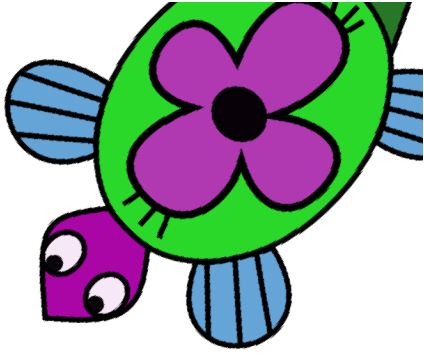


Capítulo 7

Operaciones

¿Qué ocurre si necesitamos realizar operaciones en xLogo? Disponemos de las siguientes primitivas:

7.1. Operaciones binarias

Son aquellas que implican a dos elementos.

7.1.1. Con números

Descripción	Primitiva/s	Ejemplo
Sumar dos números	`suma` ó `+`	`suma 15 20` ó `15 + 20` devuelven `35`
Multiplicar dos números	`producto` ó `*`	`producto (-5) 6` ó `(-5) * 6` devuelven `-30`
Restar dos números	`diferencia` ó `-`	`diferencia 35 (-10)` ó `35 - (-10)` devuelven `45`
Dividir dos números reales	`division` ó `/`	`division 35 3` ó `35 / 3` devuelven `11.666666666666666`
Cociente entero	`cociente`	`cociente 35 3` devuelve `11`
Resto de una división entera	`resto`	`resto 35 3` devuelve `2`
Calcular una potencia	`potencia`	`potencia 3 1.5` devuelve `5.196152422706632`

Si se trata de sumas o productos de varios números, podemos usar la forma general de `suma` y `producto`:

```
13 + 7 + 2.5                (suma 13 7 2.5)
13 * 7 * 2.5                (producto 13 7 2.5)
```

Fíjate en los paréntesis de la segunda forma; son obligatorios. El resultado puede ser simplemente mostrado por pantalla:

```
escribe 13 + 7 + 2.5          escribe (suma 13 7 2.5)
escribe 13 * 7 * 2.5          escribe (producto 13 7 2.5)
```

o puede ser usado para dibujar:

```
avanza 13 + 7 + 2.5       giraderecha (suma 13 7 2.5)
retrocede 13 * 7 * 2.5    giraizquierda (producto 13 7 2.5)
```

Si, por ejemplo, queremos calcular cuántas formas distintas tenemos de rellenar una quiniela, le pediremos a la *tortuga*:

```
escribe potencia 3 15
```

devuelve

```
1.4348907E7
```

(3^{15} = 14 348 907), el resultado se muestra en notación científica.

Analiza el siguiente procedimiento. Recuerda el uso de la barra invertida (sección 1.10.1) para generar espacios y saltos de línea. Te presentamos además la primitiva `tipea`, que escribe en el Histórico de comandos pero, a diferencia de `escribe`, no produce un salto de línea

```
para tabla
  haz "contador1 1
  haz "espacio [ \  ]
  haz "saltolinea [ \n ]
  repite 9
    [ haz "contador2 1
      repite 9
        [ tipea (:contador1 * :contador2)
          tipea :espacio
          haz "contador2 :contador2 + 1 ]
      tipea :saltolinea
      haz "contador1 :contador1 + 1 ]
fin
```

¿Qué crees que hacen las variables `contador1` y `contador2`? ¿Por qué crees que las hemos usado? ¿Se te ocurre una forma mejor de conseguir lo mismo? ¿Cómo mejorarías el aspecto con el que salen los los resultados?

