PYTHON

`map` `filter`

`reduce` `zip`

`itertools`

`list comprehensions`

`sorted` `groupby`

`any` `all`

C# LINQ

Отложенные операции:

Select, SelectMany
Where
OrderBy
GroupBy
Take, TakeWhile
Repeat
Reverse

Мгновенные операции:

Aggregate
All
Any
Last, First
Average
Max, Min
Single

JAVA 8 STREAMS

Промежуточные операции:

distinct
filter
flatMap
limit
map
peek
skip
sorted

Терминальные операции:

allMatch
anyMatch
findAny
findFirst
noneMatch
forEach
reduce

INSPIRED BY

http://markheath.net/post/python-equivalents-of-linq-methods

http://blog.lahteenmaki.net/2013/04/java-streams-vs-c-linq-vs-java6.html

… and many others

LIST COMPREHENSIONS (1)

```
res = [x**2 for x in range(10)]
```

```
res = map(lambda x: x**2, range(10)]
```

```
res = (x**2 for x in range(10))
```

```
var res = Enumerable.Range(0, 10).Select(x => x*x).ToList();
```

```
int[] res = IntStream.range(0, 10).map(x -> x*x).toArray();
```

LIST COMPREHENSIONS (2)

```
res = [x**2 for x in range(10) if x % 2]

res = map(lambda x: x**2, range(1,10,2)]
```

```
var res = Enumerable.Range(0, 10)
                      .Where(x => x % 2 ! = 0)
                      .Select(x => x*x)
                      .ToList();
```

```
int[] res = IntStream.range(0, 10)
                      .filter(x -> x % 2 != 0)
                      .map(x -> x*x)
                      .toArray();
```

LIST COMPREHENSIONS (3)

```
chessboard = [x + str(y+1) for x in "ABCDEFGH" for y in range(8)]
```

```
var chessboard = "ABCDEFGH".SelectMany(x => Enumerable.Range(1,8)
                                  .Select(y => x+y.ToString()))
                                  .ToList();
```

```
List<String> chessboard = Stream.of("A", "B", "C", "D", "E", "F", "G", "H")
                            .flatMap(s -> Stream.iterate(1, n->n+1)
                                              .limit(9).map(t->s + t) )
                            .collect(Collectors.toList());
```

LIST COMPREHENSIONS (3)

```
['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8',
 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8',
 'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8',
 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8',
 'E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8',
 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8',
 'G1', 'G2', 'G3', 'G4', 'G5', 'G6', 'G7', 'G8',
 'H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7', 'H8']
```

GENERATORS

```
def generate_children():
    yield "Ben"
    yield "Lily"
    yield "Joel"
    yield "Sam"
    yield "Annie"
```

```
for child in generate_children():
    print child
```

```
public IEnumerable<string> GenerateChildren()
{
    yield return "Ben";
    yield return "Lily";
    yield return "Joel";
    yield return "Sam";
    yield return "Annie";
}
```

```
foreach (string child in GenerateChildren())
    Console.WriteLine(child);
```

TODICTIONARY()

```
fruit = ['apples', 'oranges', 'bananas', 'pears']
dic = {f:len(f) for f in fruit}
print dic
```

```
{'pears': 5, 'apples': 6, 'oranges': 7, 'bananas': 7}
```

```
var fruits = new [] { "apples", "oranges", "bananas", "pears" };
var dic = fruits.ToDictionary(f => f, f => f.Length);

foreach (KeyValuePair<string, int> entry in dic)
{
         Console.WriteLine("{0} : {1}", entry.Key, entry.Value);
}
```

```
Map<String, Integer> res = Stream.of("apples", "oranges", "bananas", "pears")
                 .collect(Collectors.toMap(s->s, String::length));

for (Map.Entry<String, Integer> entry : res.entrySet()) {
    System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

‘STUDENT’ ENTITY

```
class Student
{
        public String Name { get; set; }
        public char Sex { get; set; }
        public int Age { get; set; }
}
```

```
class Student {
        public String name;
        public char sex;
        public int age;

        public Student(String name, char sex, int age) {
                this.name = name;
                this.sex = sex;
                this.age = age;
        }
}
```

‘STUDENT’ ENTITY

```
students = [ ('Vasya', 'm', 23),
             ('Lena', 'f', 20),
             ('Kolya', 'm', 28),
             ('Ira', 'f', 24)    ]
```

```
var students = new List<Student>()
{
         new Student { Name="Vasya", Sex='m', Age=23 },
         new Student { Name="Lena", Sex='f', Age=20 },
         new Student { Name="Kolya", Sex='m', Age=28 },
         new Student { Name="Ira", Sex='f', Age=24 }
};
```

```
List<Student> students = Arrays.asList(
                             new Student(“Vasya", 'm', 23),
                             new Student("Lena", 'f', 20),
                             new Student("Kolya", 'm', 28),
                             new Student("Ira", ‘f', 24));
```

FILTER, WHERE

```
women = [s for s in students if s[1] == 'f']
women = filter(lambda s : s[1] == 'f', students)
```

