Translation of For
For-Expressions and Higher-Order Functions

The syntax of `for` is closely related to the higher-order functions `map`, `flatMap` and `filter`.

First of all, these functions can all be defined in terms of `for`:

```scala
def mapFun[T, U](xs: List[T], f: T => U): List[U] = 
  for (x <- xs) yield f(x)

def flatMap[T, U](xs: List[T], f: T => Iterable[U]): List[U] = 
  for (x <- xs; y <- f(x)) yield y

def filter[T](xs: List[T], p: T => Boolean): List[T] = 
  for (x <- xs if p(x)) yield x
```
In reality, the Scala compiler expresses for-expressions in terms of map, flatMap and a lazy variant of filter.

Here is the translation scheme used by the compiler (we limit ourselves here to simple variables in generators)

1. A simple for-expression

   \[ \text{for } (x \leftarrow e1) \text{ yield } e2 \]

   is translated to

   \[ e1.\text{map}(x \rightarrow e2) \]
Translation of For (2)

2. A for-expression

```scala
for (x <- e1 if f; s) yield e2
```

where $f$ is a filter and $s$ is a (potentially empty) sequence of generators and filters, is translated to

```scala
for (x <- e1.withFilter(x => f); s) yield e2
```

(and the translation continues with the new expression)

You can think of withFilter as a variant of filter that does not produce an intermediate list, but instead filters the following map or flatMap function application.
3. A for-expression

    for (x <- e1; y <- e2; s) yield e3

where s is a (potentially empty) sequence of generators and filters, is translated into

    e1.flatMap(x => for (y <- e2; s) yield e3)

(and the translation continues with the new expression)
Take the for-expression that computed pairs whose sum is prime:

\[
\text{for } \{ \\
\quad \text{i <- 1 until n} \\
\quad \text{j <- 1 until i} \\
\quad \text{if isPrime(i + j)} \\
\} \text{ yield (i, j)}
\]

Applying the translation scheme to this expression gives:

\[
(1 \text{ until n}) . \text{flatMap(i =>} \\
\quad (1 \text{ until i}) . \text{withFilter(j => isPrime(i+j)))} \\
\quad . \text{map(j => (i, j)))}
\]

This is almost exactly the expression which we came up with first!
Exercise

Translate

```scala
for (b <- books; a <- b.authors if a.startsWith "Bird")
yield b.title
```

into higher-order functions.
Exercise

Translate

```scala
for (b <- books; a <- b.authors if a startsWith "Bird")
yield b.title
```

into higher-order functions.

```
books.flatMap (b =>
b.authors filter (a => a.startsWith "Bird") map (a => a.title))
```
Interestingly, the translation of for is not limited to lists or sequences, or even collections; It is based solely on the presence of the methods map, flatMap and withFilter.

This lets you use the for syntax for your own types as well – you must only define map, flatMap and withFilter for these types.

There are many types for which this is useful: arrays, iterators, databases, XML data, optional values, parsers, etc.
For and Databases

For example, books might not be a list, but a database stored on some server.

As long as the client interface to the database defines the methods map, flatMap and withFilter, we can use the for syntax for querying the database.

This is the basis of the Scala data base connection frameworks ScalaQuery and Slick.

Similar ideas underly Microsoft’s LINQ.