



Introducción a la Programación

Clases teóricas

por Pablo E. “Fidel” Martínez López

2. Procedimientos, representación y estrategia





Repaso



- **Programar es comunicar** (con máquinas y personas)
- Lenguaje de programación (Gobstones)
 - **Comandos**: describen acciones
 - **Expresiones**: describen información
- Programas
 - **Describen transformaciones de estado**
 - Hay infinitos programas **equivalentes**
 - Deben **documentarse** e **indentarse**
 - **Propósito y precondiciones**



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 }

```


```

Mover(Norte) Poner(Rojo) Mover(Norte)
Mover(Este)  Poner(Rojo) Mover(Este)
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 }



```

¡Son equivalentes!
¿Cuál es más claro?

- 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**




```
procedure DibujarCuadradoRojo() {  
  /*  */  
  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)  
}
```

Nombre

Cuerpo



- El **cuerpo** determina el efecto del comando definido
- El **nombre** es el que define al nuevo comando

```
/*  */  
program {  
  DibujarCuadradoRojo()  
  PosicionarPara2doCuadrado()  
  DibujarCuadradoRojo()  
}  
  
procedure PosicionarPara2doCuadrado() {  }  
procedure DibujarCuadradoRojo() {  }
```

Comandos
nuevos

Definiciones
de procedimientos



- El **nombre** de un procedimiento
 - Empieza con mayúsculas
 - Lleva paréntesis después (ej. `DibujarCuadradoRojo()`)
 - La primera palabra debe ser un verbo (¿por qué?)
 - Recomendamos usar camelCase para poner nombres
 - Varias palabras en mayúsculas, todas pegadas
 - Debe describir el propósito de forma rápida

Buenos nombres

```
procedure PlantarUnaFlor() {
```

```
procedure PosicionarPara2doCuadrado() {
```

```
procedure DibujarCuadradoRojo() {
```

```
procedure HuirEnElUnicornio() {
```

```
procedure PatearLaPelota() {
```



- 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

Malos nombres

```
procedure Dibujar() {
```

```
procedure Procedimiento() {
```

```
procedure P1() {
```

```
procedure HacerAlgo() {
```

```
procedure Cuadrado() {
```

¿Por qué ninguno de estos nombres es adecuado?



Propósito de procedimientos



- Un **procedimiento**, al igual que un programa, tiene un **contrato** conformado por propósito y precondiciones
 - **Propósito**: qué transformación expresa
 - **Precondiciones**: requerimientos para que funcione
- ¡Debe documentarse con el procedimiento!

```
procedure DibujarCuadradoRojo() {  
  /*  
    PROPÓSITO: dibuja un cuadrado de lado 3, hecho con bolitas rojas  
    PRECONDICIONES:  
      hay al menos 2 celdas al Norte y 2 al Este de la celda inicial  
  */  
  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)  
}
```



- El **propósito** debe expresar la transformación *completa*
- Los estados intermedios NO son relevantes

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

CORRECTO
Expresa el efecto completo

INCORRECTO
Expresa estados
intermedios

```
procedure PonerUnaBolitaDeCadaColor() {  
  /*  
    PROPÓSITO: pone una bolita azul, y  
               después una negra, y después  
               una roja y después una verde  
  */  
  Poner(Azul) Poner(Negro)  
  Poner(Rojo) Poner(Verde)  
}
```




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

```
procedure DibujarLetraF() {  
  /*  
    PROPÓSITO: dibuja una letra F  
    PRECONDICIONES:  
      hay suficiente lugar  
  */  
  Poner(Negro) Mover(Norte) Poner(Negro) Mover(Norte)  
  Poner(Negro) Mover(Norte) Poner(Negro) Mover(Norte)  
  Poner(Negro) Mover(Este) Poner(Negro) Mover(Este)  
  Poner(Negro) Mover(Este) Poner(Negro) Mover(Sur)  
  Mover(Sur) Mover(Oeste)  
  Poner(Negro) Mover(Oeste) Poner(Negro)  
}
```

¿Es correcto?
¿Qué pasa con el cabezal?



- Si no es parte del propósito mover el cabezal, lo correcto es dejarlo donde estaba

```
procedure DibujarLetraF() {  
  /*  
    PROPÓSITO: dibuja una letra F  
    PRECONDICIONES: hay suficiente lugar  
  */  
  Poner(Negro) Mover(Norte) Poner(Negro) Mover(Norte)  
  Poner(Negro) Mover(Norte) Poner(Negro) Mover(Norte)  
  Poner(Negro) Mover(Este) Poner(Negro) Mover(Este)  
  Poner(Negro) Mover(Este) Poner(Negro) Mover(Sur)  
  Mover(Sur) Mover(Oeste)  
  Poner(Negro) Mover(Oeste) Poner(Negro)  
  Mover(Sur) Mover(Oeste) Mover(Sur)  
}
```

¡No modifica la celda inicial!



- La razón de esto es que otros procedimientos luego saben donde queda el cabezal sin problemas



