LINQ Cheat Sheet

By: Mosh Hamedani

Restriction

```
context.Courses.Where(c => c.Level == 1);
```

Ordering

```
context.Courses
       .OrderBy(c => c.Name)  // or OrderByDescending
       .ThenBy(c => c.Level); // or ThenByDescending
```

Projection

```
context.Courses.Select(c => new
       {
            CourseName = c.Name,
            AuthorName = c.Author.Name  // no join required
       });

// flattens hierarchical lists
var tags = context.Courses.SelectMany(c => c.Tags);
```

Grouping

```
var groups = context.Courses.GroupBy(c => c.Level);
```

LINQ Cheat Sheet

By: Mosh Hamedani

Inner Join

Use when there is no relationship between your entities and you need to link them based on a key.

```
var authors = context.Authors.Join(context.Courses,
     a => a.Id,  // key on the left side
     c => c.AuthorId,   // key on the right side,
     (author, course) =>   // what to do once matched
        new
        {
            AuthorName = author.Name,
            CourseName = course.Name
        }
);
```

Group Join

Useful when you need to group objects by a property and count the number of objects in each group. In SQL we do this with LEFT JOIN, COUNT(*) and GROUP BY. In LINQ, we use group join.

```
var authors = context.Authors.GroupJoin(context.Courses,
     a => a.Id,  // key on the left side
     c => c.AuthorId,   // key on the right side,
     (author, courses) =>   // what to do once matched
        new
        {
            AuthorName = author.Name,
            Courses = courses
        }
);
```

LINQ Cheat Sheet

By: Mosh Hamedani

Cross Join

To get full combinations of all objects on the left and the ones on the right.

```
var combos = context.Authors.SelectMany(a => context.Courses,
    (author, course) => new {
        AuthorName = author.Name,
        CourseName = course.Name
    });
```

Partitioning

To get records in a given page.

```
context.Courses.Skip(10).Take(10);
```

Element Operators

```
// throws an exception if no elements found
context.Courses.First();
context.Courses.First(c => c.Level == 1);

// returns null if no elements found
context.Courses.FirstOrDefault();

// not supported by SQL Server
context.Courses.Last();
context.Courses.LastOrDefault();

context.Courses.Single(c => c.Id == 1);
context.Courses.SingleOrDefault(c => c.Id == 1);
```

LINQ Cheat Sheet

By: Mosh Hamedani

Quantifying

```
bool allInLevel1 = context.Courses.All(c => c.Level == 1);

bool anyInLevel1 = context.Courses.Any(c => c.Level == 1);
```

Aggregating

```
int count = context.Courses.Count();
int count = context.Courses.Count(c => c.Level == 1);

var max = context.Courses.Max(c => c.Price);
var min = context.Courses.Min(c => c.Price);
var avg = context.Courses.Average(c => c.Price);
var sum = context.Courses.Sum(c => c.Price);
```