¡Hola, computadora!

En nuestra vida cotidiana sabemos cómo hacer para comunicarnos con otras personas. Si necesitamos pedirles que hagan algo, tenemos que decirles por favor . Pero... ¿cómo hacemos si le tenemos que pedir algo a una computadora?

Para que la computadora nos entienda, vamos a tener que aprender SU lenguaje . En realidad, las computadoras entienden más de un lenguaje, muchos de ellos los vamos a aprender a lo largo de este curso.

En este capítulo vamos a conocer Gobstones, un lenguaje educativo diseñado por docentes argentinos .

Gobstone usa un tablero y bolitas de colores. El tablero es muy similar a un tablero de ajedrez: tiene diferentes celdas por las que podemos movernos. Para hacerlo, vamos a utilizar un cabezal que en todo momento está situado sobre una de las celdas del tablero y puede realizar distintas operaciones, como, por ejemplo, poner bolitas.

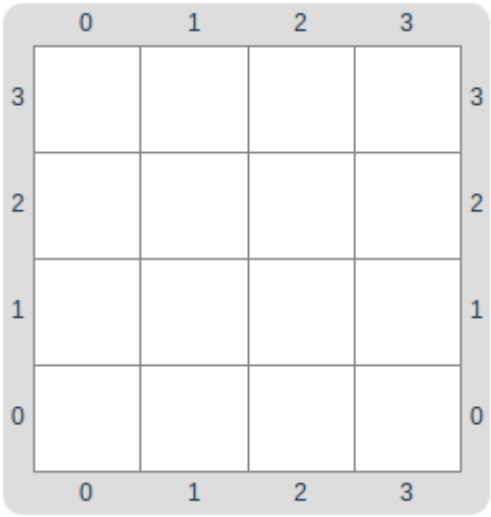
Ahora es tu turno. ¡Empecemos a programar!

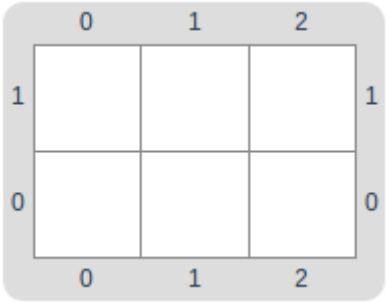
El Tablero

Para empezar a programar, el primer elemento que vamos a usar es un **tablero** cuadriculado, similar al del Ajedrez, Damas o Go.

Estos tableros pueden ser de cualquier tamaño, por ejemplo,

4x4	3x2





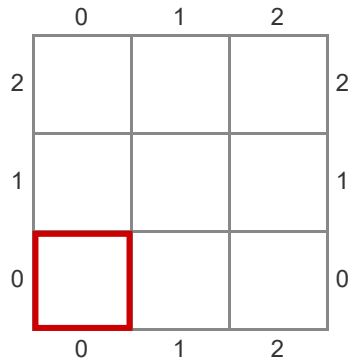
Siempre vamos a necesitar un tablero sobre el cual **ejecutar** nuestros **programas**, ¡pero despreocupate! nos vamos a encargar de crearlos por vos en cada uno de los ejercicios. Lo interesante es que un mismo programa puede ejecutarse sobre distintos tableros, potencialmente produciendo resultados diferentes.

> Para que veas lo que te decimos, presioná el botón Continuar, y vamos a generar tu primer tablero: un tablero de 3x3.

Continuar

¡Muy bien!

Tablero final

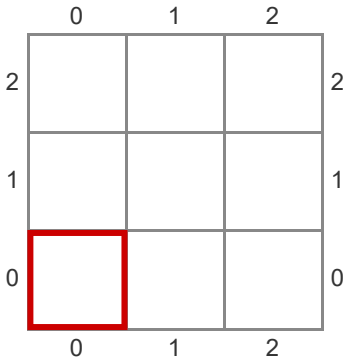


Perfecto, ¡este es tu primer **tablero** de 3x3!

Notá que ahora la **celda** (casillero) de la esquina inferior izquierda está pintada de otra forma. En el próximo ejercicio veremos por qué.

