Capítulo 9

Condicionales y Operaciones lógicas

En ocasiones será necesario decidir qué acción realizar en función de una determinada condición, lo que en programación recibe el nombre de *flujo de control de un algoritmo o programa.*

Por ejemplo, si quiero calcular una raíz cuadrada, antes debo mirar si el número es positivo o no. Si es positivo, no tendré ningún problema, pero si es negativo xLogo dará un mensaje de error, ya que la raíz cuadrada de un negativo no es un número real.

Las estructuras condicionales son uno de los pilares de la **Programación Estructurada**, es decir, una de las piezas clave en el desarrollo de programas complejos pero fáciles de leer.

9.1. Condicional

xLogo define para ello la primitiva `si`, cuya sintaxis es:

```
si condicion
  [ Acciones a realizar si la condicion es cierta ]
```

Por ejemplo, el siguiente programa dice si un número es negativo:

```
para signo :numero
  si :numero < 0
    [ escribe [El  numero es negativo] ]
fin
```

Ahora bien, no dice nada si el número es positivo o nulo. Disponemos de otra opción:

```
si condicion
  [ Acciones a realizar si la condicion es cierta ]
  [ Acciones a realizar si la condicion es falsa ]
```

de modo que podemos mejorar nuestro programa `signo`:

```
para signo :numero
  si :numero < 0
    [ escribe [El numero es negativo] ]
    [ escribe [El numero es positivo] ]
fin
```

Si queremos que los argumentos para `cierto` y `falso` estén guardados en sendas variables, no podemos usar `si`. En este caso, la primitiva correcta es:

`sisino`

En este ejemplo, xLogo mostrará un error:

```
haz "Opcion_1 [escribe "cierto]
haz "Opcion_2 [escribe "falso]
si 1 = 2 :a :b
```

ya que la segunda lista nunca será evaluada:
`¿Qué hacer con [escribe "falso]?`

La sintaxis correcta es:

```
haz "Opcion_1 [escribe "cierto]
haz "Opcion_2 [escribe "falso]
sisino 1 = 2 :a :b
```

que devolverá:

```
"falso
```

9.2. Operaciones Lógicas

Con los condicionales es muy interesante conocer las *operaciones lógicas*:

Primitiva y Argumentos	**Uso**
`y condicion_1 condicion_2` ó `condicion_1 & condicion_2`	Devuelve `cierto` si ambas condiciones son ciertas. Si una (o las dos) son falsas, devuelve `falso`
`o condicion_1 condicion_2` ó `condicion_1 | condicion_2`	Devuelve `cierto` si al menos una de las condiciones es cierta. Si las dos son falsas, devuelve `falso`
`no condicion`	Devuelve la negación de `condicion`, es decir, `cierto` si `condicion` es falsa y `falso` si `condicion` es cierta.

Para comparaciones numéricas, disponemos de cuatro operadores que pueden expresarse de dos formas:

Operador "menor"	Operador "mayor"	Operador "menor o igual"	Operador "mayor o igual"
`<`	`>`	`<=`	`>=`
`menor?`	`mayor?`	`menoroigual?`	`mayoroigual?`

si bien es evidente que no serían estrictamente necesarios:

- `a <= b` es equivalente a `no (a > b)`
- `a >= b` puede sustituirse por `no (a < b)`

Por ejemplo (los paréntesis están para entender mejor el ejemplo):

```
para raiz_con_prueba :numero
  si o (:numero > 0) (:numero = 0)
    [ escribe raizcuadrada :numero ]
fin
```

```
para raiz_con_prueba :numero
  si (:numero > 0) | (:numero = 0)
    [ escribe raizcuadrada :numero ]
fin
```

que comprueba si el número es positivo **ó** cero antes de intentar calcular la raíz. Este procedimiento podría hacerse con `no`:

```
para raiz_con_prueba :numero
  si no (:numero < 0)
    [ escribe raizcuadrada :numero ]
fin
```

y ahora comprueba que **no** sea negativo.

