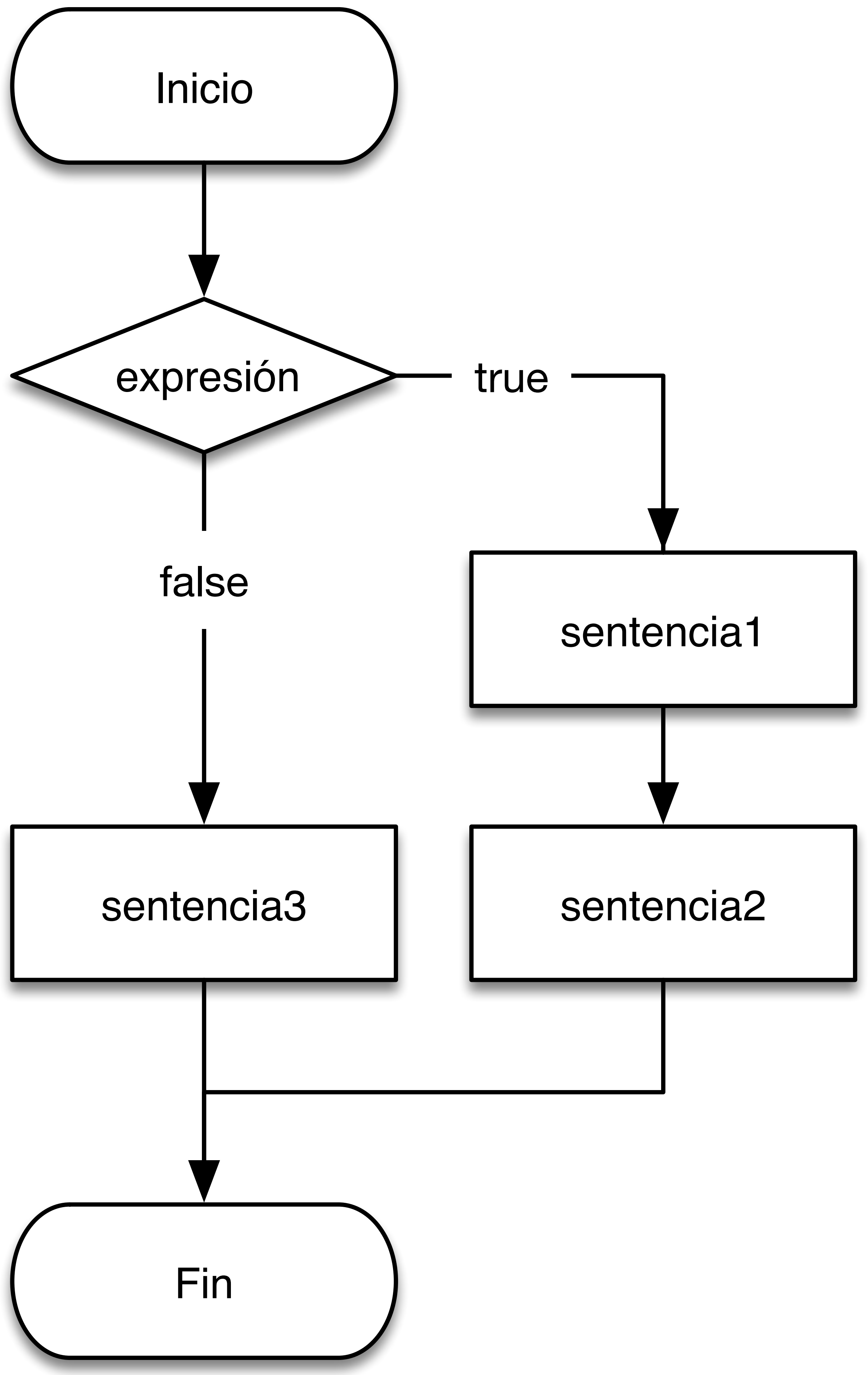
Estructuras de control

Swift 4

Alternativa simple: if

Alternativa simple: if

```
if expresión {
    sentencia1
    sentencia2
}
else {
    sentencia3
}
```



Alternativa simple: if

```swift
var temperatureInFahrenheit = 30

if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
}
```

Alternativa simple: if

```
temperatureInFahrenheit = 90

if temperatureInFahrenheit <= 32 {
    print("It's very cold. Consider wearing a scarf.")
} else if temperatureInFahrenheit >= 86 {
    print("It's really warm. Don't forget to wear sunscreen.")
} else {
    print("It's not that cold. Wear a t-shirt.")
}
```

Operadores: relacionales y lógicos

Operadores relacionales

Operador	Operación
`==`	Igual
`!=`	Distinto
`>`	Mayor que
`<`	Menor que
`>=`	Mayor o igual que
`<=`	Menor o igual que
`===`	Idéntico
`!==`	No idéntico
`c ? a:b`	Si c, entonces a. Si no c, entonces b.

