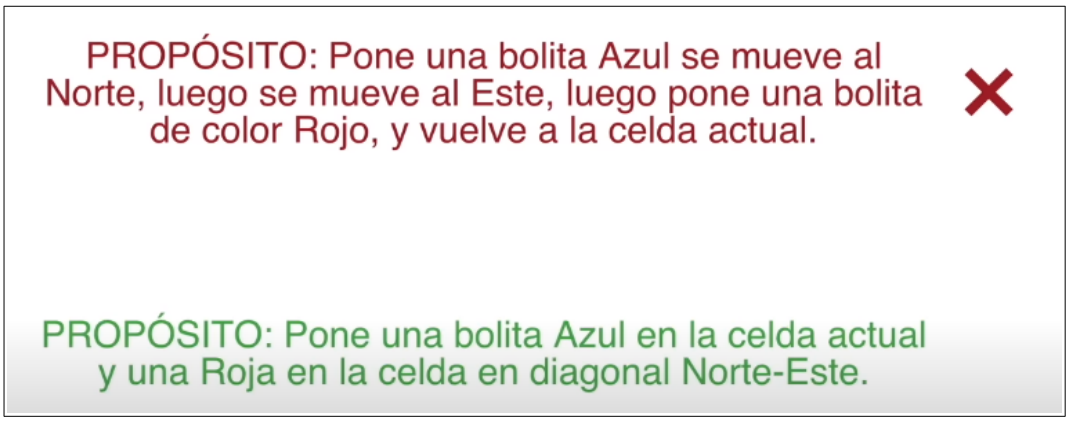
- El **<u>Propósito del contrato</u>** debería responder a **¿Qué?** (qué transformación de estado final tendrá el tablero inicial) Y **¿Dónde?** (refiriéndose a las celdas en las que se realiza la transformación de estado FINAL incluyendo el cabezal, ya que es parte de los elementos del tablero)

Entonces, en el propósito debo responder a: ¿Qué hace el programa? Y ESTO IMPLICA:

- ¿En que celdas ocurre la transformación FINAL tomando como punto de partida la "posición actual del cabezal" (el cual es aleatorio) en el estado inicial del tablero ?
- ¿Con que color/colores se representan los objetos del problema?
- ¿Dónde queda el cabezal en el estado final del programa? (siendo que si debe posicionarse en la misma celda que donde comenzó, no es una transformación de estado. Por lo tanto, no se menciona).
- Los cambios de estado INTERMEDIOS NO SON RELEVANTES.



El ¿Cómo? Esta dado por el código en el cuerpo del programa.

AHORA PRECONDICIONES:

Siempre deben hacer referencia al tablero en el estado INICIAL del tablero. ¿Por qué? Porque un programa puede probarse en tableros de diferentes dimensiones/características, por lo tanto son requerimientos indispensables que necesitará el entrono donde se ejecutará el programa para poder ejecutarse exitosamente (valga la redundancia).

Entonces, se expresan como restricciones del tablero inicial. Si no se cumple la precondición, el programa no puede cumplir con su propósito.

```
program {
    /*
        PROPÓSITO: Dibuja una línea de color Rojo de
            4 celdas de largo. El cabezal queda
            3 lugares al Este de la celda actual.
        PRECONDICIÓN: Deben haber 3 celdas al Este
            de la actual.
    */
```

Un programa que usa COMANDOS PARCIALES (como mover, sacar), siempre requerirá de precondiciones. Ya que el programa puede fallar. Sin embargo un programa que solo usa COMANDOS TOTALES (como poner) nunca fallará, por lo tanto no necesita precondiciones.

(en un tablero se no hay limite en la cantidad de bolitas que se pueden colocar en una celda)

EJEMPLO DE PROGRAMA QUE NO HACE TRANSFORMACIÓN DE ESTADO:

```
program {
    /*
      PROPÓSITO: No hace nada.
      PRECONDICIÓN: Ninguna.
    */
    Poner(Rojo)
    Sacar(Rojo)
}
```

Es necesario no hablar de dimensiones exactas del tablero ni de la posición inicial del cabezal en el contrato. Ya que deja a fuera muchos escenario posibles. Sino que deberíamos referirnos a la cantidad de celdas de una dirección a otra, haciendo referencia a los movimientos que hagamos.
Lo mismo aplica con las bolitas. (hacer referencia a colores a utilizar, más no a “la bolita de color..”).

