

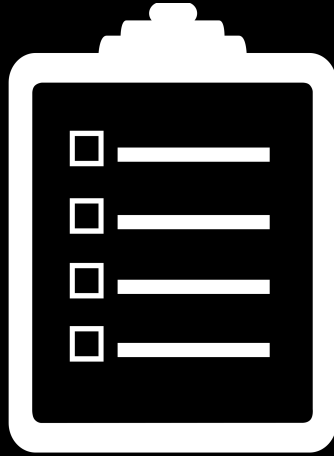
CS Bridge

Flujo de Control



Universidad de
los Andes

El plan del día



Repaso

El Bucle for

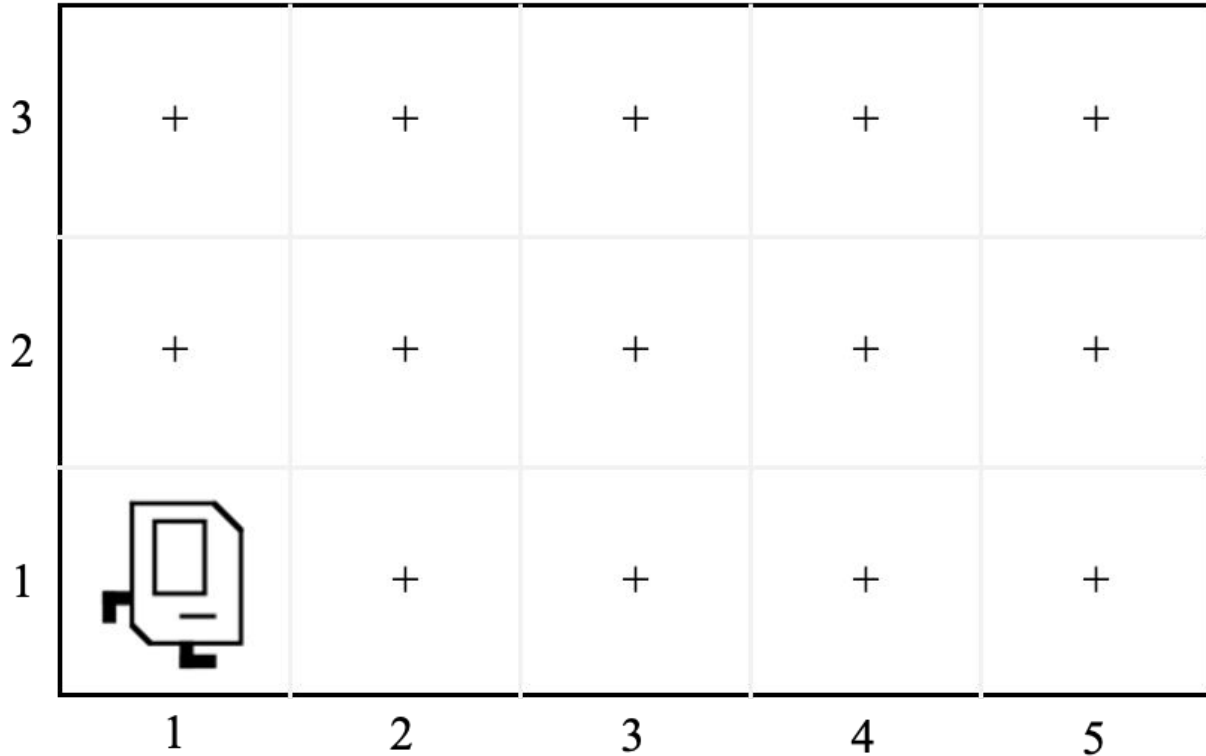