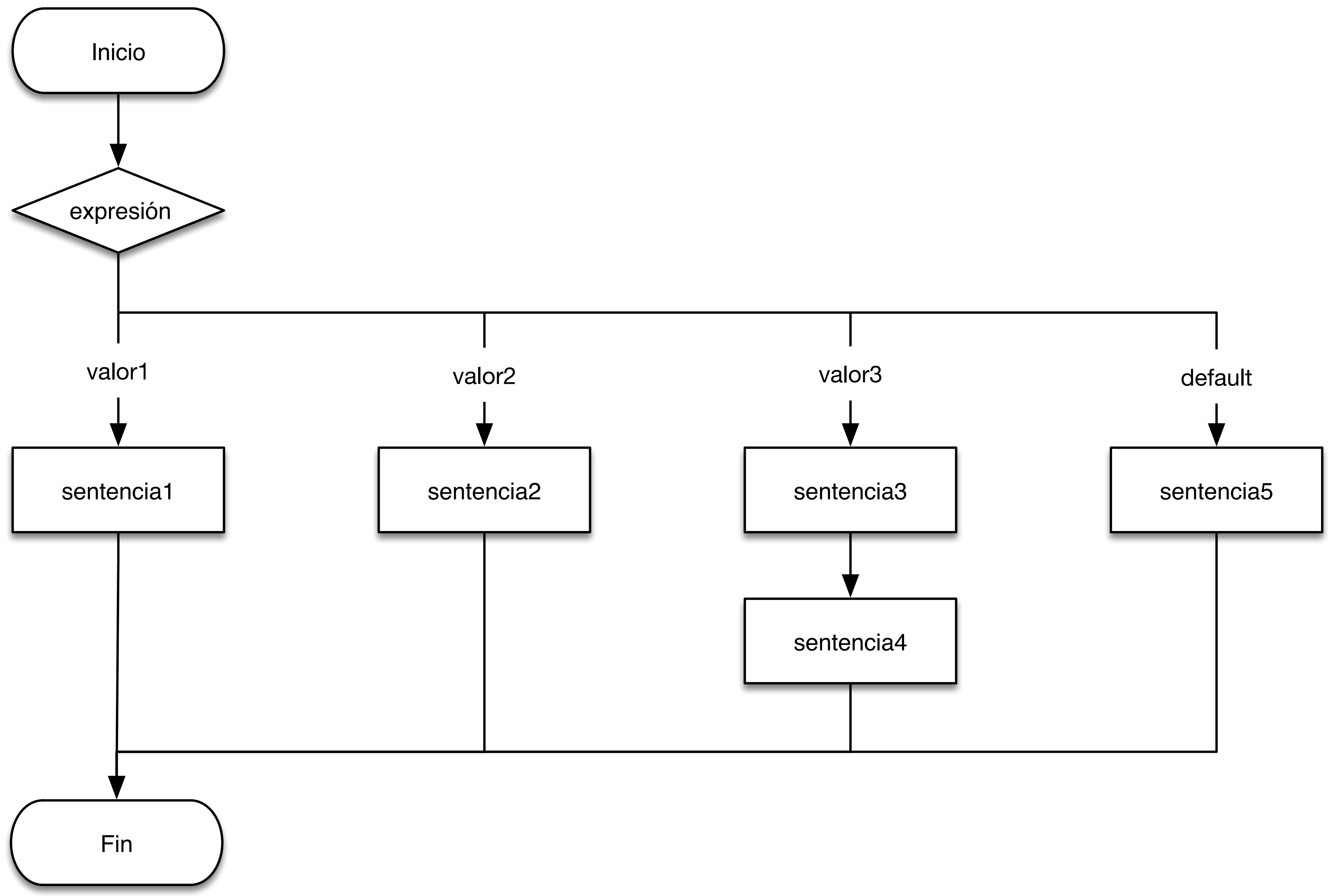
Operadores lógicos

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

Alternativa múltiple: switch

Alternativa múltiple: switch

```
switch variable {
case valor:
    sentencia
    sentencia
    ...
case valor:
    sentencia
    ...
default:
    sentencia
}
```