EL CONTRATO

- Indica qué hace el programa
- Indica cuándo funciona correctamente y cuándo no
- Ayuda a pensar el problema antes de intentar solucionarlo
- Sirve para hacer código más mantenible en el tiempo
- Sirve (más adelante) para poder reutilizar código

```
program {
    /*
      PROPÓSITO: Dibuja un cuadrado sólido de color Verde
          de 3 celdas de lado con centro en la celda actual.
      PRECONDICIÓN: Debe existir al menos una celda hacia
          cada dirección.
    */
```

} Contrato

```
procedure DibujarCuadradoRojo() {
    /*
      PROPÓSITO: Dibuja el cuadrado de la esquina
          Sur-Oeste del tablero.
      PRECONDICIÓN: El cabezal debe estar en la
          esquina Sur-Oeste del tablero.
    */
    Poner(Rojo) Mover(Este) Poner(Rojo) Mover(Este)
    Poner(Rojo) Mover(Norte) Poner(Rojo) Mover(Norte)
    Poner(Rojo) Mover(Oeste) Poner(Rojo) Mover(Oeste)
    Poner(Rojo) Mover(Sur)   Poner(Rojo) Mover(Sur)
}
```

```
program {
    /*
      PROPÓSITO: Dibuja dos cuadrados de color Rojo
          de 3 celdas de lado uno en la esquina Sur-Oeste
          del tablero y otro en la Sur-Este.
      PRECONDICIONES:
          * El tablero tiene al menos 7 celdas de ancho
            y 3 de alto.
          * El cabezal inicia en la esquina Sur-Oeste
          del tablero.
    */
    DibujarCuadradoRojo()
    IrAlBorde(Este) Mover(Oeste) Mover(Oeste)
    DibujarCuadradoRojo()
}
```

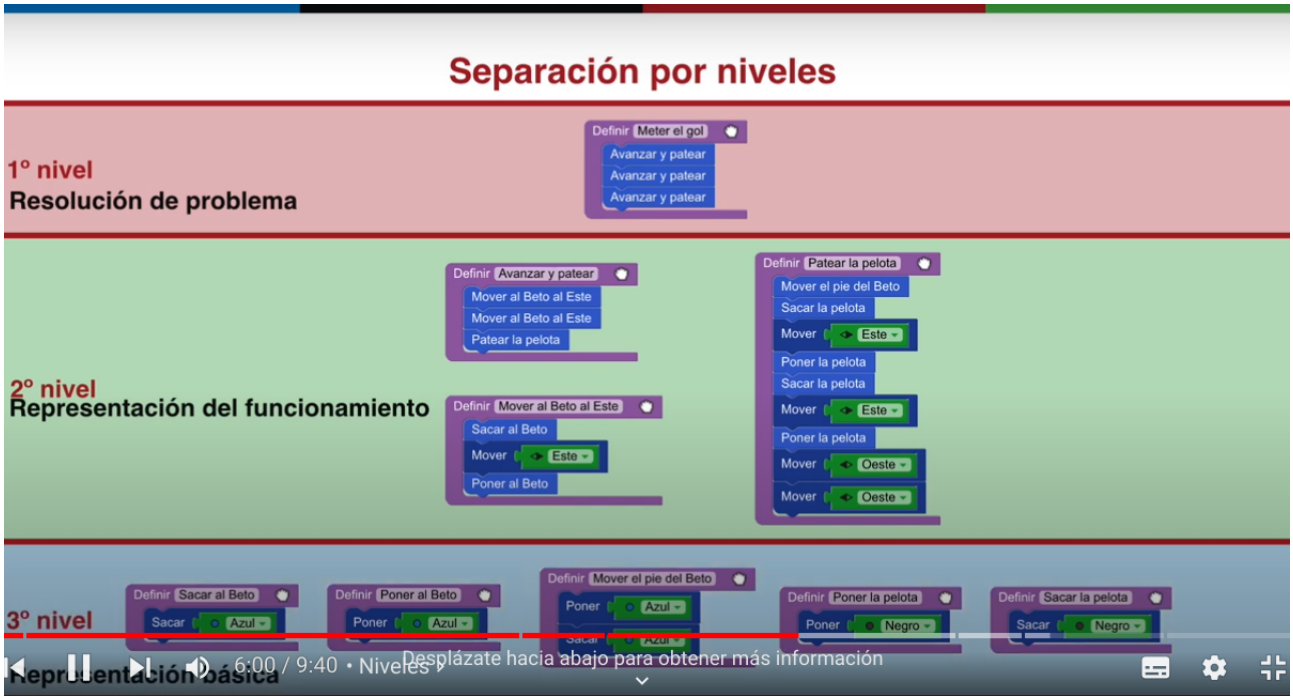
Pero si solo pone en la esquina Sur-Oeste, entonces no lo puedo usar para que ponga un cuadrado en la Sur-Este.

