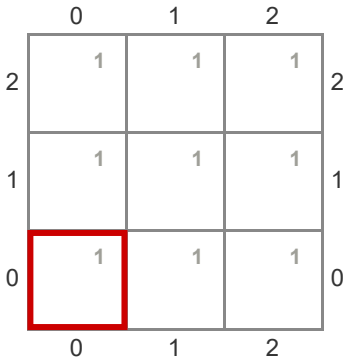
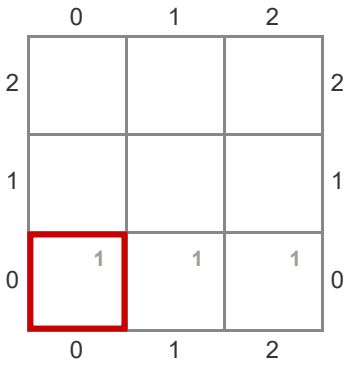
Pensando en subtareas

¡Queremos dibujar un nuevo cuadrado!



Dividiendo cada parte del problema en procedimientos más pequeños, podemos plantear la siguiente **estrategia**: construir el cuadrado como tres líneas de tres bolitas una encima de la otra. ¿A qué nos referimos con una línea de tres bolitas? A esto:



¡Arranquemos por ahí!

> Definí el procedimiento `DibujarLineaNegra3` que, como su nombre lo indica, *dibuje una línea* poniendo 3 bolitas negras consecutivas hacia el Este y dejando el cabezal donde comenzó. Invocalo en un `program`.
>
> En la Biblioteca vas a encontrar el procedimiento `VolverAtras`. ¡Eso significa que podés invocarlo sin tener que definirlo!

Solución | Biblioteca

```
1 procedure DibujarLineaNegra3() {
```

```
  repeat(2) {
    Poner(Negro)
    Mover(Este)
  }
  Poner(Negro)
  VolverAtras()
}
program {
  DibujarLineaNegra3()
}
```

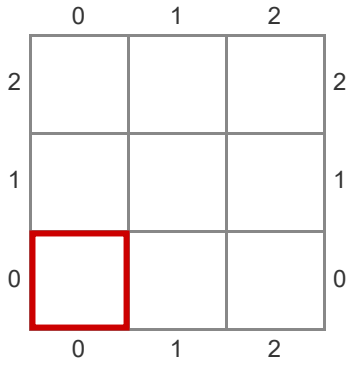
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

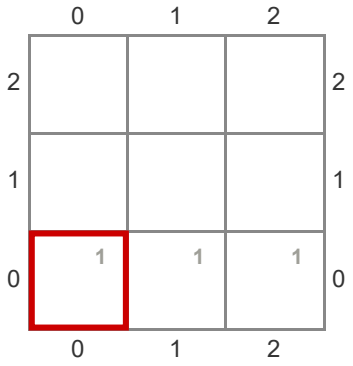
Resultados de las pruebas:

Tablero de 3 por 3

Tablero inicial

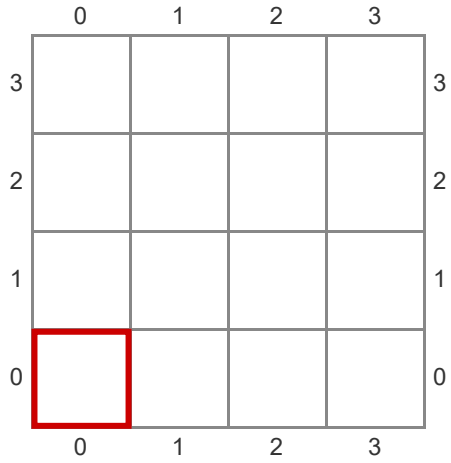


Tablero final

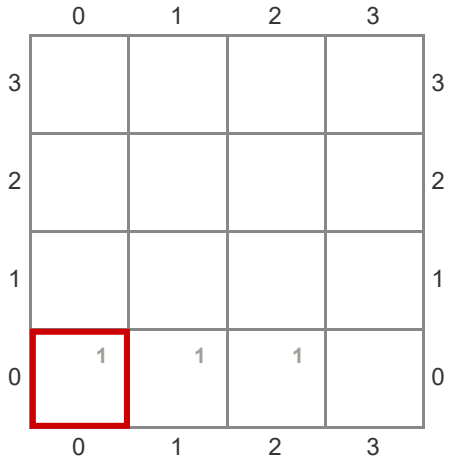


Tablero de 4 por 4

Tablero inicial



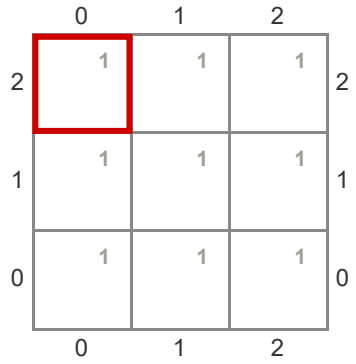
Tablero final



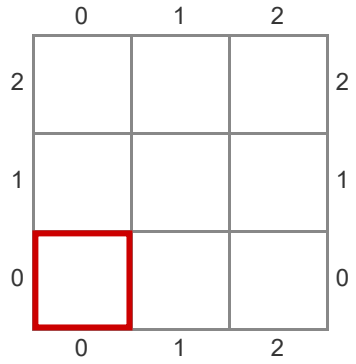
¡Perfecto! Ya tenemos nuestra línea, ya vamos a poder hacer un cuadrado negro.

