

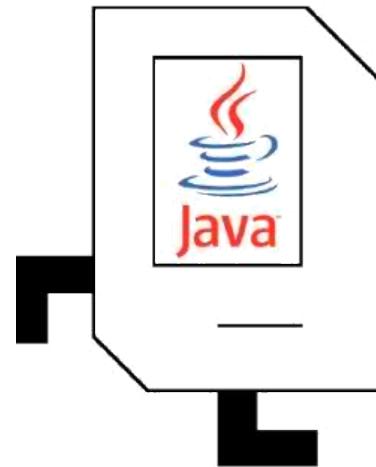


Universidad  
de los Andes

# Flujo de control

# El plan para hoy

- Repaso
- Flujo de control
  - Ciclos while
  - Instrucciones condicionales



# El plan para hoy

- Repaso 
- Flujo de control
  - Ciclos while
  - Instrucciones condicionales

# Definiendo nuevos comandos

Podemos definir nuevos comandos (métodos) para descomponer nuestros programas—¡no solo los de Karel!

```
private void nombre( ) {  
    instrucción;  
    instrucción;  
    ...  
}
```

Por ejemplo:

```
private void girarDerecha( ) {  
    girarIzquierda( );  
    girarIzquierda( );  
    girarIzquierda( );  
}
```

# Flujo de control: ciclos for

Anatomía de un ciclo for:

```
for (int i = 0; i < contar; i++) {  
    instrucción;  
    instrucción;  
    ...  
}
```

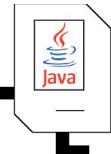
Repite las instrucciones en el cuerpo del ciclo **contar** veces.

Por ejemplo:

```
// Karel gira a la derecha  
for (int i = 0; i < 3; i++) {  
    girarIzquierda();  
}
```