LOS PROCEDIMIENTOS Y SUS CONTRATOS:

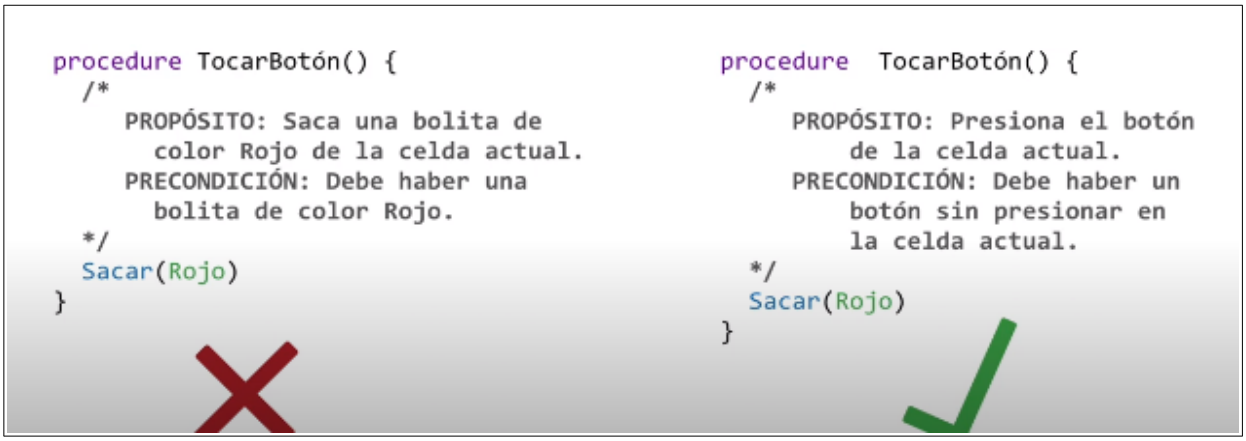
Un problema puede resolverse por niveles. Los niveles inferiores en un programa involucrarán ***procedimientos de representación:***

- Entendiendo como procedimientos de representación a los procedimientos que hacen cosas elementales con la representación y que hablan en términos de bolitas (en el cuerpo del procedimientos, nunca en el contrato).

Los procedimientos que hablen en términos del problema (en el bloque de código) están por arriba de los que hablen en términos de bolitas.