```
/*  
  PROPÓSITO: escribe "FIDEL"  
  PRECONDICIONES: hay suficiente lugar  
*/  
program {  
  DibujarLetraF()  PasarASiguienteLetra()  
  DibujarLetraI()  PasarASiguienteLetra()  
  DibujarLetraD()  PasarASiguienteLetra()  
  DibujarLetraE()  PasarASiguienteLetra()  
  DibujarLetraL()  PasarASiguienteLetra()  
}
```

```
procedure PasarASiguienteLetra() {  
  /* PROPÓSITO: mueve el cabezal para empezar a  
    dibujar la siguiente letra  
    PRECONDICIÓN: hay suficiente lugar  
  */  
  Mover(Este) Mover(Este) Mover(Este)  
  Mover(Este) Mover(Este)  
}
```

¡Siempre pasa de la misma forma!



- En otros casos es conveniente mover el cabezal
- Por ejemplo, al dibujar líneas... **¡pero debe indicarse!**

```
procedure DibujarLíneaRojaAlEste() {  
  /* PROPÓSITO:  
    * Dibuja una línea de longitud 2 hacia el Este  
    * Deja el cabezal al final de la línea  
  PRECONDICIONES:  
    * Hay al menos 2 celdas al Este de la inicial  
  OBSERVACIÓN:  
    * La línea está hecha con bolitas rojas  
  */  
  Poner(Rojo)  Mover(Este)  Poner(Rojo)  
}
```

Se prepara para seguir
dibujando...

```
procedure DibujarCuadradoRojo() {  
  /* PROPÓSITO:  
    * dibuja un cuadrado de lado 3,  
    * hecho con bolitas rojas  
  PRECONDICIONES:  
    * hay al menos 2 celdas al Norte y  
    * 2 al Este de la celda inicial  
  */  
  DibujarLíneaRojaAlEste()  
  DibujarLíneaRojaAlNorte()  
  DibujarLíneaRojaAlOeste()  
  DibujarLíneaRojaAlSur()  
}
```



Precondiciones de procedimientos



- Las **precondiciones** de un procedimiento establecen los requerimientos para que funcione
- Si la precondición no se cumple, entonces el procedimiento puede hacer cualquier cosa
 - Pero en InPr preferimos que haga BOOM

```
procedure DibujarCuadradoRojo() {  
  /*  
    PROPÓSITO: dibuja un cuadrado de lado 3, hecho con bolitas rojas  
    PRECONDICIONES:  
      hay al menos 2 celdas al Norte y 2 al Este de la celda inicial  
  */  
  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)  
}
```



- Al **invocar** a un procedimiento, como un comando, deben garantizarse sus **precondiciones**
- Puede hacerse de 2 maneras
 - Ajustando el estado para que las cumpla
 - Aumentando las precondiciones del programa

```
/*  
  PROPÓSITO: dibuja 2 cuadrados rojos, en las esquinas SurOeste y SurEste  
  PRECONDICIONES:  
    El tablero tiene al menos 3 celdas de lado (idealmente tiene al menos 7)  
    La celda inicial es la esquina SurOeste  
*/  
program {  
  DibujarCuadradoRojo()  
  IrAlBorde(Este)  
  Mover(Oeste) // Se debe ubicar al cabezal para garantizar  
  Mover(Oeste) // la precondición de DibujarCuadradoRojo  
  DibujarCuadradoRojo()  
}
```

La precondición de esta invocación está incluida en la precondición del programa

La precondición de esta invocación está garantizada por los Mover anteriores

- Las precondiciones siempre son **condiciones** que deben contestarse **por sí o no**
- No son explicaciones, ni comentarios
- Es mejor expresarlas en términos del problema

```
procedure MoverAlAlienAlEste() {  
  /*  
    PROPÓSITO: Mueve al Alien un lugar al Este  
    PRECONDICIONES:  
      Hay un alien representado en la celda inicial  
      Hay una celda al Este de la celda inicial  
  */  
  Sacar(Verde) Mover(Este) Poner(Verde)  
}
```

¡BIEN!
Es una CONDICIÓN

¡MAL!
Es una EXPLICACIÓN

```
procedure MoverAlAlienAlEste() {  
  /*  
    PROPÓSITO: Mueve al Alien un lugar al Este  
    PRECONDICIONES:  
      El alien se representa con una bolita verde  
  */  
  Sacar(Verde) Mover(Este) Poner(Verde)  
}
```



- Más de una precondition sirve para un procedimiento
- Pero vamos a preferir la que pida menos cosas
 - Precondición más débil