Dibujando un cuadrado con subtareas

¡Ya podemos dibujar nuestro cuadrado! El cual debería verse así:



El cabezal comienza en el origen, es decir, en el casillero de abajo a la izquierda:



> Definí el procedimiento `DibujarCuadradoNegroDeLado3` que invocando `DibujarLineaNegra3` dibuje un cuadrado negro sobre el tablero. Invocalo en un `program`.

¡Dame una pista!

Solución | Biblioteca

```
1 procedure DibujarCuadradoNegroDeLado3() {
2   repeat(2) {
3     DibujarLineaNegra3()
4     Mover(Norte)
5   }
6   DibujarLineaNegra3()
7 }
```

```
8
9  program {
10   DibujarCuadradoNegroDeLado3()
11 }
```

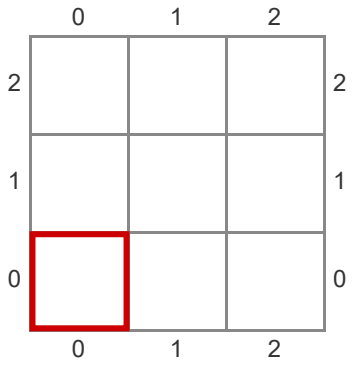
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

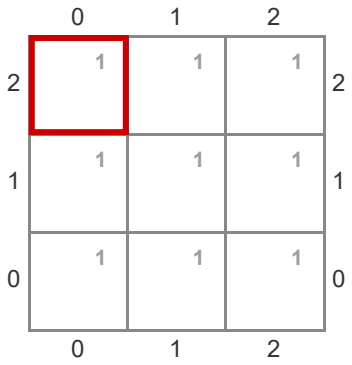
Resultados de las pruebas:

Tablero de 3 por 3

Tablero inicial

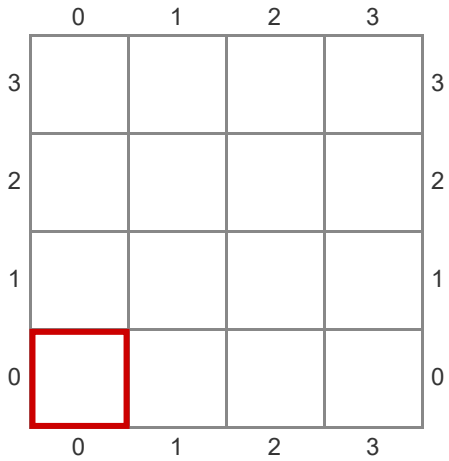


Tablero final

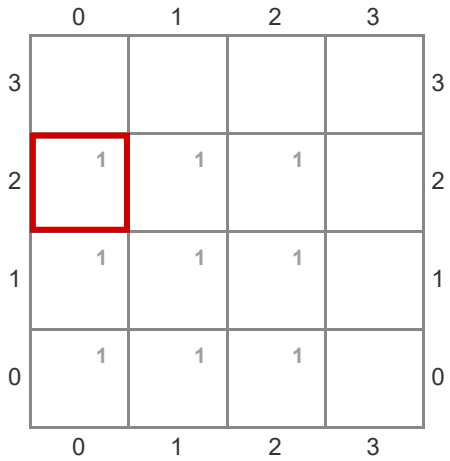


Tablero de 4 por 4

Tablero inicial



Tablero final



¡Muy bien! Ya tenemos un cuadrado negro, pero también queremos cuadrados de otros colores . Solucionemos esto haciendo nuevos procedimientos.

Color esperanza

Ya sabemos como dibujar una línea negra con 3 bolitas. Usamos esa lógica para dibujar una línea verde.

3. Color esperanza

> Definí el procedimiento `DibujarLineaVerde3` e invocalo en el `program`.

¡Dame una pista!