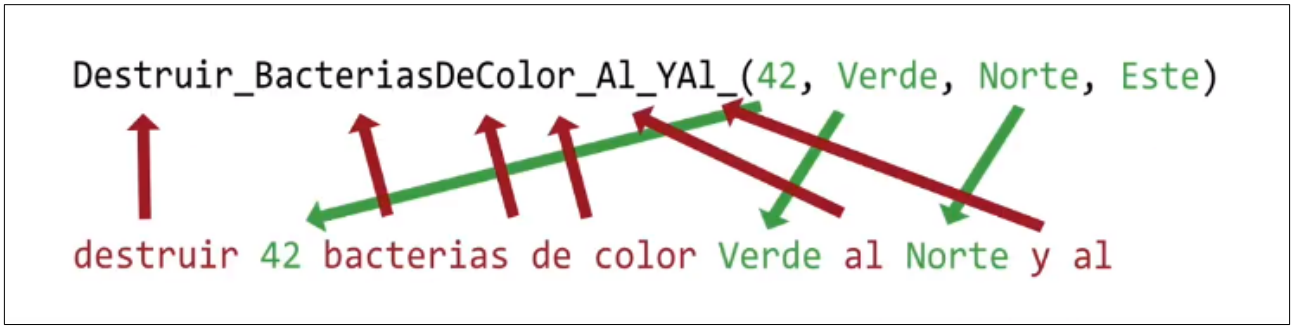


De esta forma, el contrato debe hablar SIEMPRE EN TÉRMINOS DEL PROBLEMA Y NO DE BOLITAS.

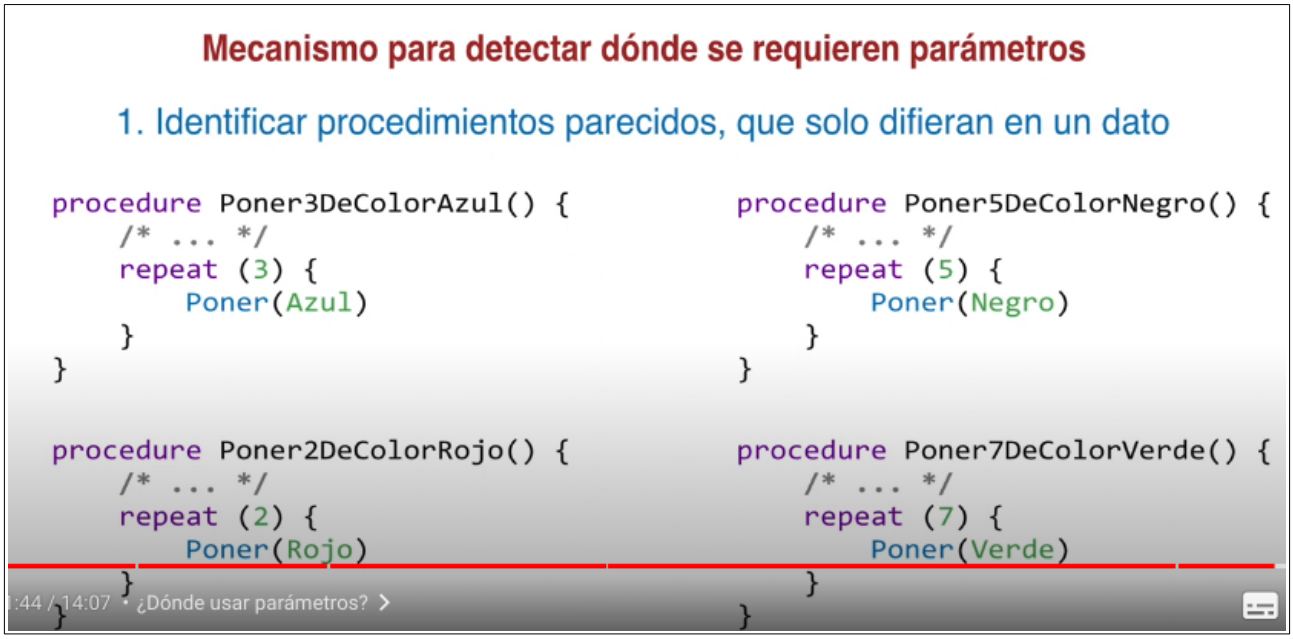


Fuente: https://www.youtube.com/watch?v=eN_aL4ty7jo&list=PLfrTDBDj636aRNJQA19LB7paPuQX_QEpm&index=6

PARÁMETROS:
https://www.youtube.com/watch?v=zi-OaXj7C5s&list=PLfrTDBDj636aRNJQA19LB7paPuQX_QEpm&index=9



Antes de parametrizar, hay que pensar si el dato que requerimos es fijo o existen en el programa procedimientos parecidos (donde se utiliza el mismo tipo de dato y es argumento de un comando o comandos que se repite/en entre los procedimientos que sean parecidos, pero la expresión(no el tipo de dato) necesita cambiar/variar durante la ejecución del programa. -siendo la única diferencia entre estos procedimientos-) .



