



# Introducción a la Programación

Clases teóricas

por Pablo E. “Fidel” Martínez López

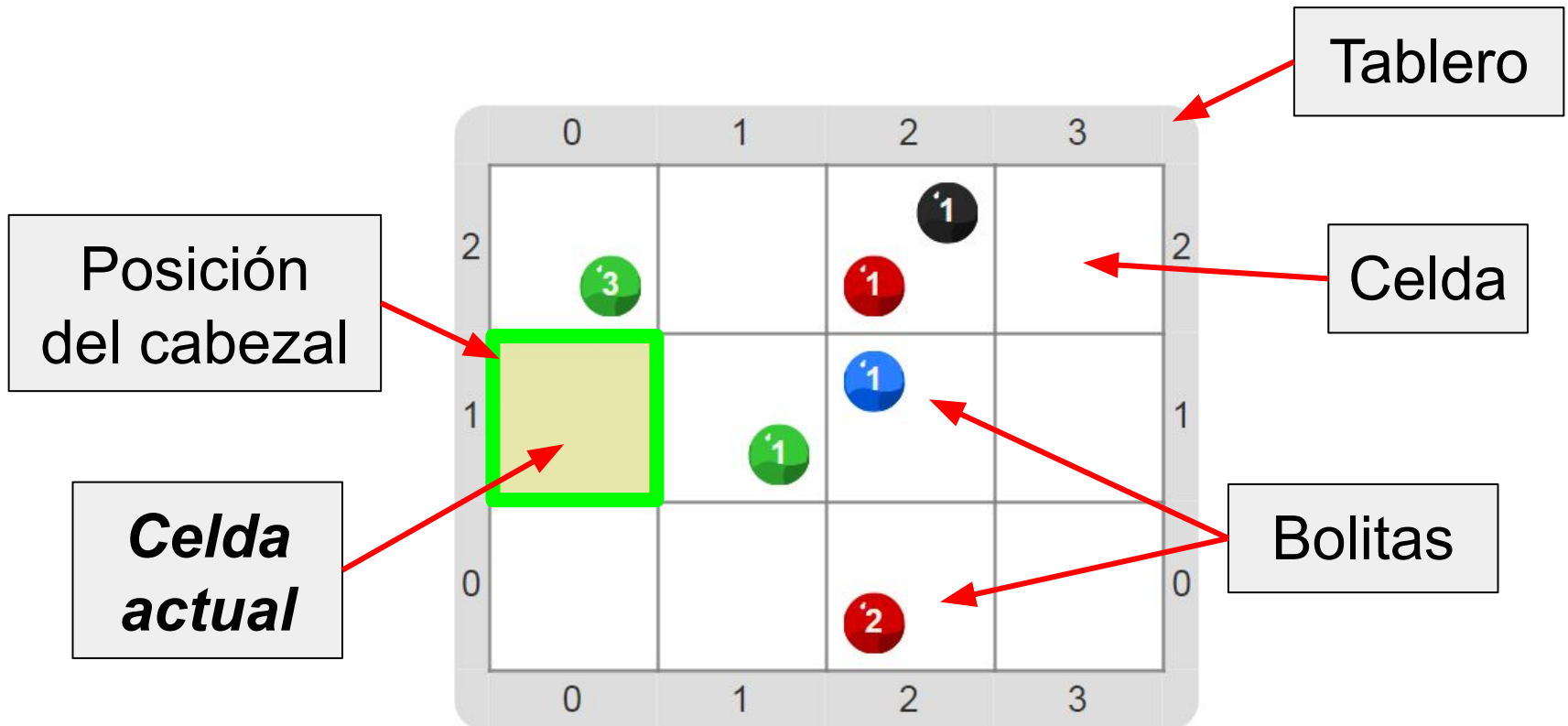
## 1. Presentación de Gobstones. Procedimientos



