Modern C# features from version 6 to 12

Every C# developer needs to know

TechWorld WithMilan
simplifying complex topics

Swipe →

C# language version history

C# 1
Events, Delegates

C# 2
Generics, Iterators

C# 3
Linq, Ext. methods

C# 4
Dynamic Programming

C# 5
Async Programming

C# 6
Static imports, interpolated strings

C# 7
Tuples, Pattern Matching

C# 8
Nullable ref. types

C# 9
Records, top-level statements

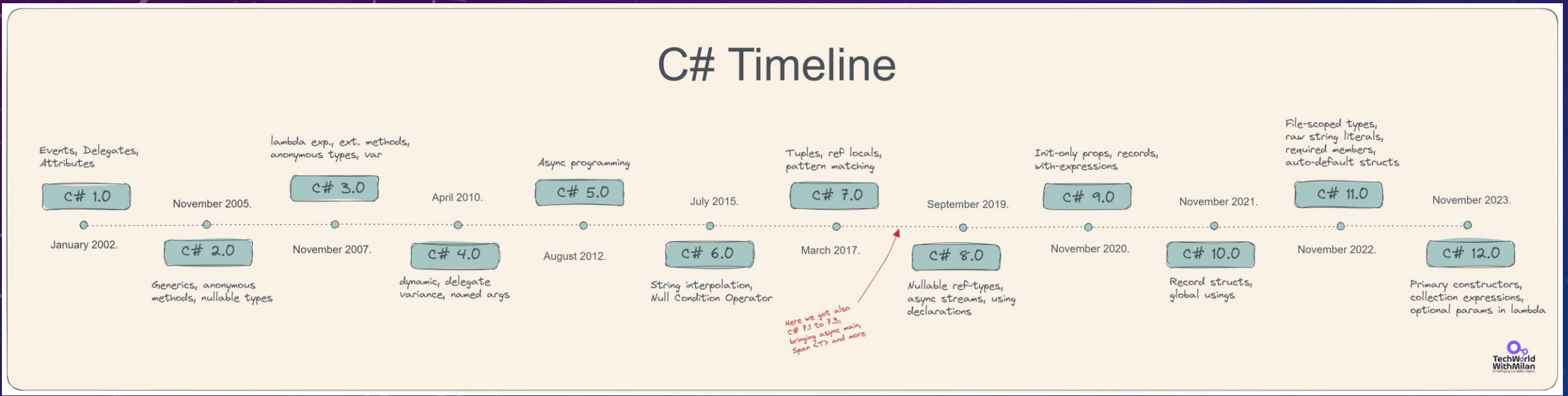
C# 10
Record Structs

C# 11
Raw String Literal

C# 12
Primary constructors

TechWorld WithMilan
simplifying complex topics

Swipe →



TechWorld WithMilan
simplifying complex topics

Swipe →

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 6 Key Features

String interpolation

Null Conditional Operator / Null Coalescing

Auto-Property Declaration & Read-Only

Collection Initializers

Expression-bodied methods and properties

Static Using

TechWorld WithMilan
simplifying complex topics

Swipe →

String Interpolation

```
int result = Add(10, 5);

Console.WriteLine("result: {0} ", result);
Console.WriteLine($"result: {result}");

// or

var aNumberAsString = $"{aDoubleValue:0.####}"
```

A possibility to replace interpolated expression by the string representation of the result

TechWorld WithMilan
simplifying complex topics

Swipe →

Null Conditional Operator / Null Coalescing

```
var address = customer?.Address;

var address = customer?.Address ?? "None";
```

Null-conditional operators provide a way to simplify null-checking code and avoid null-reference exceptions.

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 6 Features

Auto-Property Declaration & Read-Only

```
public class Person
{
    public string Name { get; private set; } = "Milan";
    public int Age { get; private set; } = 5525;

    public Person()
    {
    }
}
```

The auto-property initializer allows the assignment of properties directly within their declaration.

TechWorld WithMilan
simplifying complex topics

Swipe →

Expression-bodied methods and properties

```
public int Add(int a, int b) {
    return a + b;
}

public int Add(int a, int b) => a + b;

// or

public override string ToString() => $"{LastName}, {FirstName}";
```

The expression-bodied syntax uses the =>(fat arrow) operator to define the body of the method or property.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Collection/dictionary Initializers

```
Dictionary<string, Customer> cust = new Dictionary<string, Customer>()
{
  ["P122"] = new Customer("P122"),
  ["R200"] = new Customer("R200"),
  ["S333"] = new Customer("S333")
};
```

Collection initializers let you add items when the collection is instantiated.