Mecanismo para detectar dónde se requieren parámetros

2. Dejar un agujero en los lugares donde se usa el dato que cambia

```
procedure Poner3DeColorAzul() {
    /* ... */
    repeat (□) {
        Poner(□)
    }
}

procedure Poner2DeColorRojo() {
    /* ... */
    repeat (□) {
        Poner(□)
    }
}

procedure Poner5DeColorNegro() {
    /* ... */
    repeat (□) {
        Poner(□)
    }
}

procedure Poner7DeColorVerde() {
    /* ... */
    repeat (□) {
        Poner(□)
    }
}
```

11:48 / 14:07 • ¿Dónde usar parámetros? >

3. Dejar un único procedimiento ya que todos son iguales...
... y ajustar su nombre de forma acorde usando Mix Fix

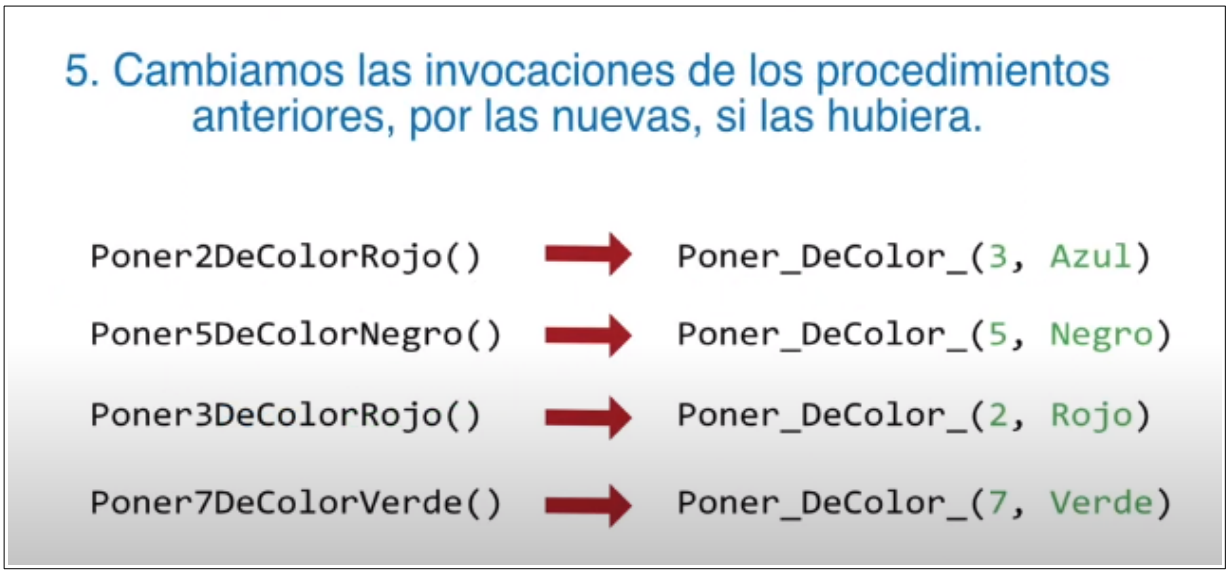
```
procedure Poner_DeColor_() {
    /* ... */
    repeat (□) {
        Poner(□)
    }
}
```

