

V1.1.2

Estructuras de
control

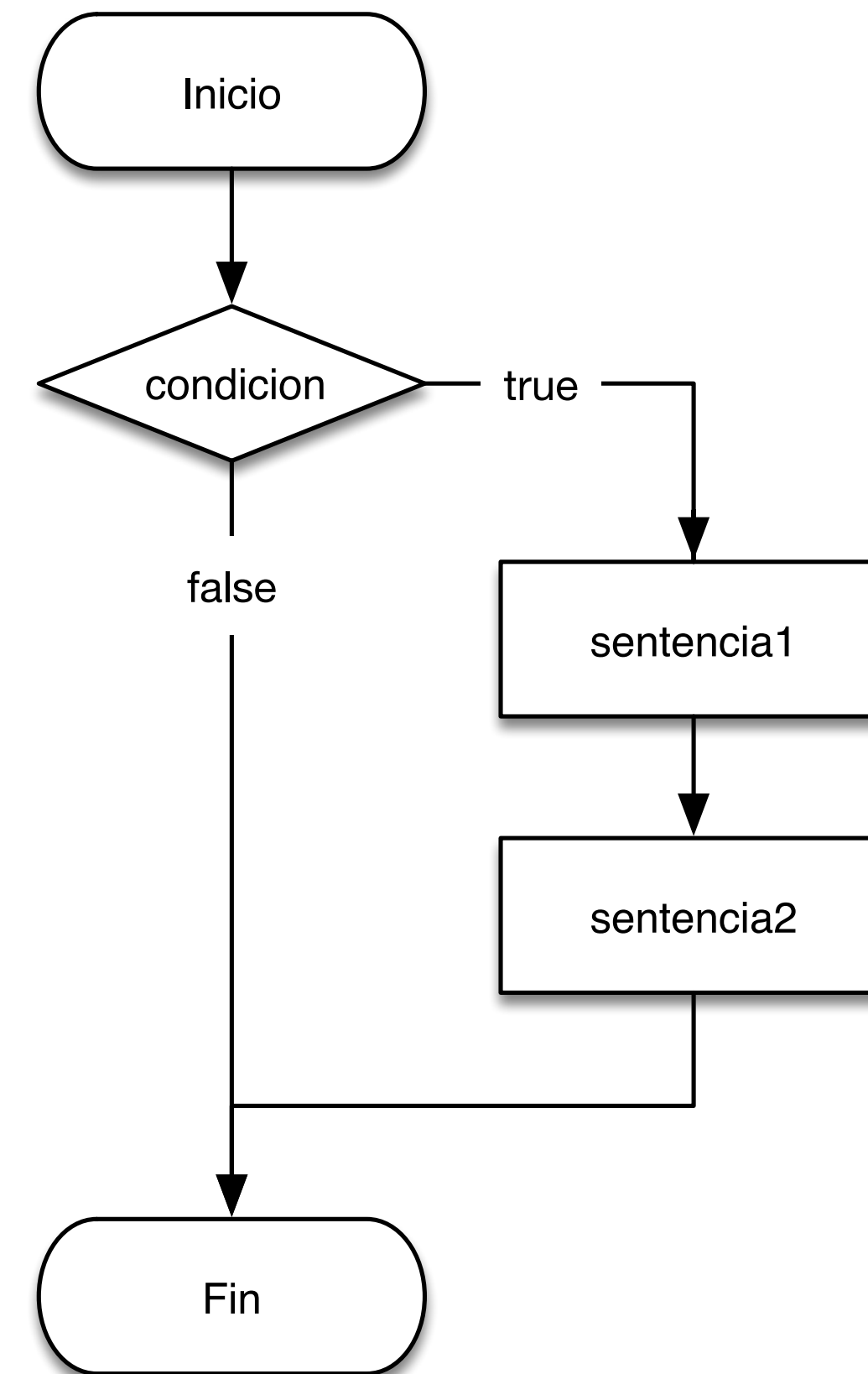


Kotlin

Alternativa simple: if