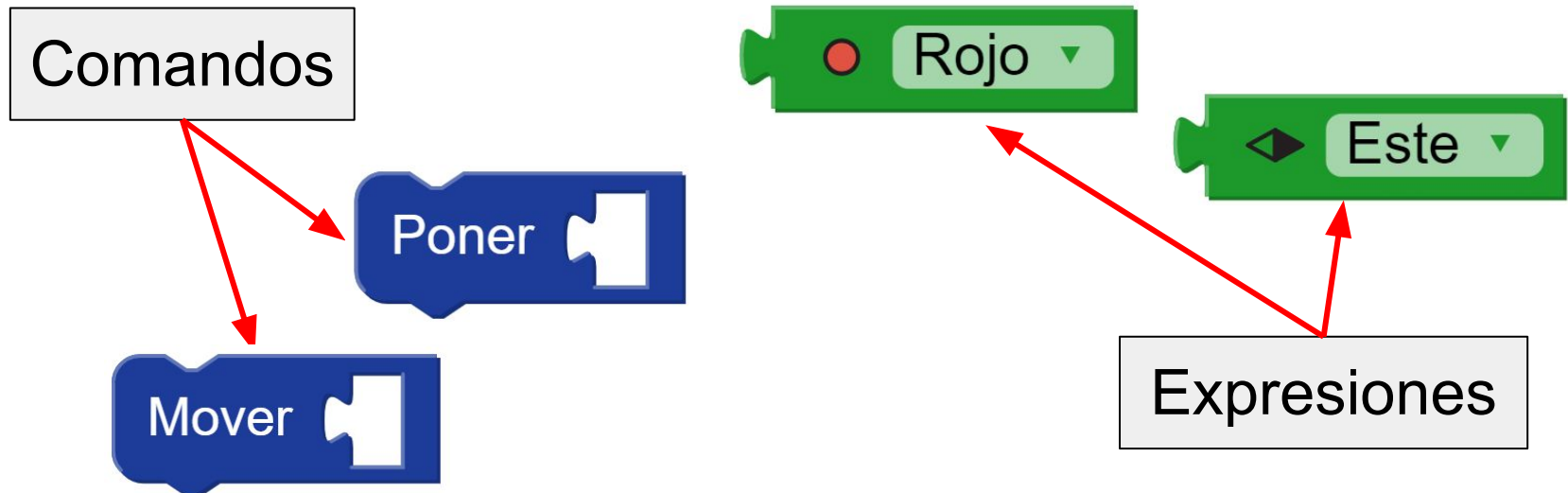
# Introducción a la programación



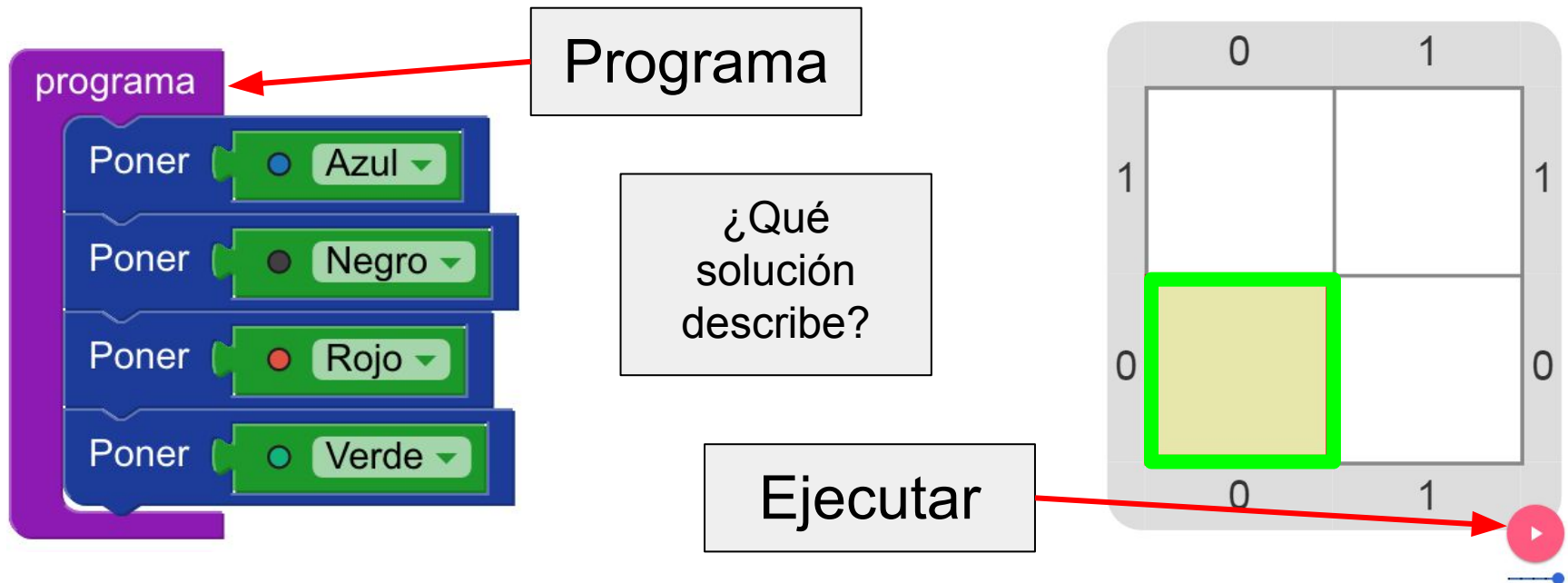
- Máquina (**cabezal**)
- Manipula **bolitas** sobre un **tablero**
- Solo puede acceder a una **celda** por vez



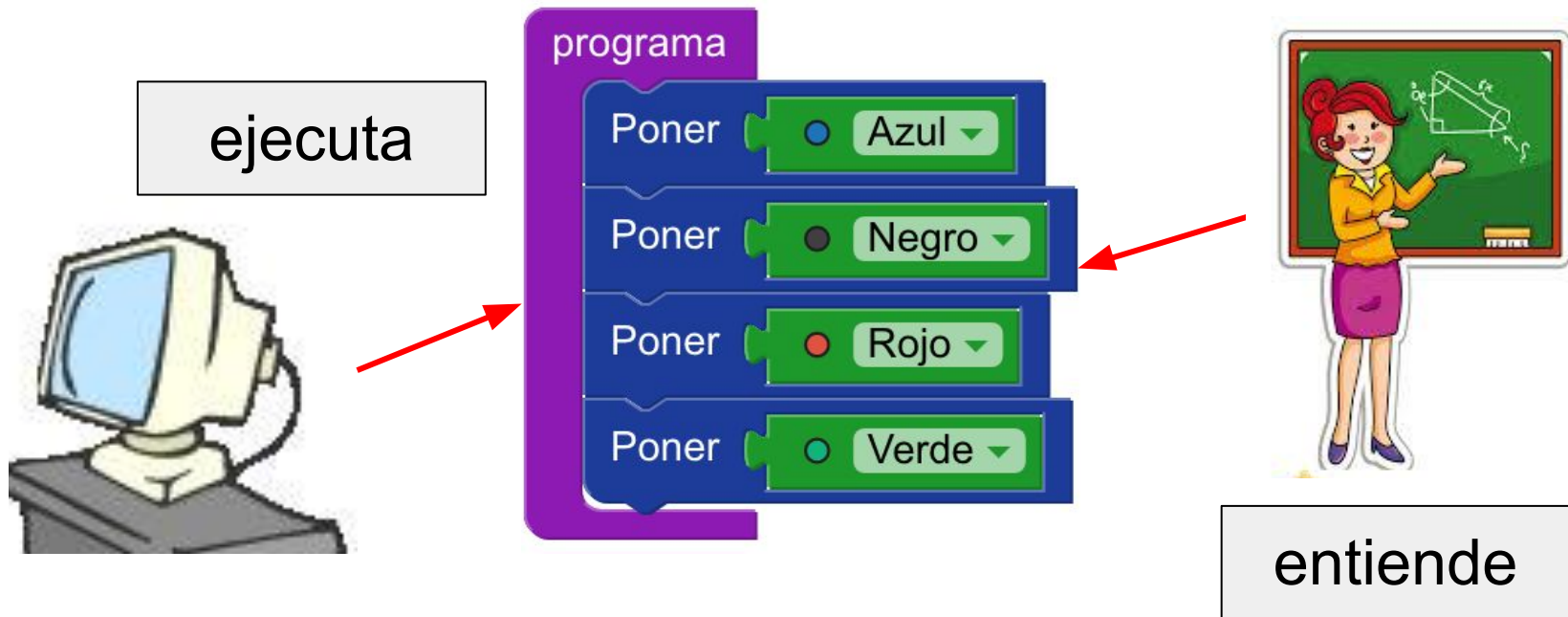
- El *cabezal*
  - realiza **acciones** al recibir instrucciones
  - brinda **información** al responder preguntas
- Se utilizan textos (o bloques) para manejar al cabezal
  - Los **comandos** son descripciones de *acciones*
  - Las **expresiones** son descripciones de *información*



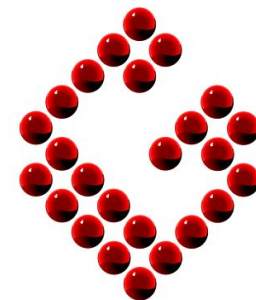
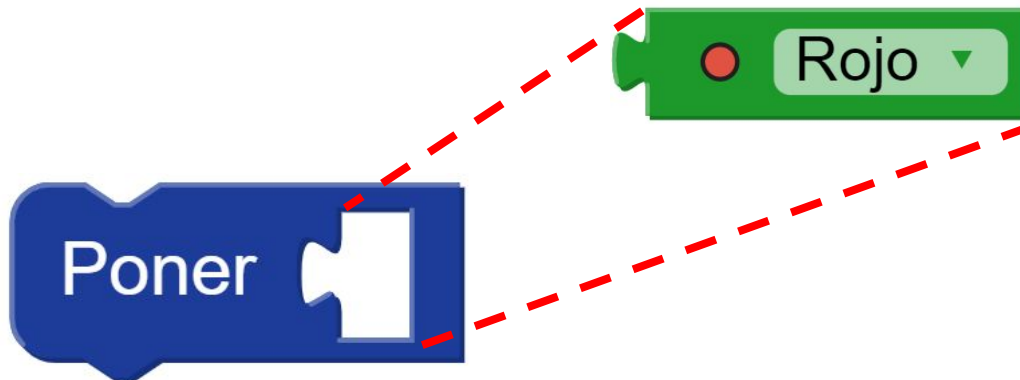
- Un **programa**
  - es un **texto** que combina comandos y expresiones
  - **describe** la solución a un problema
  - puede ser **ejecutado** por la máquina para obtener la solución propuesta



- **Programar** es **describir** la solución de un problema de forma *metódica* de manera que
  - una máquina pueda **ejecutar** la solución
  - las personas puedan **entender** la solución

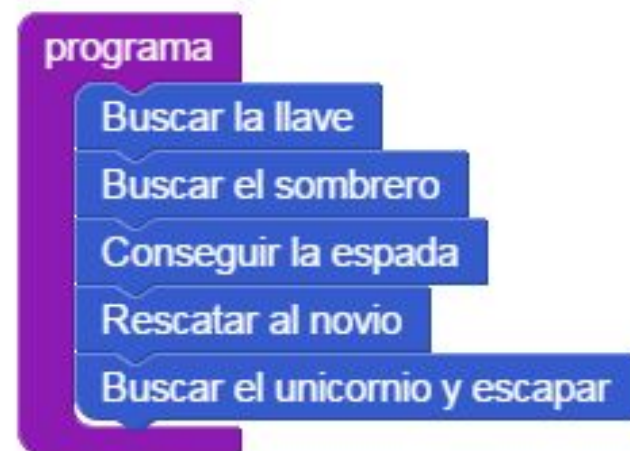


- Necesitamos **reglas** para construir programas tal que
  - la máquina pueda leerlos y ejecutarlos
  - las personas puedan leerlos y entenderlos
- **Lenguaje de programación**
  - Establece esas reglas para una máquina dada



# Gobstones

- Se puede programar bien o mal en cualquier lenguaje
  - **Solucionar** el problema **NO ALCANZA**
  - Además el programa debe **ENTENDERSE**
- Decir “*el programa funciona*” no es excusa para no hacerlo entendible



Ambos programas describen la misma solución, pero uno se entiende más que el otro.