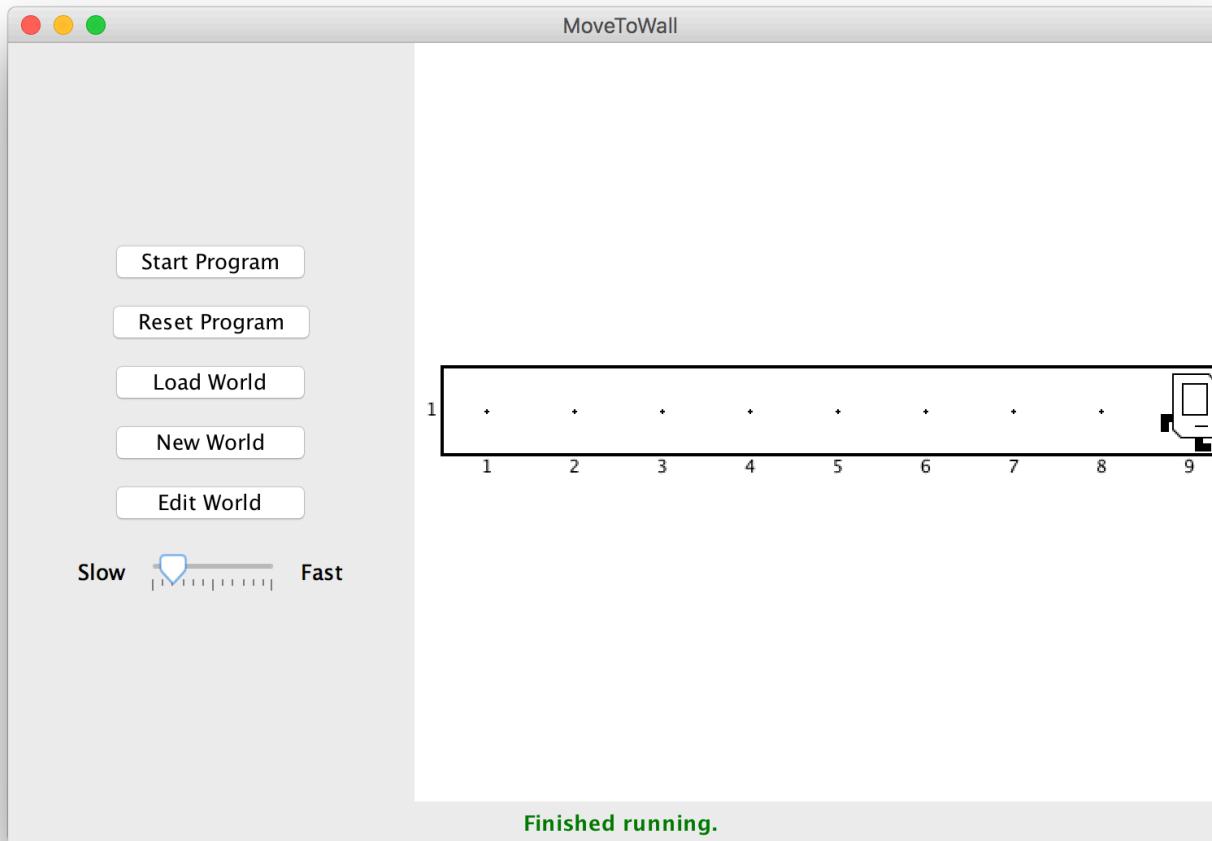
# El plan para hoy

- Repaso
- Flujo de control
  - Ciclos while
  - Instrucciones condicionales



# Flujo de control: ciclos while

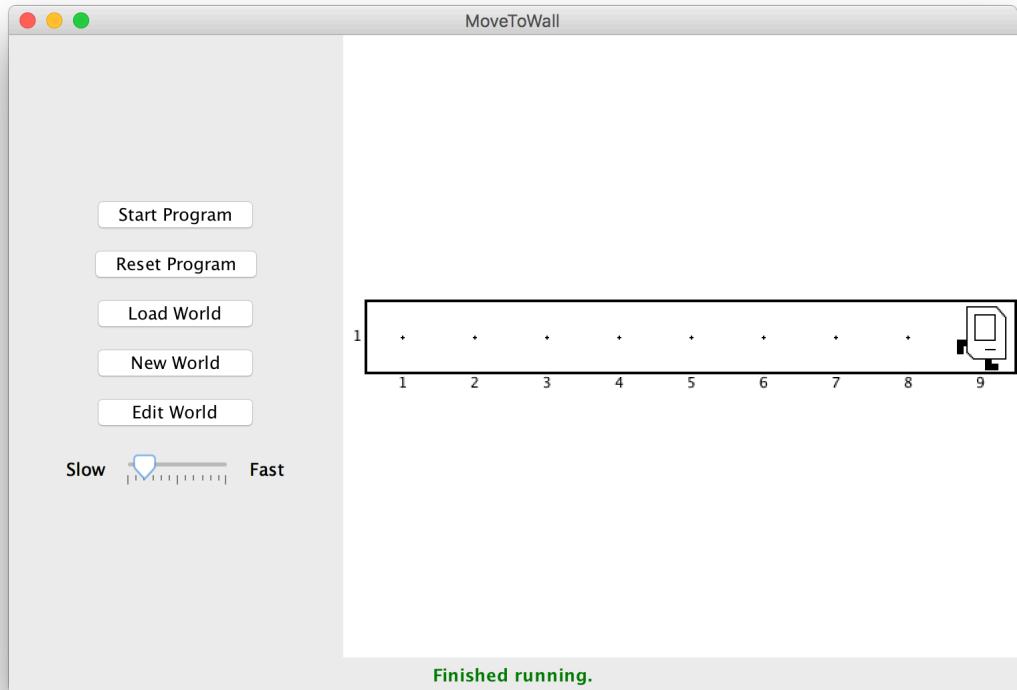
- Quiero que Karel se mueva hasta llegar a la pared. ¿Cómo lo hago?



# Flujo de control: ciclos while

Podríamos decir:

```
moverse();  
moverse();  
moverse();  
moverse();  
moverse();  
...  
...
```



¡Esto es muy repetitivo! Además, tal vez no sepamos el ancho del mundo. Queremos un programa *general*.