El Bucle while

El problema de la cerca


Instrucciones condicionales

Repaso


El mundo de Karel



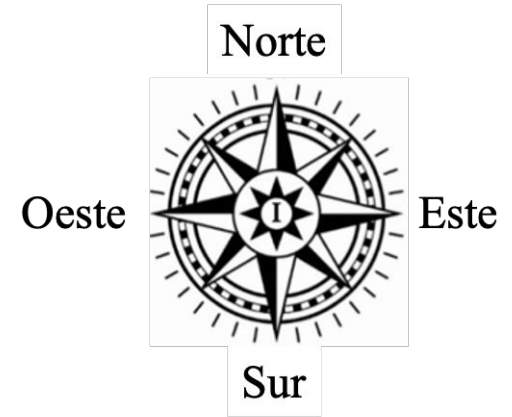
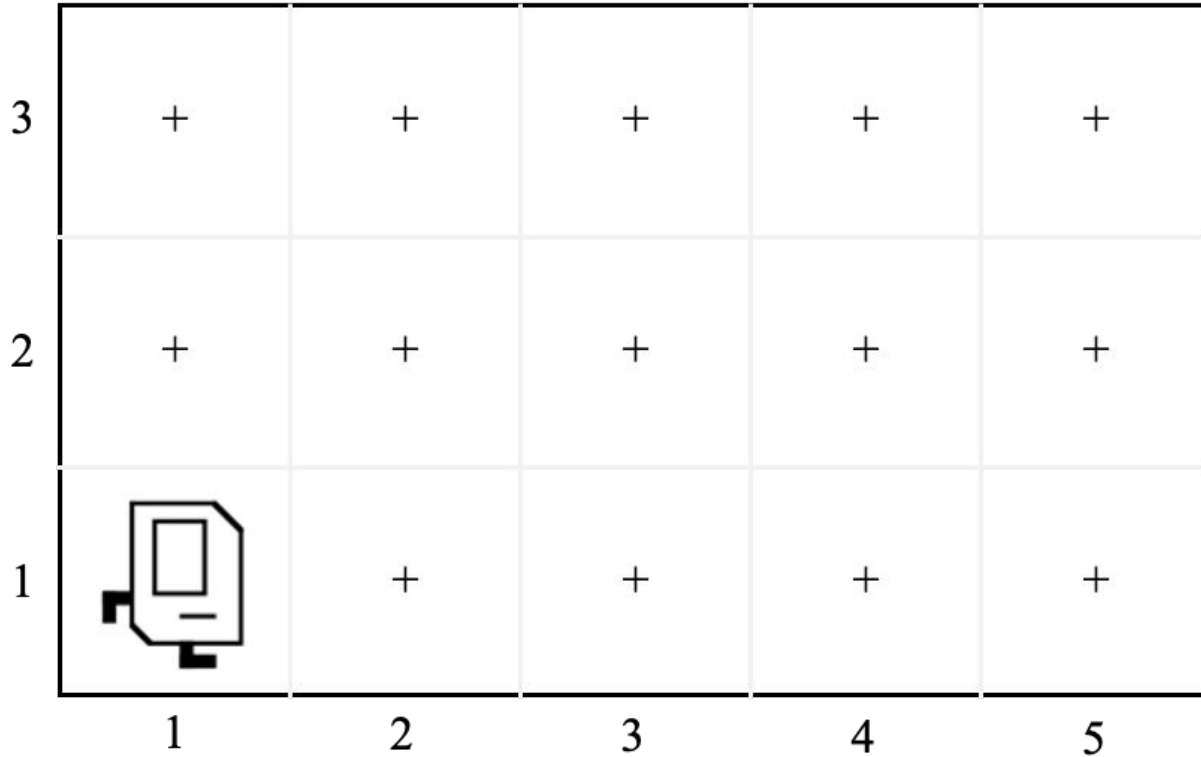
Filas