Solución | Biblioteca

```
1  procedure DibujarLineaVerde3() {
2    repeat(2) {
3      Poner(Verde)
4      Mover(Este)
5    }
6    Poner(Verde)
7    VolverAtras()
8  }
9  program {
10   DibujarLineaVerde3()
11 }
```

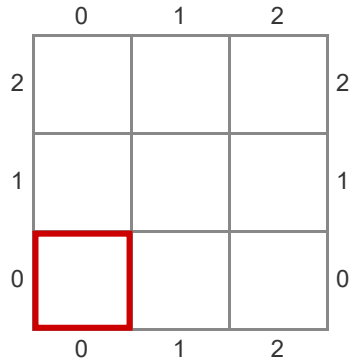
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

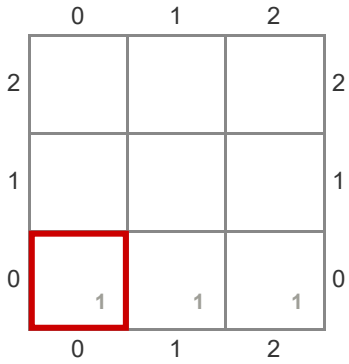
Resultados de las pruebas:

- Tablero de 3 por 3

Tablero inicial

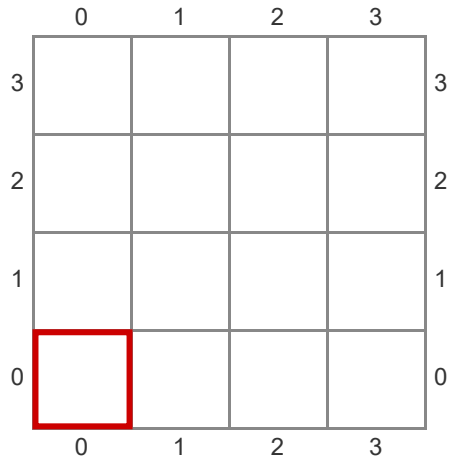


Tablero final

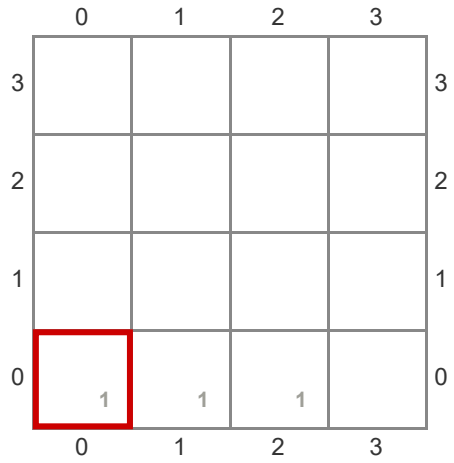


Tablero de 4 por 4

Tablero inicial



Tablero final



¡Excelente! Pero todavía nos faltan dos colores más . Ya te imaginarás lo que hay que hacer.

Los que faltan

Solo nos faltan dos colores: `Rojo` y `Azul` .

4. Los que faltan

Definí los procedimientos `DibujarLineaRoja3` y `DibujarLineaAzul3` .

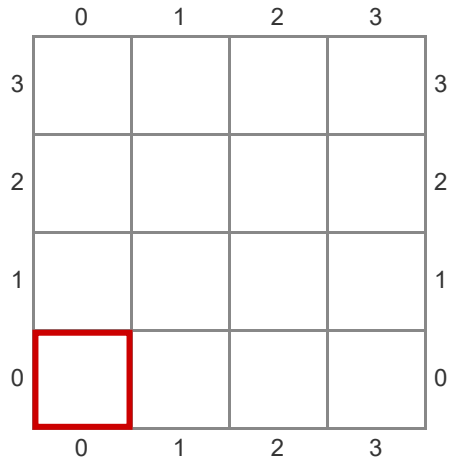
Solución | Biblioteca

```
1  procedure DibujarLineaRoja3() {
2    repeat(2) {
3      Poner(Rojo)
4      Mover(Este)
5    }
6    Poner(Rojo)
7    VolverAtras()
8  }
9
10 procedure DibujarLineaAzul3() {
11   repeat(2) {
12     Poner(Azul)
13     Mover(Este)
14   }
15   Poner(Azul)
16   VolverAtras()
17 }
18
```

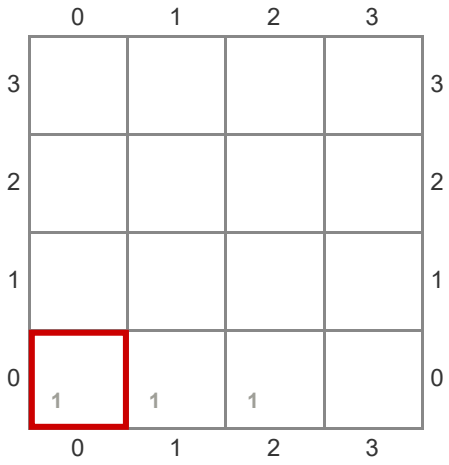
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



¡Muy bien! Peeero se volvió medio repetitiva la tarea de dibujar 4 procedimientos casi iguales ¿no? Sólo cambiaba el color, y si a esto le sumamos que además necesitamos un nuevo procedimiento por cada cuadrado, ¡la cosa se pone cada vez más aburrida! . ¿Se podrá solucionar de alguna manera?

