C# TIP

C# Pattern Matching

POORNA SOYSA

SWIPE

C# TIP

Declaration Pattern

Declaration pattern was introduced in **C# 7**, which is a powerful tool for concisely type-checking and destructuring values in conditional statements. It allows checking the run-time type of an expression and, if a match succeeds, assigns the expression result to a declared variable.

```
if(vehicle != null
    && vehicle is Car)
{
    Car car = (Car)vehicle;
    Console.WriteLine($"No.of Passengers: {car.Passengers}");
}
```

```
if (vehicle is Car car)
{
    Console.WriteLine($"No.of Passengers: {car.Passengers}");
}
```

POORNA SOYSA

SWIPE

C# TIP

Constant Pattern

Constant pattern was introduced in **C# 7**, which tests the value of an expression against a specified constant value.

In a **Constant pattern**, can use any ***constant expression***,

- an integer or floating-point numerical literal
- a char
- a string literal.
- a Boolean value true or false
- an enum value
- the name of a declared const field or local
- null

```
public static decimal CalculateToll(int passengers)
    => passengers switch
{
    1 => 12.0m,
    2 => 20.0m,
    3 => 27.0m,
    4 => 32.0m,
    _ => 0.0m,
};
```

POORNA SOYSA

SWIPE

C# TIP

Discard Pattern

Discard pattern was introduced in **C# 7**, that allows to match any expression, including null values. It's represented by the **underscore (_)** symbol.

✅ **Discard pattern** can be used in ***switch expressio***n. If do not use **Discard pattern** in the ***switch expression*** and none of the expression's patterns matches an input, the runtime throws an ***exception***.

❌ Cannot be used in ***is expression*** or ***switch statements***. In these cases, to match an expression, use a ***var pattern*** with discard: *var* ***_***.

```
static decimal CalculateTicketDiscountPrice(DayOfWeek? dayOfWeek)
    => dayOfWeek switch
{
    DayOfWeek.Monday => 0.5m,
    DayOfWeek.Tuesday => 12.5m,
    DayOfWeek.Wednesday => 7.5m,
    DayOfWeek.Thursday => 12.5m,
    DayOfWeek.Friday => 5.0m,
    DayOfWeek.Saturday => 2.5m,
    DayOfWeek.Sunday => 2.0m,
    _ => 0.0m,
};
```

POORNA SOYSA

SWIPE

C# TIP

Property Pattern

Property pattern was introduced in **C# 8,** which offers a concise and readable way to match an expression's properties or fields against nested patterns. **Property pattern** is used to match an expression if the result of the expression is not null, and if each nested pattern successfully matches the corresponding property or field of that expression result.

```
if (student != null &&
    student.Age > 18 &&
    student.Address.City == "Colombo")
{
    Console.WriteLine($"{student.Name} is eligible for the C# course.");
}
```

```
if (student is { Age: > 18, Address.City: "Colombo" })
{
    Console.WriteLine($"{student.Name} is eligible for the C# course.");
}
```

POORNA SOYSA

SWIPE

C# TIP

Positional Pattern

Positional pattern is a powerful feature introduced in **C# 8** that allows deconstructing an expression result and matching the resulting values against the corresponding nested patterns.
Positional pattern can also be used for matching ***tuple*** types.

```
public static decimal CalculateTicketDiscountPrice(int groupSize, DateTime visitDate)
    => (groupSize, visitDate.DayOfWeek) switch
{
    ( <= 0, _) => throw new ArgumentException("Group size must be positive."),
    (_, DayOfWeek.Saturday or DayOfWeek.Sunday) => 0.0m,
    ( >= 5 and < 10, DayOfWeek.Monday) => 20.0m,
    ( >= 10, DayOfWeek.Monday) => 30.0m,
    ( >= 10, _) => 15.0m,
    _ => 0.0m,
};
```

POORNA SOYSA

SWIPE

C# TIP

Relational Pattern

Relational pattern is a powerful feature introduced in **C# 9** that allows to compare the expression result with a constant using ***relational operators.*** This enables a more concise and readable way to perform conditional checks in the code.

```
public static string Classify(double measurement)
{
    if (measurement < -4.0)
        return "Too low";

    if (measurement > 10.0)
        return "Too High";

    if (double.IsNaN(measurement))
        return "Unknown";

    return "Acceptable";
}
```

```
public static string Classify(double measurement)
    => measurement switch
{
    < -4.0 => "Too low",
    > 10.0 => "Too high",
    double.NaN => "Unknown",
    _ => "Acceptable",
};
```

POORNA SOYSA

SWIPE

C# TIP

List Pattern

List pattern is a powerful feature introduced in **C# 11** that allows matching an array or list against a sequence of patterns. This enables a more concise, readable, and type-safe manner to match array or list compared to traditional ***if statements***.

```
List<int> ages = new() { 18, 22, 33, 44 };

Console.WriteLine(ages is [18, 22, 33, 44]   Output i s All Ages are matching
    ? "All Ages are matching" : "Ages are not matching");

Console.WriteLine(ages is [18, 22, 33, 44, 55] Output i s Ages are not matching
   ? "All Ages are matching" : "Ages are not matching");

Console.WriteLine(ages is [19, 22, 33, 44] Output i s Ages are not matching
 ? "All Ages are matching" : "Ages are not matching");

Console.WriteLine(ages is [18, .., > 40]  Output i s First age should be 18 and the last
     ? "First age should be 18 and last one should be greater than 40"      one should be greater than 40
     : "Ages are not matching");

Console.WriteLine(ages is [_, > 20 and < 25, _, _]
   ? "Second age should be greater than 20 and less than 25"
   : "Ages are not matching"); Output i s Second age should be greater than 20 and less
                               than 25
```

POORNA SOYSA

SWIPE

Let's spread the knowledge together!

DO YOU LIKE THIS POST?

REPOST IT!

Follow for more tips

POORNA SOYSA

Leave a comment below