3	+	+	+	+	+
2	+	+	+	+	+
1		+	+	+	+
	1	2	3	4	5

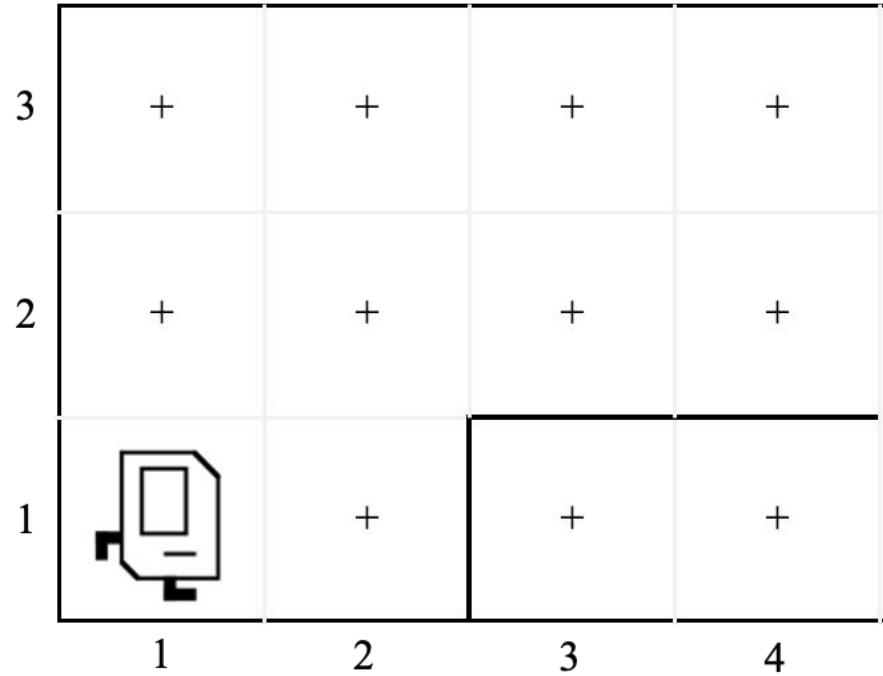
Columnas

3	+	+	+	+
2	+	+	+	+
1		+	+	+
	1	2	3	4

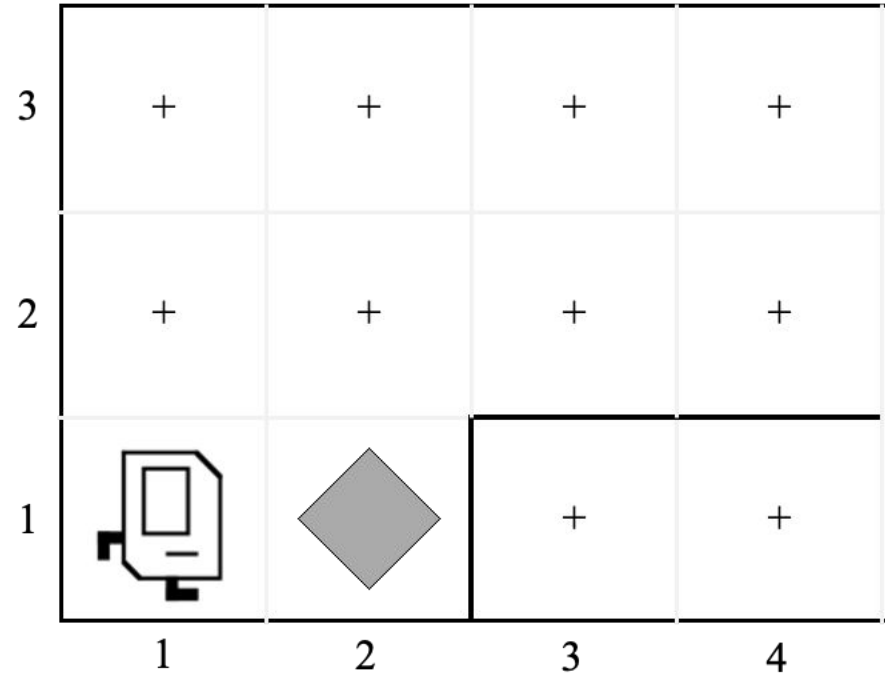
El mundo del Karel



Paredes



Conos



Entiende cuatros comandos



`move()`

`girar_izquierda()`

`recoger_cono()`

`poner_cono()`

Definición de un función

```
def nombre_de_funcion():  
    instrucción  
    instrucción  
    ...
```

Por ejemplo

```
def girar_derecha():  
    girar_izquierda()  
    girar_izquierda()  
    girar_izquierda()
```