# Flujo de control: ciclos while

En cambio, usa un ciclo **while**:

```
while (condición) {  
    statement;  
    statement;  
    ...  
}
```

Repite las instrucciones del cuerpo hasta que la *condición* ya no sea verdadera.

**MUY IMPORTANTE:** durante cada vuelta del ciclo, Karel ejecuta **todas las instrucciones** del cuerpo y **entonces** verifica la condición de nuevo.

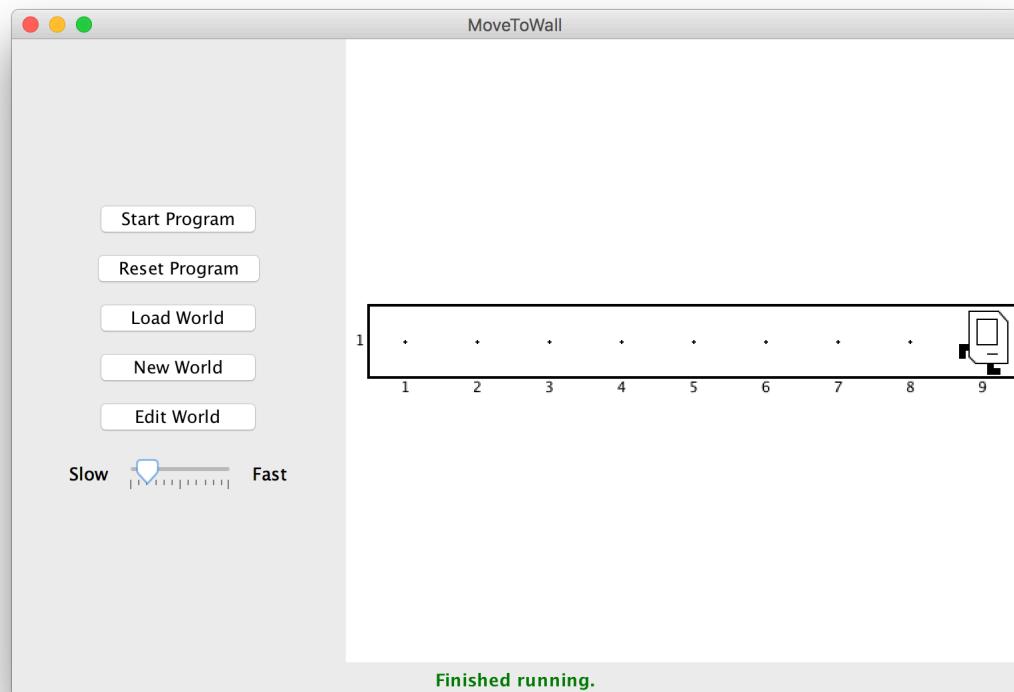
# Condiciones posibles

<i>Condición</i>	<i>Opuesto</i>	<i>Qué verifica</i>
<code>frenteDespejado()</code>	<code>frenteBloqueado()</code>	¿Hay una pared enfrente de Karel?
<code>izquierdaDespejada()</code>	<code>izquierdaBloqueada()</code>	¿Hay una pared a la izquierda de Karel?
<code>derechaDespejada()</code>	<code>derechaBloqueada()</code>	¿Hay una pared a la derecha de Karel?
<code>conosPresentes()</code>	<code>conosAusentes()</code>	¿Hay conos en esta esquina?
<code>bolsaConConos()</code>	<code>bolsaSinConos()</code>	¿Hay conos en la bolsa de Karel?
<code>rumboNorte()</code>	<code>sinRumboNorte()</code>	¿Está Karel orientada hacia el norte?
<code>rumboEste()</code>	<code>sinRumboEste()</code>	¿Está Karel orientada hacia el este?
<code>rumboSur()</code>	<code>sinRumboSur()</code>	¿Está Karel orientada hacia el sur?
<code>rumboOeste()</code>	<code>sinRumboOeste()</code>	¿Está Karel orientada hacia el oeste?