Imaginemos ahora un procedimiento que diga si la temperatura exterior es agradable o no:

```
para agradable :temperatura
  si y (:temperatura < 25) (:temperatura > 15)
    [ escribe [La temperatura es agradable] ]
    [ si no (:temperatura > 15)
      [ escribe [Hace frío] ]
      [ escribe [Hace demasiado calor] ] ]
fin
```

```
para agradable :temperatura
  si (:temperatura < 25) & (:temperatura > 15)
    [ escribe [La temperatura es agradable] ]
    [ si no (:temperatura > 15)
      [ escribe [Hace frio] ]
      [ escribe [Hace demasiado calor] ] ]
fin
```

que estudia primero si la temperatura está en el intervalo `(15 , 25)` – o sea `15 < T < 25` – y si lo está dice `La temperatura es agradable`. Si no pertenece a ese intervalo, analiza si está por debajo de él (y dice `Hace frio`) o no (y devuelve `Hace demasiado calor`). Hemos *encadenado* condicionales.

Recuerda que tanto `y` como `o` admiten una forma general y, por tanto, pueden efectar más de una comparación:

```
si (y condicion1 condicion2 condicion3 ...) [ ... ]
si (o condicion1 condicion2 condicion3 ...) [ ... ]
```

son equivalentes a:

```
si (condicion1 & condicion2 & condicion3 & ...) [ ... ]
si (condicion1 | condicion2 | condicion3 | ...) [ ... ]
```

9.3. Ejercicios

1. Modifica el procedimiento `raiz_con_prueba` para **no** usar ninguna operación lógica

2. Plantea un procedimiento `no_menor` que decida si, dados dos números, el primero es mayor o igual que el segundo y responda `sí` en caso afirmativo
   `no_menor 8 5` ⟶ `sí`
   `no_menor 3 5` ⟶ (nada)

3. Plantea el procedimiento `edad_laboral`, que compruebe si la `edad` de una persona verifica la condición `17 < edad < 65`, respondiendo `Está en edad laboral` en caso afirmativo

4. Escribe el procedimiento `múltiplo?`, que verifique si un número `dato` es múltiplo de otro `divisor`, respondiendo `Es múltiplo` o `No es múltiplo`, en cada caso.

   Pista: puedes usar `resto`, `cociente` y/o `division` (`/`)
   En caso de que sea múltiplo, la tortuga debe dibujar un rectángulo cuya base sea el divisor y su área el múltiplo
   `múltiplo? 18 5` ⟶ `No es múltiplo`
   `múltiplo? 320 40` ⟶ `Es múltiplo`

5. Diseña un procedimiento que diga si un año es bisiesto no. Recuerda que un año es bisiesto si es múltiplo de 4, pero no es múltiplo de 100 aunque sí de 400.
   `bisiesto? 1941` ⟶ `1941 no es bisiesto`
   `bisiesto? 1900` ⟶ `1900 no es bisiesto`
   `bisiesto? 2000` ⟶ `2000 sí es bisiesto`

6. Diseña el procedimiento `calificaciones` que, dada una `nota` la califique de acuerdo con el baremo usual:

Nota	$n < 5$	$5 \leq n < 6$	$6 \leq n < 7$	$7 \leq n < 9$	$9 \leq n < 10$
Calificación	Suspenso	Aprobado	Bien	Notable	Sobresaliente

7. Diseña un programa que calcule la hipotenusa de un triángulo rectángulo dados sus catetos, pero que llame a un subprocedimiento que devuelva el cuadrado de un número dado, y que además dibuje el triángulo en pantalla

   Pista: Coloca los catetos paralelos a los ejes cartesianos, y y utiliza las primitivas asociadas a las coordenadas

8. Diseña un programa que dibuje un triángulo dados sus tres lados.

Cuidado: Dados tres lados, no siempre es posible construir un triángulo. Piensa qué condición debe cumplirse para que sea posible dibujarlo y después realiza los cálculos trigonométricos necesarios.