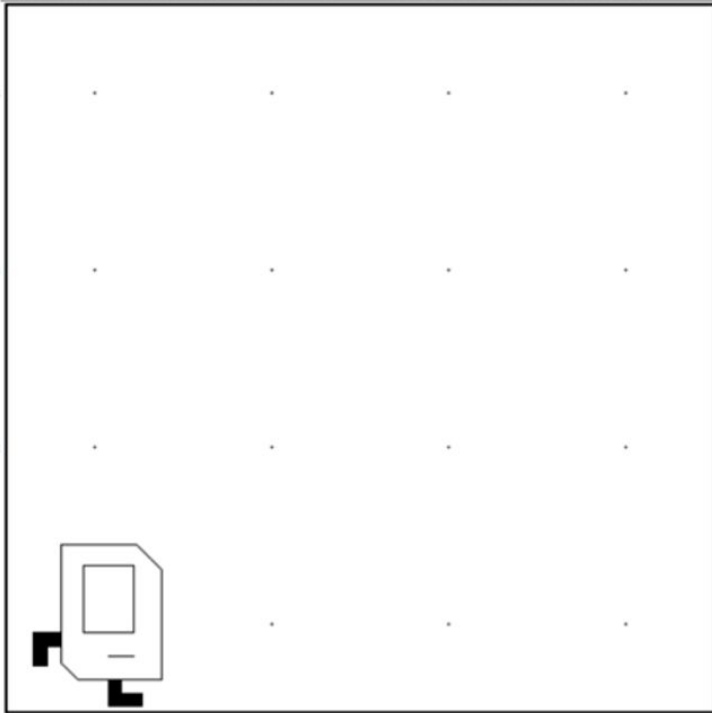
Anatomía de un programa

```
1.  def main():
2.      move()
3.      recoger_ono()
4.      move()
5.      girar_izquierda()
6.      move()
7.      girar_derecha()
8.      move()
9.      poner_cono()
10.     move()
11.
12.  def girar_derecha():
13.      girar_izquierda()
14.      girar_izquierda()
15.      girar_izquierda()
16.
```

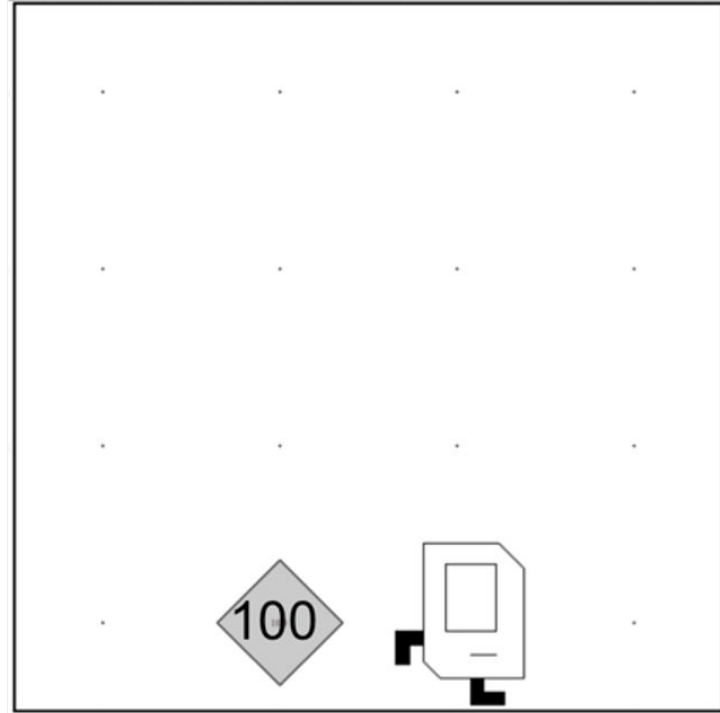
El bucle **for**

¿Poner 100 conos?

Antes



Después



¿Poner 100 conos?

Podríamos decir:

```
1.  def main():  
2.      move()   
3.      poner_cono()  
4.      poner_cono()  
5.      poner_cono()  
6.      poner_cono()  
7.      ...  
8.      poner_cono()  
9.      move()  
10.
```


¿Poner 100 conos?

Podríamos decir:

```
1. def main():  
2.     moverse()  
3.     poner_cono()  
4.     poner_cono()  
5.     poner_cono()  
6.     poner_cono()  
7.     ...  
8.     poner_cono()  
9.     moverse()  
10.
```

Pero, es muy
repetitivo.

Además, es difícil
generalizar el
programa.

Hay una solución?

El bucle **for**

El bucle `for`

```
for i in range(numero):  
    instrucción  
    instrucción  
    ...
```

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

Poner 100 conos!

Ahora podríamos decir:

```
1. def main():  
2.     move()   
3.     for i in range(100):  
4.         poner_cono()  
5.     move()  
6.
```

Poner 100 conos!

Ahora podríamos decir:

```
1. def main():  
2.     move()   
3.     for i in range(100):  
4.         poner_cono()  
5.     move()  
6.
```

Es menos **repetitivo**.
y es más fácil
cambiarlo a 1000
conos o 25 conos.