Procedimientos con agujeritos

¡Empecemos con algo fácil! Supongamos que tenemos un procedimiento llamado `Poner3Verdes`, que pone 3 bolitas verdes en un casillero, y lo queremos **generalizar** para que funcione con cualquier color que queramos (pero uno solo por vez). Lo que necesitamos es agregarle al procedimiento una especie de *agujero*...

```
procedure Poner3(color) {
  repeat(3) {
    Poner(color)
  }
}
```

...que luego pueda ser completado con el color que queramos:

```
program {
  Poner3(Negro)
  Poner3(Rojo)
}
```

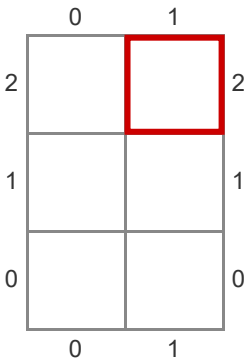
> Escribí los códigos anteriores en el editor y fijate qué pasa.

```
1  procedure Poner3(color) {
2    repeat(3) {
3      Poner(color)
4    }
5  }
6
7  program {
8    Poner3(Negro)
9    Poner3(Rojo)
10 }
```

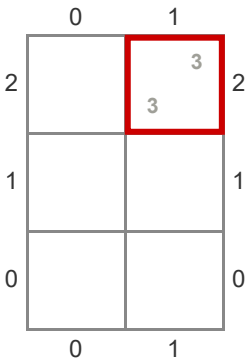
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



¿Viste qué interesante lo que hicimos?

Con un **mismo procedimiento** pudimos hacer 2 cosas casi iguales; por un lado pusimos 3 bolitas negras y por el otro 3 bolitas rojas. La diferencia estuvo en **cómo usamos** `Poner3` : la primera vez completamos el *agujero* con `Negro` y la segunda con `Rojo` .

Llenando los espacios vacíos

Entendamos qué acabamos de hacer.

Lo primero que hicimos fue definir un procedimiento, pero con una pequeña diferencia: toma un *parámetro*, llamado `color`.

```
procedure Poner3(color) {
  Poner(color)
  Poner(color)
  Poner(color)
}
```

¿Y qué es un parámetro? Son esos nombres que van entre paréntesis para ser reemplazados por valores concretos cuando invocamos al procedimiento. Por ejemplo, si lo invocamos así..

```
program {
  Poner3(Negro)
}
```

...lo que se ejecuta es:

```
Poner(Negro)
Poner(Negro)
Poner(Negro)
```

Y si lo invocamos así...

```
program {
  Poner3(Rojo)
}
```

lo que se ejecuta es:

```
Poner(Rojo)
Poner(Rojo)
```

```
Poner(Rojo)
```

Fijate como cada vez que aparece `color` se reemplaza por el valor que le *pasamos* a `Poner` .

Veamos si se entiende:

> Creá un programa que ponga tres bolitas verdes. No te olvides de invocar el procedimiento `Poner3` .

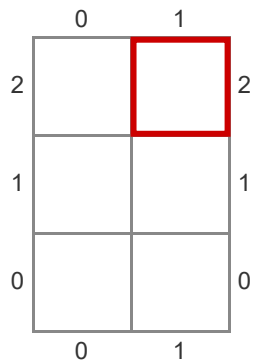
Solución | Biblioteca

```
program {
  Poner3(Verde)
}
```

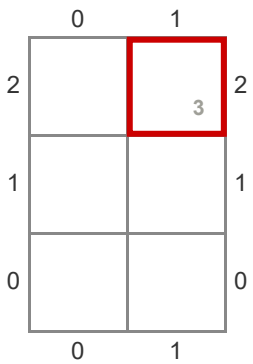
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



DibujarLinea3

¡Ahora te toca a vos!

Ya hicimos cuatro procedimientos para dibujar líneas de cada color, pero si usamos parámetros podría ser sólo uno.

> Definí el procedimiento `DibujarLinea3` que reciba un color y dibuje una línea de ese color. Despreocupate por los `program` s para invocarlo con cada uno de los colores, van por nuestra parte.

¡Dame una pista!

Solución | Biblioteca

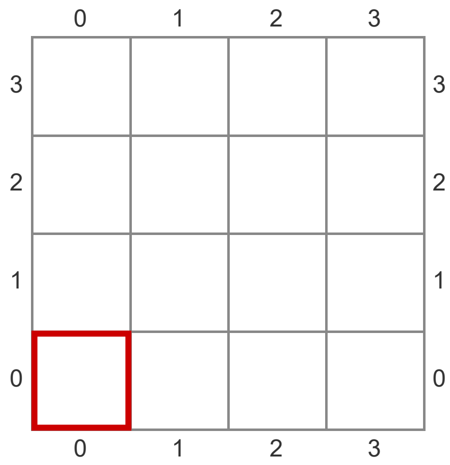
```
1 procedure DibujarLinea3(color) {
2   repeat(2) {
3     Poner(color)
4     Mover(Este)
5   }
6   Poner(color)
7   VolverAtras()
8 }
```