```
procedure DibujarCuadradoRojo() {  
  /*  
    PROPÓSITO: dibuja un cuadrado de lado 3, hecho con bolitas rojas  
    PRECONDICIONES:  
      hay 21 celdas para cada lado de la celda inicial  
  */  
}
```


¿Debería llamar a este procedimiento si hay menos de 42 celdas?

¿Podría pedirse algo menos fuerte?




Representación de problemas usando procedimientos

- ¿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

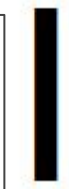
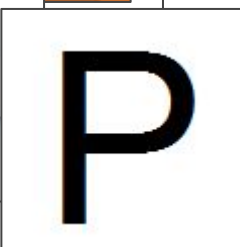
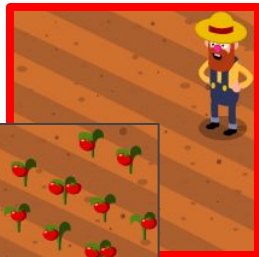
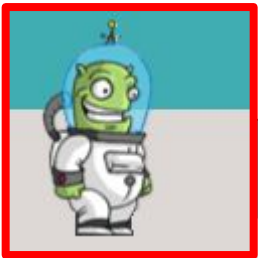
```
1 program {  
2   /*  */  
6   MoverAlienAlEste()  
7   MoverAlienAlEste()  
8   MoverAlienAlEste()  
9   TocarBotón()  
10 }
```

¡Son equivalentes!

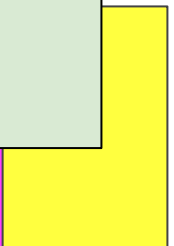
Pero en el primero sabemos en qué está pensando el programador

```
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 }
```


- Las bolitas pueden representar diferentes cosas
 - Aliens y botones
 - Letras y números
 - Partes de fotos
 - Piezas de ajedrez
 - Granjeros y tomates
 - Cartas
 - Colores
 - Plantas vs. zombies



Los dibujos ACOMPAÑAN las ideas. Pero son las bolitas las que las representan





- Las vestimentas permiten asociar estados de una celda con imágenes específicas
- Así, podemos visualizar las ideas representadas

```
- { image: Copas-07.png, when: { blue: 3, black: 207, red: 0, green: 0 } }  
- { image: Copas-10.png, when: { blue: 3, black: 210, red: 0, green: 0 } }  
- { image: Copas-11.png, when: { blue: 3, black: 211, red: 0, green: 0 } }  
- { image: Copas-12.png, when: { blue: 3, black: 212, red: 0, green: 0 } }  
- { image: Espadas-01.png, when: { blue: 3, black: 301, red: 0, green: 0 } }  
- { image: Espadas-02.png, when: { blue: 3, black: 302, red: 0, green: 0 } }  
- { image: Espadas-03.png, when: { blue: 3, black: 303, red: 0, green: 0 } }  
- { image: Espadas-04.png, when: { blue: 3, black: 304, red: 0, green: 0 } }  
- { image: Espadas-05.png, when: { blue: 3, black: 305, red: 0, green: 0 } }  
- { image: Espadas-06.png, when: { blue: 3, black: 306, red: 0, green: 0 } }  
- { image: Espadas-07.png, when: { blue: 3, black: 307, red: 0, green: 0 } }  
- { image: Espadas-10.png, when: { blue: 3, black: 310, red: 0, green: 0 } }
```

3 bolitas azules y 301 negras
representan
el ancho de espadas



- ¡Pero precisamos comandos que hablen de los elementos que queremos, y no de bolitas!
 - PROCEDIMIENTOS de representación

```
procedure PonerAlBeto() {  
  /* PROPÓSITO: poner al Beto de la celda actual  
  PRECONDICIONES:  
    * NO está el Beto en la celda actual  
  OBSERVACIONES:  
    * el Beto se representa con una bolita azul  
  */  
  Poner(Azul)  
}
```

El Beto se representa
con una bolita Azul

```
procedure SacarAlBeto() {  
  /* PROPÓSITO: sacar al Beto de la celda actual  
  PRECONDICIONES:  
    * está el Beto en la celda actual  
  OBSERVACIONES:  
    * el Beto se representa con una bolita azul  
  */  
  Sacar(Azul)  
}
```



- 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

COMANDOS

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

Mover Alien al Este

Tocar botón

programa

- Mover Alien al Este
- Mover Alien al Este
- 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

- ¡Podemos olvidarnos de cómo se representan los elementos!
 - **ABSTRACCIÓN** de los detalles

```
procedure MoverAlBetoAlEste()  
  /* PROPÓSITO: mueve al Beto  
  PRECONDICIONES:  
    * está el Beto en la celda actual  
    * no hay otras representaciones del Beto  
    en el tablero  
  */  
  SacarAlBeto()  
  Mover(Este)  
  PonerAlBeto()  
}
```

Observar cómo se expresa
el movimiento poniendo y
sacando al Beto


```
procedure PonerAlBeto() {  
  /* PROPÓSITO: poner al Beto de la celda actual  
  PRECONDICIONES:  
    * NO es  
  OBSERVACIONES:  
    * el Beto  
  */  
  Poner(Azul)  
}
```

```
procedure SacarAlBeto() {  
  /* PROPÓSITO: sacar al Beto de la celda actual  
  PRECONDICIONES:  
    * está el Beto en la celda actual  
  OBSERVACIONES:  
    * el Beto se representa con una bolita azul  
  */  
  Sacar(Azul)  
}
```





- Los procedimientos se pueden considerar como cubriendo diferentes “*niveles*”
 - a. Nivel de resolución del problema
 - b. Nivel de representación de funcionamiento
 - c. Nivel de representación básica

Nivel a.

```
procedure AvanzarYPatear() {  
  /*  */  
  MoverABetoAlEste()  
  MoverABetoAlEste()  
  PatearLaPelota()  
}
```



Observar cómo cada nivel
usa los niveles inferiores



Nivel b.