Finalmente…

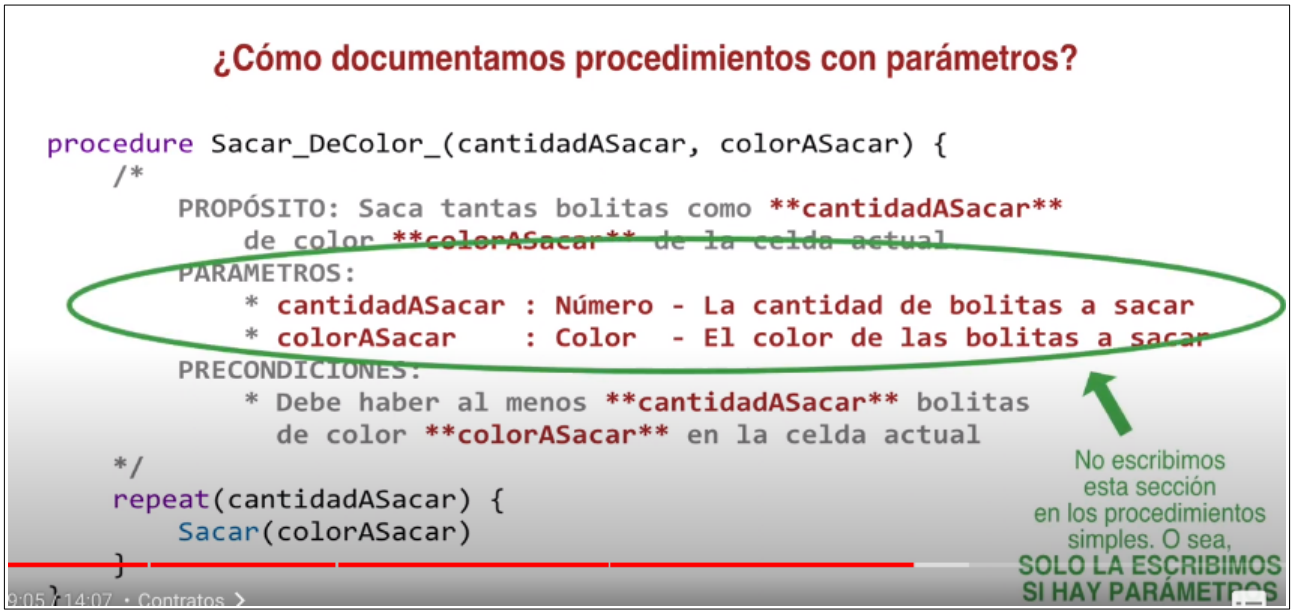
4. Agregamos un parámetro por cada agujero que tenemos

```
procedure Poner_DeColor_(cantidadAPoner, colorAPoner) {
    /* ... */
    repeat (cantidadAPoner) {
        Poner(colorAPoner)
    }
}
```

Y Ahora….



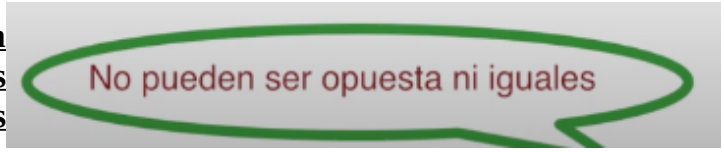
PROCEDIMIENTOS CON PARÁMETROS Y SUS CONTRATOS:
fuente: **https://www.youtube.com/watch?v=KLk4ls1Zg_Y&list=PLfrTDBDj636aRNJQA19LB7paPuQX_QEpm&index=10**



Uno mas... y así aprendemos más...

```
procedure MoverEnDiagonalAl_YAl_(dirección1, dirección2) {
    /*
        PROPÓSITO: Mueve el cabezal hacia la celda en diagonal
            hacia **dirección1** y **dirección2**
        PARÁMETROS:
            * cantidadASacar : Número - La cantidad de bolitas a sacar
            * colorASacar    : Color  - El color de las bolitas a sacar
        PRECONDICIONES:
            * Debe haber al menos una celda hacia **dirección1**.
            * Debe haber al menos una celda hacia **dirección2**.
            * **dirección1** y **dirección2** no pueden ser opuestas.
            * **dirección1** y **dirección2** no pueden ser iguales.
    */
```

<u>En este caso, las direcciones que se pasen como argumento no pueden ser opuestas ni iguales al invocar al procedimientos con parámetros.</u>



No pueden ser, por ejemplo: (Norte, Norte) ni (Norte, Sur) , al invocar el procedimientos con parámetros.

DEBO VERIFICAR QUE LOS DATOS QUE NOS PASEN COMO ARGUMENTO CUMPLAN CON EL PROPÓSITO DEL PROCEDIMIENTO MEDIANTE RESTRICCIONES COLOCADAS EN LAS PRECONDICIONES, LAS CUALES INDICARAN LO QUE NO PUEDEN HACER O SER LOS DATOS QUE SE RECIBAN.