Enviar

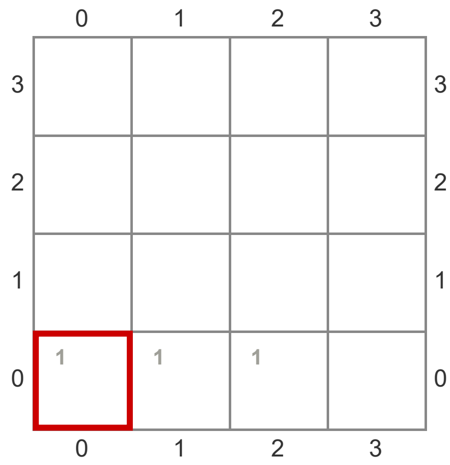
¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

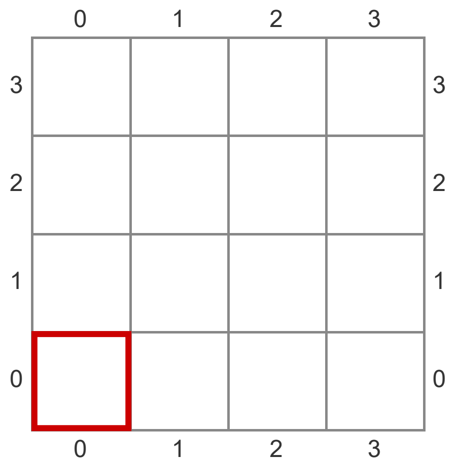
Tablero inicial



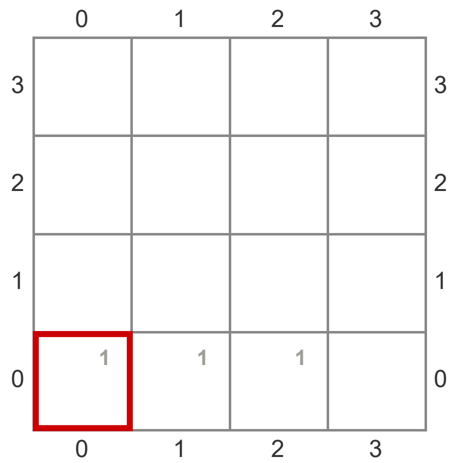
Tablero final



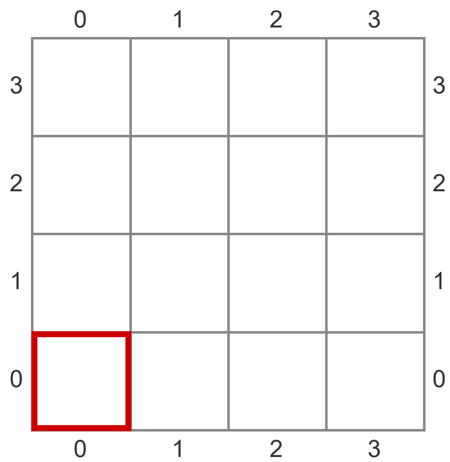
Tablero inicial



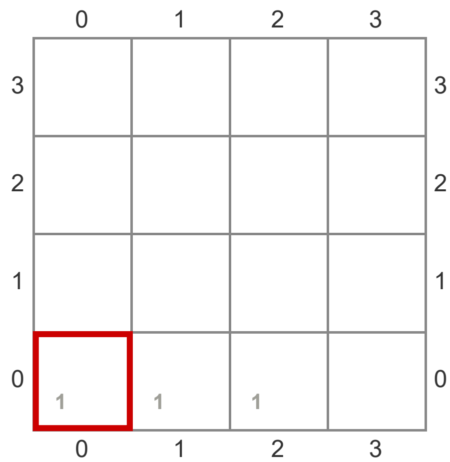
Tablero final



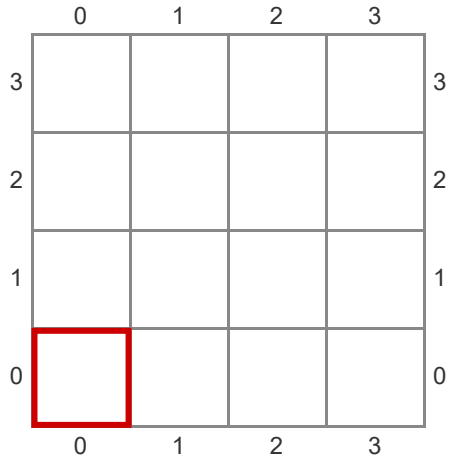
Tablero inicial



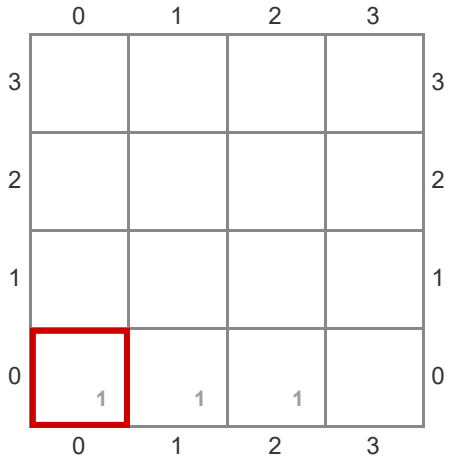
Tablero final



Tablero inicial



Tablero final



En vez de escribir `color` podríamos haber escrito `c`, `col` o incluso `pepe`, porque a la computadora no le importa el nombre que le demos a las cosas... pero a nosotros sí (y mucho), así que por eso elegimos **usar nombres que expresen bien lo que queremos decir**.

