

REFERENCIA RÁPIDA DE C#

Tipos, LINQ, async/await, colecciones, esenciales de OOP

Fundamentos

Hola Mundo

```
Console.WriteLine("Hello, World!"); // top-level (C# 10+)
// Clásico: class Program { static void Main() { ... } }
```

Compilar y Ejecutar

```
dotnet new console -n MyApp # crear proyecto
dotnet run # compilar y ejecutar
dotnet build # solo compilar
```

Variables y Constantes

```
int x = 42;
var name = "Alice"; // inferencia de tipo
const double Pi = 3.14159;
readonly int maxRetries = 3; // asignar una vez, en ctor
```

Tipos

Tipos de Valor

int Entero con signo de 32 bits
long Entero con signo de 64 bits
float Punto flotante de 32 bits (sufijo 'f')
double Punto flotante de 64 bits
decimal Alta precisión de 128 bits (sufijo 'm')
bool true / false
char Carácter Unicode de 16 bits

Tipos de Referencia

string Texto UTF-16 inmutable
object Tipo base para todos los tipos
dynamic Omite la verificación de tipos en compilación
int[] Arreglo de enteros
List<T> Lista genérica (System.Collections.Generic)

Nullable y Tuplas

```
int? age = null; // tipo de valor nullable
string? name = null; // referencia nullable (C# 8+)
var point = (X: 1, Y: 2); // tupla con nombres
Console.WriteLine(point.X);
```

Características de Cadena

```
string name = "World";
string msg = $"Hello, {name}!"; // interpolación
string path = @"C:\Users\file.txt"; // verbatim
string raw = @"raw string here"; // raw (C# 11+)
```

Control de Flujo

If / Else

```
if (x > 0) Console.WriteLine("positive");
else if (x == 0) Console.WriteLine("zero");
else Console.WriteLine("negative");
```

Switch y Pattern Matching

```
string label = x switch {
    > 0 => "positive", 0 => "zero", _ => "negative"
};
if (obj is string s && s.Length > 0) { } // pattern match
```

Bucles

```
for (int i = 0; i < 10; i++) { }
foreach (var item in collection) { }
while (condition) { }
do { } while (condition);
```

Clases

Definición de Clase

```
public class Person {
    public string Name { get; set; }
    public int Age { get; init; } // init-only (C# 9+)
    public Person(string name, int age) { Name = name; Age = age; }
}
```

Records (C# 9+)

```
public record Point(double X, double Y);
var p1 = new Point(1, 2);
var p2 = p1 with { X = 3 }; // copia no destructiva
// auto: Equals, GetHashCode, ToString, deconstruct
```

Herencia

```
public abstract class Shape { public abstract double Area(); }
public class Circle(double r) : Shape {
    public override double Area() => Math.PI * r * r;
}
```

Modificadores de Acceso

public Accesible desde cualquier lugar
private Solo la misma clase (por defecto para miembros)
protected La misma clase y clases derivadas
internal Solo el mismo assembly (por defecto para clases)
protected internal Mismo assembly o clases derivadas

Interfaces

Definición de Interface

```
public interface IShape {
    double Area();
    double Perimeter() => 0; // impl por defecto (C# 8+)
}
public class Rect(double w, double h) : IShape { public double Area() => w * h; }
```

Interfaces Comunes

IEnumerable<T> Soporte de iteración (foreach, LINQ)
IDisposable Limpieza determinista (sentencia 'using')
IComparable<T> Ordenamiento natural para clasificación
IComparable<T> Comparación de igualdad por valor
ICloneable Clonado de objetos

LINQ

Sintaxis de Método

```
var result = numbers
    .Where(n => n > 3)
    .OrderBy(n => n)
    .Select(n => n * 2)
    .ToList();
```

Sintaxis de Query

```
var result = from n in numbers
              where n > 3
              orderby n
              select n * 2;
```

Métodos LINQ Comunes

.Where(pred) Filtrar elementos
.Select(func) Proyectar / transformar elementos
.OrderBy(key) Ordenar ascendente
.GroupBy(key) Agrupar elementos por clave
.First() / .FirstOrDefault() Primer elemento (o por defecto)
.Any(pred) true si algún elemento coincide
.Count() Número de elementos
.Sum() / .Average() Agregar valores numéricos
.Distinct() Eliminar duplicados
.SelectMany(func) Aplanar colecciones anidadas

Async/Await

Método Async

```
public async Task<string> FetchAsync(string url) {
    using var client = new HttpClient();
    return await client.GetStringAsync(url);
}
```

Combinadores de Task

```
var results = await Task.WhenAll(task1, task2, task3);
var first = await Task.WhenAny(task1, task2);
```

Patrones Async

Task Retorno async void (sin resultado)
Task<T> Retorno async con resultado de tipo T
ValueTask<T> Task ligero para rutas síncronas rápidas
await foreach Iteración async sobre `IAsyncEnumerable<T>`
CancellationToken Cancelación cooperativa para operaciones async

Colecciones

Colecciones Comunes

List<T> Arreglo dinámico, acceso por índice rápido
Dictionary<K,V> Hash map, búsqueda O(1) por clave
HashSet<T> Elementos únicos, búsqueda O(1)
Queue<T> Colección FIFO
Stack<T> Colección LIFO
LinkedList<T> Lista doblemente enlazada
SortedDictionary<K,V> Ordenado por clave (basado en árbol)

Uso de Dictionary

```
var dict = new Dictionary<string, int> {
    ["Alice"] = 90, ["Bob"] = 85
};
dict.TryGetValue("Alice", out int score);
foreach (var (key, val) in dict) { }
```

Colecciones Inmutables

```
using System.Collections.Immutable;
var list = ImmutableList.Create(1, 2, 3);
var newList = list.Add(4); // retorna nueva lista
```

Propiedades

Sintaxis de Propiedad

```
public string Name { get; set; }
public int Age { get; private set; }
public string Email { get; init; } // init-only
public string Display => $"{Name} ({Age})"; // calculada
```

Indexadores

```
public double this[int row, int col] {
    get => data[row, col];
    set => data[row, col] = value;
}
```

Patrones de Propiedad

{ get; set; } Propiedad auto de lectura-escritura
{ get; } Solo lectura (asignar en constructor)
{ get; init; } Solo lectura después de inicialización (C# 9+)
{ get; private set; } Lectura pública, escritura privada
=> expression Propiedad de expresión (calculada)

Excepciones

Try / Catch / Finally

```
try { int result = int.Parse(input); }
catch (FormatException ex)
{ Console.Error.WriteLine(ex.Message); }
catch (Exception ex) when (ex is not OutOfMemoryException) { }
finally { /* siempre se ejecuta */ }
```

Sentencia Using

```
using var file = File.OpenRead("data.txt");
// file.Dispose() se llama automáticamente al finalizar el scope
// equivalente a try/finally con Dispose()
```

Excepciones Comunes

ArgumentNullException Argumento nulo pasado al método
ArgumentOutOfRangeException Argumento fuera del rango válido
InvalidOperationException Operación inválida para el estado actual

NullReferenceException Desreferencia de objeto nulo
KeyNotFoundException Clave no encontrada en el diccionario

NotImplementedException Método aún no implementado

Excepción Personalizada

```
public class AppException : Exception {
    public int Code { get; }
    public AppException(string msg, int code)
        : base(msg) { Code = code; }
}
```