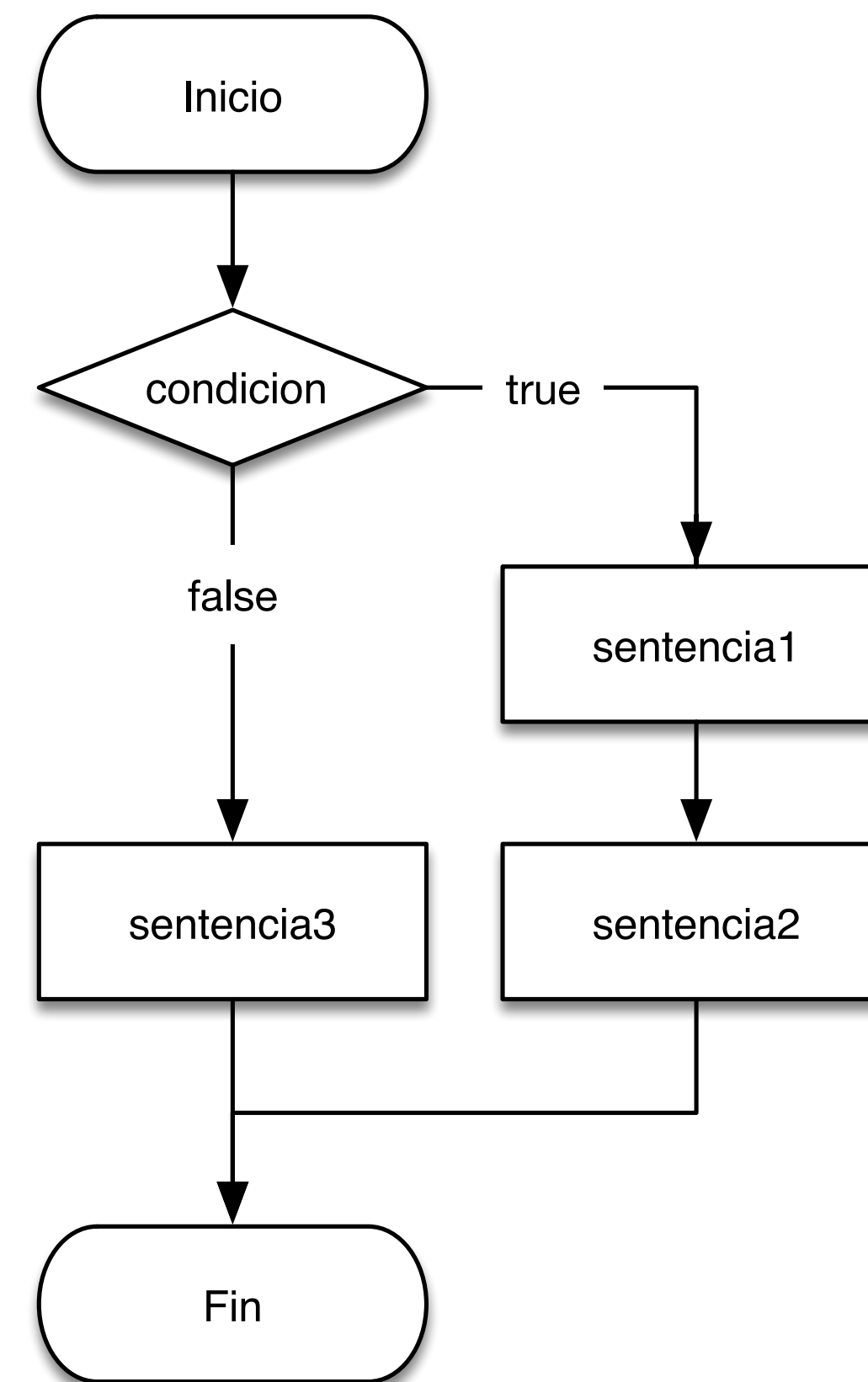
Alternativa simple: if

```
if (condicion) {  
    sentencia1  
    sentencia2  
}
```