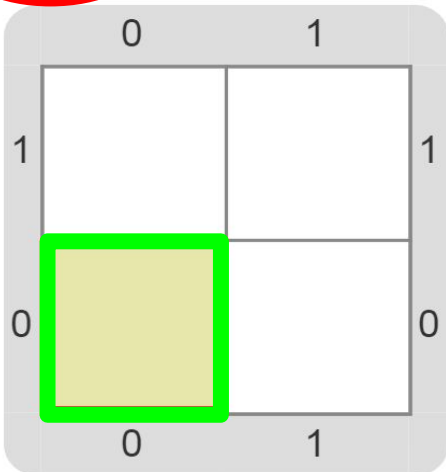


- ¿Qué tipo de problemas podemos resolver?
  - transformaciones de **estado**
  - transformaciones de **información**
- En Gobstones, se transforma un **tablero inicial** en un **tablero final**

Tablero inicial

Tablero inicial 1

Tablero final



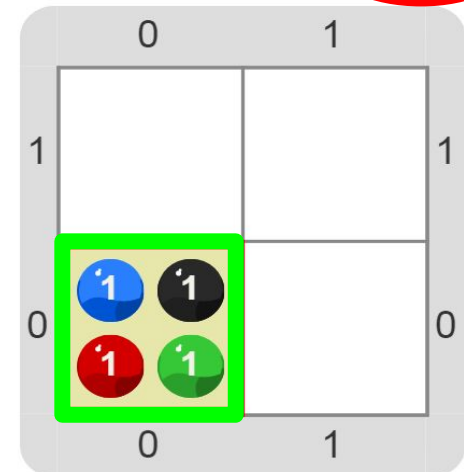
programa



Tablero final

Tablero inicial 1

Tablero final





- En distintas ejecuciones el tablero inicial puede
  - ser uno de varios posibles (quizás solo uno)
  - ser cualquiera
- El **mismo programa** debe funcionar en **todos** los tableros iniciales dados como posibles

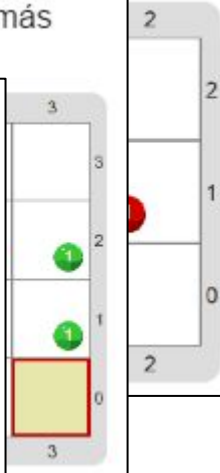
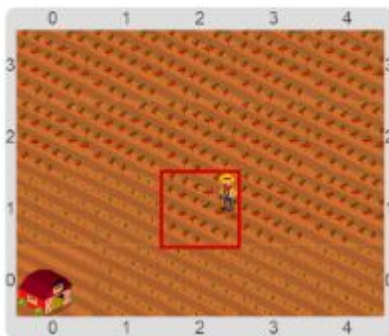
Siempre el mismo

4.2 Limpiar la cruz roja

3.3 Una víbora más inquieta

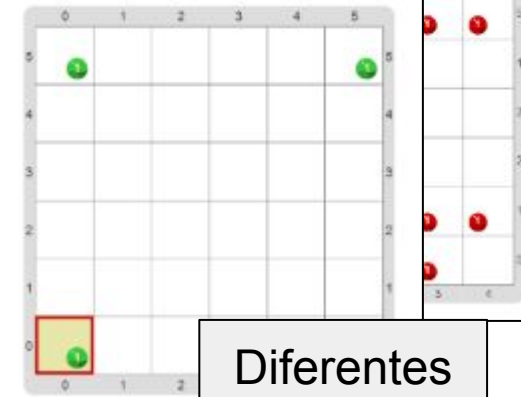
2 Gobstones

1.2 El docente, el granjero y las bolitas



5.2 Lugar para los corners

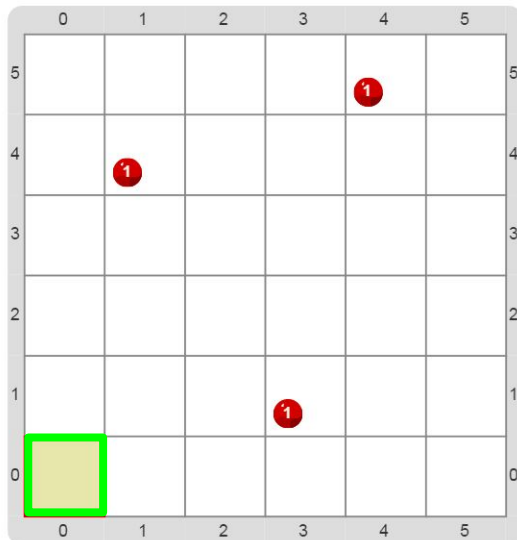
5.1 Marquemos las puntas



Diferentes posibles



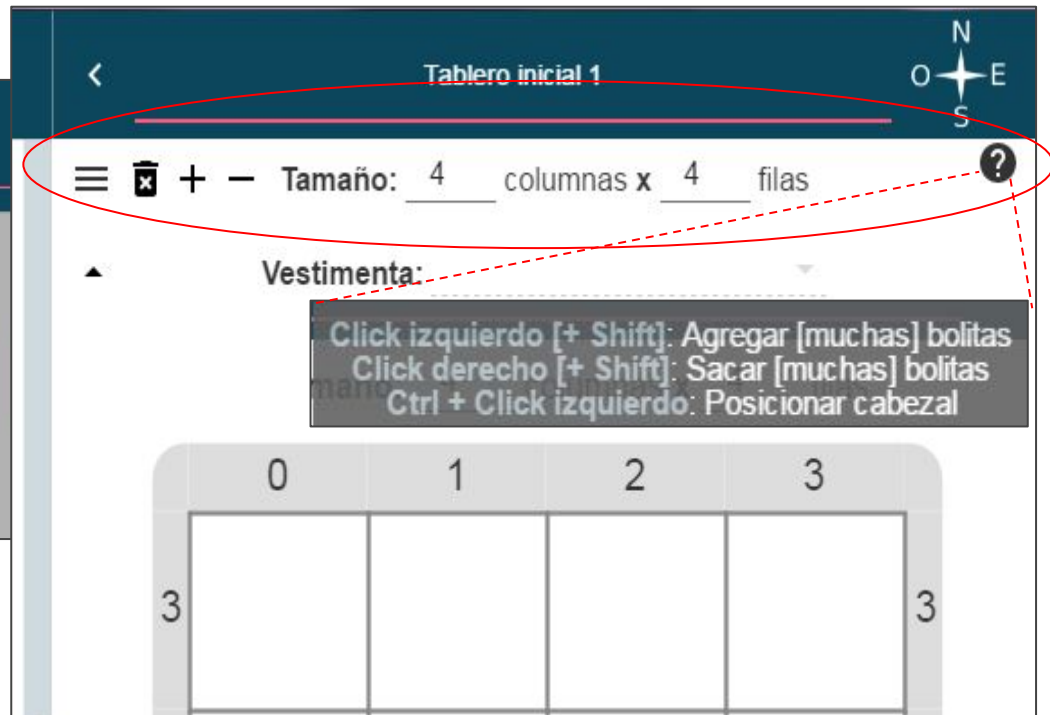
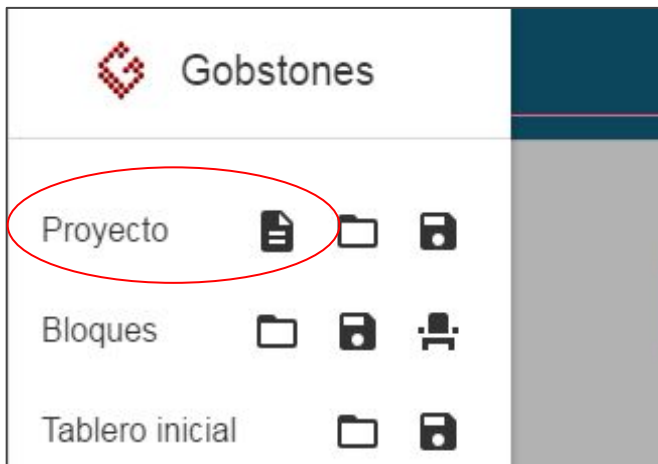
- ¿Cómo indicar cuáles tableros son los posibles?
  - informar el **estado** de cada tablero posible
- Estado del tablero inicial
  - tamaño del tablero
  - cantidad y posición de las bolitas
  - posición del cabezal (**celda actual**)