7.1.2. Con listas

No sólo pueden efectuarse operaciones con números. Las palabras y las frases (listas de palabras) pueden *concatenarse* (ponerse una a continuación de la otra). Disponemos de las primitivas `frase` y `lista` (haremos un estudio más profundo de las listas en el capítulo 10). Por ejemplo:

```
escribe frase [El gato es ] [gris]
```

muestra en pantalla:

```
El gato es gris
```

pero:

```
escribe lista [El gato es ] [gris]
```

muestra en pantalla:

```
[El gato es] [gris]
```

es decir, crea una lista cuyos elementos son los argumentos, lo que en este caso lleva a obtener una *lista de listas.*

También es posible concatenar listas con números o variables. Por ejemplo, si la variable `area` contiene el valor `250` podemos pedirle a XLOGO:

```
escribe frase [La superficie es ] :area
```

que proporciona:

```
La superficie es 250
```

ya que `frase` ha concatenado la *lista* `La superficie es` y el valor de `:area`

Concatenar listas con variables es una forma de que los mensajes de XLOGO a la hora de presentar los resultados se entiendan mejor.

Ejemplo: Vamos a crear un procedimiento que calcule el área de un triángulo dándole la base y la altura

```
Recuerda que el área de un triángulo es: A = b * h / 2
```

```
para area_triangulo :base :altura
  haz "area (:base * :altura) / 2
  escribe :area
fin
```

o bien:

```
para area_triangulo :base :altura
  haz "area (division (producto :base :altura) 2)
  escribe :area
fin
```

Para ejecutarlo, escribimos:

```
area_triangulo 3 5
```

Es posible ser un poco más elegante. ¿Tú sabrías lo que hace este programa sólo viendo los resultados o, incluso, leyéndolo? Cambiemos la penúltima línea:

```
para area_triangulo :base :altura
  haz "area (division (producto :base :altura) 2)
  escribe frase [El área del triángulo es ] :area
fin
```

o bien:

```
para area_triangulo :base :altura
# Este procedimiento calcula el área de un triángulo
# pidiendo su base y altura
  haz "area (division (producto :base :altura) 2)
  haz "texto frase [El área del triángulo de base] :base
  haz "texto frase :texto [y altura]
  haz "texto frase :texto :altura
  haz "texto frase :texto [es]
  haz "texto frase :texto :area
  escribe :texto
fin
```

que al ejecutarlo:

```
area_triangulo 3 5
```

proporciona:

```
El area del triangulo de base 3 y altura 5 es 7.5
```

¿No se entiende mejor?

Observa otra capacidad del lenguaje xLogo. Hemos *reutilizado* la variable `texto` varias veces, incluyendo en su definición **a ella misma**. Esto es útil cuando no quieres definir varias variables para un proceso que se refiere a una misma cosa (en este caso ir aumentando palabra a palabra el texto a mostrar en pantalla).

Podríamos haber aprovechado la *forma general* de la primitiva `frase`:

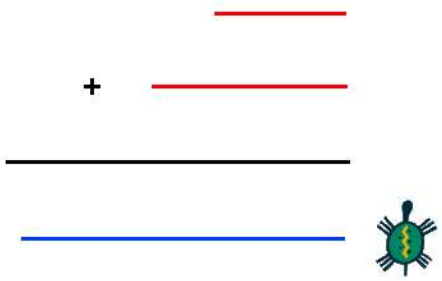
```
para area_triangulo :base :altura
# Este procedimiento calcula el área de un triángulo
# pidiendo su base y altura
  haz "area (division (producto :base :altura) 2)
  escribe (frase [El área del triángulo de base ] :base [y altura ] :altura [es ] :area)
fin
```

7.2. Ejercicios

1. Modifica el procedimiento `tabla` presentado como ejemplo para que muestre la *tabla de restar* y la *tabla de multiplicar*. ¿Qué observas?

2. Crea un procedimiento `acumula` que reciba dos números (`n1` y `n2`) como argumentos, dibuje dos segmentos cuyas longitudes sean precisamente `n1` y `n2` separados por un signo "más" y a continuación dibuje el segmento cuya longitud sea la suma de `n1` y `n2` precedido del signo "igual".

3. Modifica el procedimiento anterior para que los segmentos aparezcan colocados como en las sumas tradicionales: los sumandos unos encima de otros, el signo "más" a la izquierda, una línea de "operación" y el resultado en la parte inferior. Usa colores para diferenciar los sumandos del signo y la línea y éstos del resultado



(Nota que en este caso los segmentos empiezan a dibujarse desde la derecha)

4. A partir de los procedimientos anteriores, construye los procedimientos `sustrae` que en lugar de sumas hagan "restas de segmentos".