# Flujo de control: ciclos while

La sintaxis de un ciclo **while**:

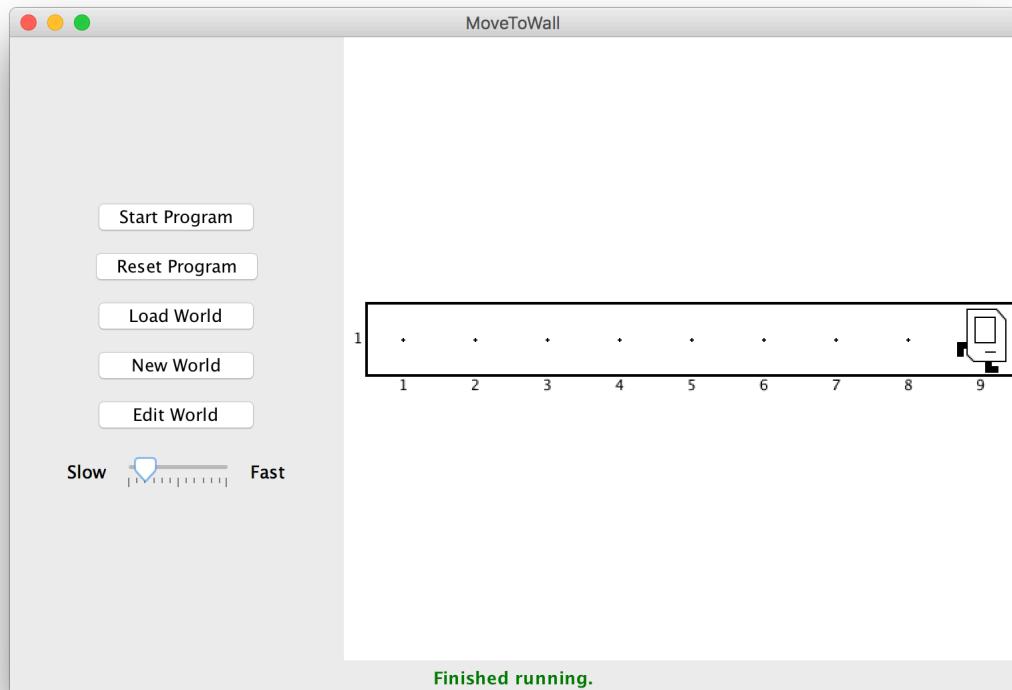
```
while (condición) {  
    instrucción;  
    instrucción;  
    ...  
}
```



# Flujo de control: ciclos while

Ahora podemos decir:

```
while (frenteDespejado()) {  
    moverse();  
}
```



Es menos repetitivo y funciona en un mundo de *cualquier* tamaño

# Flujo de control: ciclos while

Los ciclos **while** también pueden tener condiciones *múltiples*:

```
// "y"  
while (frenteDespejado() && conosPresentes()) {  
    ...  
}
```

```
// "o"  
while (rumboEste() || derechaDespejada()) {  
    ...  
}
```

# Flujo de control: ciclos while

ciclos **while** también pueden tener condiciones *múltiples*:

```
// "y"  
while (frenteDespejado() && conosPresentes()) {  
    ...  
}  
  
// "o"  
while (rumboEste() || derechaDespejada()) {  
    ...  
}
```

# Flujo de control: ciclos while

ciclos **while** también pueden tener condiciones *múltiples*:

```
// "y"  
while (frenteDespejado() && conosPresentes()) {  
    ...  
}  
  
// "o"  
while (rumboEste() || derechaDespejada()) {  
    ...  
}
```