- Ejemplo
  - Tablero de 6x6
  - Hay 1 bolita roja en 3 celdas fijas
  - Cabezal en la esquina SurOeste

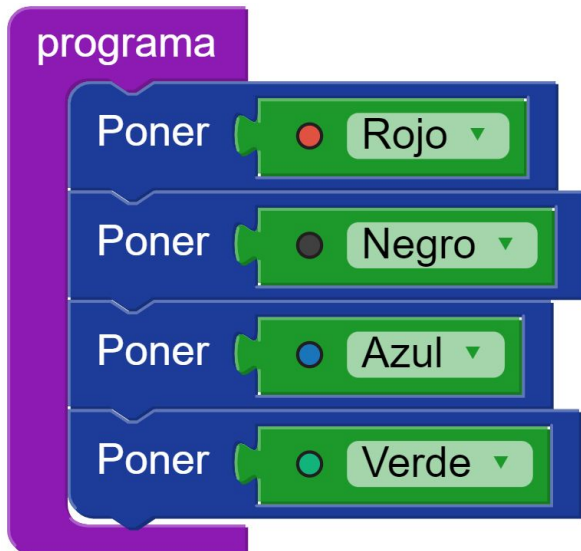


- En los proyectos predefinidos, los tableros iniciales vienen dados, y usualmente no podemos cambiarlos
- Pero para ejercicios sueltos o para probar nuestras ideas, debemos **construir nuestros propios tableros**
- En un **proyecto nuevo**, se puede editar el tablero y guardarlo para usos futuros

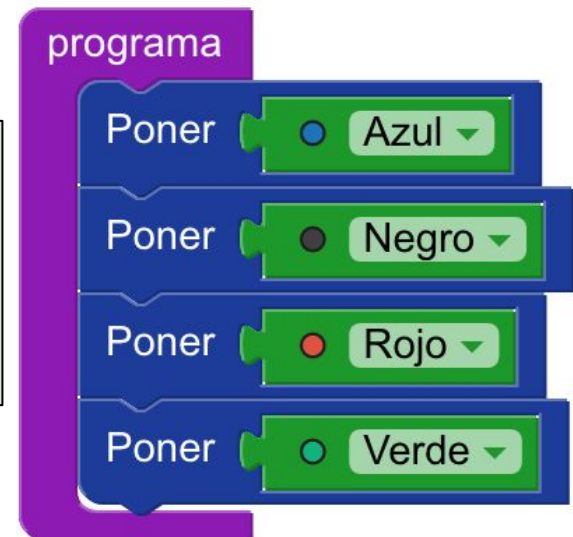




- En general solo importa la transformación completa
  - Qué problema resuelve
- Los estados intermedios no son relevantes
  - Cómo se resuelve es menos importante
- Programas distintos pueden resolver el mismo problema
  - ***Programas equivalentes***



- Son distintos programas
- Pero resuelven el mismo problema





- Usualmente existen **INFINITOS programas equivalentes** para resolver un mismo problema.
- A medida que los programas se vuelven complejos, las soluciones de diferentes personas son muy distintas.
- Por eso es importante **NO COMPARTIR** código al *empezar a aprender a programar*
  - Cada quien debe buscar expresar **SUS** ideas
  - (Cuando ya sabés programar, podés compartir)



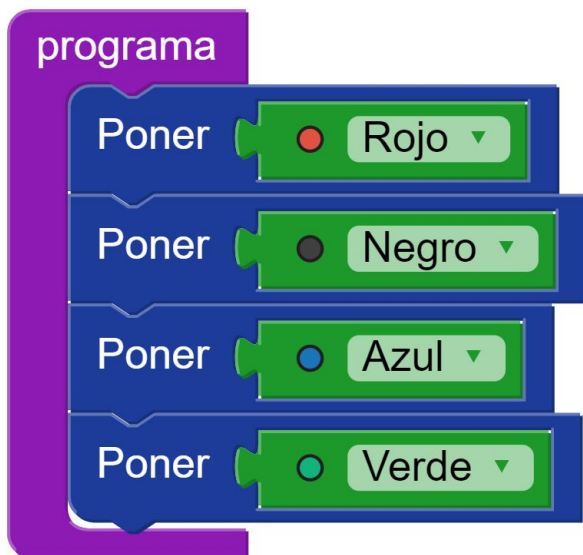


## Programas en texto



- Los bloques son una facilidad para comenzar
- Pero para programar profesionalmente debe usarse ***texto***
- El texto debe seguir ***reglas precisas (sintaxis)***
- *Por atrás, los bloques en realidad producen texto*