	Prueba distintas posibilidades, observando qué ocurre cuando `n1` es mayor que `n2` y viceversa.

5. Crea un procedimiento `superficie` con dos argumentos, `n1` y `n2`, dibuje un rectángulo cuyos lados midan `n1` y `n2` y calcule su área.

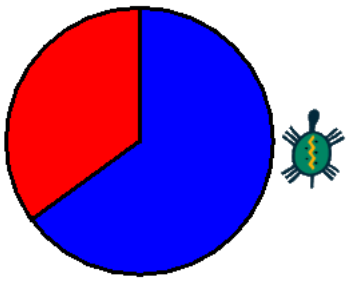
6. Echa un vistazo a tu clase, y cuenta el número de chicos y de chicas que hay. Con esa información, crea el procedimiento `quesitos` que:

   a) Calcule cuántos/as estudiantes hay en tu clase

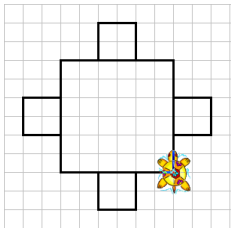
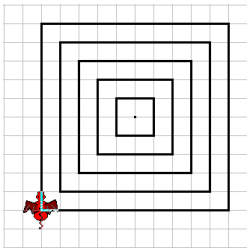
   b) Dibuje una circunferencia. Para ello usa la primitiva `circulo`, que necesita un argumento (`número`), y dibuja una circunferencia de radio `número` centrada en la posición actual de la tortuga

c) Divida el círculo en dos partes proporcionales al número de chicas y chicos y los coloree con distintos colores

Por ejemplo, si en una clase hay un 40 % de chicos y un 60 % de chicas, el dibujo debería ser algo así:

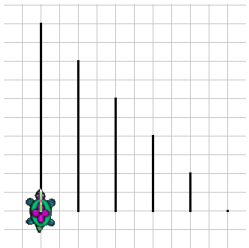
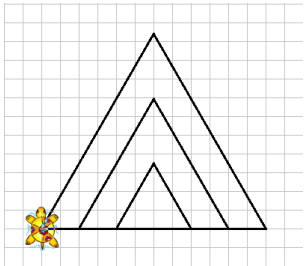


7. Con el procedimiento `cuadrado` definido en el capítulo anterior, dibuja:

a)  *b*) 

8. Programa un procedimiento que dibuje una fila horizontal de `n` baldosas cuadradas cuyo lado sea la variable `lado` y que aparezcan centradas en pantalla

9. Dibuja:

a)  *b*) 

10. Escribe un procedimiento `rayos`, que dibuje los `n` radios de longitud `largo` de una rueda (sólo los radios)

7.3. Operaciones unitarias

Son aquellas que sólo necesitan un elemento:

Descripción	Primitiva/s	Ejemplo
Hallar la raíz cuadrada de un número	`raizcuadrada` ó `rc`	`raizcuadrada 64` ó `rc 64` devuelven `8`
Cambiar el signo	`cambiasigno` ó `-`	`cs 5` devuelve `-5`
Valor absoluto de un número	`absoluto` ó `abs`	`abs (-5)` devuelve `5`
Truncamiento de un número decimal	`trunca`	`trunca 3.565` devuelve `3`
Redondear un número al entero más cercano	`redondea`	`redondea 3.565` devuelve `4`
Generar un número aleatorio comprendido entre `0` y `n-1`	`azar n`	`azar 6` devuelve un entero entre `0` y `5`
Generar un número aleatorio comprendido entre `0` y `1`	`aleatorio`	`aleatorio` devuelve un número entre `0` y `1`

La primitiva `truncar` puede usarse para conseguir un número entero cuando nos aparece en notación científica. En el apartado 7.1.1 calculábamos cuántas formas distintas tenemos de rellenar una quiniela, y escribimos:

```
escribe potencia 3 15
```