Poner 100 conos!

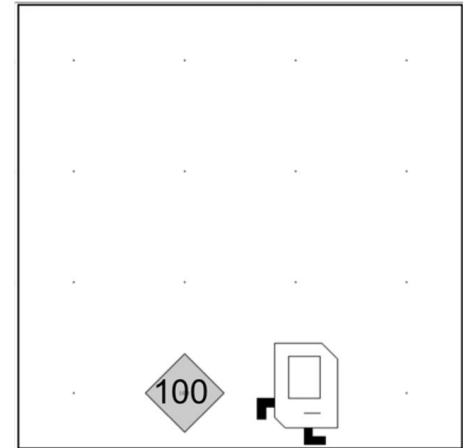
Ahora podríamos decir:

```
1. def main():  
2.     move()   
3.     for i in range(100):  
4.         poner_cono()  
5.     move()  
6.
```

Poner 100 conos!

Ahora podríamos decir:

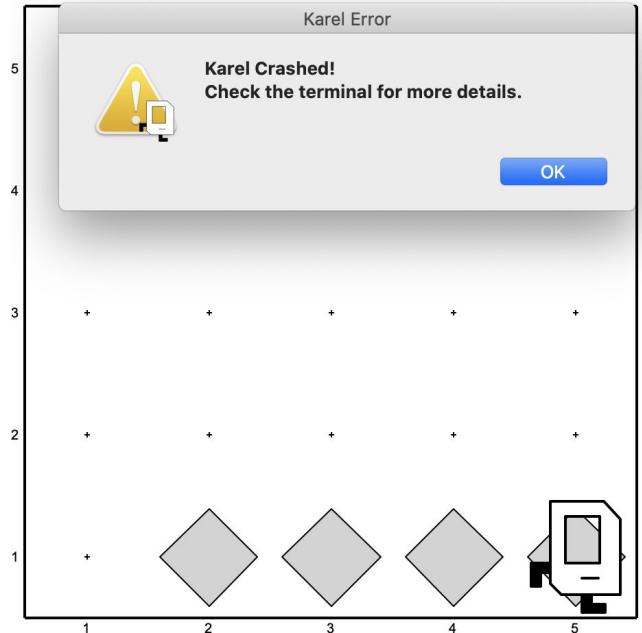
```
1. def main():  
2.     move()   
3.     for i in range(100):  
4.         poner_cono()  
5.     move()  
6.
```



Poner 100 conos!

Ahora podríamos decir:

```
1. def main():  
2.     move()   
3.     for i in range(100):  
4.         poner_cono()  
5.         move()  
6.
```



El bucle **for**: mas ejemplos

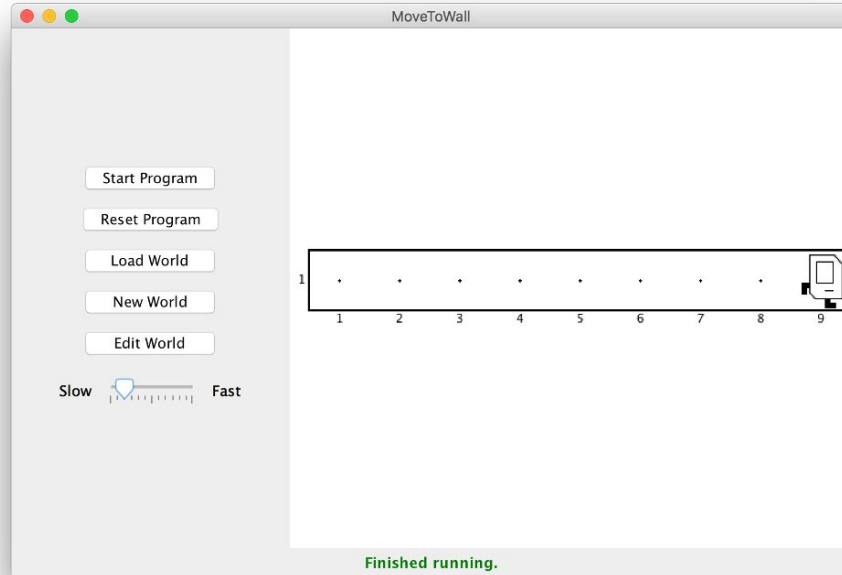
```
1. def girar_derecha():  
2.     for i in range(3):  
3.         girar_izquierda()  
4.
```

```
1. def dar_una_vuelta():  
2.     for i in range(3):  
3.         moverse()  
4.         girar_izquierda()  
5.
```

El bucle **while**

¿Hasta la pared?