Alternativa simple: if

```
if (condicion) {  
    sentencia1  
    sentencia2  
}  
else {  
    sentencia3  
}
```



Alternativa simple: if

```
val max = if (a > b) a else b
```

- En Kotlin el `if` es una expresión, por lo que se puede asignar a una variable
- Si se usa como expresión, debe llevar `else`

Alternativa simple: if

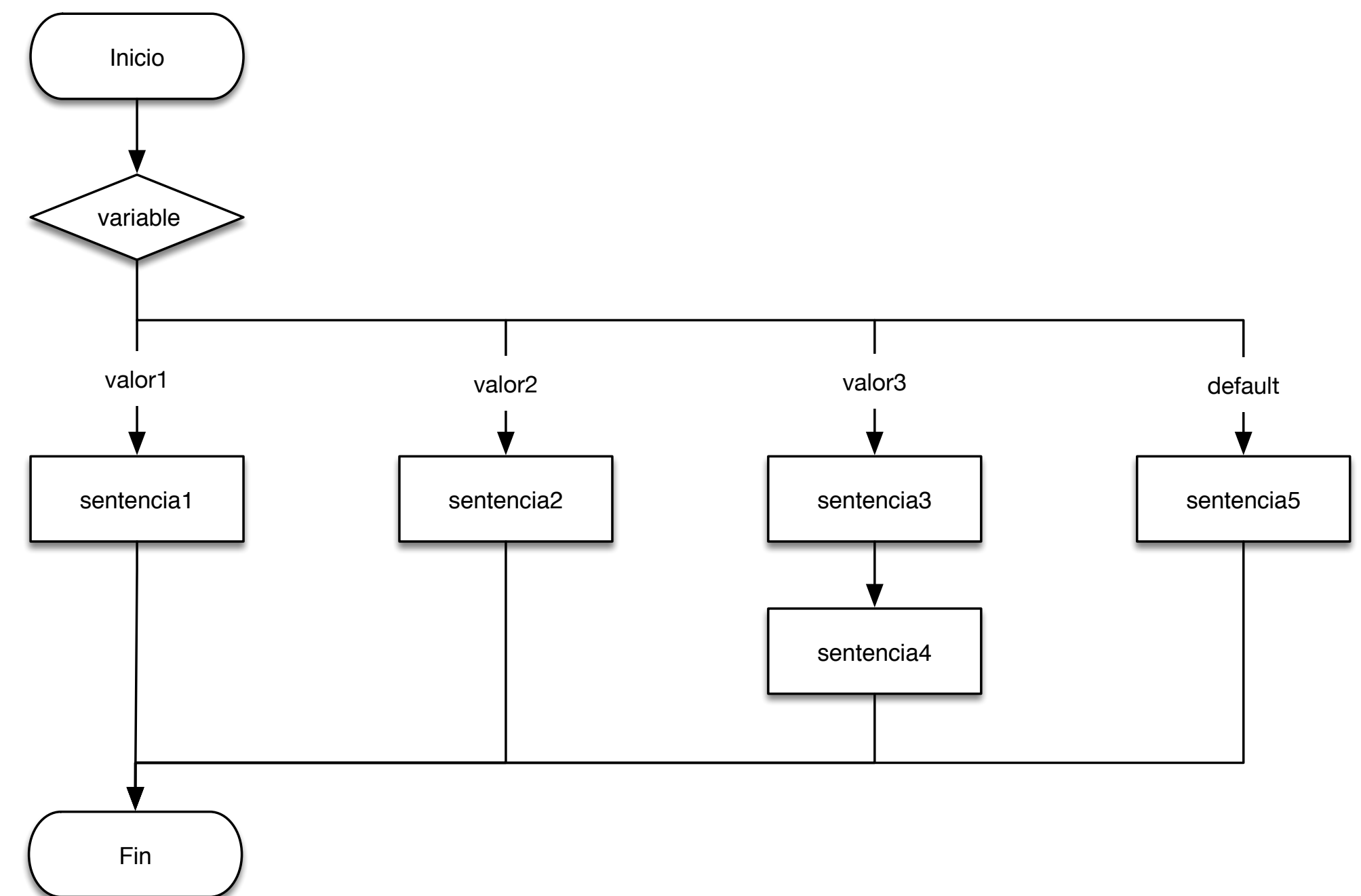
```
val max = if (a > b) {  
    print("Choose a")  
    a  
} else {  
    print("Choose b")  
    b  
}
```