# Resumen de ciclos

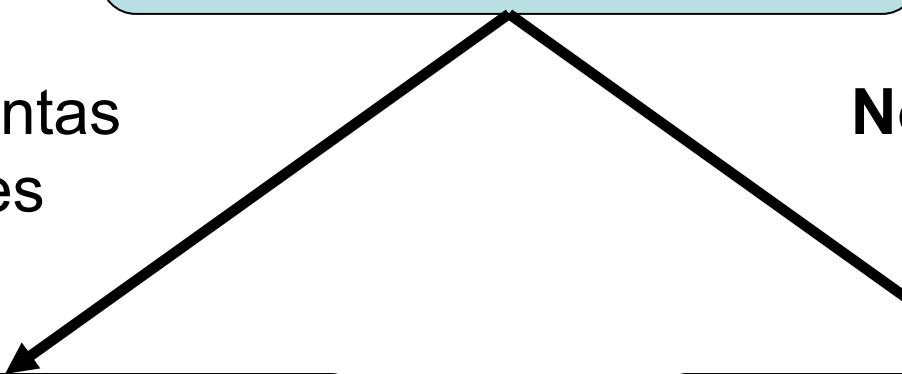
¡Quiero que Karel repita  
unos comandos!

Sé cuantas  
veces

ciclo for

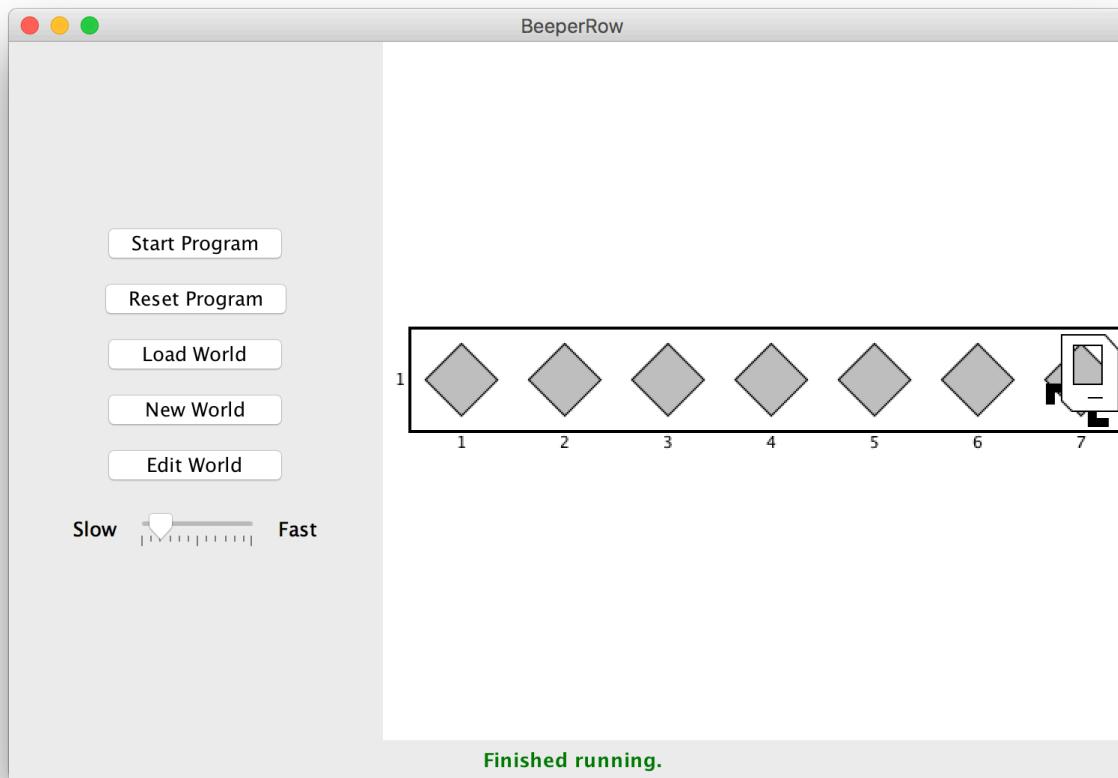
No sé cuantas  
veces

ciclo while



# Fila de conos

- Quiero que Karel llene una fila con conos. ¿Cómo lo hago?