TechWorld WithMilan
simplifying complex topics

Swipe →

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 7 Key Features

- Tuples / ValueTuples
- Pattern Matching
- Throw Expressions
- Local Functions
- Improving to Literals
- Out Variables
- Returning by Reference
- Async Main
- Default Literal Expressions

TechWorld WithMilan
simplifying complex topics

Swipe →

Tuples

```
public Tuple<string, int> GetStudent (string id) {
    return new Tuple<string, int>("Mike", 19);
}

public void Test() {
     Tuple<string, int> info = GetStudent ("1110-836");
     Console.WriteLine($"Name: {info.Item1}, Age: {info.Item2}");
}
```

Hold multiple values in a variable.

TechWorld WithMilan
simplifying complex topics

Swipe →

Value Tuple

```
(string name, int age) GetStudent(string id) {
    return (name: "Mike", age: 19);
}

public void Test() {
    (string name, int age) info = GetStudent("1110-836");
    Console.WriteLine($"Name: {info.name}, Age: {info.age}");
}
```

The ValueTuple structure is a value-type representation of the Tuple.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Pattern Matching

```
public static void IsPattern(object o)
{
        if (o is null) throw new ArgumentNullException(nameof(o));
}
```

Pattern matching determines if a variable is of a specific type and holds a particular value.

The const pattern

TechWorld WithMilan
simplifying complex topics

Swipe →

Pattern Matching

```
if (o is Person p)
{
    Console.WriteLine($"it's a person: {p.FirstName}");
}
```

The type pattern

TechWorld WithMilan
simplifying complex topics

Swipe →

Pattern Matching

```
if (o is var x)
{
    Console.WriteLine($"it's type {x?.GetType()?.Name}");
}
```

The var pattern

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Pattern Matching

```
public static void SwitchPattern(object o)
{
    switch (o)
    {
        case null:
            Console.WriteLine("it's a constant pattern");
            break;
        case Person p when p.FirstName.StartsWith(“Ge"):
            Console.WriteLine($"a Ge person {p.FirstName}");
            break;
        case Person p:
            Console.WriteLine($"any other person {p.FirstName}");
            break;
        // …
    }
}
```

The switch statement

TechWorld WithMilan
simplifying complex topics

Swipe →

Local Functions

```
public int Fibonacci(int x)
{
    if (x < 0) throw new ArgumentException();
    return Fib(x).current;

    (int current, int previous) Fib(int i)
    {
        if (i == 0) return (1, 0);
        var (p, pp) = Fib(i - 1);
        Console.WriteLine($"{p}");
        return (p + pp, p);
    }
}
```

Allows local methods to be defined within a method

TechWorld WithMilan
simplifying complex topics

Swipe →

Out Variables

```
if (int.TryParse(input, out var quantity))
{
    WriteLine(quantity);
}
else
{
    WriteLine("Quantity is not a valid integer!");
}

WriteLine($"{nameof(quantity)}: {quantity}");
```

Declare a variable where it is being passed as an out argument.

TechWorld WithMilan
simplifying complex topics

Swipe →

Returning By Reference

```
var i = 52;
ref var ri = ref i;     // reference to i
ri = 53;

Console.WriteLine(i);  // prints 53
```

The ref modifier has been extended with local variables and return values.

TechWorld WithMilan
simplifying complex topics

Swipe →

Async Main

```
// Before:

static void Main(string[] args)
{
    MainAsync(args).GetAwaiter().GetResult();
}

static async Task MainAsync(string[] args)
{
    // asynchronous code
}

// Now:

public static Task Main(string[] args);

public static Task<int> Main(string[] args);
```

Allow await to be used in an application's Main / entry point method

TechWorld WithMilan
simplifying complex topics

Swipe →

Default Literal Expressions

```
int numeric = default;
Object reference = default;
DateTime value = default;
T defaultValue = default;
```

Allow await to be used in an application's Main / entry point method

TechWorld WithMilan
simplifying complex topics

Swipe →

Private Protected

- A member may be accessed by containing class or derived classes declared in the same assembly.
- `protected internal` allows access by derived classes or classes in the same assembly.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Span <T>

```
string array = { "a", "b", "c" };

Span<string> span = array; // Implicit cast from T[] to Span<T>

Console.Write(span.Length);
```

Ref struct that is allocated on the stack rather than on the managed heap.

