LINQ CHEAT SHEET

Les exemples ci-dessous sont tous basés sur les définitions suivantes :

```
// Classes
class Order
{
    public int Id { get; set; }
    public string Number { get; set; }
    public decimal Amount { get; set; }
    public int CustomerId { get; set; }
}

class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
}

// Listes
List<Customer> customers ;
List<Order> orders ;
```

Filtrer (Retourne une liste)	Where

```
// Les commandes d'un certain client
List<Order> ordersOfCustomer = orders.Where(o => o.Id == 3).ToList();

// Les commandes de clients choisis
List<Order> ordersOfCustomers = orders
    .Where(o => o.Id == 3 || o.Id == 7)
    .ToList();
```

Identifier (Retourne un seul élément)	Single ElementAt	First Last

```
// Un certain client
// Lancera une exception si :
//  - customers est vide
//  - le client n'existe pas
//  - il y a plus d'un client avec cet Id
Customer cust = customers.Single(
    c => c.Id == 84
);
```

LINQ CHEAT SHEET

```
// Un certain client
// Lancera une exception si :
//  - customers est vide
//  - il y a plus d'un client avec cet Id
// Retournera null si le client n'existe pas
Customer cust = customers.SingleOrDefault(
    c => c.Id == 84
);
```

```
// Le premier de la liste
Customer cust = customers.First();
```

```
// Le premier de la liste dont le nom commence par 'C'
Customer cust = customers.First(c => c.Name.StartsWith('C'));
```

```
// Le dernier de la liste dont le nom commence par 'C'
Customer cust = customers.Last(c => c.Name.StartsWith('C'));
```

```
// Le 4ème (!) client de la liste
Customer cust = customers.ElementAt(3);
```

NOTE:

`Single, Last, First, ElementAt` lanceront des exceptions si la liste fournie est vide ou s',il n'y a rien à retourner.

`SingleOrDefault, LastOrDefault, FirstOrDefault, ElementAtOrDefault` lanceront des exceptions si la liste fournie est vide , mais ils retourneront `null` s'il n'ont rien trouvé.

Trier	OrderBy

```
// Trier par client
List<Order> ordersByCustomerId = orders.OrderBy(o => o.CustomerId).ToList();

// Trier par montant, en descendant
List<Order> biggestOrders = orders.OrderByDescending(o => o.Amount).ToList();
```

```
// Trier sur plusieurs colonnes
List<Order> myOrders = orders.
    OrderBy(o => o.CustomerId)
    .ThenByDescending(o => o.Amount)
    .ToList();
```

LINQ CHEAT SHEET

Joindre
(réunir des valeurs de deux objets)

Join

```
// Tous les clients, avec toutes leurs commandes
var customerOrders = customers.Join(orders,
    c => c.Id, o => o.CustomerId,
    (c, o) => new
    {
        CustomerId = c.Id,
        CustomerName = c.Name,
        OrderId = o.Id,
        OrderAmount = o.Amount,
    }
);
```

Paginer

Take Skip

```
// Les trois plus grosses commandes
List<Order> top3 = orders
    .OrderByDescending(o => o.Amount)
    .Take(3)
    .ToList();
```

```
// Ignorer les trois plus grosses commandes !!
List<Order> smallest = orders
    .OrderByDescending(o => o.Amount)
    .Skip(3)
    .ToList();
```

NOTE:
Le type de la clé de groupe sera le même que l'attribut sur lequel on regroupe. Dans l'exemple ci-dessus, `group.key` est un `int` parce que `Order.CustomerId` est un `int`.

Distinguer

Distinct

```
var numbers = new[] { 1, 2, 2, 3, 4, 4, 5 };
var distinctNumbers = numbers.Distinct();
// { 1, 2, 3, 4, 5 }
```

LINQ CHEAT SHEET

Combiner deux listes
(operations ensemblistes)

Zip
Except
Intersect
Union
Concat
SequenceEqual

```
var numbers = new[] { 1, 2, 3 };
var words = new[] { "one", "two", "three" };

// On parcourt les deux listes en parallèle comme une fermeture éclair (zip)
var zipped = numbers.Zip(words, (n, w) => $"{n}: {w}");

foreach (var item in zipped)
{
    Console.WriteLine(item);
}
// 1: one
// 2: two
// 3: three
```

```
var numbers1 = new[] { 1, 2, 3 };
var numbers2 = new[] { 3, 4, 5 };
// On réunit ces deux listes sans créer de doublons
var unionNumbers = numbers1.Union(numbers2);
// { 1, 2, 3, 4, 5 }
```

```
var numbers1 = new[] { 1, 2, 3 };
var numbers2 = new[] { 3, 4, 5 };
// On assemble les deux listes complètes, même si elles créent des doublons
var concatenatedNumbers = numbers1.Concat(numbers2);
// { 1, 2, 3, 3, 4, 5 }
```

```
var numbers1 = new[] { 1, 2, 3 };
var numbers2 = new[] { 3, 4, 5 };
// On ne conserve que les chiffres qui appartiennent aux deux listes
var intersectNumbers = numbers1.Intersect(numbers2);
// { 3 }
```

```
var numbers1 = new[] { 1, 2, 3, 4 };
var numbers2 = new[] { 3, 4, 5 };
// On garde que les chiffres de la première liste qui ne sont pas dans la
deuxième
var exceptNumbers = numbers1.Except(numbers2);
// { 1, 2 }
```