9. Plantea el procedimiento `mismo_signo`, que decida si dos números no nulos tienen el mismo signo. **Pista**: Comprueba el signo de su producto

9.4. Booleanos

Una variable o primitiva es *booleana* si sus únicos valores posibles son `cierto` o `falso`. En xLogo un booleano es la respuesta a las primitivas terminadas con `?`

Primitivas	Argumentos	Uso
`cierto`	`ninguno`	Devuelve `"cierto`
`falso`	`ninguno`	Devuelve `"falso`
`palabra?`	`variable`	Devuelve `cierto` si `a` es una palabra, `falso` si no.
`numero?`	`variable`	Devuelve `cierto` si `a` es un número, `falso` si no.
`entero?`	`un número`	Devuelve `cierto` si `a` es un número entero, `falso` si no.
`lista?`	`variable`	Devuelve `cierto` si `a` es una lista, `falso` si no.
`vacio?`	`variable`	Devuelve `cierto` si `a` es una lista vacía o una palabra vacía, `falso` si no.
`iguales?`, `=`	`a b`	Devuelve `cierto` si `a` y `b` son iguales, `falso` si no.
`antes?`, `anterior?`	`a b` (dos palabras)	Devuelve `cierto` si `a` está antes que `b` siguiendo el orden alfabético, `falso` si no.
`miembro?`	`a b`	Si `b` es una lista, determina si `a` es un elemento de `b`. Si `b` es una palabra, determina si `a` es un caracter de `b`.
`cuadricula?`	`no`	Devuelve `cierto` si la cuadrícula está activa, `falso` si no.
`ejex?`,	`no`	Devuelve `cierto` si está activo el eje de abscisas (eje X), `falso` si no.
`ejey?`,	`no`	Devuelve `cierto` si está activo el eje de ordenadas (eje Y), `falso` si no.
`bajalapiz?`, `bl?`	`ninguno`	Devuelve la palabra `cierto` si el lápiz está abajo, `falso` si no.
`visible?`	`ninguno`	Devuelve la palabra `cierto` si la tortuga está visible, `falso` si no.
`primitiva?`, `prim?`	`una palabra`	Devuelve `cierto` si la palabra es una primitiva de xLogo, `falso` si no.

Primitivas	Argumentos	Uso
`procedimiento?`, `proc?`	`una palabra`	Devuelve `cierto` si la palabra es un procedimiento definido por el usuario, `falso` si no.
`variable?`, `var?`	`una palabra`	Devuelve `cierto` si la palabra es una variable definida por el usuario, `falso` si no.

En la sección 5.1 te pedimos que hicieras pruebas con determinados nombres de procedimientos. Si queremos estar seguros de que un `nombre` está "libre" antes de usarlo, podemos preguntar:

```
variable? "nombre
```

y en caso que que la variable `nombre` no haya sido definida, asignarle un valor:

```
si no (variable? "nombre) & no (procedimiento? "nombre)
  [ haz "nombre pi ]
```

Las primitivas `cuadricula?`, `ejes?`, ... permiten controlar aspectos que serán importantes, por ejemplo, al colorear regiones (sección 13.2.3).

Analiza ahora el siguiente procedimiento:

```
para cuadrado_cortado
  haz "paso cambiasigno 50
  repite 4
    [ repite 3
        [ si bajalapiz? [subelapiz] [bajalapiz]
          avanza 75 ]
      giraderecha 90 ]
fin
```

¿Eres capaz de reproducir su resultado?

9.5. Ejercicios

1. Utilizando las primitivas `listavars` y `listaprocs`, intenta crear un procedimiento que investigue si una determinada palabra es una variable o un procedimiento sin usar las primitivas `variable?` o `procedimiento?` (ni sus formas cortas)

2. Diseña un procedimiento que compruebe si la cuadrícula está activa o no.

   - Si está activa, que la borre y dibuje otra de dimensiones `[100 * 100]`
   - Si no lo está, que dibuje una de dimensiones `[50 * 50]`

   A continuación, que sitúe a la tortuga en una posición generada por dos números aleatorios de la serie:

   ```
   { -505 , -455 , -405 , ... , 405 , 455 , 505 }
   ```

y rellene o no el cuadrado en el que se encuentra siguiendo la secuencia de un tablero de ajedrez.

Finalmente, repetir la secuencia varias veces hasta ver si se consigue dibujar un tablero de ajedrez