Si usamos

```
escribe truncar potencia 3 15
```

nos devuelve:

```
14348907
```

Ejemplo: Queremos un procedimiento que calcule el cociente y el resto de la división entera entre `A` y `B`, es decir, conseguir lo que proporcionan las primitivas `cociente` y `resto`, pero sin usarlas.

```
para division_entera :A :B
  haz "C truncar (:A / :B)
  escribe :C
  escribe :A - :B * :C
fin
```

La primitiva `azar` devuelve números aleatorios comprendidos entre `0` y `n - 1`. ¿Cómo harías para simular el lanzamiento de un dado? Lee con cuidado la sección 7.8 para evitar conclusiones erróneas.

7.4. Ejercicios

1. Modifica el procedimiento `division_entera` de modo que devuelva los resultados indicando con mensajes qué es cada número.
2. Plantea un procedimiento `Pitagoras` que refiriéndose a un triángulo rectángulo, acepte los valores de los catetos y calcule el valor de la hipotenusa.
3. Intenta escribir un procedimiento `suerte`, que genere un número al azar del conjunto `{20 , 25 , 30 , 35 , 40 , 45 , 50}`.[1]
4. Plantea un procedimento `dados`, que simule el lanzamiento de **dos** dados y cuya salida sea la suma de ambos
5. El procedimiento `juego` con el que presentamos a la tortuga en el capítulo 3 ubica las "piedras" de forma aleatoria. Piensa cómo harías para ubicar `n` "piedras" circulares y coloreadas en posiciones aleatorias
6. Plantea un procedimiento que simule un sorteo de Lotería Nacional, esto es, generar un número de 5 cifras por extracción consecutiva de 5 "bolas" con valores comprendidos entre 0 y 9 y colocarlas una a continuación de otras (recuerda la prmitiva `tipea`)

En un sorteo de la "Lotería Primitiva" hay 49 bolas numeradas en un único bombo, y de él se extraen consecutivamente 6 bolas. ¿Podríamos usar la primitiva `azar` para simular un sorteo de la Primitiva?

7.5. Cálculo superior

xLogo puede evaluar funciones trigonométricas y logarítmicas.

Descripción	Primitiva/s	Ejemplo
Calcular el seno de un ángulo	`seno` o `sen`	`seno 45`
Calcular el coseno de un ángulo	`coseno` o `cos`	`coseno 60`
Calcular la tangente de un ángulo	`tangente`, `tg` o `tan`	`tangente 135`
Obtener el ángulo cuyo seno es `n`	`arcoseno`, `arcsen` o `asen`	`arcsen 0.5`
Obtener el ángulo cuyo coseno es `n`	`arcocoseno`, `arccos` o `acos`	`arccos 1`
Obtener el ángulo cuya tangente es `n`	`arcotangente`, `arctg` o `atan`	`arctg 1`
Evaluar el logaritmo decimal de un número	`log` ó `log10`	`log 100`
Evaluar el logaritmo neperiano de un número	`logneperiano` ó `ln`	`ln 100`
Exponencial de un número (e^x)	`exp`	`exp 2`

[1] Recuerda que `azar 5 + 10` es lo mismo que `azar (5 + 10)`, es decir, `azar 15`. Igualmente `10 + azar 5` es equivalente a `(azar 5) + 10`

7.6. Precisión en los cálculos

En ocasiones, necesitaremos trabajar con más precisión de la que el programa permite en sus ajustes iniciales. Si queremos que xLogo realice los cálculos utilizando un mayor número de dígitos, debemos utilizar la primitiva `pondigitos` o `pondecimales`.

Las dos formas de esta primitiva se deben a las diferencias de nivel académico a las que podemos enfrentarnos, siendo más fácil entender para un niño pequeño que la precisión aumenta con *los decimales*. Por lo demás, su sintaxis es muy simple:
`pondecimales n` o `pondigitos n`
fija la precisión del cálculo:

1. Por defecto, xLogo usa 16 dígitos.
2. Si `n` es negativa, se vuelve al valor inicial (el valor inicial es -1).
3. Si `n` es cero, todos los números se redondean a la unidad.