Fijate también que elegimos un nuevo nombre para nuestro nuevo procedimiento porque ahora sirve para cualquier color. Esto tampoco es estrictamente necesario para la computadora, pero es super importante para los humanos que van a leer y escribir el código. Imaginate que si le pusieramos `DibujarLineaNegra3` (o `DibujarLineaAzul3`) al usarlo para dibujar una línea roja quedaría `DibujarLineaNegra3(Rojo)` . Si bien va a funcionar, no se entendería bien qué es lo que hace: ¿una línea negra? ¿una línea roja? ¿una línea negra y roja?.

¡Es importantísimo poner buenos nombres para programar bien!

Reglas del Espacio de Consultas

DibujarCuadradoDeLado3

¡Hora del último pasito! Ya definimos un procedimiento para poder dibujar cuadrados negros (`DibujarCuadradoNegroDeLado3`), pero todo este asunto de los parámetros surgió cuando quisimos hacer cuadrados de distintos colores.

> Invocando `DibujarLinea3` , definí el procedimiento `DibujarCuadradoDeLado3` que recibe un `color` y dibuja un cuadrado de 3x3 de ese color.

¡Dame una pista!

Solución | Biblioteca

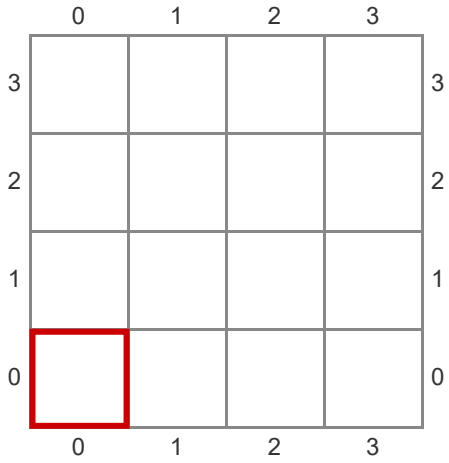
```
procedure DibujarCuadradoDeLado3(color) {
  repeat(2) {
    DibujarLinea3(color)
    Mover(Norte)
  }
  DibujarLinea3(color)
}
```

Enviar

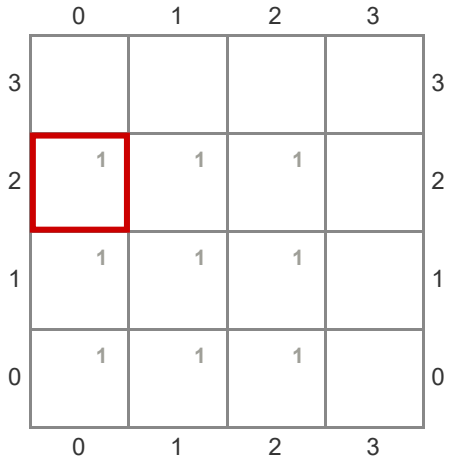
¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