- Las ramas pueden ser bloques
- La última expresión será el valor que se asigne

Alternativa múltiple: when

Alternativa múltiple: when

```
when (variable) {  
    valor1 -> sentencia1  
    valor2 -> sentencia2  
    valor3 -> {  
        sentencia3  
        sentencia4  
    }  
    else -> sentencia5  
}
```



Alternativa múltiple: when

- Puede utilizarse como sentencia o expresión
- Si se usa como expresión, el resultado será el de la rama seleccionada
- Si la rama es un bloque, se asignará el valor de la última expresión
- Si se usa como expresión, debe llevar `else`

Casos múltiples

```
when (x) {  
    0, 1 -> print("x == 0 or x == 1")  
    else -> print("otherwise")  
}
```

Evaluación de expresiones arbitrarias

```
when (x) {  
    parseInt(s) -> print("s encodes x")  
    else -> print("s does not encode x")  
}
```

Evaluación de rangos

```
when (x) {  
  in 1..10 -> print("x is in the range")  
  in validNumbers -> print("x is valid")  
  !in 10..20 -> print("x is outside the range")  
  else -> print("none of the above")  
}
```

Evaluación de tipos

```
fun hasPrefix(x: Any) = when(x) {  
    is String -> x.startsWith("prefix")  
    else -> false  
}
```

Operadores: relacionales y lógicos

Operadores relacionales

Operador	Operación
==	Igual (estructural)
!=	Distinto
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
===	Idéntico (referencial)
!==	No idéntico

Operadores lógicos

Operador	Operación
!	Negación lógica, NOT
&&	Conjunción lógica, AND
	Disyunción lógica, OR