Entonces, así como se pueden establecer restricciones sobre el tablero, también se pueden establecer restricciones sobre los datos que se utilicen en el procedimientos siempre que permitan que el propósito se cumpla.

Deberemos tener tantos argumentos, como parámetros tengamos y siempre respetando el orden en el que los parámetros fueron definidos.

“Una precondiciones es una restricción sobre el tablero inicial del programa, pero también puede ser una restricción sobre la información inicial de un procedimiento (osea que los datos del parámetro también pueden tener restricciones y deben cumplir una condición especifica en el tablero para que el propósito se cumpla)”.

EXPRESIONES Y TIPOS DE DATOS: (PAG106)
fuente: https://www.youtube.com/watch?v=PzVjqnMBvxw&list=PLfrTDBDj636aRNJQA19LB7paPuQX_QEpm&index=11

TIPOS DE DATOS: Numérico, Color, Dirección, Booleano.

EXPRESIONES EN GOBSTONE: TRABAJAN COMO SENSORES PARA EL CABEZAL. LE PERMITEN SABER SI HAY O NO BOLITAS DE CIERTO COLOR EN LA CELDA ACTUAL, CUANTAS HAY, Y SI PUEDE O NO MOVERSE DE MANERA SEGURA SIN CAERSE DEL TABLERO.

- **nroDeBolitas(<color>)** → toma una expresión de tipo Color como argumento, pero nroDeBolitas es una expresión de tipo NUMERO. (LE INDICA AL CABEZAL CUANTAS BOLITAS DE UN COLOR HAY EN LA CELDA ACTUAL)

- **puedeMover(<dirección>)** → representa un booleano

- **hayBolitas(<color>)** → representa un booleano

“pueden haber expresiones que también tomen argumentos y esos argumentos deben tener un tipo especifico.”

OPERACIONES PREDEFINIDAS (especificas para cada tipo de dato) PARA CONSTRUIR EXPRESIONES:

- Operaciones sobre números: +, -, *, div, mod, exponencial. -son del tipo numérico

- Operaciones sobre Colores: minColor() y maxColor() - son del tipo color

- Operaciones sobre direcciones: minDir(), maxDir() - son del tipo dirección

- Operaciones sobre Booleanos: not, &&, minBool(), maxBool(), || -tipo booleano

NOTAS ———————————————————————————

- *Las operaciones y expresiones que representan un Booleano se utilizan en condicionales.*

- *Las operaciones y expresiones que representan un numero se utilizan en repeticiones.*

OPERACIONES QUE SE PUEDEN UTILIZAR CON TODOS LOS TIPOS DE DATOS PARA CREAR EXPRESIONES BOOLEANAS:

- ==, !=, <, >, <=, >= -Permiten realizar comparaciones entre expresiones del mismo tipo.

CASOS ESPECIALES:

- previo, siguiente, opuesto.

EJEMPLO DE ERROR DE TIPOS DE DATOS:

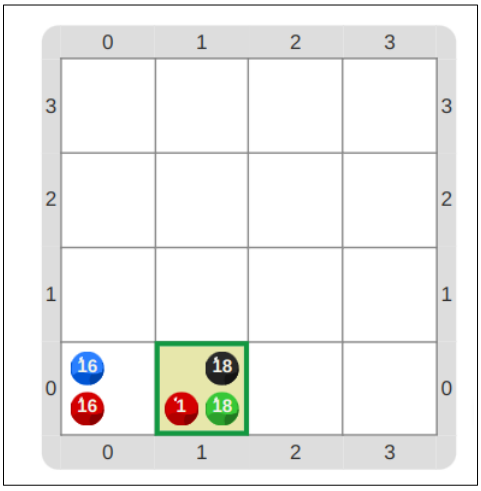
```
16 procedure PonerUnaDecada(){
17     repeat(nroBolitas(minColor())){
18         Poner(Rojo)
19     }
20     Mover(Este)
21     repeat (nroBolitas(maxColor())){
22         Poner(Negro)
23     }
24     if(maxColor()== 1){
25         Mover(Este)
26         Poner(Negro)
27     }else{
28         Poner(Rojo)
29     }
30 }
31
```

BOOM

Los tipos de las expresiones no coinciden: la primera es un color y la segunda es un número.