```
var women = students.Where(s => s.sex=='f').ToList();
```

```
List<Student> women = students.stream()
                      .filter(s -> s.sex == 'f')
                      .collect(Collectors.toList());
```

ANY, ALL

```
print  any(s[2] > 25 for s in students)
print  all(s[2] < 25 for s in students)
```

```
var areExperienced = students.Any(s => s.age > 25);
Console.WriteLine(areExperienced);
var areYoung = students.All(s => s.age < 25);
Console.WriteLine(areYoung);
```

```
Boolean areExperienced = students.stream().anyMatch(s -> s.age > 25);
System.out.println(areExperienced);
Boolean areYoung = students.stream().allMatch(s -> s.age < 25);
System.out.println(areYoung);
```

MIN, MAX

```
print   max(s[2] for s in students)
print   min(s[2] for s in students)
```

```
var maxAge = students.Max(s => s.age);
Console.WriteLine(maxAge);
var minAge = students.Min(s => s.age);
Console.WriteLine(minAge);
```

```
Student youngest = students.stream()
                   .min( (s1, s2) -> Integer.compare(s1.age, s2.age) )
                   .get();

System.out.println(youngest.age);
```

TAKE, SKIP

```
from   itertools import islice
print  list(islice(students, 1, 3))
```

```
var some = students.Skip(1).Take(2).ToList();

foreach (var s in some)
        Console.WriteLine(s.name);
```

```
List<Student> some = students.stream()
                         .skip(1).limit(2)
                         .collect(Collectors.toList());

for (Student s : some)
        System.out.println(s.name + ", " + s.age);
```

TAKEWHILE, SKIPWHILE

```
from itertools import takewhile
some = takewhile(lambda s: len(s[0]) > 3, students)
print list(some)
```

```
var some = students.TakeWhile(s => s.name.Length > 3).ToList();

foreach (var s in some)
        Console.WriteLine(s.name);
```

```
List<Student> some = students.stream()
                .collect(Collectors.partitioningBy(s -> s.name.length() > 3))
                .get(true);
```

SELECT, MAP

```
print list(map(lambda s: (s[0].upper(), s[1], s[2]), students))
```

```
def capitalize_name(s):
        return s[0].upper(), s[1], s[2]

print list(map(capitalize_name, students))
```

```
var some = students.Select(s => {
                        s.name = s.name.ToUpper();
                        return s; })
                .ToList();
```

```
List<Student> filtered = students.stream()
                .map(s -> {
                        s.name = s.name.toUpperCase();
                        return s;
                }
                .collect(Collectors.toList());
```

REDUCE

```
print  reduce(lambda x, y: (x[0] + ', ' + y[0], 'a', x[2] + y[2]), students)
```

```
var res = students.Aggregate(
         (x,y) => new Student(x.name += ", " + y.name, 'a', x.age + y.age) )
```

```
Student res = students.stream()
         .reduce((x,y) -> new Student(x.name + ", " + y.name, 'a', x.age + y.age))
         .orElse(new Student("", 'a', 0));
```

SORTED, ORDERBY

```
print  sorted(students, key=lambda s: s[2])
```

```
var sortedByAge = students.OrderBy(s => s.age).ToList();
```

```
List<Student> sortedByAge = students.stream()
          .sorted((s1,s2) -> Integer.compare(s1.age, s2.age))
          .collect(Collectors.toList());
```

GROUPBY

```
from itertools import groupby

students = sorted(students, key=lambda s: s[1])
for sex, info in groupby(students, lambda s: s[1]):
        print list(info)
```

```
var groupedBySex = students.GroupBy(s => s.sex).ToList();

foreach (var group in groupedBySex)
{
        Console.WriteLine(group.Key);
        foreach (var s in group)
                Console.WriteLine(s.name);
}
```

```
Map<Character, List<Student>> res = students.stream()
                  .collect(Collectors.groupingBy(s->s.sex));
```

DISTINCT

```
string[] names = { "Pete", "Ann", "Pete", "Luke" };

var uniques = names.Distinct().ToList();
```

```
String[] names = { "Pete", "Ann", "Pete", "Luke" };

Stream.of(names).distinct().forEach(System.out::println);
```

ZIP

```
fruit = ['apples', 'oranges', 'bananas', 'pears']
recipes = ['pie', 'juice', 'milkshake']
print list(zip(fruit,recipes))
```

```
string[] names = { "Pete", "Ann", "Pete", "Luke" };
string[] lastnames = { “Johnson", “Lee", “Harris", “Gable" };

var fullnames = names.Zip(lastnames, (n,l) => n + " " + l).ToList();
```

LINQ SYNTHAX

```
string[] fullNames = { "Anne Williams", "John Fred Smith", "Sue Green" };

var query =
  from fullName in fullNames
  from name in fullName.Split()
  orderby fullName, name
  select name + " ; full: " + fullName;
```

```
var query = fullNames
  .SelectMany (fName => fName.Split().Select (name => new { name, fName } ))
  .OrderBy (x => x.fName)
  .ThenBy  (x => x.name)
  .Select  (x => x.name + "; full: " + x.fName);
```