Alternativa múltiple: switch

```
let someCharacter: Character = "z"

switch someCharacter {
case "a":
    print("The first letter of the alphabet")
case "z":
    print("The last letter of the alphabet")
default:
    print("Some other character")
}
```

Alternativa múltiple: switch

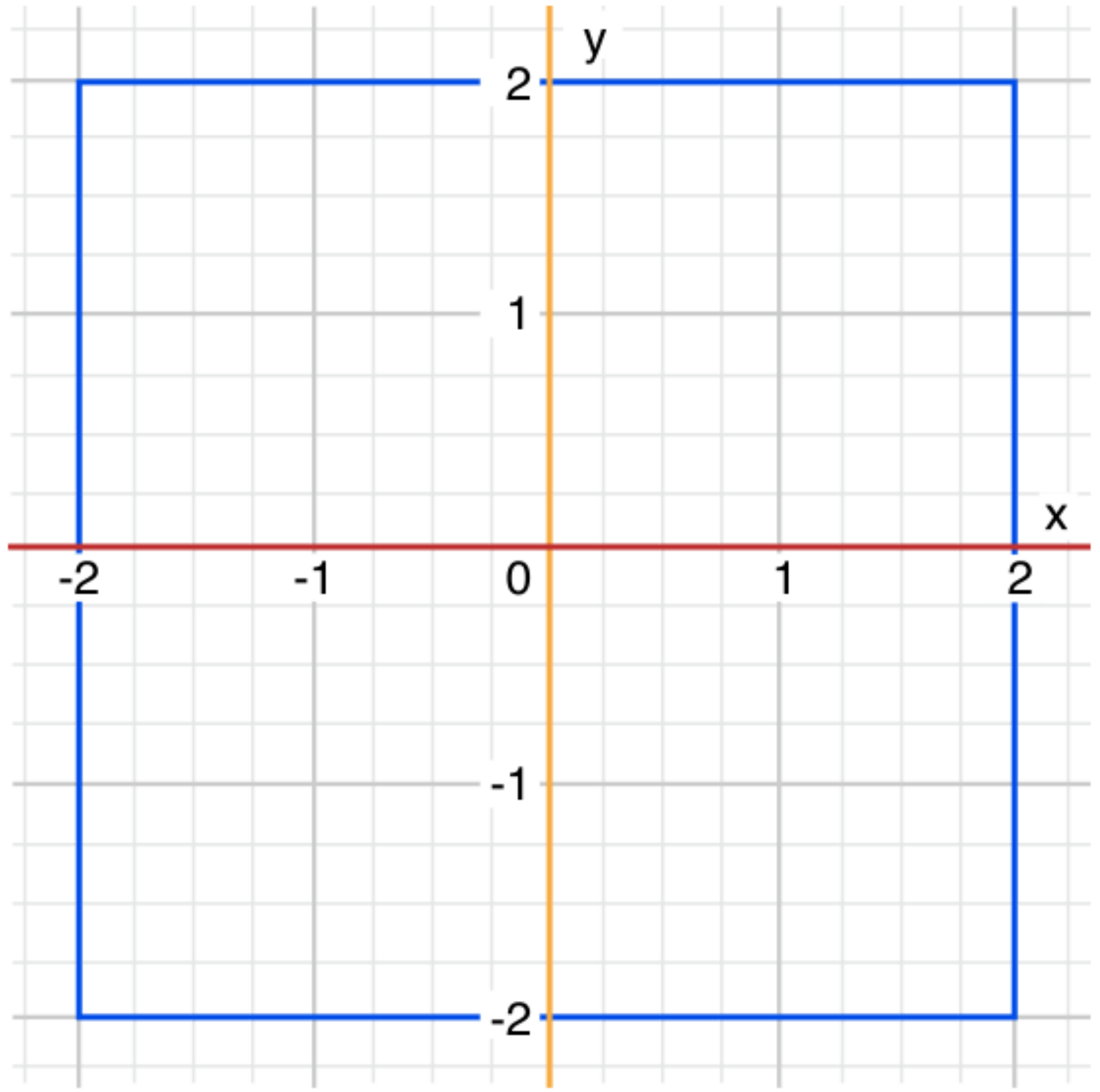
- A diferencia de en C o Java, no hace falta `break` en cada caso
- No hay *fallthrough* automático
- No puede haber casos vacíos
- Debe evaluar todos los casos posibles o tener `default`
- Se puede afinar más la condición usando `where`
- Admite intervalos y tuplas