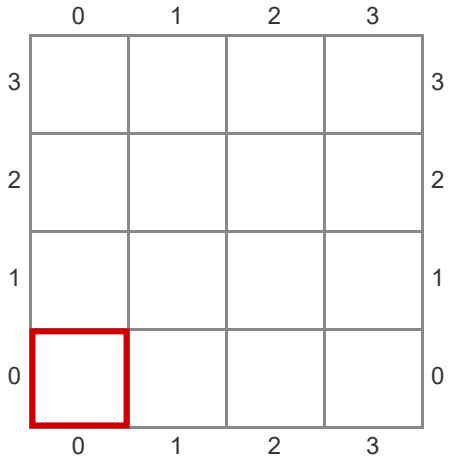
Tablero inicial



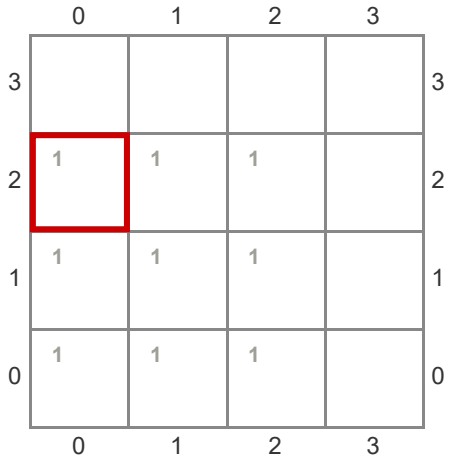
Tablero final



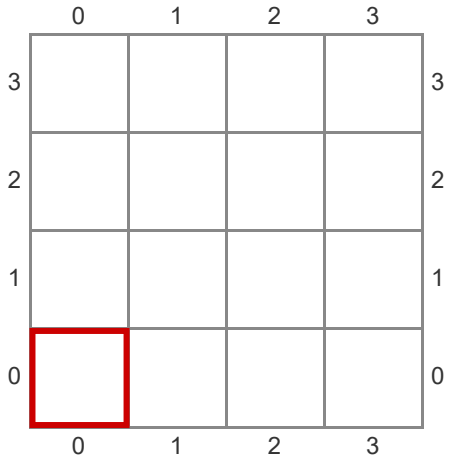
Tablero inicial



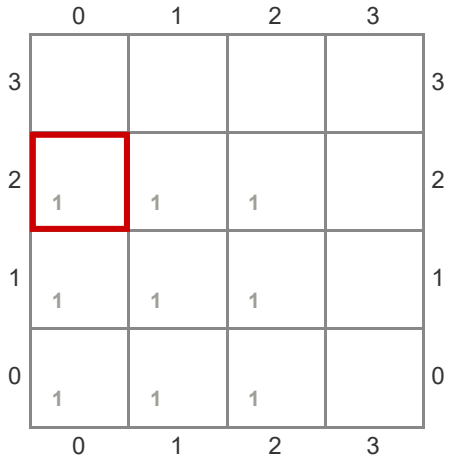
Tablero final



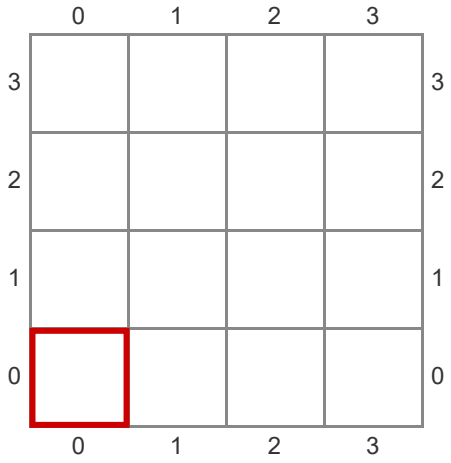
Tablero inicial



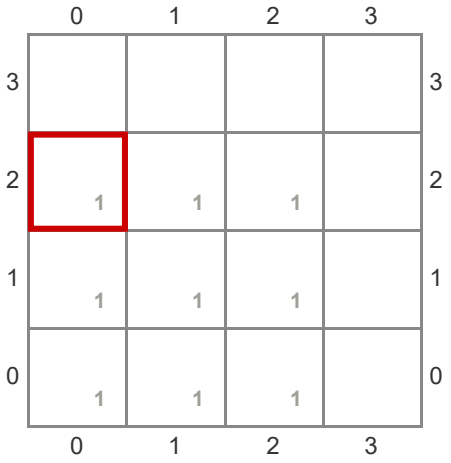
Tablero final



Tablero inicial



Tablero final



Genial, ¡logramos crear un procedimiento que nos sirve para cualquier color!

Un procedimiento puede no tener parámetros (como pasa en `VolverAtras` o `DibujarLineaNegra3`) o un parámetro (como `Mover`, `Poner` o `DibujarLinea3`), pero ¿puede tener más de uno?

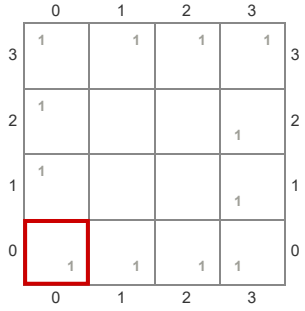
Pasando varios parámetros

¿Y si queremos que `DibujarLinea3` sirva también para dibujar líneas en cualquier dirección? Sin dudas tenemos que decirle al procedimiento además del `color`, en qué `direccion` debe dibujar la línea; y para eso vamos a necesitar un nuevo **parámetro**. Por suerte, ¡los procedimientos también pueden tener más de un **parámetro**!

¿Y cómo se hace esto? Muy fácil, al igual que como hacemos al escribir, vamos a **separar cada parámetro usando comas** de esta manera:

```
procedure DibujarLinea3(color, direccion) {
  Poner(color)
  Mover(direccion)
  Poner(color)
  Mover(direccion)
  Poner(color)
}
```

Creá un `program` que invoque la nueva versión de `DibujarLinea3` (no tenés que definirla, sólo invocarla) y dibuje un cuadrado multicolor como este:



No te preocupes por la posición final del cabezal.

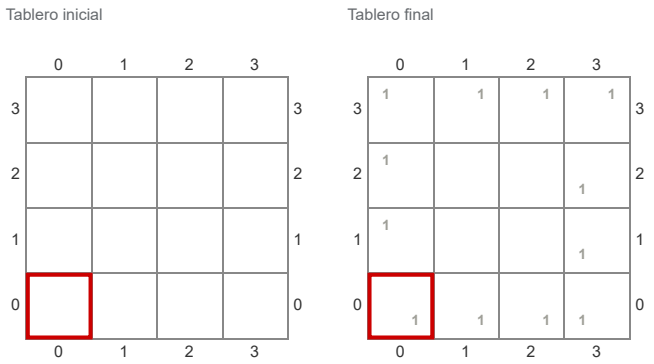
¡Dame una pista!