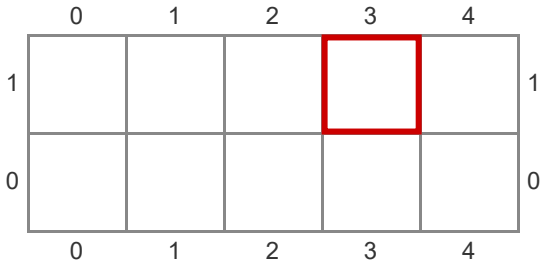
El Cabezal

El tablero generado en el ejercicio anterior tenía una **celda** marcada:



¿Y eso por qué? Porque nuestra máquina tiene un **cabezal**, que en todo momento está situado sobre una de las celdas del tablero y puede realizar distintas operaciones sobre ella (paciencia, ya las vamos a conocer).

Por ejemplo, el siguiente es un tablero de 5x2, con el cabezal en la segunda fila y la cuarta columna.



¡Algo importante! Contamos las filas hacia arriba, y las columnas hacia la derecha. La primera **fila** es la de **abajo** de todo, y la primera **columna** es la de la **izquierda**.

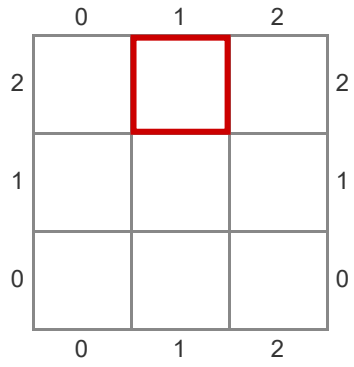
Vamos a ver otro ejemplo?

> Presioná Continuar y generaremos un tablero 3x3 con el cabezal en la segunda columna y tercera fila.

Continuar

¡Muy bien!

Tablero final



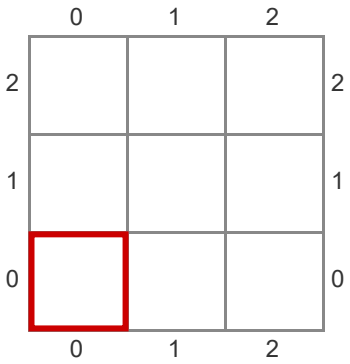
Como podrás ver podemos crear muchísimos tableros distintos, pero ¿el cabezal se va a quedar siempre en el mismo casillero?

Que comience el movimiento

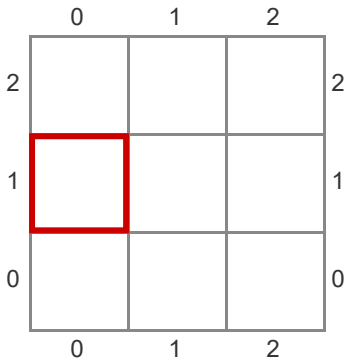
Hasta ahora lo que vimos no fue muy emocionante, porque no te enseñamos cómo darle instrucciones a la máquina y sólo te mostramos un tablero. En este ejercicio vamos a aprender una de las órdenes que podemos darle a la máquina: mover el cabezal.

Por ejemplo, partiendo de un tablero **inicial** vacío con el cabezal en el origen (abajo a la izquierda), podemos fácilmente crear un programa que mueva el cabezal una posición hacia el **norte**:

Inicial



Final



El **código** del programa (es decir, el **texto** de la descripción de la solución que le daremos a la computadora) que logra esto es el siguiente:

```
program {
  Mover(Norte)
}
```

> ¿No nos creés? Escribí el código anterior en el editor y dale Enviar.

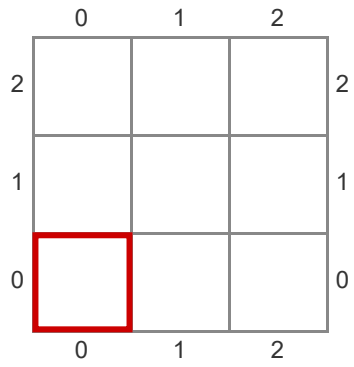
¡Dame una pista!

```
1 program {
2   Mover(Norte)
3 }
```

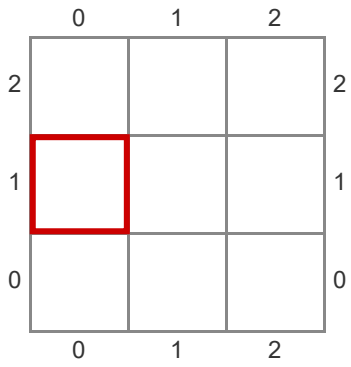
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



¡Felicitaciones, creaste tu primer programa! Éste, al ser ejecutado por la máquina, provoca que el cabezal se mueva una posición hacia el Norte.