Switch con intervalos

```swift
let approximateCount = 62
let countedThings = "moons orbiting Saturn"
var naturalCount: String

switch approximateCount {
case 0:
    naturalCount = "no"
case 1..<5:
    naturalCount = "a few"
case 5..<12:
    naturalCount = "several"
case 12..<100:
    naturalCount = "dozens of"
case 100..<1000:
    naturalCount = "hundreds of"
default:
    naturalCount = "many"
}

print("There are \(naturalCount) \(countedThings).")
```

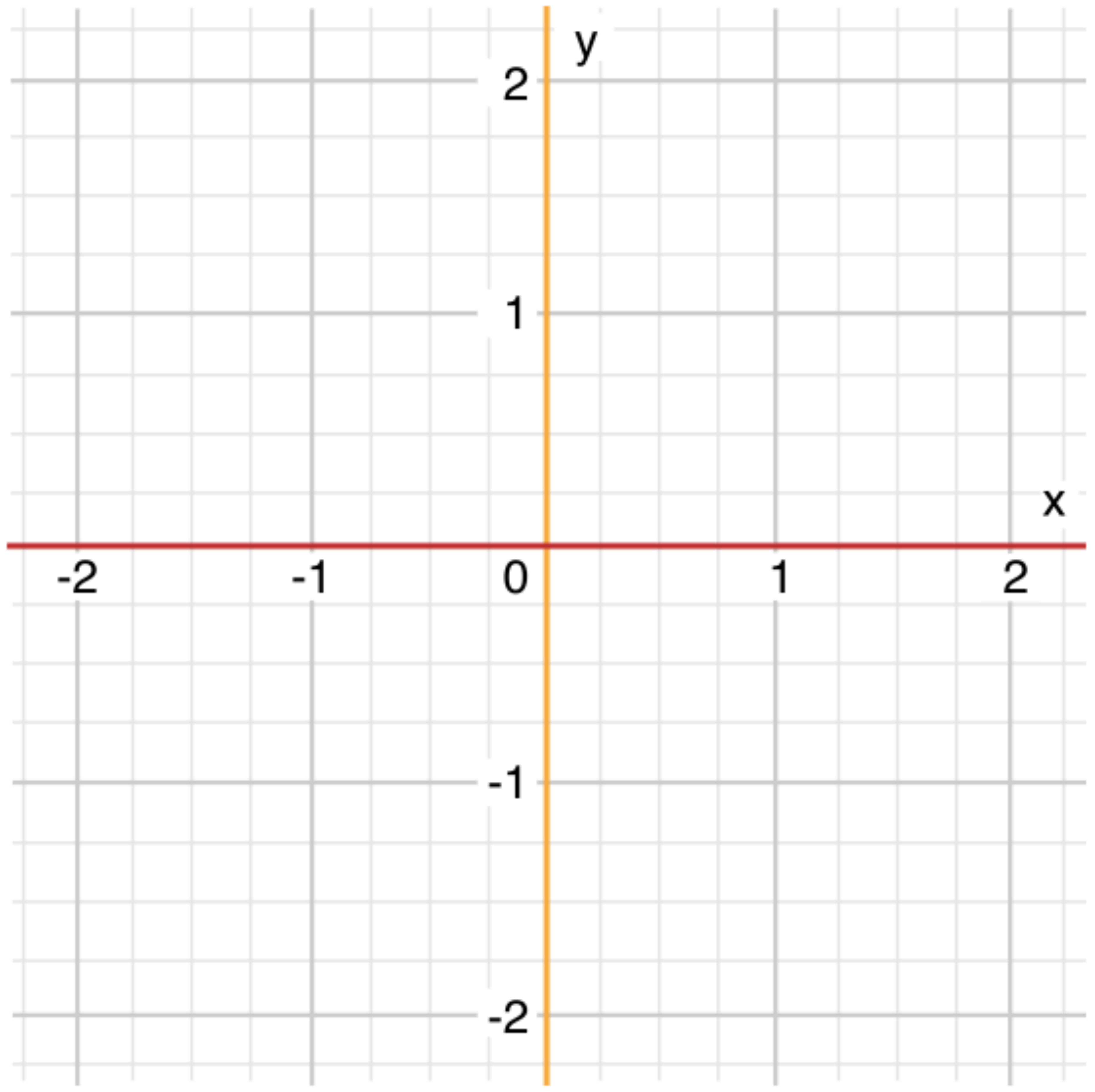
Switch con tuplas

```
let somePoint = (1, 1)

switch somePoint {
case (0, 0):
    print("(0, 0) is at the origin")
case (_, 0):
    print("(\(somePoint.0), 0) is on the x-axis")
case (0, _):
    print("(0, \(somePoint.1)) is on the y-axis")
case (-2...2, -2...2):
    print("(\(somePoint.0), \(somePoint.1)) is
inside the box")
default:
    print("(\(somePoint.0), \(somePoint.1)) is
outside of the box")
}
```