```
procedure MoverAlBetoAlEste() {  
  /*  */  
  SacarAlBeto()  Mover( Este)  PonerAlBeto()  
}
```

¡NO MEZCLAR
LOS NIVELES!

Nivel c.


```
procedure PonerAlBeto() {  
  /*  */  
  Poner( Azul)  
}
```

```
procedure SacarAlBeto() {  
  /*  */  
  Sacar( Azul)  
}
```





- ¿Cómo sería mezclar niveles?
 - Por ejemplo, usar bolitas en un nivel superior
 - O no ser consistente con el uso de los procedimientos ya definidos

¡NO MEZCLAR
LOS NIVELES!

```
procedure AvanzarYPatear() {  
  /*  */  
  MoverABetoAlEste()  
  SacarAlBeto() Mover(Este) Poner(Azul)  
  PatearLaPelota()  
}
```

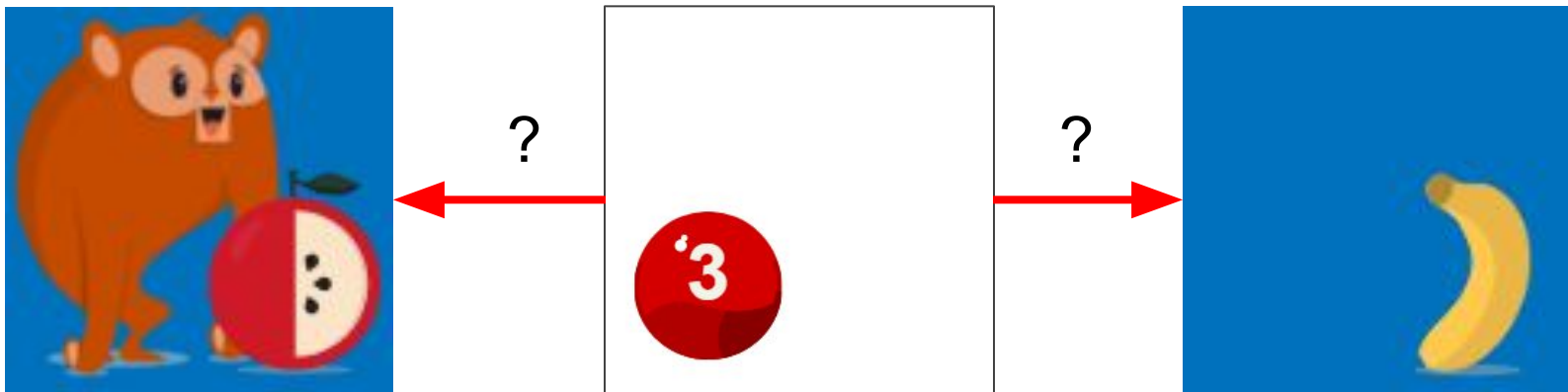
¿Por qué una vez dice
MoverABetoAlEste, y la otra no?

```
procedure MoverABetoAlEste() {  
  /*  */  
  SacarAlBeto() Mover(Este) Poner(Azul)  
}
```