Repetitivas

Repetitivas

$0 \rightarrow n$

$1 \rightarrow n$

n

`while`

`do-while`

`for`

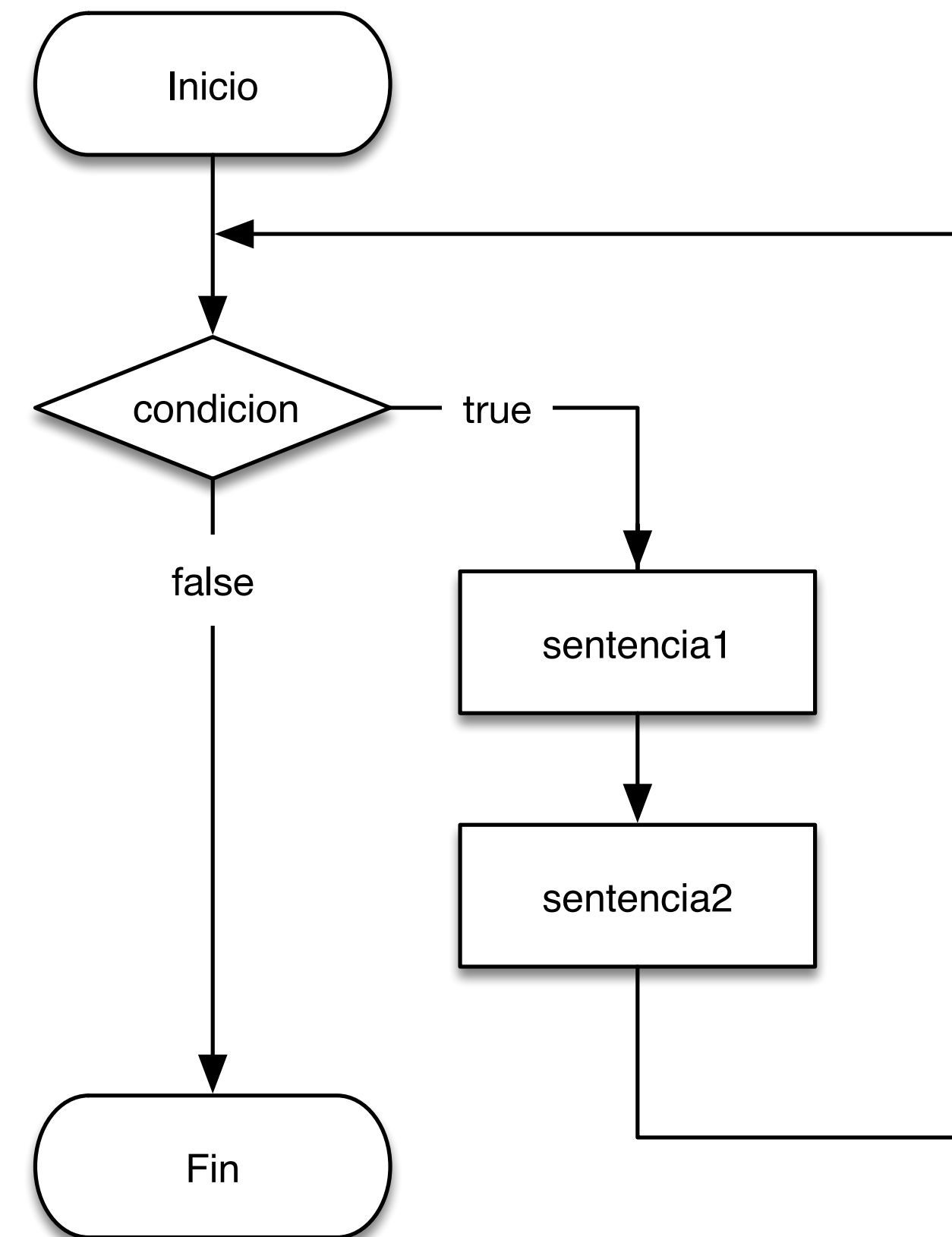
Puede que nunca se ejecute

Se ejecuta por lo menos una vez

Recorre los elementos de un
intervalo o colección

while

```
while (condicion) {  
    sentencia1  
    sentencia2  
}
```

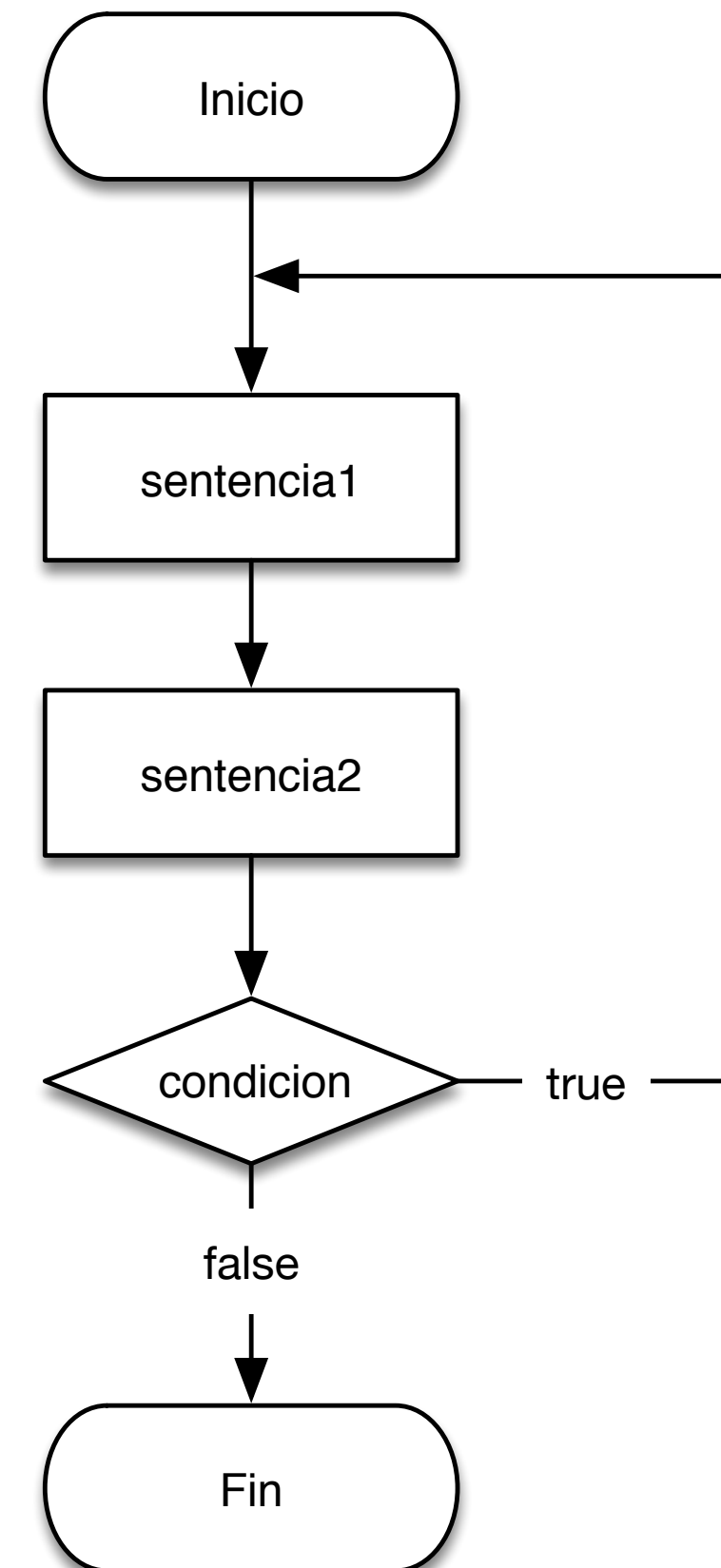


while

```
while (x > 0) {  
    x--  
}
```

do-while

```
do {  
    sentencia1  
    sentencia2  
} while (condicion)
```



do-while

```
do {  
    val y = retrieveData()  
} while (y != null) // y is visible here!
```

for

```
for (item in coleccion) sentencia
```

for

```
for (item: Int in ints) {  
    println(item)  
}
```


Recorrido de un array con índices

```
for (i in array.indices) {  
    print(array[i])  
}
```

Transferencia de control

- Se puede poner `break` dentro de un bucle para cortar la repetición actual y forzar a que el bucle termine
- Se puede utilizar `continue` dentro de un bucle para terminar la repetición actual y pasar a la siguiente
- Se pueden utilizar etiquetas para definir a quien afecta un posible `break` o `continue`

Rangos

Operaciones con rangos

Operador	Operación	Ejemplo	Valores
..	Rango cerrado	1..5	1, 2, 3, 4, 5
until	Rango abierto	1 until 5	1, 2, 3, 4
downTo	Descendente	5 downTo 1	5, 4, 3, 2, 1
step	Variación	1..5 step 2	1, 3, 5

Ejemplos

```
if (i in 1..10) { // equivalent of 1 <= i && i <= 10
    println(i)
}
```

```
for (i in 1..4) print(i) // prints "1234"
```

```
for (i in 4 downTo 1) print(i) // prints "4321"
```

```
for (i in 1..4 step 2) print(i) // prints "13"
```

```
for (i in 1 until 10) { // i in [1, 10), 10 is excluded
    println(i)
}
```

Excepciones

Gestión de excepciones

```
try {  
    // some code  
}  
catch (e: SomeException) {  
    // handler  
}  
finally {  
    // optional finally block  
}
```

Gestión de excepciones

```
val a: Int? = try { parseInt(input) } catch (e: NumberFormatException) { null }
```