TechWorld WithMilan
simplifying complex topics

Swipe →

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

C# 8 Key Features

- Nullable Ref. Types
- Null Coalescing Operator
- Switch expressions
- Indices and ranges
- Default interface methods
- Using variables

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Nullable Reference Types

```
<LangVersion>10.0</LangVersion>
<Nullable>Enable</Nullable>
```

Allow you to spot places where you're unintentionally dereferencing a null value.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

New operators: ? And !

```
// this will make sure we can assign a null value to that variable
string? a = null;

// this will remove the warning and let us try to call `.Length`
var l = a!. Length;
```

? - to tell the compiler that this object can be null.

! - is used to tell the compiler that we know that the value won't be null.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Null coalescing operator

```
// Before
var possibleNullValue = possibleNullValue ?? "default value";

// Now this is also allowed
var possibleNullValue ??= "default value";
```

Allow you to assign a default value in case of a null.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Switch Expression with pattern-matching

```
public decimal CalculateToll(object vehicle)
{
    return vehicle switch
    {
        Car c⇒ 2.00m,
        Taxi t ⇒ 3.50m,
        Bus b ⇒ 5.00m,
        DeliveryTruck t ⇒ 10.00m,
        { } ⇒ throw new ArgumentException("Not a known vehicle type", nameof(vehicle)),
        null ⇒ throw new Argument NullException (nameof(vehicle))
    };
}
```

- Enable to use switch in a context of an expression
- Matching on types

TechWorld WithMilan
simplifying complex topics

Swipe →

Switch Expression with pattern-matching

```
public decimal CalculateToll(object vehicle)
{
    return vehicle switch
    {
        Car { Passengers: 0 } => 2.00m + 0.50m,
        Car { Passengers: 1 } => 2.0m,
        Car { Passengers: 2 } => 2.0m - 0.50m,
        Car _ => 2.00m - 1.0m,

        Taxi { Fares: 0 } => 3.50m + 1.00m,
        Taxi { Fares: 1 } => 3.50m,
        Taxi { Fares: 2 } => 3.50m - 0.50m,
        Taxi _ => 3.50m - 1.00m,

        Bus b => 5.00m,
        DeliveryTruck t => 10.00m,
        { } => throw new ArgumentException("Not a known vehicle type", nameof(vehicle)),
        null => throw new ArgumentNullException(nameof(vehicle))
    };
}
```

Matching on properties

TechWorld WithMilan
simplifying complex topics

Swipe →

Switch Expression with pattern-matching

```
public decimal CalculateToll(object vehicle)
{
    return vehicle switch
    {
        Car c => c.Passengers switch
        {
            0 => 2.00m + 0.5m,
            1 => 2.0m,
            2 => 2.0m - 0.5m,
            _ => 2.00m - 1.0m
        },

        Taxi t => t.Fares switch
        {
            0 => 3.50m + 1.00m,
            1 => 3.50m,
            2 => 3.50m - 0.50m
            _ => 3.50m - 1.00m
        },
        // ...
    };
}
```

TechWorld WithMilan
simplifying complex topics

Nesting

Swipe →

Indices and ranges

```
private string[] words = new string[]
{
            // index from start     // index from end
    "The",  // 0                    ^4
    "quick" // 1                    ^3
    "brown" // 2                    ^2
    "fox"   // 3                    ^1
}

var allWords = words[..];     // contains "The" through "fox
var firstPhrase = words[..4]; // contains "The" through "fox"
var lastPhrase = words[2..];  // contains "brown" and "fox"
var lazyDog = words[^2..^0];  // contains "brown" and "fox"
```

Provide clear syntax to access a single element or a range of elements in a sequence.

TechWorld WithMilan
simplifying complex topics

Swipe →

Default interface methods

```
interface IOutput
{
    sealed void PrintException(Exception e) => PrintMessageCore($"Exception: {e}");

    protected void PrintMessageCore(string message);

    protected static void PrintToConsole(string message) => Console.WriteLine(message);
}

class ConsoleOutput : IOutput
{
    void IOutput.PrintMessageCore(string message)
    {
        IOutput.PrintToConsole(message);
    }
}
```

Add new methods to an interface without breaking existing implementations.