Pero programar no se trata de copiar y pegar código. Acompañanos al próximo ejercicio para entender qué es lo que hicimos exactamente.

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas

Que siga el movimiento

Entendamos qué es lo que acabamos de hacer: ¡crear un programa!

Todo programa tiene exactamente un `program`: una sección del código que declara los comandos (acciones) que queremos que la máquina realice sobre el tablero **inicial**. Al **ejecutar** un programa obtendremos un tablero **final**.

La sintaxis de un `program` es bastante simple:

1. escribimos una línea (renglón) que diga `program`, seguido de una llave de apertura: `{`
2. a continuación, los comandos: uno por línea
3. y finalmente, una última llave que cierra la que abrimos anteriormente `}`

Vamos a ver algunos ejemplos de `program`s:

- uno que no hace nada

```
program {
}
```

- uno que mueve el cabezal **una** posición hacia el norte

```
program {
  Mover(Norte)
}
```

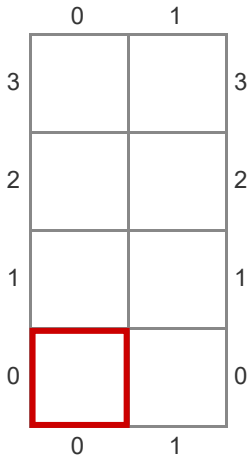
- uno que mueve el cabezal **dos** posiciones hacia el norte

```
program {
  Mover(Norte)
  Mover(Norte)
}
```

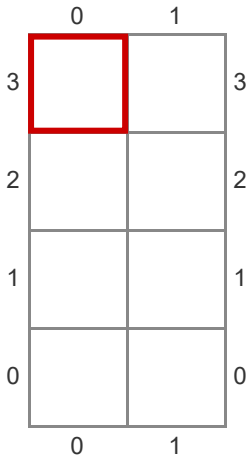
¡Te toca a vos!

Creá un programa que en un tablero de 2x4 con el cabezal en el origen (la celda de abajo a la izquierda), mueva el cabezal tres veces hacia el norte:

Inicial



Final



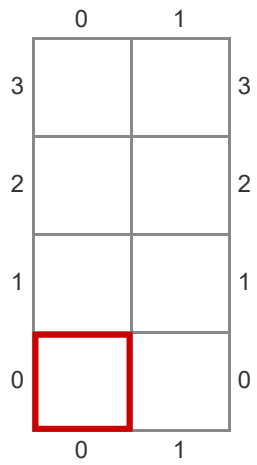
¡Dame una pista!

```
program {
  Mover(Norte)
  Mover(Norte)
  Mover(Norte)
}
```

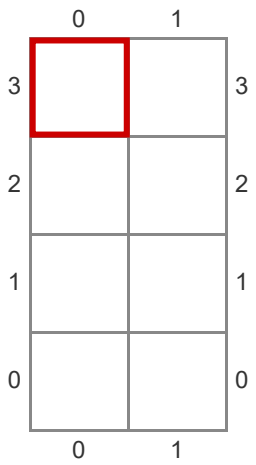
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



Los lenguajes de programación son creados con algunas palabras que solo se pueden utilizar con un fin determinado. Se las llama **palabras reservadas** . En *Gobstones*, el lenguaje que estamos utilizando, `program` es una palabra reservada.