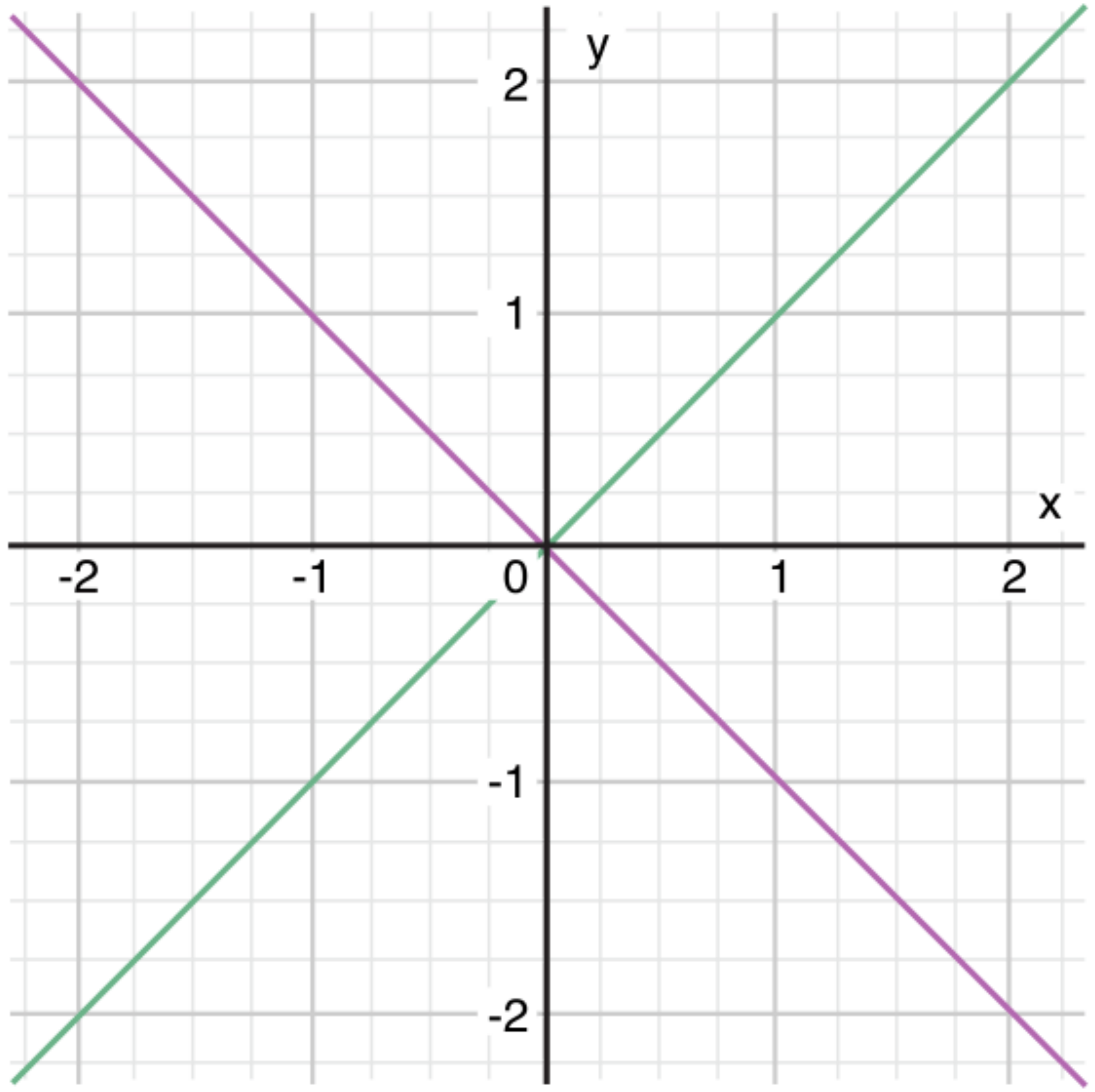
Value bindings

```
let anotherPoint = (2, 0)

switch anotherPoint {
case (let x, 0):
    print("on the x-axis with an x value of \(x)")
case (0, let y):
    print("on the y-axis with a y value of \(y)")
case let (x, y):
    print("somewhere else at (\(x), \(y))")
}
```