# Fila de conos

*Demostración*

# El problema de la cerca



*¡8 segmentos de cerca, pero hay 9 postes!*

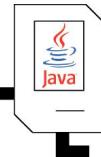
# Estructura de la cerca

Útil cuando quieres hacer un ciclo con varias instrucciones, pero quieres hacer una parte de ese grupo una vez más.

```
ponerCono();           // poste  
while (frenteDespejado()) {  
    moverse();          // cerca      Opción 1  
    ponerCono();         // poste  
}  
  
while (frenteDespejado()) {  
    ponerCono();         // poste  
    moverse();          // cerca      Opción 2  
}  
ponerCono();           // poste
```

# El plan para hoy

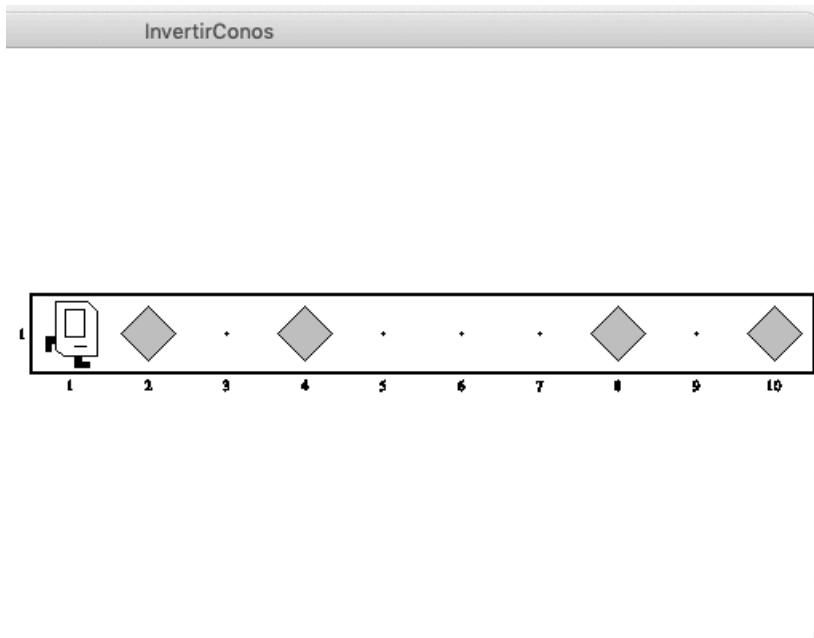
- Repaso
- Flujo de control
  - Ciclos while
  - Instrucciones condicionales



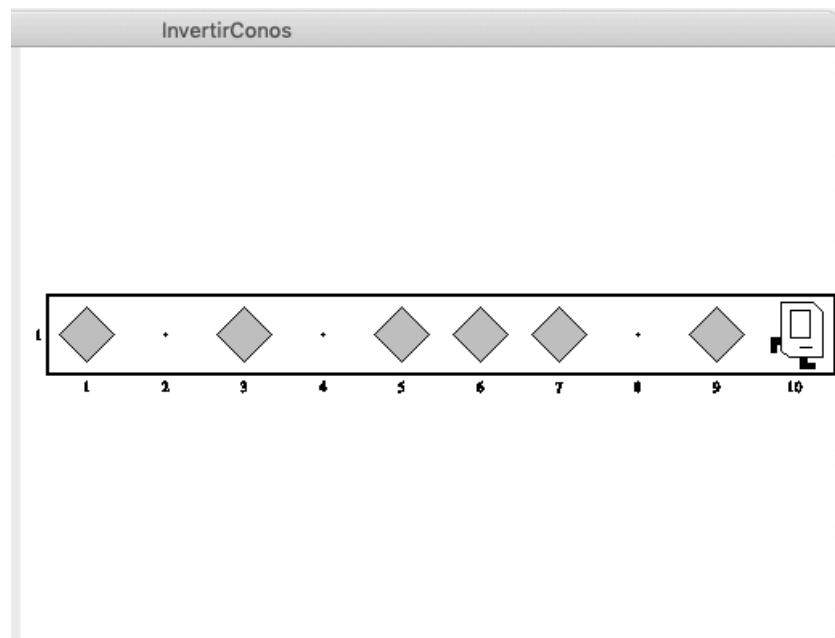
# Instrucciones condicionales

- Quiero que Karel ponga conos a lo largo de la fila, pero si ya hay un cono en una esquina, debería recogerlo en vez de poner uno nuevo. ¿Cómo lo hago?

