36:
For Comprehensions

Quick Introduction

For Comprehensions

- We already met the following construct where we print each element of a list on a separate line using the *foreach* method which is defined in *Traversable*.

```
List(1,2,3) foreach println
```

- But we could also write this, which has the same effect:

```
for (i <- List(1,2,3)) println(i)
```

We can spell this with or without the hyphen

- This form is called a “for-comprehension” and it is “syntactic sugar” for the first statement (the one involving *foreach*).
- Thus, the type yielded by this for-comprehension is: *Unit*.

For Comprehensions (2)

- Actually, there are two quite different forms of for-comprehension:

```
for (i <- List(1,2,3)) println(i)
for (i <- List(1,2,3)) yield i.toString
```

  - As noted in the previous slide, the first form yields *Unit*. (i.e. it works through side-effects).

- Notice the “yield” in the second form. What does it do?
  - The result of the second statement is a *List[String]* with the elements “1”, “2”, and “3”
- We will discover (next lecture) that this second form is syntactic sugar for:

```
List(1,2,3) map { i: Int => i.toString }
```

For Comprehensions (3)

- We will find that there is much more to these for-comprehensions. They are an important way of making functional code more readable by a human.