Switch con where

```
let yetAnotherPoint = (1, -1)

switch yetAnotherPoint {
case let (x, y) where x == y:
    print("(\(x), \(y)) is on the line x == y")
case let (x, y) where x == -y:
    print("(\(x), \(y)) is on the line x == -y")
case let (x, y):
    print("(\(x), \(y)) is just some arbitrary 
point")
}
```



Casos compuestos

```swift
let someCharacter: Character = "e"

switch someCharacter {
case "a", "e", "i", "o", "u":
    print("\(someCharacter) is a vowel")
case "b", "c", "d", "f", "g", "h", "j", "k", "l", "m",
     "n", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z":
    print("\(someCharacter) is a consonant")
default:
    print("\(someCharacter) is not a vowel or a consonant")
}
```

Transferencia de control

- Se puede poner `break` en un caso para cortar la ejecución y forzar a que el switch termine
- El uso de `break` permite escribir casos vacíos en el switch (un comentario no basta, daría error)

Fallthrough

```
let integerToDescribe = 5

var description = "The number \(integerToDescribe) is"

switch integerToDescribe {
case 2, 3, 5, 7, 11, 13, 17, 19:
    description += " a prime number, and also"
    fallthrough
default:
    description += " an integer."
}

print(description)
```

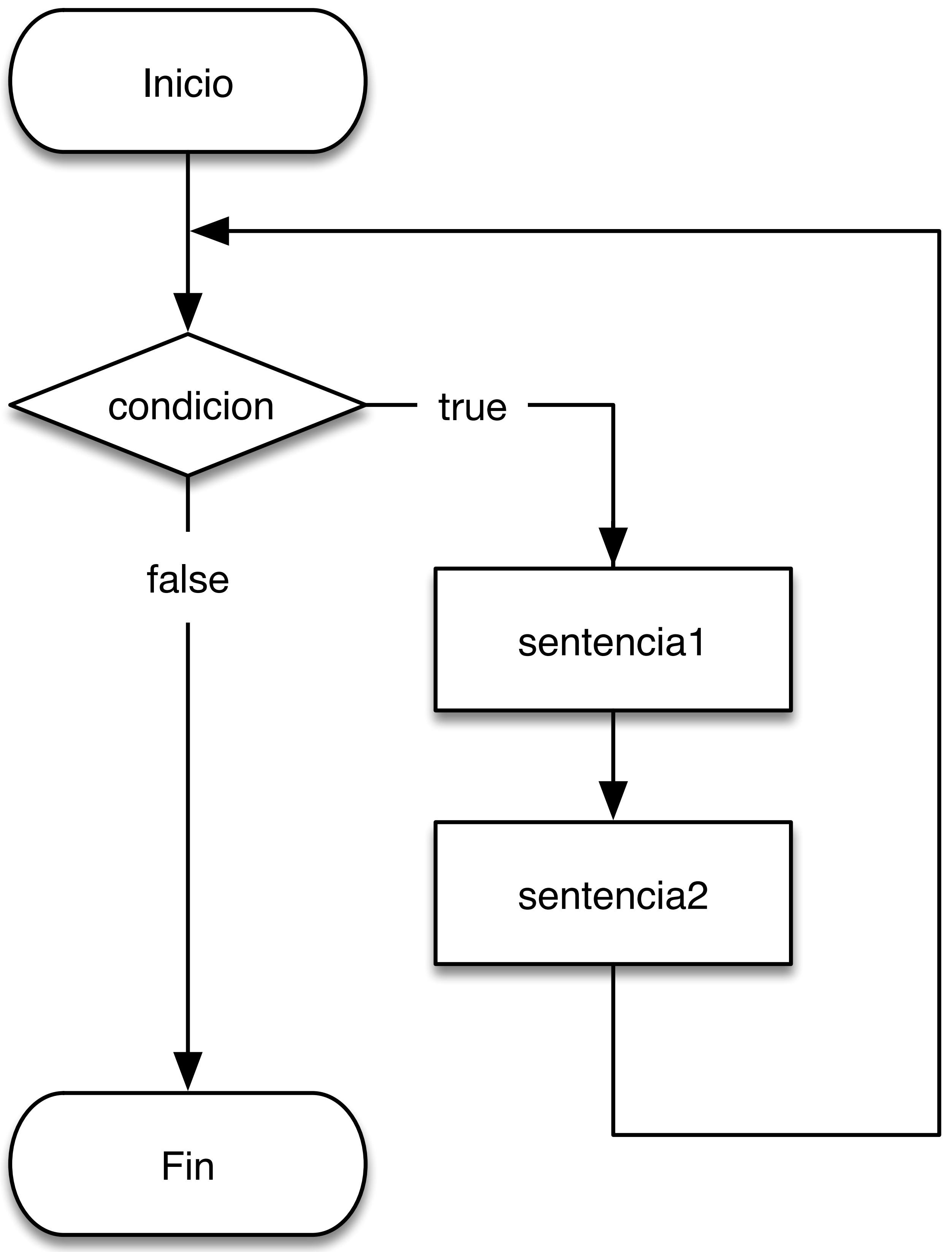
Repetitivas

Repetitivas

0 → n	1 → n	n
`while`	`repeat-while`	`for-in`
Puede que nunca se ejecute	Se ejecuta por lo menos una vez	Recorre los elementos de un intervalo o colección

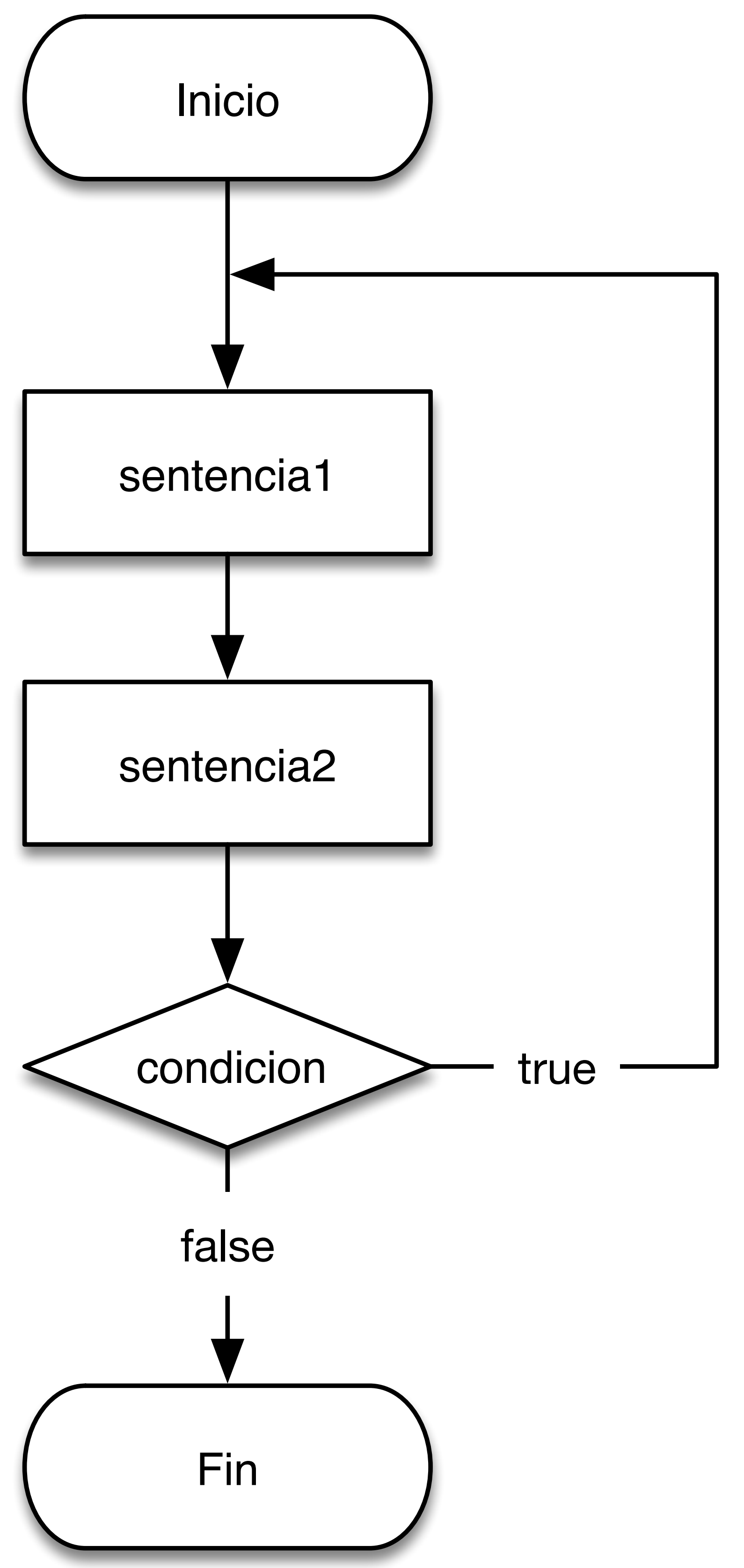
while

```
var i = 0

while i < 3 {
    print("W: El valor de i es: \(i)")
    i += 1
}
```



