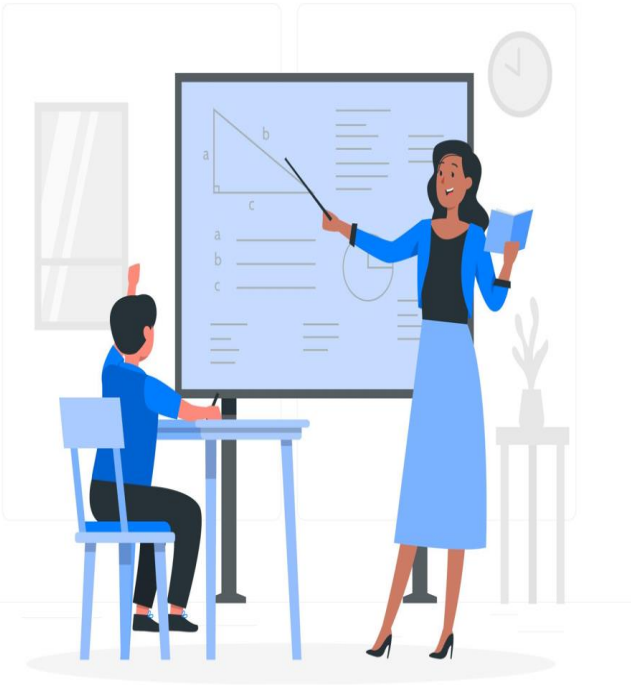
# Enum

- By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1.
- Enumerations (enums) make your code much more readable and understandable.
- Every enum type automatically derives from System.Enum

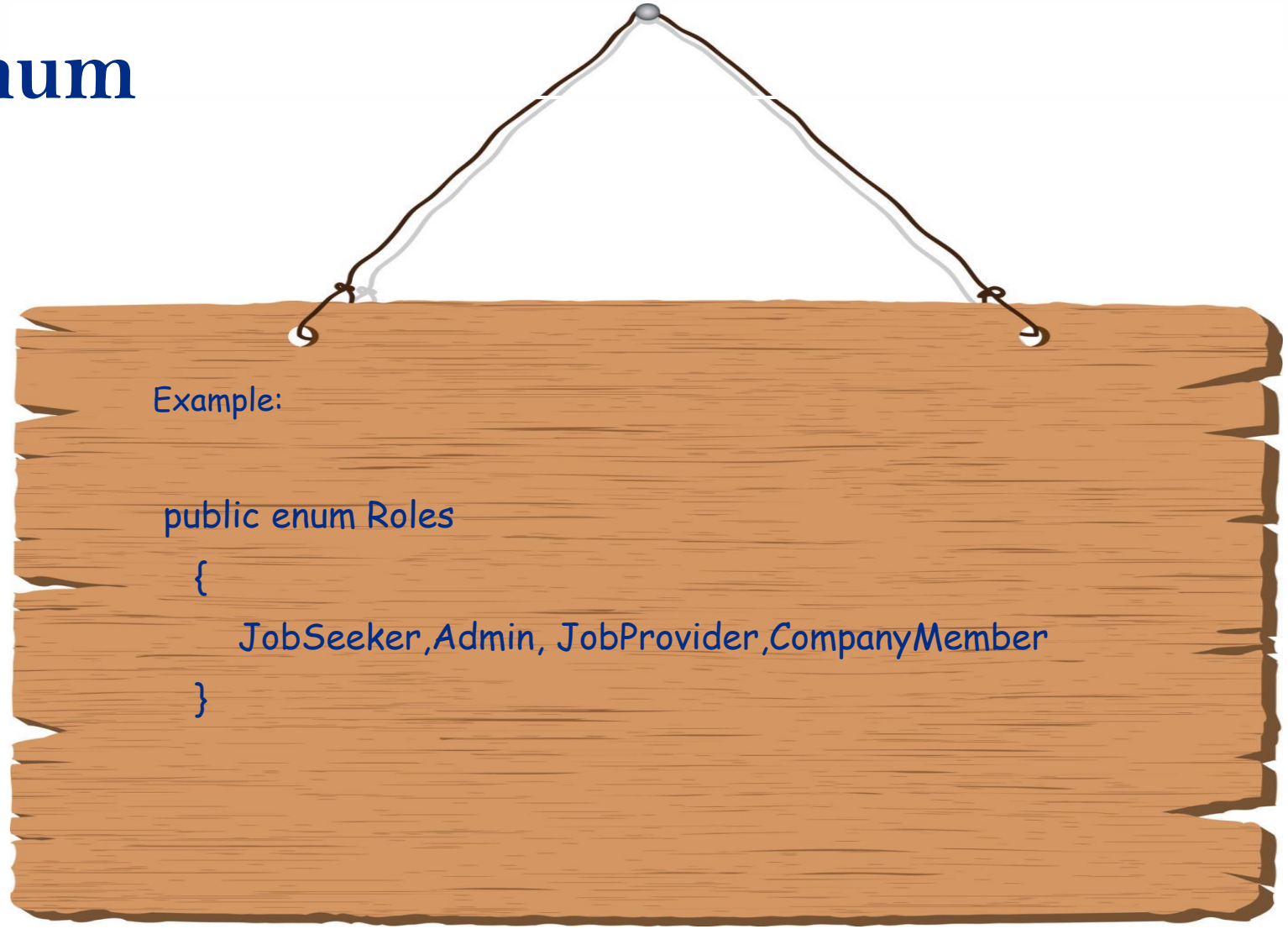


# Enum

- The attributes is optional and is used to hold additional declarative information.
- Modifiers are new, public, protected, internal and private.
- The keyword enum must be followed by an identifier that names the enum.
- The underlying type that specifies the storage allocated for each enumerator. It can be one of the integral types except char. The default is int..
- The enumerator-list contains the identifiers which are separated by commas