```
procedure MoverABetoAlEste() {  
  /*  */  
  SacarAlBeto() Mover(Este) PonerAlBeto()  
}
```

Si dice **SacarAlBeto**, ¿por qué no
dice **PonerAlBeto**?

- ¡Al elegir la representación, debe tenerse en cuenta que no se mezclen representaciones!
 - Ej: si queremos representar, usando bolitas rojas
 - un mono con 1 bolita,
 - una manzana con 2 bolitas,
 - una banana con 3 bolitas
 - ¿Qué pasa si el mono y la manzana están en la misma celda?



¿Un mono con la manzana? ¿O una banana?



Expresar estrategia usando procedimientos

- Al resolver un problema, es mejor empezar pensando una **estrategia de solución**
 - ¿Qué cosas hay que hacer para lograr la solución?
 - **Subtareas**, tareas necesarias para llegar a la solución
 - No pensar detalles, sino cosas grandes

¿En qué pensamos al organizar un viaje a Córdoba?

Por ejemplo:

- el viaje,
- el alojamiento,
- los paseos, etc.

SUBTAREAS

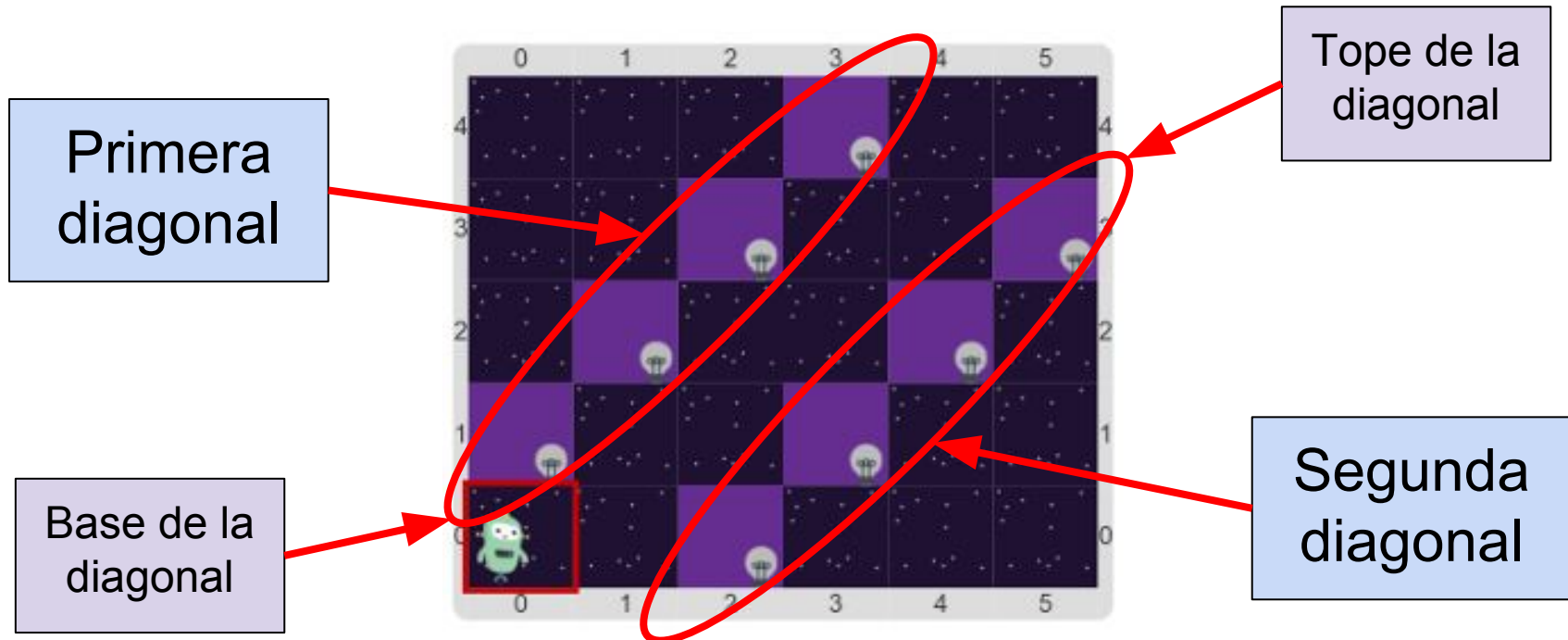


NO pensamos en

- Cuántos pasos dar para salir del aula
- Los movimientos para pararse del asiento

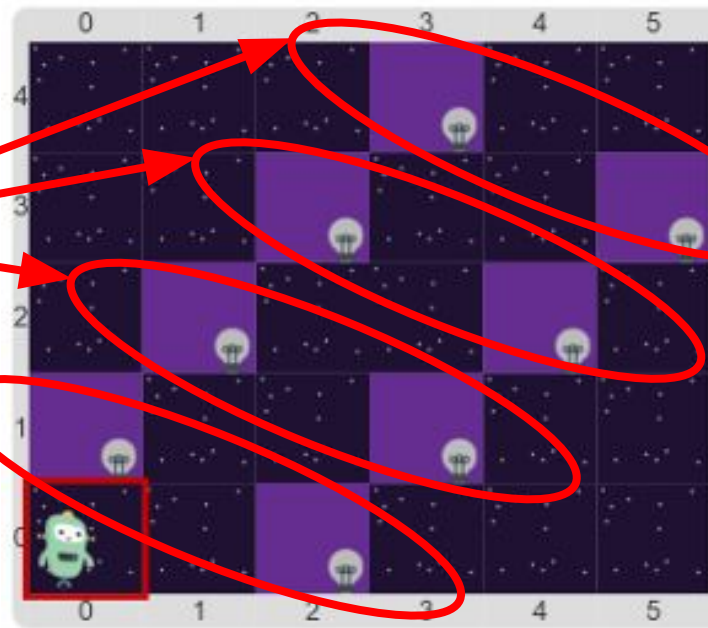
ACCIONES INDIVIDUALES

- Para pensar una **estrategia** que solucione un problema
 - Pensar una forma de dividir el problema en partes (dar nombre a las subtareas)
 - Nombrar cada una de las partes (***terminología***)



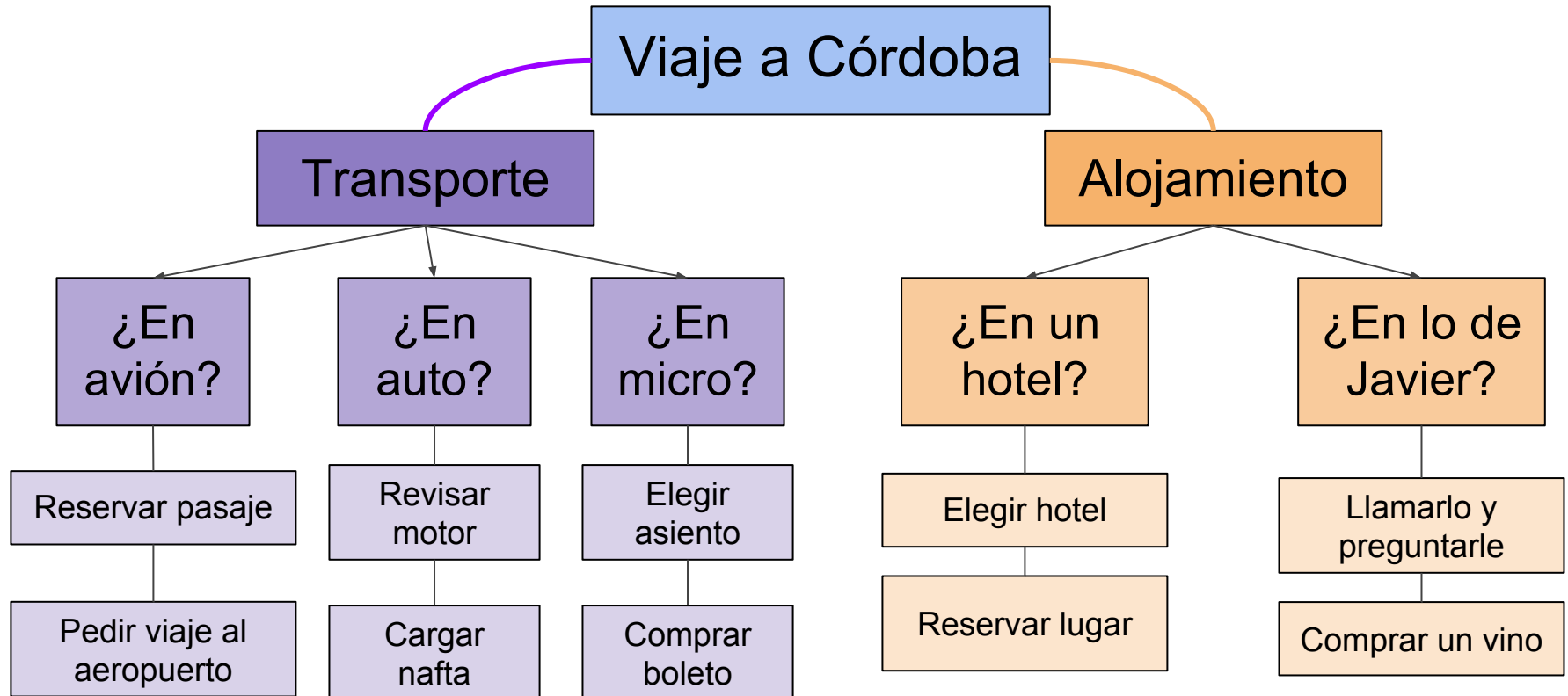
- Cada problema admite diversas estrategias de solución
- Cada estrategia implica una **forma de mirar** el problema
 - Ej: por diagonales vs. por “rieles”

“Rieles”



¡Distintas personas ven el problema de distintas maneras!


- Una vez dividida la tarea en subtarefas solucionar cada parte por separado
 - ¡Usando el mismo método!





- ¡Una subtarea puede tener subtareas!
 - Cada problema se puede dividir en partes
 - La técnica se puede aplicar en diferentes niveles