repeat-while

```
var j = 0

repeat {
    print("RW: El valor de j es: \(j)")
    j += 1
} while j < 3
```



for-in

```swift
for index in 1...5 {
    print("\(index) times 5 is \(index * 5)")
}
```

for-in

```
let base = 3
let power = 10

var answer = 1

for _ in 1...power {
    answer *= base
}

print("\(base) to the power of \(power) is \(answer)")
```

for-in

```swift
let names = ["Anna", "Alex", "Brian", "Jack"]
for name in names {
    print("Hello, \(name)!")
}


let numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
for (animalName, legCount) in numberOfLegs {
    print("\(animalName)s have \(legCount) legs")
}
```

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`

Operadores: rangos

Operadores de rango

Operador	Operación	Ejemplo	Valores
`n...m`	Rango cerrado	`1...5`	1, 2, 3, 4, 5
`n..<m`	Rango semicerrado	`1..<5`	1, 2, 3, 4

`n...`
`...n`
	Rango cerrado por un lado	`2...`
		`...2`
			2, 3, 4, ... final
			comienzo ... 1, 2

`..<n`	Rango semicerrado por un lado	`..<2`	0, 1

Rango cerrado

```
for index in 1...5 {
    print("\(index) times 5 is \(index * 5)")
}
```

Rango semicerrado

```swift
let names = ["Anna", "Alex", "Brian", "Jack"]
let count = names.count

for i in 0..<count {
    print("Person \(i + 1) is called \(names[i])")
}
```

Rangos de un solo lado

```
for name in names[2...] {
    print(name)
}

for name in names[...2] {
    print(name)
}

for name in names[..<2] {
    print(name)
}
```

Gestión de errores

Excepciones

```
func canThrowAnError() throws {
    // this function may or may not throw an error
}

do {
    try canThrowAnError()
    // no error was thrown
} catch {
    // an error was thrown
}
```

Salida temprana

```
func greet(person: [String: String]) {
    guard let name = person["name"] else {
        return
    }

    print("Hello \(name)!")

    guard let location = person["location"] else {
        print("I hope the weather is nice near you.")
        return
    }

    print("I hope the weather is nice in \(location).")
}

greet(person: ["name": "John"])

greet(person: ["name": "Jane", "location": "Cupertino"])
```

Comprobar versión de API

Comprobar versión de API

```
if #available(iOS 10, macOS 10.12, *) {
    // Use iOS 10 APIs on iOS, and use macOS 10.12 APIs on macOS
} else {
    // Fall back to earlier iOS and macOS APIs
}
```