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



¿Hasta la pared? (Intento 1)

Podríamos decir:

```
1.  def main():  
2.      move()   
3.      move()   
4.      move()   
5.      move()   
6.      ...  
7.      move()   
8.
```

¿Hasta la pared? (Intento 2)

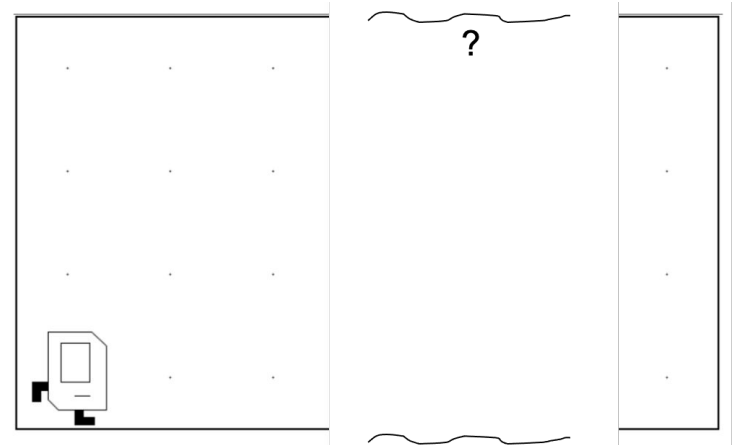
Podríamos decir:

```
1. def main():  
2.     for i in range(??):  
3.         move()se()  
4.
```

¿Hasta la pared? (Intento 2)

Podríamos decir:

```
1. def main():  
2.     for i in range(??):  
3.         moveerse()  
4.
```



Hay una solución?

El bucle `while`

El bucle *while*

while *condición*:

instrucción

instrucción

...

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

Condiciones posibles

<i>Condición</i>	<i>Opuesto</i>	<i>Qué verifica</i>
frente_despejado()	frente_bloqueado()	¿Hay una pared enfrente de Karel?
izquierda_despejada()	izquierda_bloqueada()	¿Hay una pared a la izquierda de Karel?
derecha_despejada()	derecha_bloqueada()	¿Hay una pared a la derecha de Karel?
conos_presentes()	conos_ausentes()	¿Hay conos en esta esquina?
rumbo_norte()	sin_rumbo_norte()	¿Está Karel orientada hacia el norte?
rumbo_este()	sin_rumbo_este()	¿Está Karel orientada hacia el este?
rumbo_sur()	sin_rumbo_sur()	¿Está Karel orientada hacia el sur?
rumbo_oeste()	sin_rumbo_oeste()	¿Está Karel orientada hacia el oeste?

Hasta la pared!