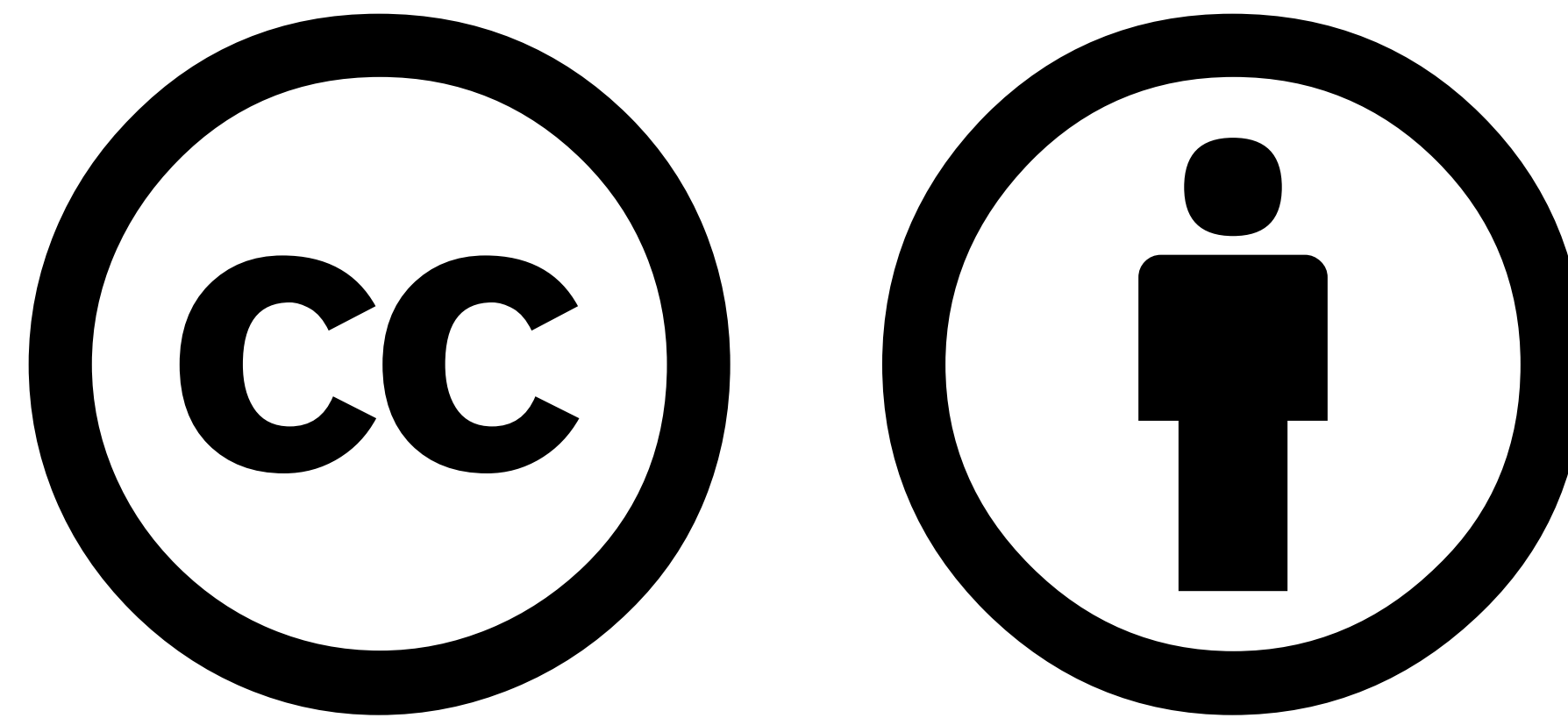

El tipo Nothing

```
val s = person.name ?: throw IllegalArgumentException("Name required")
```

El tipo Nothing

```
fun fail(message: String): Nothing {  
    throw IllegalArgumentException(message)  
}
```

```
val s = person.name ?: fail("Name required")  
println(s)           // 's' is known to be initialized at this point
```



Excepto si se especifica lo contrario, esta presentación está bajo licencia

<https://creativecommons.org/licenses/by/4.0/>

© 2017 Ion Jaureguialzo Sarasola. Algunos derechos reservados.