Presentación

Programación en Julia: Primeros pasos

3. Control de flujo

Benjamín Pérez

Héctor Medel

Condicionales

Los condicionales nos ayudan a realizar operaciones dependiendo si una expresión es verdadera o falsa.

La sintaxis es: `if...elseif...else...end`

Ejemplo 1: Condicional clásico

```
let
    var = 7;
    if var > 10
        println("var tiene valor $var y es mayor que 10")
    elseif var < 10
        println("var tiene valor $var y es menor que 10")
    else
        println("var tiene valor $var y es 10")
    end

end
```

```
var tiene valor 7 y es menor que 10
```

El condicional puede contener varios `elseif` o puede omitirse `else`.

Ejemplo 2: Operador ternario simple

Para el caso de condiciones sencillas podemos usar el operador `?`.

```
enunciado ? acción_si_es_verdad : acción_si_es_falso
```

```
15
```

```
let
    a = 10;
    b = 15;
    z = a > b ? a : b
end
```

Ejemplo 3: Operador ternario encadenado

```
let
    var = 7
    varout = "var tiene valor $var"
    cond = var > 10 ? "y es mayor que 10." : var < 10 ? "y es menor que 10" : "y es
    10"
    println("$varout $cond")
end
```

```
var tiene valor 7 y es menor que 10
```

Ejemplo 4: Evaluación mínima (shor-circuit evaluation)

En este caso el segundo argumento es evaluado sólo si el primero no se alcanza; es un: `if...only`

Su sintaxis:

```
if <condicion>
    <accion>
end
```

Se escribe como: `<condicion> && <accion>`

```
if !<condicion>
    <accion>
end
```

Se escribe como: `<condicion> || <accion>`

```
n debe ser no negativo

  1. error(::String) @ error.jl:35
  2. (::Main.var"workspace#2".var"#sqroot#1")(::Int64) @ Local: 3
  3. top-level scope @ Local: 6
```

```
let
    function sqroot(n::Int)
        n >= 0 || error("n debe ser no negativo")
        n == 0 && return 0
    end
    sqroot(-4)
end
```

Evaluación repetida

Las ciclos sobre colección o acciones repetidas se hacen usando `for`. Se puede usar `while` para repeticiones con condición. Además, se puede influenciar la ejecución con `break` y `continue`.

for loops

Forma general

```
for i in coleccion
    #alguna acción a evaluar sobre los elementos de coleccion
end
```

Ejemplo 1: for bucle

```
let
    for n = 1:10
        println(n^3)
    end
end
```

```
1
8
27
64
125
216
343
512
729
1000
```

Ejemplo 2: for bucle sobre elementos de arreglo

```
let
    arr = [x^2  for x in 1:10]
    for i in 1:length(arr)
        println("el $i-ésimo elemento es $(arr[i])")
    end
end
```

```
el 1-ésimo elemento es 1
el 2-ésimo elemento es 4
el 3-ésimo elemento es 9
el 4-ésimo elemento es 16
el 5-ésimo elemento es 25
el 6-ésimo elemento es 36
el 7-ésimo elemento es 49
el 8-ésimo elemento es 64
el 9-ésimo elemento es 81
el 10-ésimo elemento es 100
```

Ejemplo 3: Función enumerate

```
let
    arr = [x^2  for x in 1:10]
    for (ix,val) in enumerate(arr)
        println("el $ix-ésimo elemento es $val")
    end
end
```

```
el 1-ésimo elemento es 1
el 2-ésimo elemento es 4
el 3-ésimo elemento es 9
el 4-ésimo elemento es 16
el 5-ésimo elemento es 25
el 6-ésimo elemento es 36
el 7-ésimo elemento es 49
el 8-ésimo elemento es 64
el 9-ésimo elemento es 81
el 10-ésimo elemento es 100
```

Ejemplo 4: for bucles anidaddos

```
let
    for n = 1:5
        for m = 1:5
            println("$n * $m = $(n*m)")
        end
    end
end
```

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```

Ejemplo 5: Bucle externo

```
let
    for n = 1:5, m = 1:5
        println("$n * $m = $(n*m)")
    end
end
```

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```

while bucle

Cuando necesitamos hacer un bucle considerando una condición se puede usar `while`

Ejemplo 1: while bucle

```
let
    a = 10; b = 15;
    while a < b
        println(a)
        a +=1
    end
end
```

```
10
11
12
13
14
```

Ejemplo 2: while bucle usando arreglo

```
let
    arr = [1,2,3,4];
    while !isempty(arr)
        println(pop!(arr))
    end
end
```

```
4
3
2
1
```

Break

A veces es conveniente parar un loop cuando una condición es alcanzada. Eso se puede hacer usando `break`

Ejemplo 1: `break` en while loop

```
let
    a = 10; b = 150;
    while a < b
        print(a," ")
        a += 1
        if a >= 50
            break
        end
    end
end
```

```
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 3
5 36 37 38 39 40 41 42 43 44 45 46 47 48 49
```

Ejemplo 2: `break` en for loop

```
let
    arr = rand(1:10,10);
    println(arr)
    searched = 4
    for (ix,curr) in enumerate(arr)
        if curr == searched
            println("El elemento buscado $searched se encuentra en la posición $ix")
            break
        end
    end
end
```

```
[8, 1, 10, 7, 3, 4, 3, 9, 1, 8]
El elemento buscado 4 se encuentra en la posición 6
```

Continue

Si queremos saltar una (o varias) repeticiones dentro de un loop podemos usar el comando `continue`.

Ejemplo 1

```
let
    for n in 1:10
        if 3<= n <= 6
            continue
        end
        println(n)
    end
end
```

Nota: Scope

Los bloques `for` y `while` introducen un scope nuevo en las variables; es decir, las variables definidas en esos bloques son locales, únicamente viven ahí y no podemos obtener información de ellas.

En general, explícitamente podemos etiquetar las variables en dos tipos: `global` o `local`

- `global`: Esto indica que queremos usar las variables fuera de los bloques.
- `local`: Esto indica que queremos definir una nueva variable dentro de nuestro ambiente y usarla únicamente ahí (opcional)

```
begin
    x = 9;
    function funscope(n)
        x = 0 # x es una variable local
        for i = 1:n
            local x # declara a x como variable local dentro del loop
            x = i + 1
            if x == 7
                println("Esta es la x local en el loop: $x")
            end
        end
        x
        println("Esta es una variable local dentro de la función: $x")
        global x = 10 # Esto declara a x como variable global
    end

    funscope(10)
    println("Esta es el valor global de x: $x")
end
```

```
Esta es la x local en el loop: 7
Esta es una variable local dentro de la función: 0
Esta es el valor global de x: 10
```