Con bloques



comparar

Con texto

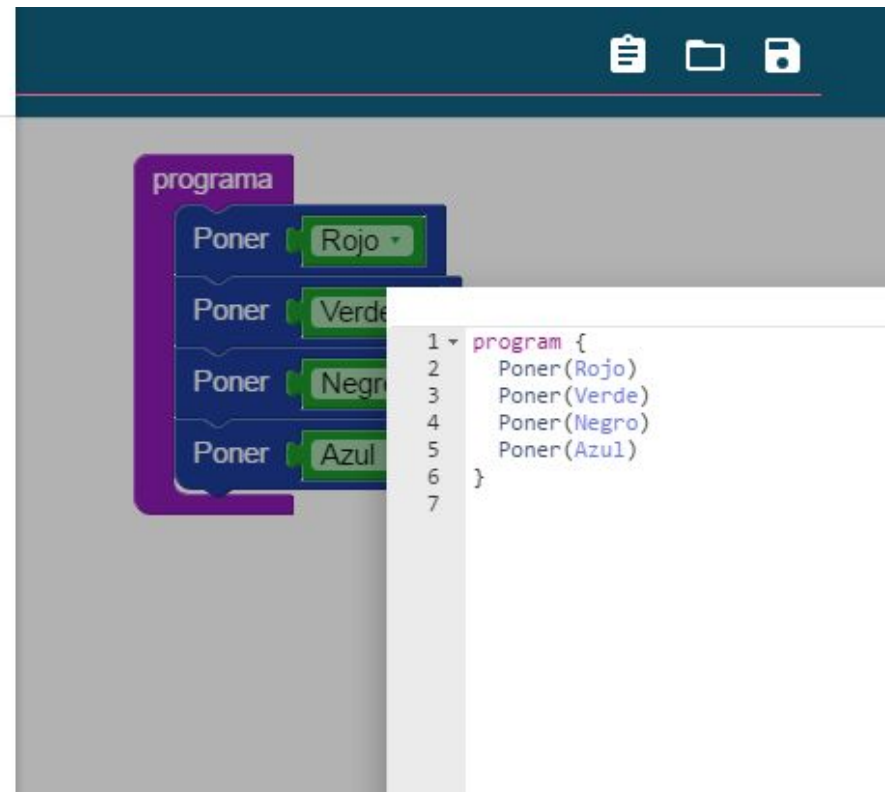
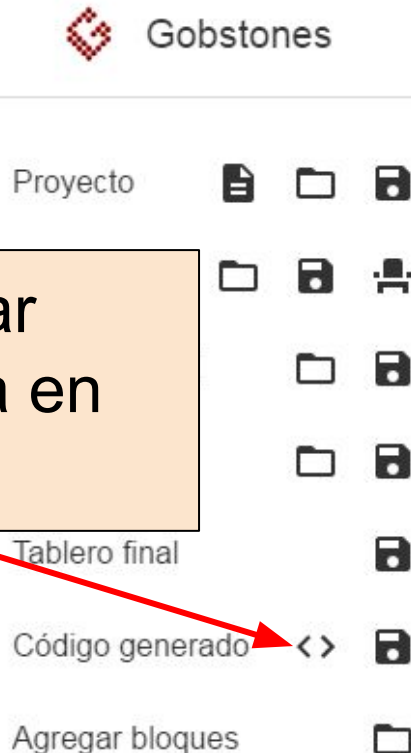
```
program {  
  Poner(Rojo)  
  Poner(Negro)  
  Poner(Azul)  
  Poner(Verde)  
}
```





- Se puede obtener automáticamente el texto a partir de los bloques
- Por ahora el camino inverso no está disponible...

Generar  
programa en  
texto



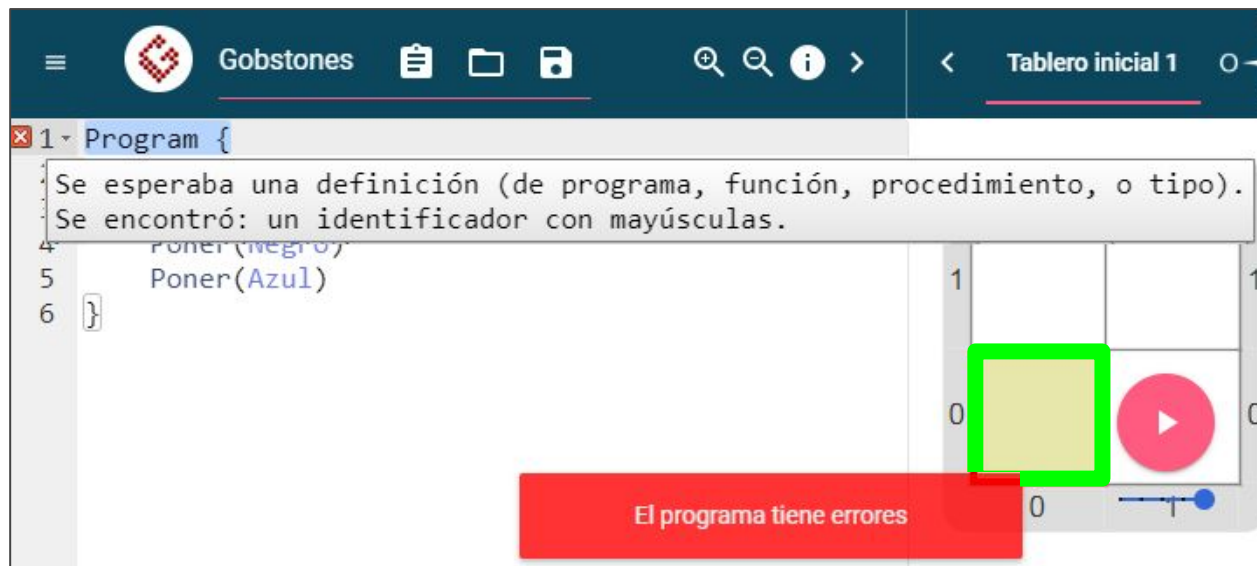


- El programa comienza con la **palabra clave** `program` y sus comandos van entre llaves { ... }
- Los comandos comienzan con **mayúscula**
  - Ej. `Poner`, `Mover`, `Sacar`, `IrAlBorde`
- Los valores literales también empiezan con **mayúscula**
  - Ej. `Rojo`, `Verde`, `Norte`, `Este`

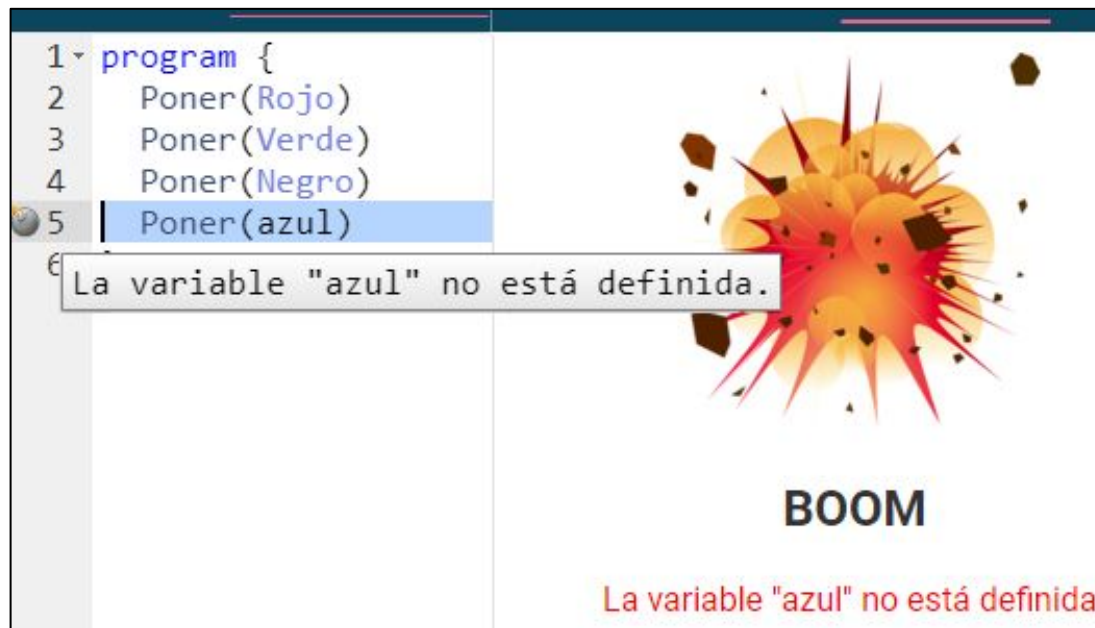
```
program {  
    Poner(Rojo)  
    Poner(Verde)  
    Poner(Negro)  
    Poner(Azul)  
}
```



- Gobstones requiere que se use la palabra exacta
  - No sirve **Program**, ni **programa**, ni **PROGRAMA**
  - Tampoco sirve **azul** ni **AZUL**, ni otras variantes
- Si no se respetan las reglas, se produce un error
  - Los mensajes de error usualmente son raros y poco claros, especialmente al comienzo



- Cuando da un error, observar el punto donde se produjo el error, o un poco antes
  - Buscar qué palabra no sigue las reglas de sintaxis
  - o qué símbolo falta (por ejemplo, una llave }
- Arreglarlo y volver a probar





- Los comandos primitivos llevan un ***argumento***
  - Ej. **Poner** (**Rojo**) , **Mover** (**Norte**)
- El argumento
  - es un valor que le da información al comando
  - se pone entre paréntesis después del comando

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

Argumentos



## Indentación



- Los programas en texto pueden contener cualquier cantidad de espacios y líneas en blanco
- Sin embargo, deben seguirse ciertas reglas de estilo
  - El programa no falla si no se siguen
  - ¡pero puede ser muy difícil de leer!

