C# LINQ Cheat Sheet

What is LINQ?

LINQ (Language Integrated Query) allows querying in-memory collections and external data sources (like SQL, XML). Supports query syntax and method syntax (fluent). Works on IEnumerable<T> and IQueryable<T>.

Common LINQ Methods

- Where: Filters based on a predicate.
- Select: Projects each element.
- First / FirstOrDefault: Returns first element or default if none.
- Any / All: Checks if any/all items match a condition.
- Count / Sum / Max / Min / Average: Aggregate operations.
- OrderBy / ThenBy / Reverse: Sorting.
- GroupBy / Join / SelectMany: Complex data transformations.

Deferred vs Immediate Execution

Deferred: Query is not executed until iterated (e.g., IEnumerable).
Immediate: Methods like ToList(), ToArray(), Count() execute the query immediately.

IEnumerable vs IQueryable

- IEnumerable<T>: In-memory filtering. Evaluated by .NET runtime.
- IQueryable<T>: Builds expression trees. Evaluated by the data source (e.g., SQL Server).
Use IQueryable<T> with Entity Framework for better performance and server-side evaluation.

ToList() vs ToListAsync()

- ToList(): Synchronously executes and materializes results into a List<T>.
- ToListAsync(): Asynchronously executes with EF Core (requires Microsoft.EntityFrameworkCore).
Use ToListAsync() in ASP.NET Core controllers and async workflows to avoid blocking threads.

First() vs FirstOrDefault() vs Single()

- First(): Returns first element. Throws exception if none.
- FirstOrDefault(): Returns default if no match (null or default(T)).
- Single(): Expects exactly one element. Throws if 0 or >1.
Use Single() when you're certain only one result should exist.

Chaining and Composition

LINQ methods can be chained for complex queries:
Example: users.Where(u => u.IsActive).OrderBy(u => u.Name).Select(u => u.Email).ToList();

Best Practices

- Use async variants like ToListAsync() to avoid blocking.
- Avoid loading unnecessary data - project only needed fields.

C# LINQ Cheat Sheet

- Cache or reuse queries if possible.
- Use AsNoTracking() for read-only EF Core queries to boost performance.

Lazy Loading

Lazy Loading:
- Defers the loading of related data until it's explicitly accessed.
- In Entity Framework Core, enable with proxies (Microsoft.EntityFrameworkCore.Proxies).
- Requires navigation properties to be virtual and context setup to allow lazy loading.
- Pros: Reduces initial load. Cons: May cause N+1 query issues.
- Use explicit loading (e.g., context.Entry(entity).Collection(x => x.Related).Load()) for more control.
- Prefer eager loading (Include) or projection (Select) when predictable performance is needed.