```

procedure DibujarCuadradoRojo() {
  /*  */
  DibujarLíneaRojaAlEste()
  DibujarLíneaRojaAlNorte()
  DibujarLíneaRojaAlOeste()
  DibujarLíneaRojaAlSur()
}

```


versus

```

procedure DibujarLíneaRojaAlSur() {
  procedure DibujarLíneaRojaAlOeste() {
    procedure DibujarLíneaRojaAlNorte() {
      procedure DibujarLíneaRojaAlEste() {
        /*  */
        Poner(Rojo)  Mover(Este)
        Poner(Rojo)  Mover(Este)
      }
    }
  }
}

```


```

procedure DibujarCuadradoRojo() {
  /*  */
  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)
}


```




- El código debe expresar la estrategia elegida
 - ¡Recordar poner buenos nombres a los procedimientos!

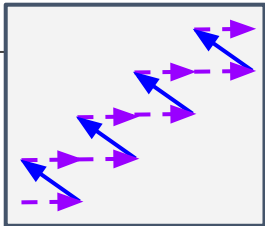
```
procedure EncenderTodasLasLuces() {  
    /*  */  
    IrALaBaseDeLaPrimeraDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
    IrALaBaseDeLaSegundaDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```


Comparar ambos procedimientos
¡Son equivalentes!
¿Pero comunican lo mismo?

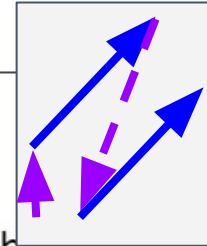
```
procedure Resolver() {  
    /*  */  
    Ir1()  
    Subir()  
    Ir2()  
    Subir()  
}
```



- Cada problema admite diversas estrategias de solución
- No todas implican el mismo esfuerzo
 - ¿Cómo elegir cuál estrategia?
 - ¿Mejor poco código, o mejor menos movimientos?