TechWorld WithMilan
simplifying complex topics

Swipe →

Using variables

```
// Before
using (var stream = new FileStream("", FileMode.Open))
{
  // ...
}

// Now
using var stream = new FileStream("", FileMode.Open);
```

> Instead of using a block, it can be defined as a variable.

TechWorld WithMilan
simplifying complex topics

Swipe →

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 9 Key Features

- Top-level statements
- Records
- Init-only setters
- Improved Pattern Matching
- Target Typing Improvements
- Covariant returns

TechWorld WithMilan
simplifying complex topics

Swipe →

Top-level statements

```
// Now this is your main app
System.Console.WriteLine("HelloWord!");
```

No boilerplate code

TechWorld WithMilan
simplifying complex topics

Swipe →

Records

```
// Positional params
public record Person(string FirstName, string LastName);

// Regular definition
public record Person
{
    public string FirstName { get; init; }
    public string LastName { get; init; }
}
```

Bridge the gap between class and struct types. They are immutable.

TechWorld WithMilan
simplifying complex topics

Swipe →

Init-only setters

```
public record Order
{
    public int Id { get; init; }
    public string Status { get; init; }
}

// Allowed
var order = new Order { Id = 100, Status = "Created" };

// Not-Allowed (Error)
order.Status = "Updated"
```

Init-only properties can only be set when the object is initialized

TechWorld WithMilan
simplifying complex topics

Swipe →

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 10 Key Features

- Global usings
- File-scoped namespaces
- Constant interpolated strings
- Record structs
- Extended property patterns
- Improvements of structure types

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Global usings

```
global using System.Diagnostics;
global using System.Text;
```

Imported in the whole application.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

File-scoped namespaces

```
namespace Models;

public class ViewModel
{
    // …
}
```

A namespace that has the whole file as its scope

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Constant Interpolated Strings

```
const string constAttribute = "awesome";
const string constMessage = $"C# is {constAttribute}.";
```

Create a constant using string interpolation

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 11 Key Features

- Static abstract interface methods
- Pattern matching - List patterns
- Required properties
- Raw string literals (""") interpolated
- UTF-8 string literals
- Developer productivity

TechWorld WithMilan
simplifying complex topics

Swipe →

Static abstract interface methods

C# 11 Features

```
var result = AddAll(new[] {1, 2, .3, 4, 5}));
WriteLine(result);

T AddAll<T>(T[] values) where T : INumber<T>
{
    T result = T.Zero;
    foreach(var value in values)
    {
        result += value;
    }
    return result;
}
```

An interface is allowed to specify abstract static members

TechWorld WithMilan
simplifying complex topics

Swipe →

Pattern matching - List patterns

```
var result = AddAll(new[] {1, 2, .3, 4, 5}));
WriteLine(result);

T AddAll<T>(T[] values) where T : INumber<T> => values switch
{
    [] => T.Zero,
    [var single] => single,
    [var first, .. var rest] => first + AddAll(rest),
};
```

Support patterns in lists of elements.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Required properties

```
var milan = new Student { FirstName = "Milan", LastName = "Milanovic" };

public class Person
{
    public required  string FirstName { get; init; }
    public string? MiddleName { get; init; }
    public required string LastName { get; init; }
}

public class Student : Person
{
    public int ID { get; init; }
}
```

Forces clients of the code to initialize "the required" field or parameter.

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 11 Features

Raw string literals (""") with interpolation

```
var s = $$"""
    <element attr="content""">
      <body>
        {{{2 + 2}}}
          </body>
    </element>
    """";

Console.WriteLine(s);

// Result:
// <element attr="content""">
//    <body>
//      {4}
//    </body>
// </element>
```

Can prefix multiple “$” signs to the interpolated string.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

UTF-8 string literals

```
var u8 = "This is a UTF-8 string!"u8;
```

Create UTF-8 strings more easily

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Developer productivity

```
[return: NotNullIfNotNull(nameof(url))]
string? GetTopLevelDomainFromFullUrl(string? url)
```

> The nameof operator can be used with method parameters.

