UNIVERSIDAD COMPLUTENSE DE MADRID

ntic master School

# Scala

## Bucles y expresiones

Enero de 2023

# Agenda

- **Bucles y expresiones *for***
- **Equivalencia funcional**



# 1 Bucles y expresiones *for*

# Bucles *for*


```
for(seq) block
```

```
scala> for (n <- 1 to 3) println(n)
1
2
3
```

- Bucle for devuelve Unit (valor vacío).
- Aplica cambios de estado externos (i.e. impresión por consola).
- *seq* puede contener **generadores**, **filtros** y **definiciones**.
- *block* aplica los cambios de estado externos y el resultado es ignorado

# Expresiones *for*


```
for(seq) yield expr
```

- No es un bucle, porque devuelve un resultado.
- *seq* puede contener **generadores**, **filtros** y **definiciones**.
- *expr* crea un elemento del resultado.

# Generadores


```
elem <- collection
```

```
scala> for (n <- Vector(1, 2, 3)) yield n + 1
res14: scala.collection.immutable.Vector[Int] = Vector(2, 3, 4)

scala> for (n <- Vector(1, 2, 3)) yield "#" + n
res15: scala.collection.immutable.Vector[String] = Vector(#1, #2, #3)

scala> for (n <- Set(1, 2, 3)) yield "#" + n
res16: scala.collection.immutable.Set[String] = Set(#1, #2, #3)
```

- Los generadores determinan la iteración.
- *collection* contiene los datos sobre los que se itera.
- *elem* será la variable local ligada al elemento de cada iteración.
- El tipo del primer generador (*collection*) determina el tipo del resultado.

# Generadores múltiples


- Se pueden anidar la iteración de múltiples estructuras de datos.
- Para hacerlo, los generadores se pueden separar por punto y coma (;) para hacerlo en una línea o definiendo un bloque usando llaves ({ }) y cada generador en una línea distinta.
- *to* es un generador de colecciones.

# Generadores con filtros


```
if expr
```

```
scala> for {
     | n <- 1 to 3 if n % 2 == 1
     | m <- 1 to n
     | } yield n * m
res20: scala.collection.immutable.IndexedSeq[Int] = Vector(1, 3, 6, 9)
```

- Los filtros controlan la iteración.
- *expr* se debe evaluar a true (*Boolean*) para permitir usar un valor del generador.
- Los filtros se definen inmediatamente después del generador en la misma línea.

# Generadores con definición


```
x = expr
```

```
scala> for {
     | time <- times
     | hours = time.hours if hours > 12
     | } yield (hours - 12) + "pm"
```

- Definiciones = Definición de variable local
- Las definiciones también pueden tener asociado un filtro en la misma línea que añada un control extra.

# Ejercicio: Usa for-expressions


- Añade el método `stopsAt` a `JourneyPlanner` que tome como parámetro `station` de tipo `Station`:
  - debe devolver un `Set` de `Tuple2` de `Time` y `Train`
  - devolverá las paradas de todos los `Train` en la `station` data
  - Hint: para la implementación usa for-expression con dos generadores y un filtro

# 2 Equivalencia funcional

# Traducción de expresiones *for*

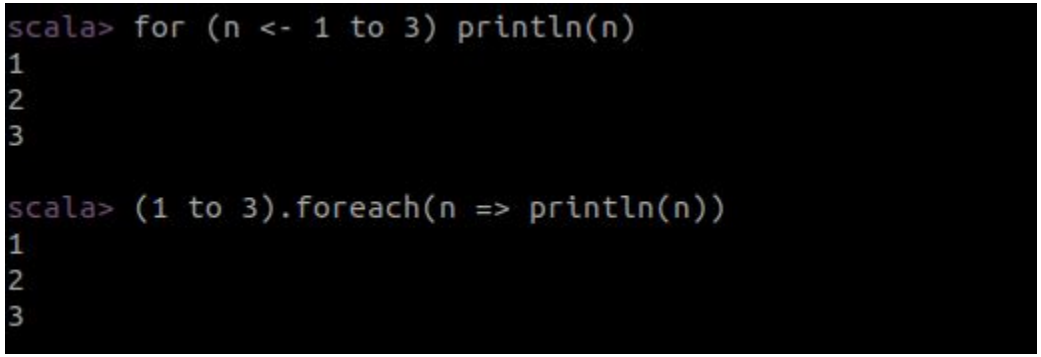

- Las expresiones *for* pueden traducirse en llamadas anidadas de *flatMap*, *map* y *withFilter*.

```
scala> for (n <- 1 to 3) yield n + 1
res26: scala.collection.immutable.IndexedSeq[Int] = Vector(2, 3, 4)

scala> (1 to 3).map(n => n + 1)
res27: scala.collection.immutable.IndexedSeq[Int] = Vector(2, 3, 4)
```

```
scala> for (n<- 1 to 3; m <- 1 to n) yield n * m
res0: scala.collection.immutable.IndexedSeq[Int] = Vector(1, 2, 4, 3, 6, 9)

scala> (1 to 3).flatMap(n => (1 to n).map(m => n * m))
res1: scala.collection.immutable.IndexedSeq[Int] = Vector(1, 2, 4, 3, 6, 9)
```

- Si tenemos que iterar, empecemos con una expresión *for*

# Traducción de los bucles *for*




- Los bucles *for* se pueden traducir a llamadas de *foreach*.