# Enum



Example:

```
public enum Roles
  {
      JobSeeker,Admin, JobProvider,CompanyMember
  }
```

# Type Conversions

Type conversion is converting one type of data to another type. It is also known as Type Casting.
In C# Type casting has two forms

1. Implicit type conversion
2. Explicit type conversion

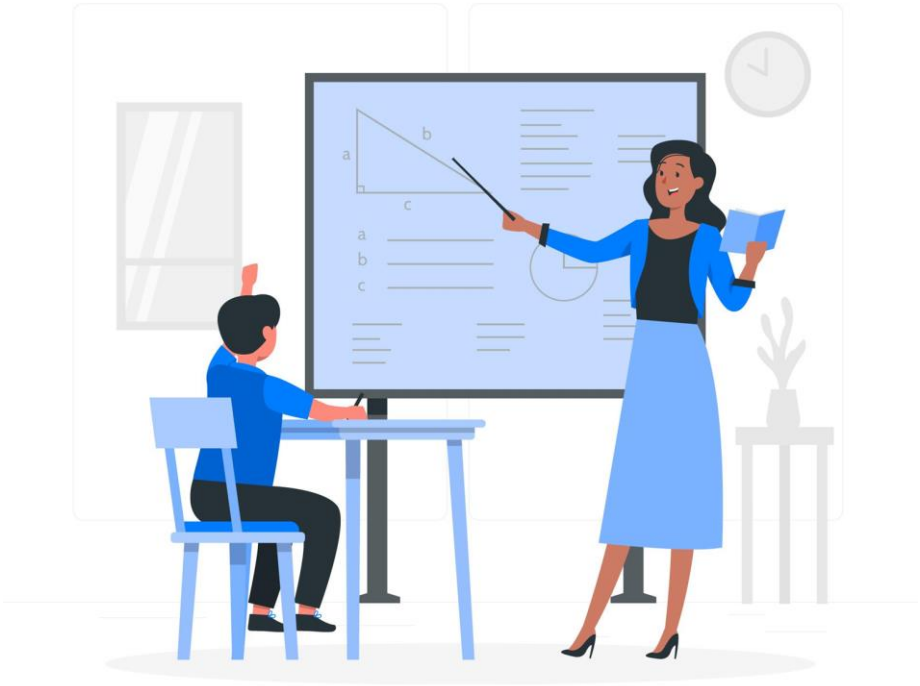
Implicit Casting (automatically) - converting a smaller type to a larger type size
char -> int -> long -> float -> double

Explicit Casting (manually) - converting a larger type to a smaller size type
double -> float -> long -> int -> char

# Implicit type conversion

- conversions from smaller to larger integral types

```
int salary= 250000;
long sal= salary;
// implicit conversion from int type to long type
```



# Explicit type conversion

Explicit type conversion:- These conversions are done explicitly by users using the pre-defined functions.

Example:

```
double myDouble = 9.78;
int myInt = (int) myDouble;

Console.WriteLine(myDouble);   // Outputs 9.78
Console.WriteLine(myInt);      // Outputs 9
```

```
class Program
 {
   static void Main(string[] args)
   {
     int myInt = 10;
     double myDouble = 5.25;
     bool myBool = true;

     Console.WriteLine(Convert.ToString(myInt));    // Convert int
to string
     Console.WriteLine(Convert.ToDouble(myInt));    // Convert int
to double
     Console.WriteLine(Convert.ToInt32(myDouble));  // Convert
double to int
     Console.WriteLine(Convert.ToString(myBool));   // Convert bool
to string
     }
 }
```

# Structures

A structure is a value type data type. It helps you to make a single variable hold related data of various data types. The struct keyword is used for creating a structure.

Structures are used to represent a record.

For example, declare the job structure

```
struct Books {
                public string title;
                public string Name;
                public int Salary;
             };
```

# Conclusion

As we conclude Chapter 3, we've navigated through pivotal concepts in C# programming, expanding our toolkit for effective coding.

- **Arrays:**

Arrays provide a powerful way to store and manage collections of data. They enhance our ability to work with multiple values efficiently.

- **Different Types of Arrays:**

We explored various array types, from single-dimensional to multidimensional arrays. This versatility equips us to handle diverse data structures.

- **Enums:**

Enums bring clarity to our code by allowing us to define named integral constants. This enhances code readability and makes it more maintainable.

- **Type Conversions:**

Understanding type conversions is essential for handling data of different types. We explored implicit and explicit conversions, ensuring smooth interaction between different data types.

# Thank You