TechWorld WithMilan
simplifying complex topics

Swipe →

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 12 Key Features

- Collection expressions
- Primary constructors
- Alias any type
- Default lambda parameters
- Inline arrays
- Interceptors

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 12 Features

Collection expressions

```
int[] x1 = new int[] { 1, 2, 3, 4 };
int[] x2 = Array.Empty<int>();
WriteByteArray(new[] { (byte)1, (byte)2, (byte)3 });
List<int> x4 = new() { 1, 2, 3, 4 };
Span<DateTime> dates = stackalloc DateTime[] { GetDate(0), GetDate(1) };
WriteByteSpan(stackalloc[] { (byte)1, (byte)2, (byte)3 });
```

```
int[] x1 = [1, 2, 3, 4];
int[] x2 = [];
WriteByteArray([1, 2, 3]);
List<int> x4 = [1, 2, 3, 4];
Span<DateTime> dates = [GetDate(0), GetDate(1)];
WriteByteSpan([1, 2, 3]);
```

Collection expression introduce new syntax, [a1, a2, a3, etc], to create common collection values. ,

TechWorld WithMilan
simplifying complex topics

Swipe →

C# 12

Features

Primary Constructors

```
public class BankAccount(string accountID, string owner)
{
    public string AccountID { get; } = accountID;
    public string Owner { get; } = owner;

    public override string ToString() => $"Account ID: {AccountID}, Owner: {Owner}";
}
```

It can be created in any class or struct to initialize a member field or property.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Alias any type

```
using FileManager = System.IO.File;

string fileContent = FileManager.ReadAllText(@"c:\temp\textFile.txt");

using Point = (int x, int y); // Tuple type
```

Now the using directive can point to any type, not just named ones.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

Default Parameters in Lambdas

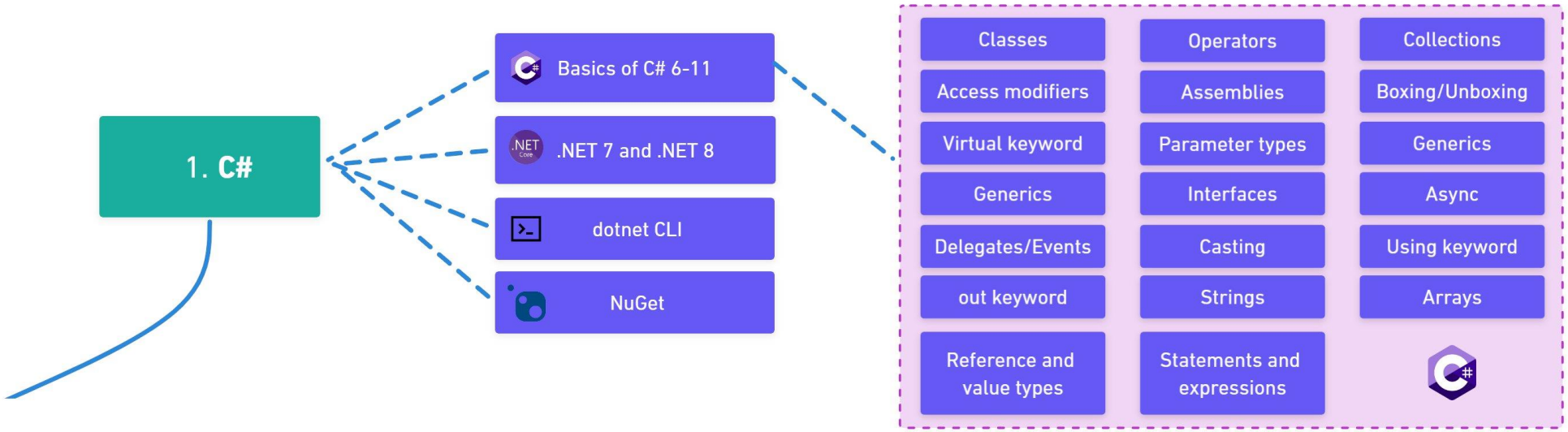
```
var increment = (int value, int inc = 2) => value + inc;
Console.WriteLine(increment(5, 5));
```

Declare default parameters in a lambda expression.

TechWorld WithMilan
simplifying complex topics

Swipe ⟶

.NET Developer Roadmap 2023.



https://github.com/milanm/DotNet-Developer-Roadmap

TechWorld WithMilan
simplifying complex topics

Swipe →

TechWorld WithMilan

simplifying complex topics

milanmilanovic

milan_milanovic

newsletter.techworld-with-milan.com