Antes



Después



# Instrucciones condicionales

Para ejecutar una instrucción condicional, usa **if**:

```
if (condición) {  
    instrucción;  
    instrucción;  
    ...  
}
```

Ejecuta las instrucciones en el cuerpo *una vez si la condición es verdadera*.

# Instrucciones condicionales

También puedes incluir una instrucción **else**:

```
if (condición) {  
    instrucción;  
    instrucción;  
    ...  
} else {  
    instrucción;  
    instrucción;  
    ...  
}
```

Ejecuta el primer grupo de instrucciones si la **condición** es verdadera; ejecuta el segundo grupo si no.

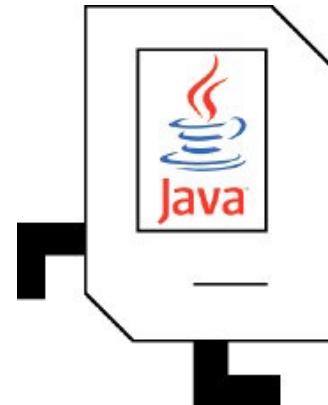
# Instrucciones condicionales

Ahora podemos decir:

```
while (frenteDespejado()) {  
    if (conosPresentes()) {  
        recogerCono();  
    } else {  
        ponerCono();  
    }  
    moverse();  
}  
if (conosPresentes()) {  
    recogerCono();  
} else {  
    ponerCono();  
}
```