The screenshot shows a code editor window titled "Gobstones". The code is as follows:

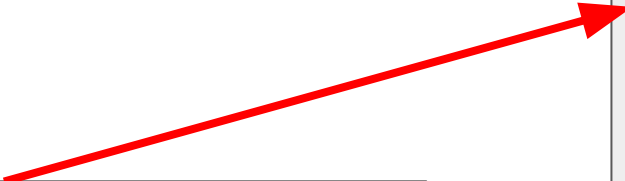
```
1 program
2   {
3   Poner(Verde)
4
5
6
7   Poner(Negro)
8
9
10
11 Poner(Azul) }
```

The code is poorly formatted with inconsistent indentation and many blank lines. A red box with the text "Programa difícil de leer" (Difficult program to read) is overlaid on the bottom right of the code editor.



- Un error típico es poner todo en la misma columna
  - No aceptamos programas así
- Debe verse claro qué cosa es parte de qué otra
  - Ej: Los comandos son parte del programa

- No queda totalmente claro que los comandos son parte del programa



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





- Usar los espacios con un buen estilo se conoce con el nombre de ***indentación*** adecuada
- Las reglas de indentación no son fijas, pero conviene seguir algunas reglas preestablecidas

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

- Los comandos empiezan todos en la misma columna
- Van más adentro que la palabra **program**
- La llave que abre, justo después que **program**
- La llave de cierre en la misma columna que **program**

A medida que aprendamos nuevas herramientas, iremos agregando algunas reglas de indentación



## Comentarios en el código



- Los programas en texto pueden contener texto que sea *ignorado* por la computadora
- Este texto a ignorar se utiliza para varios propósitos

```
/* El autor es Fidel
   (mostrar documentación) */
program {
    Poner(Rojo)      // El orden puede ser otro (comentario)
    Poner(Verde)
    Poner(Negro)
    // Poner(azul)
    // FALLA
    // (anular un comando)
}
```

- Usos típicos
  - dejar comentarios
  - documentar
  - anular un comando



- Al texto ignorado se lo llama **comentario**
- Se indica con símbolos especiales
- Hay de dos tipos: de línea y de párrafo

- De párrafo
  - Comienza con `/*`
  - Termina con `*/`
- De línea
  - Comienza con `//`
  - Va hasta el fin de línea

```
/* El autor es Fidel
   (Este es de párrafo) */
program {
    Poner(Rojo)
    Poner(Verde)
    Poner(Negro)
    Poner(Azul)
    // ARREGLADO
    // (Este es de línea)
}
```



- En bloques se pueden poner máximo un comentario por cada bloque
- Para anular un comando hay que desactivar el bloque

The diagram shows a purple 'programa' block containing several 'Poner' (Set) blocks. The first 'Poner' block is set to 'Rojo' (Red) and has a blue question mark icon. The second 'Poner' block is set to 'Verde' (Green) and also has a blue question mark icon. A third 'Poner' block is set to 'Negro' (Black). A context menu is open over the 'Verde' block, showing options: 'Duplicar', 'Añadir comentario', 'Contraer bloque', 'Desactivar bloque', 'Eliminar 2 bloques', and 'Ayuda'. The 'Añadir comentario' and 'Desactivar bloque' options are circled in red. Two callout boxes point to the first two 'Poner' blocks: the first says 'Este comando tiene un comentario' and the second says 'Este comando está desactivado'. Below the diagram, two text boxes are present: 'Comentar el bloque' with a red arrow pointing to the 'Añadir comentario' option, and 'Desactivar bloque' with a red arrow pointing to the 'Desactivar bloque' option.

Este comando tiene un comentario

Este comando está desactivado

programa

Poner Rojo

Poner Verde

Poner Negro

Poner

Duplicar

Añadir comentario

Contraer bloque

Desactivar bloque

Eliminar 2 bloques

Ayuda

Comentar el bloque

Desactivar bloque



# Contratos



- Un programa describe la solución a un problema
  - En términos de transformar un tablero en otro
- ¿Cómo saber qué problema soluciona?
  - **PROPÓSITO** es lo que debería hacer el programa
  - Es importante escribirlo como comentario

```
program {  
    /* PROPÓSITO: Poner una bolita de cada color en  
       la celda actual  
    */  
    Poner (Azul)  
    Poner (Rojo)  
    Poner (Verde)  
    Poner (Negro)  
}
```



- Cuando el efecto de un programa coincide con su propósito, decimos que es un programa ***CORRECTO***
- Si en cambio hace cosas diferentes, es incorrecto
- Hablamos, entonces de ***corrección*** de programas
  - Por eso es importante documentar el propósito

```
program {  
    /* PROPÓSITO: Poner una bolita de cada color en  
       la celda actual  
    */  
    Poner(Azul)  
    Poner(Rojo)  
    Poner(Verde)  
}  
// ¿Hace lo que debe hacer?  
// O sea, ¿es CORRECTO?
```





- ¡No hay que olvidarse del cabezal!
- No es lo mismo dejarlo en cualquier lado...

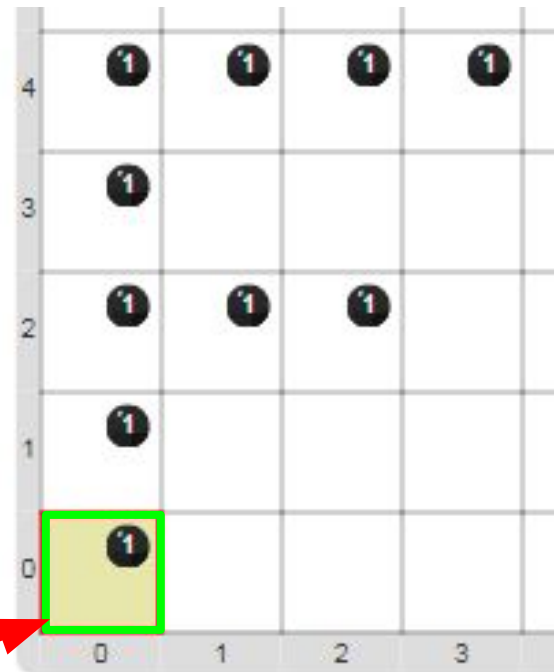
```
1 program {
2   /* PROPÓSITO:
3     "dibuja" una letra F
4   */
5   Poner(Negro) Mover(Norte)
6   Poner(Negro) Mover(Norte)
7   Poner(Negro) Mover(Norte)
8   Poner(Negro) Mover(Norte)
9   Poner(Negro) Mover(Este)
10  Poner(Negro) Mover(Este)
11  Poner(Negro) Mover(Este)
12  Poner(Negro) Mover(Sur)
13  Poner(Negro) Mover(Sur)
14  Poner(Negro) Mover(Oeste)
15  Poner(Negro) Mover(Oeste)
16  Poner(Negro)
17 }
```