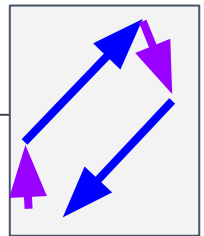
```
procedure EncenderTodasLasLuces() {  
  /**/  
  EncenderUnRiel()  
  EncenderUnRiel()  
  EncenderUnRiel()  
  EncenderUnRiel()  
}
```



```
procedure EncenderTodasLasLuces() {  
  /**/  
  IrALaBaseDeLaPrimeraDiagonal()  
  EncenderLucesDeUnaDiagonalHaciaArriba()  
  IrALaBaseDeLaSegundaDiagonal()  
  EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```




```
procedure EncenderTodasLasLuces() {  
  /**/  
  IrALaBaseDeLaPrimeraDiagonal()  
  EncenderLucesDeUnaDiagonalHaciaArriba()  
  IrAlTopeDeLaSegundaDiagonal()  
  EncenderLucesDeUnaDiagonalHaciaAbajo()  
}
```

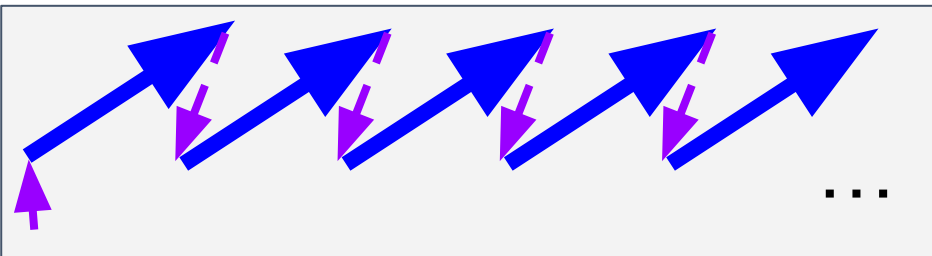


Tres estrategias
distintas

- Algunas estrategias son más fáciles de generalizar
 - La uniformidad es una aliada
 - Detectar “patrones” dentro de un problema es una parte importante de armar la estrategia


```
procedure EncenderTodasLasLuces() {  
    /*  */  
    IrAlaBaseDeLaPrimeraDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
    repeat(6) {  
        IrAlaBaseDeLaSiguienteDiagonal()  
        EncenderLucesDeUnaDiagonalHaciaArriba()  
    }  
}
```

¿Cómo hacer con las
otras estrategias?
¡Es más complicado!






- Cada subtarea se expresa mediante un procedimiento
 - Las tareas deberían ser de “nivel” similar

```
procedure EncenderTodasLasLuces() {  
    /*  */  
    IrALaBaseDeLaPrimeraDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
    IrALaBaseDeLaSegundaDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```

OK

No es
bueno
mezclar
niveles

Comparar


```
procedure EncenderTodasLasLuces() {  
    /*  */  
    MoverALuchoAl_ (Norte)  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
    IrALaBaseDeLaSegundaDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```


FEO




- Cada subtarea se expresa mediante un procedimiento
 - Cada procedimiento debería expresar a UNA tarea
 - ¡No mezclar partes de diferentes tareas!

Tampoco es
bueno mezclar
tareas

```
procedure EncenderTodasLasLuces() {  
  /*  */  
  EncenderLucesDePrimeraDiagonalHaciaArriba()  
  IrALaBaseDeLaSegundaDiagonal()  
  EncenderLucesDeSegundaDiagonalHaciaArriba()  
}
```

```
procedure EncenderLucesDePrimeraDiag  
  /*  */  
  MoverALuchoA1_(Norte)  
  EncenderLuz()  
  MoverALuchoA1NorteEste()
```


FEO

```
procedure EncenderLucesDeSegundaDiag  
  /*  */  
  EncenderLuz()  
  MoverALuchoA1NorEste()
```

¡Dos
procedimientos
diferentes por
UN comando
fuera de lugar!




- Cada subtarea se expresa mediante un procedimiento
 - Recordar poner buenos nombres
 - ¡No mezclar partes de diferentes tareas!

```
procedure EncenderTodasLasLuces() {  
  /*  */  
  IrALaBaseDeLaPrimeraDiagonal() OK  
  EncenderLucesDeUnaDiagonalHaciaArriba()  
  IrALaBaseDeLaSegundaDiagonal()  
  EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```

No es
bueno
mezclar
niveles


Comparar


```
procedure EncenderTodasLasLuces() {  
  /*  */  
  MoverALuchoAl_ (Norte) FEO  
  EncenderLucesDeUnaDiagonalHaciaArriba()  
  IrALaBaseDeLaSegundaDiagonal()  
  EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```





- Puede haber diferentes criterios sobre cómo agrupar
 - **LavarLosPlatos**, ¿es parte de **HacerDeComer**?
 - ¿Y **AccionarLaDescargaDelInodoro**?
¿Es parte de la tarea de **IrAlBaño**? ¿O lo debe hacer otro?
(Similar a **BorrarElPizarrón**... ¿Quién lo debe hacer?)

¿Dónde poner el **VolverALaBase**?