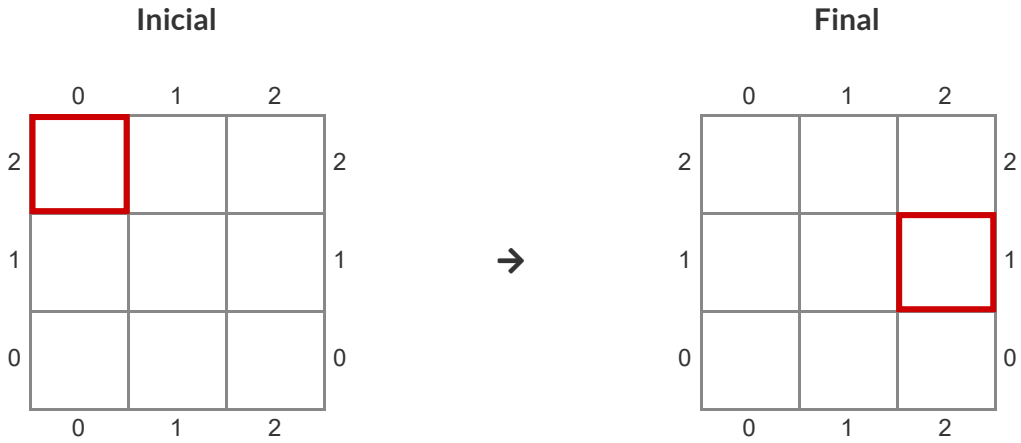
Como ya sabemos que nuestros programas son ejecutados por la máquina, de ahora en más diremos *"creá un programa que haga ..."* en vez de *"creá un programa que* ***provoque que la máquina*** *haga ..."*.

Pero ojo: la máquina sigue estando ahí y no hay que olvidarla, sólo hacemos esto para escribir un poco menos.

Para todos lados

Como te imaginarás, el cabezal no sólo se puede mover hacia el Norte, y un programa puede combinar cualquier tipo de movimientos.

Creá un programa que mueva el cabezal dos posiciones hacia el `Este` y una hacia el `Sur`, produciendo el siguiente efecto:



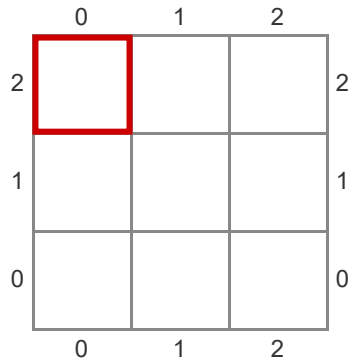
¡Dame una pista!

```
program {
  Mover(Este)
  Mover(Este)
  Mover(Sur)
}
```

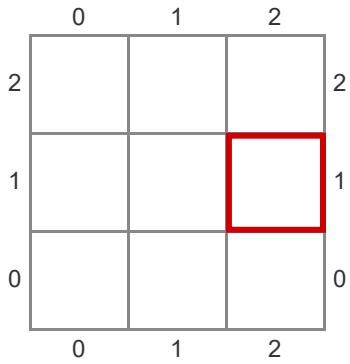
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



Notá que estos dos programas hacen lo mismo:

```
program {
  Mover(Este)
  Mover(Este)
  Mover(Sur)
}
```

```
program {
  Mover(Este)
  Mover(Sur)
  Mover(Este)
}
```

Moraleja: como te decíamos al principio ¡no hay una sóla forma de resolver un problema!

Y además, el orden, **a veces**, no es tan importante. Acompañanos a entender mejor esto.

Reglas del Espacio de Consultas

El orden de las cosas

Cuando trabajamos en Gobstones, hacemos las cosas en un cierto orden. Por ejemplo, si tenemos este programa:

```
program {
  Mover(Norte)
  Mover(Este)
}
```

una forma posible de leerlo (llamada **operacional**) es como lo haría una máquina, en orden, de arriba hacia abajo:

1. primero se mueve al norte: `Mover(Norte)`
2. luego se mueve al este: `Mover(Este)`

Y de hecho **se ejecuta de esa forma**. Esto es *cómo* lo hace.

Pero, los humanos, solemos pensar en función del resultado final, es decir, resaltamos el **objetivo** del programa. Nos importa más *qué* hace, y no cómo. Esta manera denotacional nos llevaría a decir que, simplemente, **mueve el cabezal al noreste**.

Por eso hay varias formas de resolver un mismo problema: podemos crear varios programas que hagan lo mismo (el *qué*), pero que lo hagan de forma diferente (el *cómo*).

> Veamos si entendiste esto: creá otro programa que haga lo mismo que el de arriba (mover hacia el noreste), pero de manera distinta. **Ojo:** tiene que funcionar en un tablero de 2x2.