# Resumen

- Repaso
- Flujo de control
  - Ciclos while
  - Instrucciones condicionales

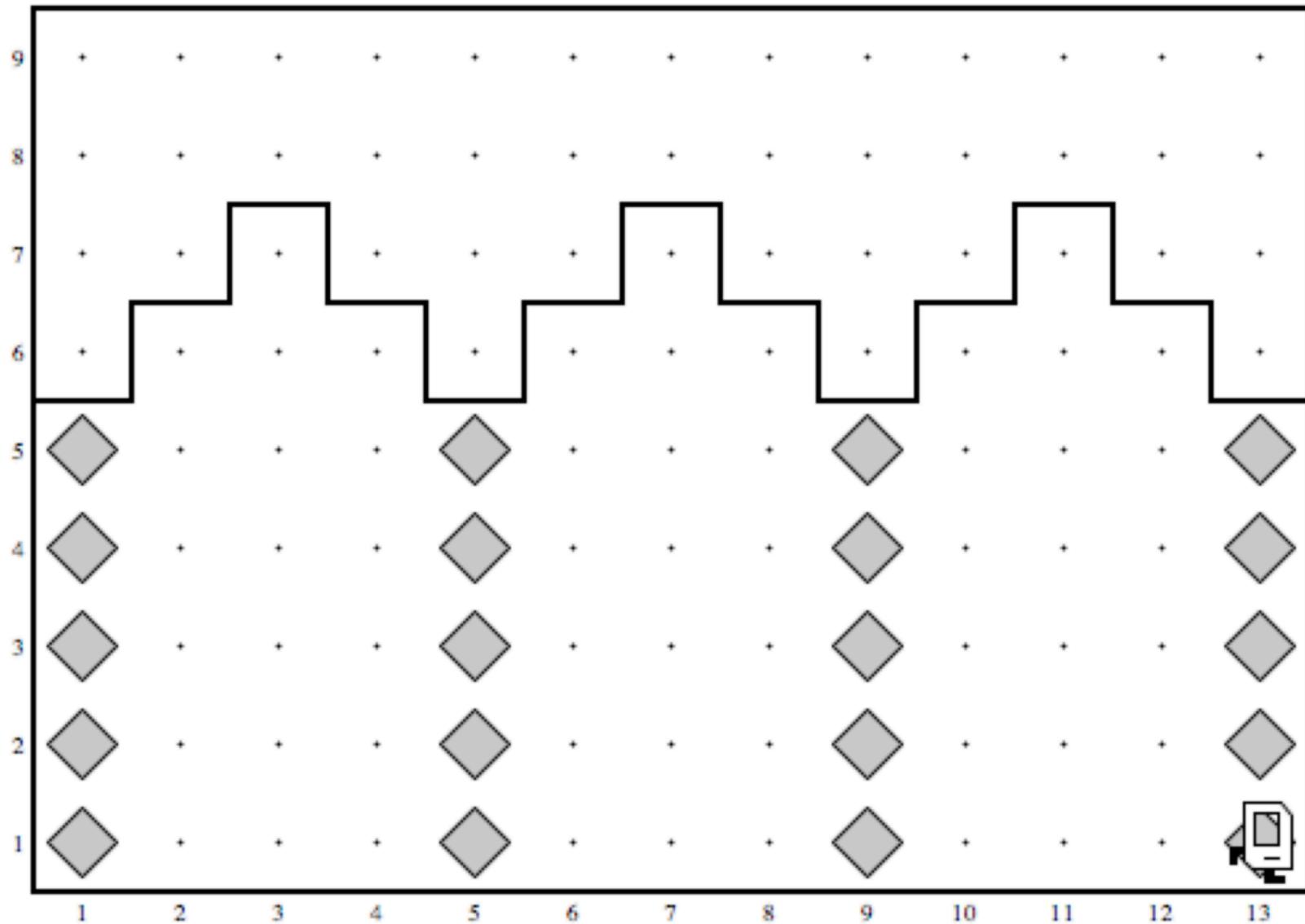


**La próxima clase:** Variables

# Práctica

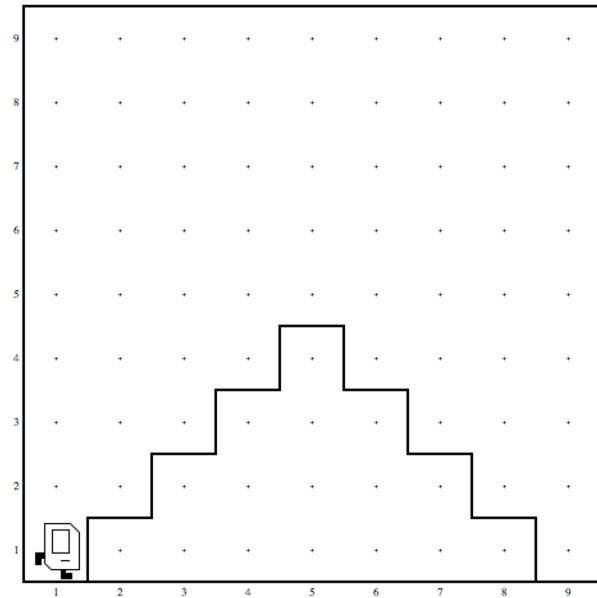
*Tareas por la tarde*

# 1. Construir Palacio Nariño

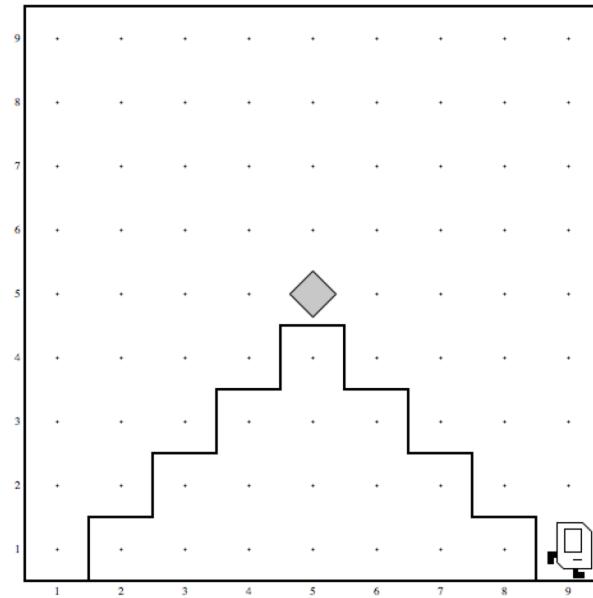


# 2. Montaña Karel

Antes



Después



### 3. Pintor Aleatorio (Bonus)