LINQ CHEAT SHEET

```
var numbers1 = new[] { 1, 2, 3 };
var numbers2 = new[] { 1, 2, 3 };
var numbers3 = new[] { 3, 2, 1 };
// On détermine si les listes sont identiques
// attention : on ne peut pas faire cela avec (numbers1 == numbers2)    !!!!
var areEqual = numbers1.SequenceEqual(numbers2);
// true
var areEqual2 = numbers1.SequenceEqual(numbers3);
// false
```

Générer

Range

Repeat

```
// Sequence 1,2,3,4,5
List<int> numbers = Enumerable.Range(1, 5).ToList();
```

```
// Sequence 1,1,1,1,1
List<int> numbers = Enumerable.Repeat(1, 5).ToList();
```

ATTENTION:

Si vous voulez faire cela avec des objets:

```
IEnumerable<Object> objects = Enumerable.Repeat(new Object(), 10);
```

Cela va un instancier un seul objet et mettre 10 fois la référence à cet objet dans la sequence! Par consequent, dès que vous changerez un seul attribute de cet objet, vous les changerez tous!!

Grouper
(et utiliser les groupes)

GroupBy
Count

Min
Max

Average
Aggregate

```
// Compter le nombre de commande par client:
var orderCounts = orders
    .GroupBy(o => o.CustomerId)
    .Select(group => new
        {
            CustID = group.Key,
            TotalOrders = group.Count()
        }
    );
```

LINQ CHEAT SHEET

```
// Regrouper les commandes par catégories sur la bas de leur montant:
// catégorie 0 = de 0 à 100.-, catégorie 1 = de 100 à 200.-, etc...
// Pour chaque catégorie, on veut savoir le nombre de commandes,
// ainsi que le min et le max de la catégorie
var orderGroups = orders.GroupBy(
    order => Math.Floor(order.Amount / 100), // key selector
    order => order.Amount,              // element selector
    (category, amounts) => new
    {
        Key = category,
        Count = amounts.Count(),
        Min = amounts.Min(),
        Max = amounts.Max()
    });                          // result selector
```

```
// compter le nombre d'éléments du groupe
Console.WriteLine(orders.Count());
```

```
Console.WriteLine($"Chiffre d'affaire : {orders.Sum(o => o.Amount)}");
Console.WriteLine($"Plus petite commande : {orders.Min(o => o.Amount)}");
Console.WriteLine($"Plus grosse commande : {orders.Max(o => o.Amount)}");
Console.WriteLine($"En moyenne : {orders.Average(o => o.Amount)}");

// Trouver la plus grosse commande (toute la commande,pas seulement le montant)
Order biggest = orders
    .Where(o => o.Amount == orders.Max(o => o.Amount))
    .First();
```

```
// Une autre manière de trouver laplus grosse commande
Order sameBiggest = orders.Aggregate(
    new Order(),                                // Valeur initiale : vide
    (a, b) => a.Amount > b.Amount ? a : b);     // Critère de comparaison
```

Transformer	Select ToArray	ToList ToDictionnary	ToLookup

```
// Transformer une liste de commande (Orders) en une liste d'objets anonymes
var smallOrders = orders.Select(o => new
    {
        OrderID = o.Id,
        Cost = o.Amount
    }
);

// Transformer une liste de commande (Orders) en une liste de Tuples
var smallOrders = orders.Select(o => (OrderID : o.Id, Cost : o.Amount));
```

LINQ CHEAT SHEET

```
// La liste des noms de clients
List<string> customerNames = customers.Select(cust => cust.Name).ToList();

// Un tableau de noms de clients
string[] arrayNames = customerNames.ToArray();
```

```
// Un dictionnaire qui permet de retrouver très rapidement un objet Customer
Dictionary<int, Customer> col = customers.ToDictionary(c => c.Id);

// Un dictionnaire qui permet de retrouver la plus grosse commande d'un client
Dictionary<string,double> biggestOrderOfEachCustomer = orders
    .Join(
        customers,
        ord => ord.CustomerId,
        cust => cust.Id,
        (ord, cust) => (name: cust.Name, amount: (double)ord.Amount)
    )
    .GroupBy(ord => ord.name)
    .ToDictionary(
        g => g.Key,
        g => g.Max(ord => ord.amount)
        );
```

```
// Un index qui permet de retrouver les commandes d'un client
ILookup<int, Order> customerOrdersLookup = orders.ToLookup(o => o.CustomerId);
```

Contenu tiré en grande partie de https://vslapp.wordpress.com/wp-content/uploads/2011/11/linq-cheatsheet.pdf

La référence complète de trouve ici : https://learn.microsoft.com/fr-fr/dotnet/csharp/linq/standard-query-operators/