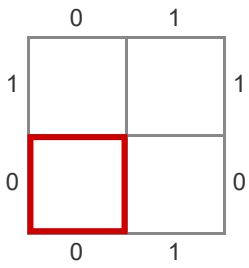
¡Dame una pista!

```
1 program {
2   Mover(Este)
3   Mover(Norte)
4 }
```

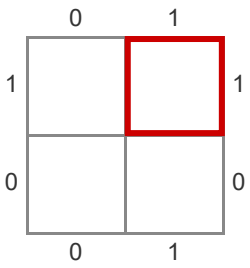
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



¡Perfecto!

Recién creamos un programa en el que el orden no afecta lo que queremos hacer. Pero esto no será siempre así, en breve nos toparemos con problemas en los que hay que tener más cuidado con el orden de los comandos.

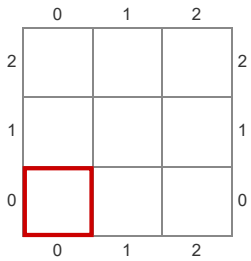
Sí, esto también se puede romper

Hasta ahora veníamos tratando de esquivar el asunto, pero seguro ya habrás pensado que tiene que haber alguna forma de romper esto (incluso es posible que ya te haya pasado).

Si bien **nunca** vamos a querer que nuestro programa se rompa, es algo que definitivamente **te va a pasar muchas veces**. Pero no es motivo para frustrarse ni mucho menos, te aseguramos que a todo el mundo le pasó alguna vez (bueno, también 2, 3, 100, 800 veces...).

Dicho esto, te vamos a mostrar una forma de hacerlo, simplemente para que no te asustes *tanto* cuando te pase de verdad .

> ¿Y cómo es esa forma? Descubrilo vos: partiendo del tablero que te mostramos acá abajo, creá un programa que provoque que el cabezal se salga fuera de los límites.



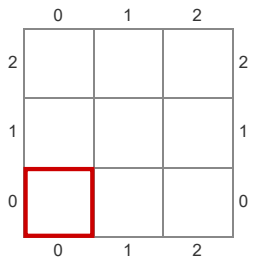

```
program {
  Mover(Sur)
 }
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡BOOM!

Tablero inicial



Tablero final

BOOM

```
[2:3]: No se puede mover hacia la dirección Sur: cae afuera del tablero.
```

¡BOOOOOOOOOOOOOOOOM!

Ey, ¿qué pasó?

Tu programa falló, se rompió, o como lo llamamos en el universo Gobstones: **hizo BOOM**.

Y, ¿qué significa esto?

Que los comandos que le diste a la computadora no se pueden ejecutar, y hay algo que vas a tener que cambiar para que funcione. En este ejercicio eso no tiene sentido porque lo hicimos a propósito, pero tenelo en cuenta para cuando falles en el futuro.

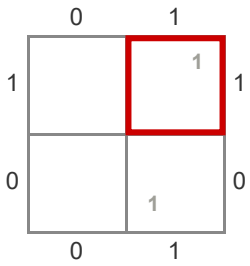
Nuestras primeras bolitas

Genial, ya entendiste cómo mover el cabezal del tablero usando la operación `Mover` y las direcciones (`Sur` , `Oeste` , etc). Vayamos un paso más allá: las **bolitas**.

En cualquier celda de nuestro tablero podemos poner `bolitas` . Las hay de distintos colores:

- rojas (`Rojo`);
- azules (`Azul`);
- negras (`Negro`);
- y verdes (`Verde`).

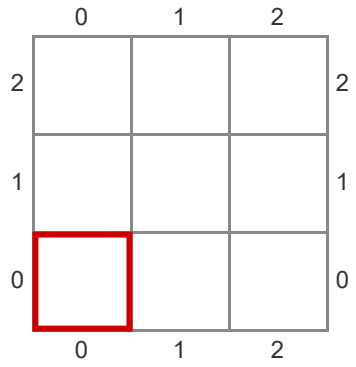
Por ejemplo, este es un tablero con una bolita roja y una negra:



Además de moverse, el cabezal también puede poner bolitas en la **celda actual**. Para eso contamos con la operación `Poner` , que le dice al cabezal que deposite una bolita del color dado:

```
program {
  Poner(Rojo)
}
```

> ¡Probá este programa! Escribí el código en el editor, envialo y verás lo que pasa al ejecutarlo sobre este tablero:

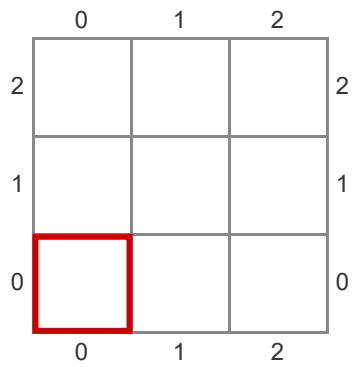



```
program {
  Poner(Rojo)
}
```

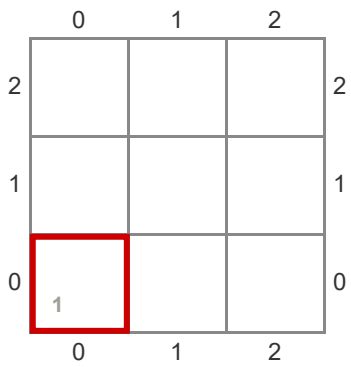
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



¡Felicitaciones! Acabás de escribir un programa que pone una bolita roja en la celda actual.

Esta guía fue desarrollada por Franco Bulgarelli, Federico Aloi, Gustavo Trucco, Daniela Villani bajo los

términos de la Licencia Creative Commons Compartir-Igual, 4.0.

Información importante

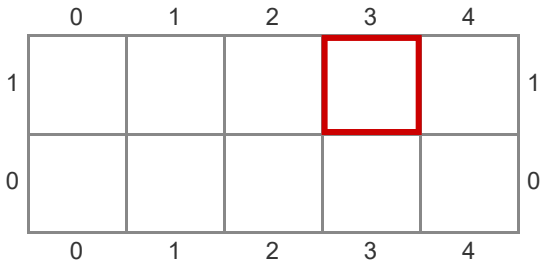
Términos y Condiciones

Reglas del Espacio de Consultas

Más y más bolitas

Algo interesante de nuestros tableros es que en sus celdas podemos poner cualquier cantidad de bolitas de cualquier color.

Por ejemplo, si tenemos este tablero:



y ejecutamos el siguiente programa:

```
program {
  Poner(Rojo)
  Poner(Rojo)
  Poner(Azul)
  Poner(Verde)
  Poner(Rojo)
}
```

el cabezal colocará en la celda actual tres bolitas rojas, una azul y una verde.

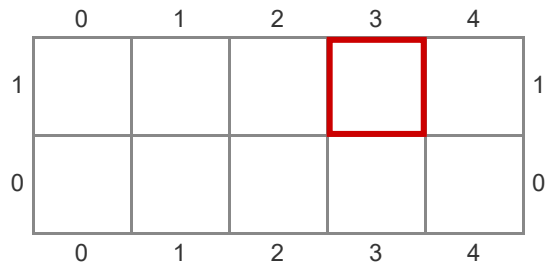
> ¡Escribí este programa en el editor y fijate cómo queda el tablero!

```
1 program {
2   Poner(Rojo)
3   Poner(Rojo)
4   Poner(Azul)
5   Poner(Verde)
6   Poner(Rojo)
7 }
```

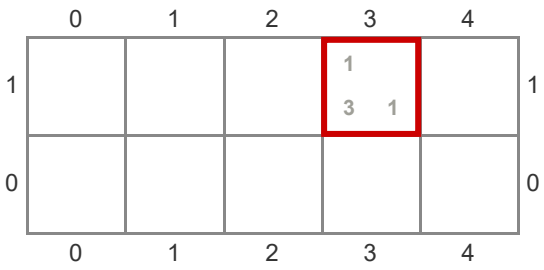
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



Notá que en este problema, si cambiamos el orden en que *llamamos* (usamos a) `Poner` , el resultado no cambia: siempre nos terminará quedando un tablero con tres bolitas rojas, una azul y una verde.

Por ejemplo, los siguientes dos programas también resuelven este mismo problema:

```
program {
  Poner(Rojo)
  Poner(Rojo)
  Poner(Rojo)
  Poner(Verde)
  Poner(Azul)
}
```