```
procedure EncenderTodasLasLuces() {  
    /*  */  
    IrALaBaseDeLaPrimeraDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
    IrDeBaseABase()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```

```
procedure EncenderTodasLasLuces() {  
    /*  */  
    IrALaBaseDeLaPrimeraDiagonal()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
    VolverALaBaseDeLaDiagonal()  
    IrDeBaseABase()  
    EncenderLucesDeUnaDiagonalHaciaArriba()  
}
```

¿Acá?

```
procedure EncenderLucesDeUnaDiagonalHaciaArriba() {  
    /*  */  
    EncenderLaDiagonalSubiendo()  
    VolverALaBaseDeLaDiagonal()  
}
```

¿O acá?

- Recordar que cada procedimiento debe tener su propio contrato correctamente documentado
 - Nombre, propósito, precondiciones
 - Es mejor expresar el contrato en términos de la subtarea y no del problema general

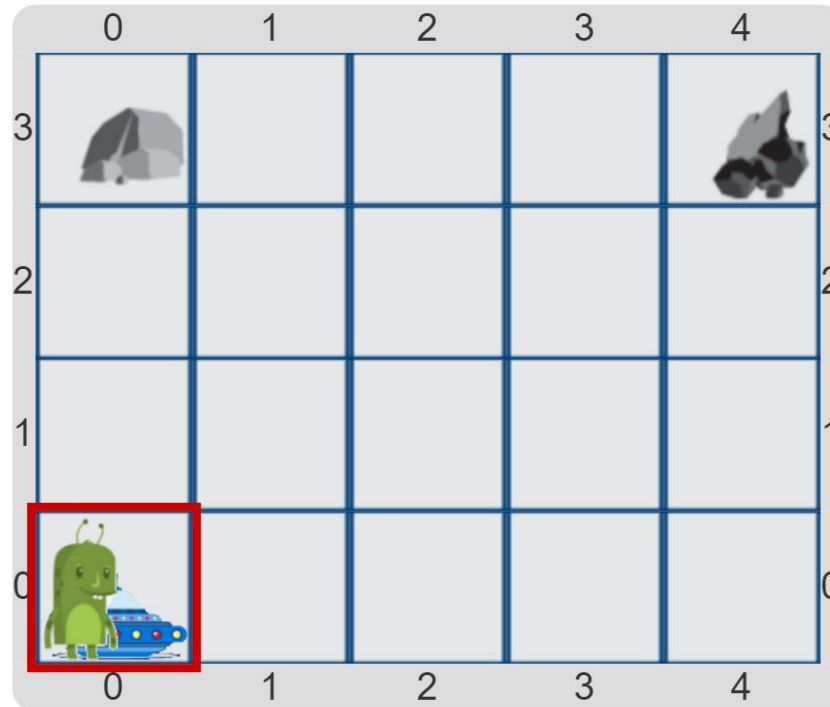
```
procedure DibujarLíneaRojaAlEste() {  
  /* PROPÓSITO:  
    * dibuja una línea roja con bolitas  
  PRECONDICIONES:  
    * hay suficiente lugar  
  */  
  Poner(Rojo) Mover(Este)  
  Poner(Rojo) Mover(Este)  
}
```

```
procedure DibujarLadoSurDelCuadrado() {  
  /*  
    PROPÓSITO: dibuja un lado del cuadrado  
  PRECONDICIONES:  
    hay suficiente lugar  
  */  
  Poner(Rojo) Mover(Este)  
  Poner(Rojo) Mover(Este)  
}
```

¿Cuál de las dos opciones es mejor, y por qué?



- EJERCICIO: “Reparando la nave”
 - Ayudar al marciano a reparar su nave
 - Precisa 3 unidades de carbón y 3 de hierro
 - Solo puede llevar una por vez...





● Procedimientos primitivos de “Reparando la nave”

- **MoverMarcianoAl_**
- **PonerCarbónEnLaNave**
- **PonerHierroEnLaNave**
- **VolverACasa**
- **AgarrarCarbón**
- **AgarrarHierro**

```
/* PROPÓSITO:  
  * mover al marciano un lugar en la  
  | dirección dada dejando el cabezal  
  | sobre el marciano  
PRECONDICIONES  
  * el marciano está en la celda actual  
  * hay una celda en la dirección dada  
*/
```

```
/* PROPÓSITO:  
  * hacer que el marciano se vaya en  
  | la nave  
PRECONDICIONES  
  * el marciano y la nave  
  | están en la celda actual  
  * la nave está completamente reparada  
*/
```

```
/* PROPÓSITO:  
  * dejar una unidad de carbón en la  
  | nave  
PRECONDICIONES  
  * el marciano y la nave están en la  
  | celda actual  
  * el marciano tiene un carbón  
*/
```

```
/* PROPÓSITO:  
  * tomar una unidad de carbón  
  | de la celda actual  
PRECONDICIONES  
  * el marciano está en la celda actual  
  | y hay al menos un carbón en ella  
  * el marciano no está cargando nada  
*/
```



Cierre

- **Procedimientos**

- Definición de nuevos comandos
 - Brindan ***abstracción*** para los comandos
- Permiten **expresar** diversas cosas
 - ***Representación*** de información y primitivas del dominio del problema a solucionar
 - ***Estrategia*** de solución y subtarefas
- Aportan legibilidad, claridad y modificabilidad
- Pueden ser reutilizados muchas veces