Solución | Biblioteca

```
1  program {
2    DibujarLinea3(Verde,Este)
3    Mover(Este)
4    DibujarLinea3(Rojo,Norte)
5    Mover(Norte)
6    DibujarLinea3(Negro,Oeste)
7    Mover(Oeste)
8    DibujarLinea3(Azul,Sur)
9    Mover(Sur)
10 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas



Como habrás notado, al usar los procedimientos debemos darle un valor a cada uno de sus parámetros **respetando el orden** en que fueron definidos.

A estos valores concretos que usamos cuando invocamos a un procedimiento los llamamos *argumentos*.

La ley, el orden y el BOOM

Recién te decíamos que el orden en que pasabamos los **argumentos** era importante pero nunca te dijimos por qué . ¡Vamos a verlo!

> Creá un programa cualquiera que invoque `DibujarLinea3` , pero esta vez intentá invocarlo con los argumentos invertidos.

¡Dame una pista!

Solución | Biblioteca

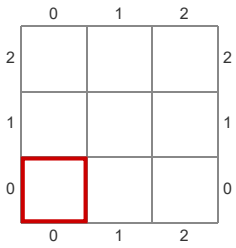
```
program  {
  DibujarLinea3(Este, Rojo)
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡BOOM!

Tablero inicial



Tablero final

BOOM

```
[5:3]: El parámetro de "Poner" debería ser un color pero es una dirección.
```

Ok, hizo BOOM , pero ¿qué quiere decir eso de `El parámetro de Poner debería ser un color` ?

Al pasar los argumentos al revés, donde se esperaba un color llegó una dirección; entonces cuando intentamos `Poner` una `dirección` provocamos la autodestrucción del cabezal . `Poner(Norte)` o `Poner(Este)` no se pueden ejecutar porque no tienen ningún sentido.

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas

Un argumento para dos parámetros

Ya vimos que pasa cuando pasamos los argumentos desordenados pero vamos a hacer un experimento más . ¿Qué crees que va a pasar si a un procedimiento le pasamos menos argumentos de los que necesita?

> Creá un programa que invoque a `DibujarLinea3` pero pasándole sólo un argumento.

Solución | Biblioteca

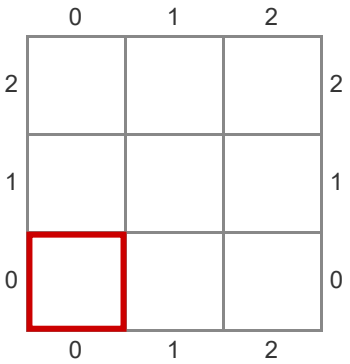
```
1 program  {
2   DibujarLinea3(Rojo)
3 }
4
5
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡BOOM!

Tablero inicial



Tablero final

BOOM

```
[2:3]: El procedimiento "DibujarLinea3" espera recibir 2 parámetros pero se lo inv
```

¡BOOM de nuevo ! Como te imaginarás, si le pasamos más argumentos de los que espera pasará lo mismo.

Entonces es importante recordar que al invocar un procedimiento **no debemos**:

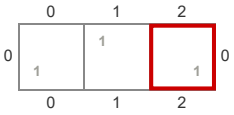
- pasarle menos o más argumentos de los que necesita;
- pasarle los argumentos en un orden diferente al que espera.

La tercera es la vencida

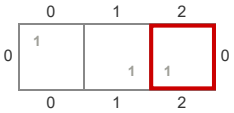
Para terminar esta lección vamos a definir un procedimiento llamado `Triada` ¡que recibe tres parámetros!

`Triada` recibe tres colores por parámetro y pone tres bolitas, una al lado de la otra hacia el Este, en el mismo orden en que se reciben. El cabezal empieza en el origen y debe terminar sobre la última bolita de la tríada.

Por ejemplo: `Triada(Rojo, Azul, Verde)` nos da como tablero resultante:



mientras que `Triada(Azul, Verde, Rojo)` :



> Definí el procedimiento `Triada` .

¡Dame una pista!

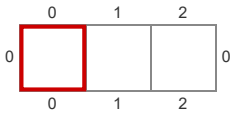
```
procedure Triada(color1, color2, color3) {
  Poner(color1)
  Mover(Este)
  Poner(color2)
  Mover(Este)
  Poner(color3)
}
```

Enviar

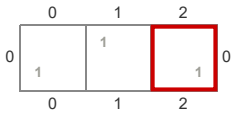
¡Muy bien! Tu solución pasó todas las pruebas

Resultados de las pruebas:

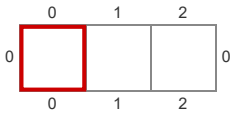
Tablero inicial



Tablero final



Tablero inicial



Tablero final