De modo equivalente, la primitiva `decimales` devuelve el número de dígitos utilizados en el cálculo.

En el apartado 7.1 calculábamos el número de combinaciones distintas en una quiniela (3^{15}) usando la primitiva `potencia`: `escribe potencia 3 15` –>`1.4348907E7`.
Si tecleamos antes: `pondecimales 0`, la misma orden nos devuelve ahora `14348907`
Un ejemplo más avanzado se encuentra en la página 101.

7.7. Ejercicios

Pista para los tres problemas siguientes: Puedes usar las primitivas `repite` y `contador`[2] y una variable en la que ir guardando los resultados parciales.

1. ¿Sabrías crear un procedimiento `poten :x :n` que calcule x^n (supuesto que `n` es un número natural) sin usar la primitiva `potencia`?
2. Crea un procedimiento `factorial :n`, que calcule el factorial del número `n`.

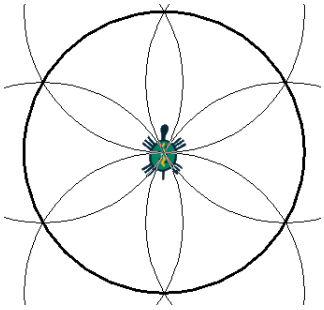
   Recuerda: `n! = n * (n - 1) * ...* 2 * 1`
3. Intenta ahora conseguir un procedimiento `suma_potencias :n` que calcule la suma `2 + 4 + 8 + ...+` 2^n

   Para los problemas siguientes necesitarás hacer algunos cálculos trigonométricos antes de ponerte a programar.

[2] Échale un vistazo a la sección 11.1.1 para entenderlas bien

4. Plantea un procedimiento `poligono_regular`, que dibuje un polígono regular de `n` lados de longitud `lado` inscrito en una circunferencia

5. La flor *hexapétala* (conocida como flor de agua o flor galana en Asturias) es muy sencilla de dibujar únicamente con un compás:



   Plantea un procedimiento `flor` que haga uso de la primitiva `arco` (`arco` necesita tres argumentos: el radio, el ángulo donde empieza y el ángulo donde termina, medido siempre en forma absoluta, es decir, 0° es **siempre** hacia arriba, independientemente de hacia dónde "mire" la tortuga) y que reciba un argumento `radio` para dibujar la flor galana.

6. Plantea un procedimiento con dos argumentos: `radio` y `n`, que dibuje un círculo con ese `radio`, y que inscriba en él un polígono de `n` lados.

7. (**Este problema requiere importantes conocimientos de geometría y/o dibujo técnico**) Plantea un procedimiento que dibuje una flor, pidiendo el `radio` y el número `n` de pétalos que tiene.

7.8. Prioridad de las operaciones

Utilizando la forma corta, `+`, `-`, `*` y `/`, xLogo realiza las operaciones (como no podía ser de otra manera) obedeciendo a la prioridad de las mismas. Así si escribimos:

```
escribe 3 + 2 * 4
```

xLogo efectúa primero el producto y luego la suma, siendo el resultado `11`.

Como sabemos, la presencia de paréntesis modifica el orden en que se deben realizar las operaciones. Por ello, si escribimos:

```
escribe (3 + 2) * 4
```

xLogo efectuará la suma antes que el producto, y el resultado será `20`.

Hay que tener cuidado, y esto es **muy importante**, si se usan las primitivas `suma`, `diferencia`, `producto`, `division`, `potencia`, ..., ya que **internamente** tienen una prioridad inferior a las anteriores. Este es un comportamiento que estamos intentando solucionar, ya que incumple las reglas por todos conocidas. Hasta entonces, aconsejamos utilizar los paréntesis para evitar sorpresas desagradables:

```
escribe (seno 20) + 10          --> 10.3420201433256687
escribe (raizcuadrada 3) + 1    --> 2.7320508075688772
escribe (azar 6) + 1            --> 1, 2, 3, 4, 5 o 6
```