LINQ CHEAT SHEET

⬜ Query Syntax
🟧 Lambda Syntax

Filtering

```
var col = from o in Orders
          where o.CustomerID == 84
          select o;
```

```
var col2 = Orders.Where(o => o.CustomerID == 84);
```

Return Anonymous Type

```
var col = from o in orders
          select new
          {
              OrderID = o.OrderID,
              Cost = o.Cost
          };
```

```
var col2 = orders.Select(o => new
               {
                   OrderID = o.OrderID,
                   Cost = o.Cost
               }
    );
```

Ordering

```
var col = from o in orders
          orderby o.Cost ascending
          select o;
```

```
var col2 = orders.OrderBy(o => o.Cost);
```

```
var col3 = from o in orders
           orderby o.Cost descending
           select o;
```

```
var col4 = orders.OrderByDescending(o => o.Cost);
```

```
var col9 = from o in orders
  orderby o.CustomerID, o.Cost descending
           select o;
```

```
//returns same results as above
var col5 = from o in orders
           orderby o.Cost descending
           orderby o.CustomerID
           select o;
//NOTE the ordering of the orderby's
```

```
var col6 = orders.OrderBy(o => o.CustomerID).
               ThenByDescending(o => o.Cost);
```

Joining

```
var col = from c in customers
          join o in orders on
          c.CustomerID equals o.CustomerID
          select new
          {
              c.CustomerID,
              c.Name,
              o.OrderID,
              o.Cost
          };
```

```
var col2 = customers.Join(orders,
    c => c.CustomerID,o => o.CustomerID,
    (c, o) => new
        {
            c.CustomerID,
            c.Name,
            o.OrderID,
            o.Cost
        }
    );
```

Grouping

```
var OrderCounts = from o in orders
        group o by o.CustomerID into g
        select new
        {
            CustomerID = g.Key,
            TotalOrders = g.Count()
        };
```

```
var OrderCounts1 = orders.GroupBy(
         o => o.CustomerID).
         Select(g => new
         {
             CustomerID = g.Key,
             TotalOrders = g.Count()
         });
```

NOTE:
the grouping's key is the same type as the grouping value. E.g. in above example grouping key is an int because o.CustomerID is an int.

LINQ CHEAT SHEET

⬜	Query Syntax
🟧	Lambda Syntax

Paging (using Skip & Take)

```
//select top 3
var col = (from o in orders
           where o.CustomerID == 84
           select o).Take(3);
```

```
var col2 = orders.Where(
           o => o.CustomerID == 84
           ).Take(3);
```

```
//skip first 2 and return the 2 after

var col3 = (from o in orders
            where o.CustomerID == 84
            orderby o.Cost
            select o).Skip(2).Take(2);
```

```
var col3 = (from o in orders
            where o.CustomerID == 84
            orderby o.Cost
            select o).Skip(2).Take(2);
```

Element Operators (Single, Last, First, ElementAt, Defaults)

```
//throws exception if no elements
var cust = (from c in customers
           where c.CustomerID == 84
           select c).Single();
```

```
var cust1 = customers.Single(
            c => c.CustomerID == 84);
```

```
//returns null if no elements
var cust = (from c in customers
            where c.CustomerID == 84
            select c).SingleOrDefault();
```

```
var cust1 = customers.SingleOrDefault(
            c => c.CustomerID == 84);
```

```
//returns a new customer instance if no elements
var cust = (from c in customers
            where c.CustomerID == 85
            select c).DefaultIfEmpty(
              new Customer()).Single();
```

```
var cust1 = customers.Where(
            c => c.CustomerID == 85
            ).DefaultIfEmpty(new Customer()).Single();
```

```
//First, Last and ElementAt used in same way
var cust4 = (from o in orders
             where o.CustomerID == 84
             orderby o.Cost
             select o).Last();
```

```
var cust5 = orders.Where(
            o => o.CustomerID == 84).
            OrderBy(o => o.Cost).Last();
```

```
//returns 0 if no elements
var i = (from c in customers
         where c.CustomerID == 85
         select c.CustomerID).SingleOrDefault();
```

```
var j = customers.Where(
        c => c.CustomerID == 85).
        Select(o => o.CustomerID).SingleOrDefault();
```

NOTE:

`Single, Last, First, ElementAt` all **throw exceptions** if source sequence is empty.

`SingleOrDefault, LastOrDefault, FirstOrDefault, ElementAtOrDefault` all **return default(T)** if source sequence is empty. i.e. NULL will be returned if T is a reference type or nullable value type; default(T) will be returned if T is a non-nullable value type (int, bool etc). This can be seen in the last example above.

LINQ CHEAT SHEET

⬜	Query Syntax
🟧	Lambda Syntax

Conversions

ToArray

```
string[] names = (from c in customers
                  select c.Name).ToArray();
```

ToDictionary

```
Dictionary<int, Customer> col = customers.ToDictionary(c => c.CustomerID);

Dictionary<string, double> customerOrdersWithMaxCost = (from oc in

      (from o in orders
      join c in customers on o.CustomerID equals c.CustomerID
      select new { c.Name, o.Cost })

      group oc by oc.Name into g
      select g).ToDictionary(g => g.Key, g => g.Max(oc => oc.Cost));
```

ToList

```
List<Order> ordersOver10 = (from o in orders
      where o.Cost > 10
      orderby o.Cost).ToList();
```

ToLookup

```
ILookup<int, string> customerLookup =
           customers.ToLookup(c => c.CustomerID, c => c.Name);
```