¿Son equivalentes?  
¿Cuál es correcto?

```
1 program {
2   /* PROPÓSITO:
3     "dibuja" una letra F
4   */
5   Poner(Negro) Mover(Norte)
6   Poner(Negro) Mover(Norte)
7   Poner(Negro) Mover(Norte)
8   Poner(Negro) Mover(Norte)
9   Poner(Negro) Mover(Este)
10  Poner(Negro) Mover(Este)
11  Poner(Negro) Mover(Este)
12  Poner(Negro) Mover(Sur)
13  Poner(Negro) Mover(Sur)
14  Poner(Negro) Mover(Oeste)
15  Poner(Negro) Mover(Oeste)
16  Poner(Negro)
17
18  // Volver al inicio
19  Mover(Sur) Mover(Oeste)
20  Mover(Sur)
21 }
```

- Si el propósito no lo indica, el cabezal debería quedarse donde comenzó

```
1 program {  
2   /* PROPÓSITO:  
3     "dibuja" una letra F  
4   */  
5   Poner(Negro) Mover(Norte)  
6   Poner(Negro) Mover(Norte)  
7   Poner(Negro) Mover(Norte)  
8   Poner(Negro) Mover(Norte)  
9   Poner(Negro) Mover(Este)  
10  Poner(Negro) Mover(Este)  
11  Poner(Negro) Mover(Este)  
12  Poner(Negro) Mover(Sur)  
13      Mover(Sur)  
14      Mover(Oeste)  
15  Poner(Negro) Mover(Oeste)  
16  Poner(Negro)  
17  
18  // Volver al inicio  
19  Mover(Sur)  Mover(Oeste)  
20  Mover(Sur)  
21 }
```



Solo dibuja, pero al final,  
queda en el mismo lugar



- ¿Cual es el propósito de cada comando?
- ¿Siempre consigue cumplirlo?

```
program {  
  /* PROPÓSITO: Poner una  
    bolita roja en la celda  
    actual  
  */  
  Poner(Rojo)  
}
```

```
program {  
  /* PROPÓSITO: Sacar una  
    bolita roja de la celda  
    actual (si puede)  
  */  
  Sacar(Rojo)  
}
```

```
program {  
  /* PROPÓSITO: Mover el  
    cabezal una celda  
    al Este (si puede)  
  */  
  Mover(Este)  
}
```

¿Cómo saber  
cuándo va a  
funcionar?

**PRECONDICIONES**



- Las **precondiciones** son las condiciones necesarias para garantizar que un problema puede resolverse
- Deben escribirse junto con el propósito

```
program {  
  /* PROPÓSITO: Poner una bolita  
    roja en la celda actual  
  PRECONDICIONES:  
    Ninguna (siempre funciona)  
  */  
  Poner(Rojo)  
}
```

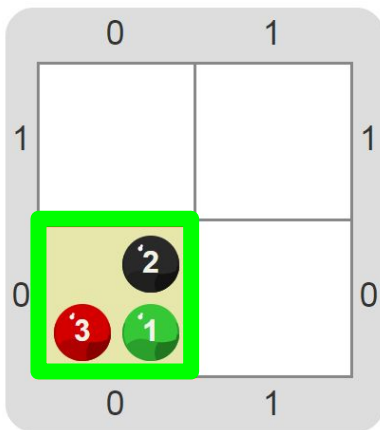
```
program {  
  /* PROPÓSITO: Sacar una bolita  
    roja de la celda actual  
  PRECONDICIONES:  
    Hay al menos una bolita  
    roja en la celda actual  
  */  
  Sacar(Rojo)  
}
```

```
program {  
  /* PROPÓSITO: Mover el cabezal  
    una celda al Este  
  PRECONDICIONES:  
    hay al menos una celda al  
    Este de la celda actual  
  */  
  Mover(Este)  
}
```

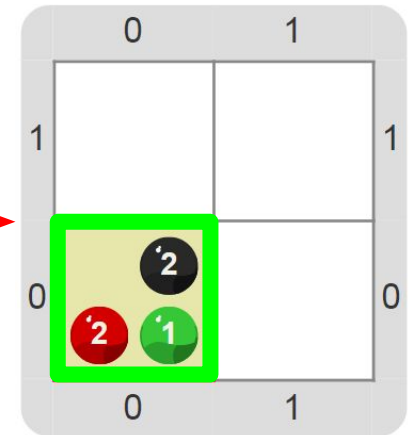
¿Y cómo se usan las precondiciones?



- Las **precondiciones** se dan en términos del estado del tablero inicial
- Si la precondición se cumple, un programa correcto debe cumplir su propósito



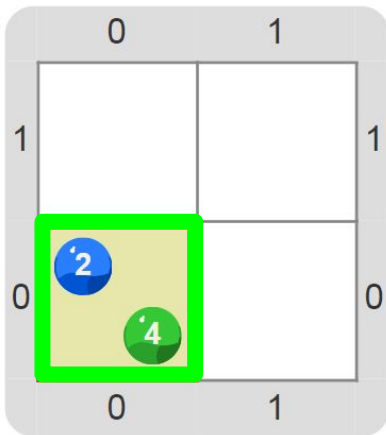
```
program {  
  /* PROPÓSITO:  
    Sacar una bolita roja de  
    la celda actual  
  PRECONDICIONES:  
    Hay al menos una bolita  
    roja en la celda actual  
  */  
  Sacar(Rojo)  
}
```



¿Y si la precondición no se cumple?



- Si la precondición NO se cumple, el programa puede comportarse de cualquier manera
  - En esta materia, el programa debe fallar con BOOM
  - Es mejor para establecer precondiciones



```
program {  
  /* PROPÓSITO:  
    Sacar una bolita roja de  
    la celda actual  
  PRECONDICIONES:  
    Hay al menos una bolita  
    roja en la celda actual  
  */  
  Sacar(Rojo)  
}
```



**BOOM**

No se puede sacar una bolita de color  
Rojo: no hay bolitas de ese color.

El BOOM es una clara muestra de  
que el programa falló



- Al conjunto de propósito y precondiciones lo llamamos **CONTRATO** de un programa
- El contrato establece qué debe hacer un programa y en qué situaciones va a funcionar correctamente

```
program {  
  /* PROPÓSITO:  
    Sacar dos bolitas rojas de la celda actual  
  PRECONDICIONES:  
    Hay al menos DOS bolitas rojas  
    en la celda actual  
  */  
  Sacar(Rojo)  
  Sacar(Rojo)  
}
```

Las precondiciones del contrato dependen de la combinación de comandos del código



- El contrato de un programa puede no ser tan fácil de determinar
  - ¡La combinación de los comandos afecta las precondiciones!

```
program {  
  /* PROPÓSITO:  
    No hacer nada (dejar el tablero igual)  
  PRECONDICIONES:  
    Ninguna (funciona siempre)  
  */  
  Poner(Rojo)  
  Sacar(Rojo)  
}
```

Este comando **Sacar** siempre funciona, porque el **Poner** garantiza su precondición





- El contrato ES PARTE del programa
- Debe aparecer como comentario para que se pueda
  - determinar si el programa es correcto
  - saber en qué condiciones va a funcionar

```
1 program{
2     Sacar(Negro) Mover(Norte) Mover(Este)
3     Sacar(Negro) Mover(Norte)
4     Sacar(Negro) Mover(Sur)   Mover(Sur)
5     Mover(Este)
6     Sacar(Negro) Mover(Oeste) Mover(Oeste)
7 }
```