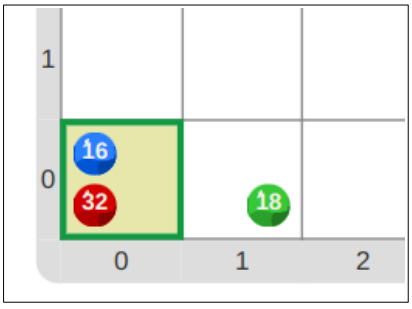
EJEMPLO CON TIPOS DE DATOS NUMÉRICOS:

```
procedure PonerUnaDecada(){
    repeat(nroBolitas(minColor())){
        Poner(Rojo)
    }
    Mover(Este)
    repeat (nroBolitas(maxColor())){
        Poner(Negro)
    }
    if(hayBolitas(Rojo)){
        Mover(Este)
    }else{
        Poner(Rojo)
    }
}
```



MAS EJEMPLOS CON TIPOS DE DATOS NUMÉRICOS:

```
procedure PonerUnaDecada(){
    repeat(2*(nroBolitas(Azul))){
        Poner(Rojo)
    }
}
```



LAS OPERACIONES DE MÍNIMO Y MÁXIMO PERMITEN ARMAR REPETICIONES INDEXADAS. (TODAVÍA NO LO VES)

PROGRAMAR A LA DEFENSIVA:

FUNCIONES SIMPLES

FUNCIONES PRIMITIVAS: Es una forma de definir expresiones personalizadas. En este sentido, serán expresiones creadas por el que diseña la actividad.

Así como los procedimientos me permiten definir comandos, **las funciones me permiten** o le permiten al programador **definir expresiones**.
A diferencia de los comandos, las expresiones **tienen un TIPO DE DATO**, los comando describen acciones y no tienen TIPO…..

Esto es importante ya que *dentro del return() de una función solo se permitirán* :

- expresiones primitivas, otras funciones y valores literales. →(Azul, 2, Este)

Cabe reiterar que un PARÁMETRO es una expresión y las funciones pueden recibir parámetros.
No se permitirán :

- condicionales, bucles o ciclos, ni comandos (primitivas o procedimientos).
- Las expresiones no pueden recibir comando como argumentos, solo otras expresiones.

¿Cuándo uso una función?

- **Funciones de representación:**
  Me permite tener niveles de abstracción con las expresiones utilizadas en los argumentos. De esta forma, ya no necesitaría colocar *observaciones* donde explique qué objetos serán representados mediante un color determinado de bolitas. Sino que *la función tendrá como nombre el objeto a representar* y en *el return devolverá el elemento con el cual se representa a ese objeto.*

  //representa un árbol con una bolita verde y peso ($) con una bolita negra.

```
Function arbol(){          Function peso(){
    return (Verde)             return (Negro)
}                          }
```

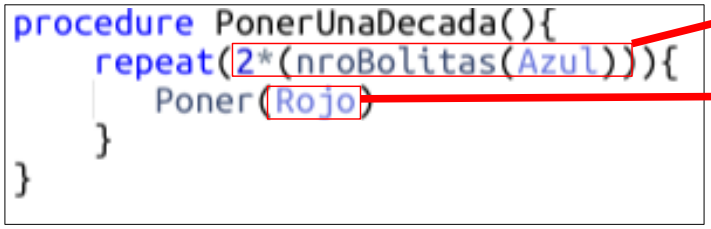
  De esta forma utilizo a la expresión (creada por la función) como argumento de un procedimiento y el código se comprende mejor.

  arbol() *será la forma correcta de invocar a una expresión*

- **Para aclarar las cantidades que representan a un elemento.**
  return (nroBolitas(peso()))

- **Cuando necesito realizar cálculos/operaciones con los elementos a representar y devuelvo únicamente el resultado final:**
  return**(** CantidadDePesosEnCuenta() - CantidadDePesosAExtraer())

- **Cuando necesito hacer una pregunta sobre la condición o estado de un elementos para determinar su valor de verdad:** if (*hayManzana()) then…..*



Este es el código sin abstracción en las expresiones.