```
program {
  Poner(Rojo)
  Poner(Azul)
  Poner(Rojo)
  Poner(Verde)
  Poner(Rojo)
}
```

Esta guía fue desarrollada por Franco Bulgarelli, Federico Aloi, Gustavo Trucco, Daniela Villani bajo los términos de la Licencia Creative Commons Compartir-Igual, 4.0.

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas

Poné tus primeras bolitas

Como te habrás dado cuenta, estos tableros son un poco mágicos, podemos poner en una celda tantas bolitas como queramos: 2, 4, 12, 50, 1000. ¡No hay ningún límite!

Esto es algo muy interesante que ocurre al programar: podemos trabajar con cantidades tan grandes como queramos.

Ah, y ahora te toca a vos: creá un programa que ponga cuatro bolitas rojas y tres bolitas negras en la celda actual.

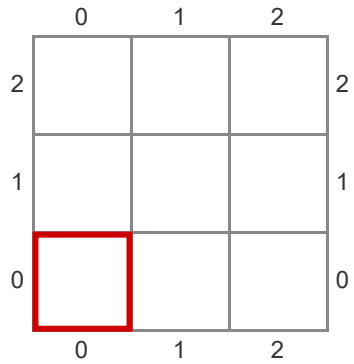
¡Dame una pista!

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

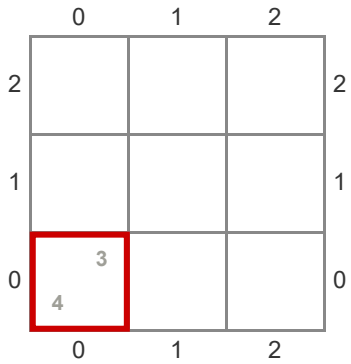
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



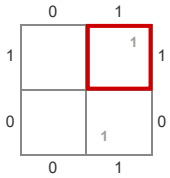
Sacar Bolitas

De la misma forma que hay un "poner bolita" (`Poner`), tenemos un "sacar bolita" (`Sacar`), que quita exactamente una bolita del color dado.

Por ejemplo, el siguiente programa saca dos bolitas de la posición inicial.

```
program {
  Sacar(Rojo)
  Sacar(Rojo)
}
```

Sabiendo esto, creá un programa que elimine **sólo** la bolita roja de este tablero. ¡Tené cuidado! Prestá atención a la posición del cabezal .



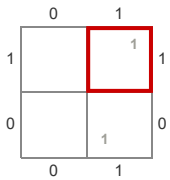
¡Dame una pista!

```
program {
  Mover(Sur)
  Sacar(Rojo)
}
```

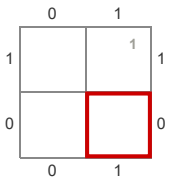
Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final



¿Y si no hubiera ninguna bolita para sacar?

Información importante

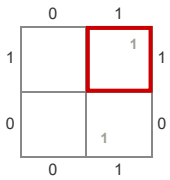
Términos y Condiciones

Reglas del Espacio de Consultas

Cuando no hay bolitas

Cada vez que usamos `Sacar` , tenemos que tener más cuidado que con `Poner` , porque...

> ¿Querés saber por qué? Intentá sacar una bolita verde o azul de este tablero y descubrilo.



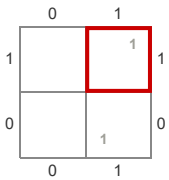

```
program {
  Sacar(Verde)
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡BOOM!

Tablero inicial



Tablero final

BOOM

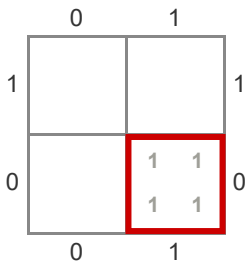
```
[2:3]: No se puede sacar una bolita de color Verde: no hay bolitas de ese color.
```

Claro, otra vez **BOOM**.

Esta vez lo que pasó fue que el cabezal intentó sacar una bolita de un color que no había, y como no sabía qué hacer se autodestruyó. Esto te va a pasar siempre que intentes sacar una bolita que no exista, así que ¡a prestar atención!

Limpiar celda

Un último esfuercito: usando `Sacar`, creá un programa que elimine todas las bolitas de este tablero:

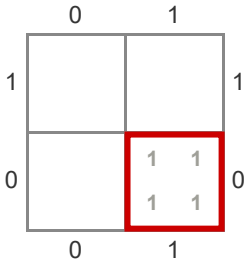



```
program {
  Sacar(Rojo)
  Sacar(Azul)
  Sacar(Negro)
  Sacar(Verde)
}
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

Tablero inicial



Tablero final