Uno de estos dos programas no es considerado adecuado en esta materia...

```
1 program{
2     /* PROPÓSITO: Quita la ceniza del dibujo
3     PRECONDICIONES:
4         hay cenizas (representadas por bolitas
5         negras) en los 4 lugares indicados en
6         este diagrama
7             _ x _
8             _ x _
9             x _ x
10        La celda actual es la inferior
11        izquierda del diagrama.
12    */
13    Sacar(Negro) Mover(Norte) Mover(Este)
14    Sacar(Negro) Mover(Norte)
15    Sacar(Negro) Mover(Sur)   Mover(Sur)
16    Mover(Este)
17    Sacar(Negro) Mover(Oeste) Mover(Oeste)
18 }
```



# Procedimientos


- Los comandos primitivos se pueden poner en **secuencia** para indicar varias acciones una detrás de otra
- Salvo para problemas MUY simples, esto es poco claro

```
1 program {  
2   /*  */  
6   Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte)  
7   Poner(Rojo) Mover(Este)  Poner(Rojo) Mover(Este)  
8   Poner(Rojo) Mover(Sur)   Poner(Rojo) Mover(Sur)  
9   Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)  
10  IrAlBorde(Este) Mover(Oeste) Mover(Oeste)  
11  Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte)  
12  Poner(Rojo) Mover(Este)  Poner(Rojo) Mover(Este)  
13  Poner(Rojo) Mover(Sur)   Poner(Rojo) Mover(Sur)  
14  Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)  
15 }
```

¡La documentación está!


La indentación ayuda,  
pero no es suficiente

- La falta de claridad se evidencia más si hay que modificar el programa para cambiar algo
  - Es difícil saber dónde y qué cambiar

```
1 program {  
2  /**/  
6  Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte)  
7  Poner(Rojo) Mover(Este) Poner(Rojo) Mover(Este) Poner(Rojo) Mover(Este)  
8  Poner(Rojo) Mover(Sur) Poner(Rojo) Mover(Sur) Poner(Rojo) Mover(Sur)  
9  Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)  
10 IrAlBorde(Este) Mover(Oeste) Mover(Oeste) Mover(Oeste)  
11 Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte)  
12 Poner(Rojo) Mover(Este) Poner(Rojo) Mover(Este) Poner(Rojo) Mover(Este)  
13 Poner(Rojo) Mover(Sur) Poner(Rojo) Mover(Sur) Poner(Rojo) Mover(Sur)  
14 Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)  
15 }
```

¿Qué cambió respecto  
del anterior?

- Sería mejor tener un comando para dibujar cuadrados
- Pero Gobstones no puede tener uno por cada cosa que se nos ocurra...
- ¿Podemos definir nuestros propios comandos?


```
1 program {  
2   /*  */  
6   DibujarCuadradoRojo()  
7   PosicionarPara2doCuadrado()  
8   DibujarCuadradoRojo()  
9 }
```

¡Son equivalentes!  
¿Cuál es más claro?



```
9   Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)  
10  IrAlBorde(Este) Mover(Oeste) Mover(Oeste)  
11  Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte)  
12  Poner(Rojo) Mover(Este) Poner(Rojo) Mover(Este)  
13  Poner(Rojo) Mover(Sur) Poner(Rojo) Mover(Sur)  
14  Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)  
15 }
```




- Un **procedimiento** permite definir un comando nuevo
- ¿Cómo definimos un procedimiento?
  - En GobstonesJr, ver “**DEFINICIONES**” en la caja de herramientas
  - En GobstonesSr, se usa la palabra clave **procedure**

```
1 program {  
2   /*  */  
6   DibujarCuadradoRojo()  
7   PosicionarPara2doCuadrado()  
8   DibujarCuadradoRojo()  
9 }
```

¡La definición es parecida a la de program!

```
11 procedure PosicionarPara2doCuadrado() {  
12   /*  */  
16   IrAlBorde(Este) Mover(Oeste) Mover(Oeste)  
17 }  
18  
19 procedure DibujarCuadradoRojo() {  
20   /*  */  
24   Poner(Rojo) Mover(Este) Poner(Rojo) Mover(Este)  
25   Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte)  
26   Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)  
27   Poner(Rojo) Mover(Sur) Poner(Rojo) Mover(Sur)  
28 }
```

- Un **procedimiento**, al igual que un programa, tiene un **cuerpo** conformado por comandos entre llaves { ... }
- Pero a diferencia del programa, tiene un **nombre**
  - Es el que se usará como comando
  - Empieza con mayúsculas
  - La primera palabra es un verbo (¿por qué?)
  - Lleva paréntesis después (ej. `DibujarCuadradoRojo()`)
  - Recomendamos usar CamelCase para poner nombres
    - Varias palabras, todas pegadas, donde cada palabra la comenzamos con mayúsculas
  - Debe describir el propósito de forma rápida

```
19 ▸ procedure DibujarCuadradoRojo() {  
20 ▸     /*  */  
24 ▸     Poner(Rojo)  Mover(Este)  Poner
```

- El nombre de un procedimiento
  - Debe estar vinculado con su propósito
  - Es parte de la documentación del programa
  - Debe comenzar con un verbo
  - Debe poderse entender con facilidad

```
procedure Dibujar() {
```

```
procedure P1() {
```

```
procedure Procedimiento() {
```


```
procedure HacerAlgo() {
```

```
procedure Cuadrado() {
```

¿Se entiende por qué ninguno de estos nombres es adecuado para el procedimiento anterior?




- ¿Para qué sirven los procedimientos?
  - Aportan **claridad** (si están bien definidos y nombrados)
  - Facilitan la **reutilización** y la **modificación**
  - Permiten **expresar** la solución en términos del problema y no de bolitas

```
1 program {  
2   /*  */  
6   MoverAlienAlEste()  
7   MoverAlienAlEste()  
8   MoverAlienAlEste()  
9   TocarBotón()  
10 }
```

¡Son equivalentes!

Pero en el primero sabemos en qué está pensando el programador

```
1 program {  
2   /*  */  
6   Sacar(Verde)    Mover(Este)  Poner(Verde)  
7   Sacar(Verde)    Mover(Este)  Poner(Verde)  
8   Sacar(Verde)    Mover(Este)  Poner(Verde)  
9   Sacar(Rojo)  
10 }
```



- Un proyecto de GobstonesWeb puede traer definidos procedimientos que expresen las primitivas del problema
- Los llamamos ***procedimientos primitivos***

¡La única diferencia es quién define esos procedimientos!

COMANDOS  
Comandos primitivos  
**Mis procedimientos**  
EXPRESIONES  
Valores literales  
DEFINICIONES  
Procedimientos

Mover al Alien al Este  
Tocar botón

programa

Mover Alien al Este  
Tocar botón

programa

Mover al Alien al Este  
Mover al Alien al Este  
Mover al Alien al Este  
Tocar botón

Definir Mover al Alien al Este

Sacar Verde  
Mover Este  
Poner Verde

Definir Tocar botón

Sacar Rojo



Cierre



# Cierre



- **Programar es comunicar**
  - una máquina **ejecuta** el código
  - las personas deben **entender** el código
- **Lenguaje de programación (Gobstones)**
  - **Comandos**: describen acciones
    - Comandos primitivos, PROCEDIMIENTOS
  - **Expresiones**: describen información
  - Debemos aprender las reglas de sintaxis
- **El programa**
  - puede ser un texto o estar hecho con bloques y
  - **describe** soluciones a problemas expresados como
  - **transformaciones de estado** (tablero inicial en final)
  - y si es la misma transformación son **equivalentes**



- **Indentar** es organizar el texto
  - para mostrar las jerarquías, y
  - hacer el texto más entendible
- **Documentar** es agregar información para las personas
  - **Contrato: propósito y precondiciones**
  - Se utilizan **comentarios**, que es texto que la máquina ignora
- Los **procedimientos** nos permiten
  - definir nuestros propios comandos
  - expresar los problemas en forma más clara
  - no tener que repetir código innecesariamente