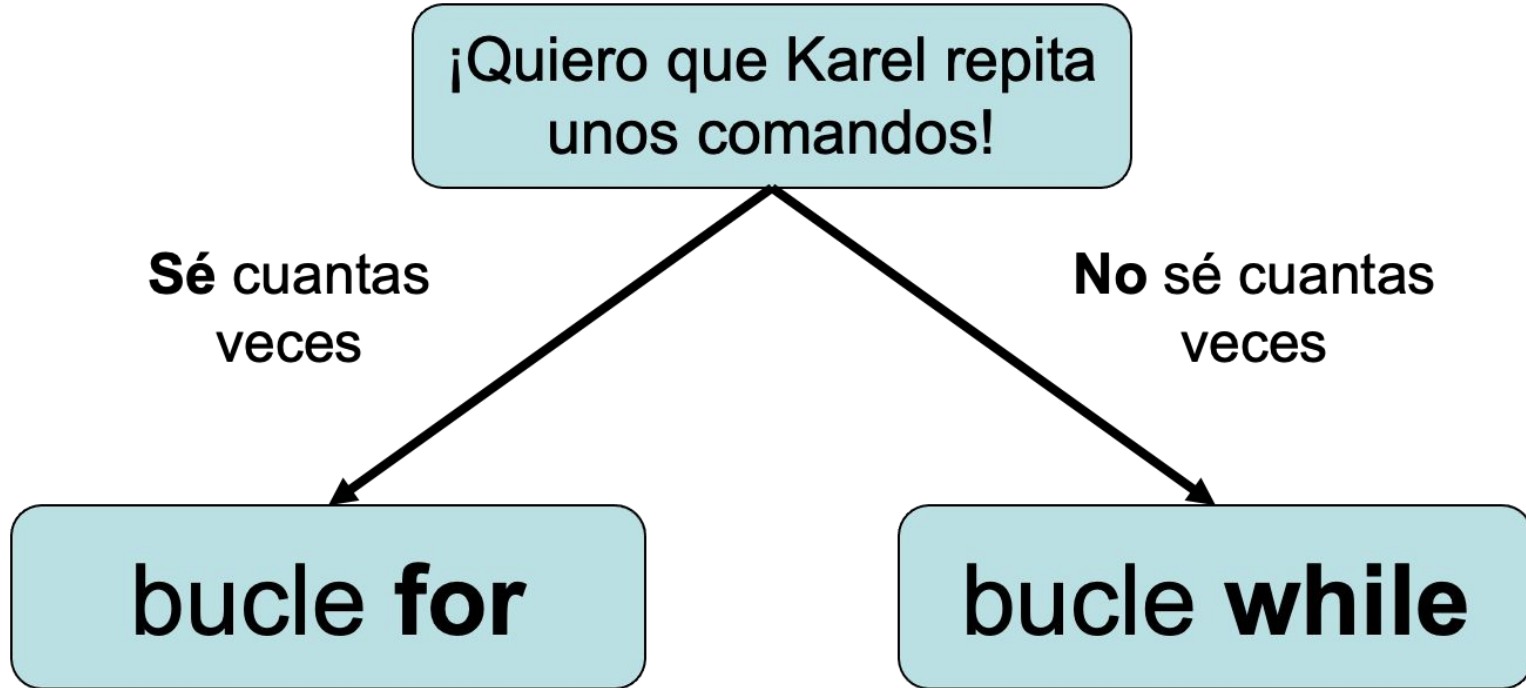
Ahora podríamos decir:

```
1. def main():  
2.     while frente_despejado():  
3.         moverse()  
4.  
5.
```

Este programa
funciona en un mundo
de ***cualquier*** tamaño

Resumen de bucles

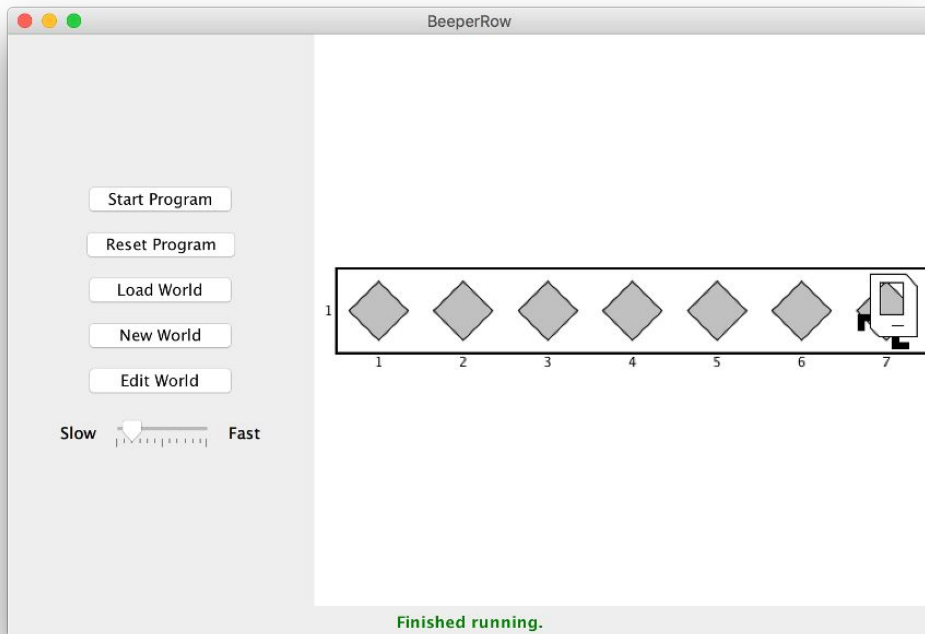
Resumen de bucles



Preguntas?

Línea de conos

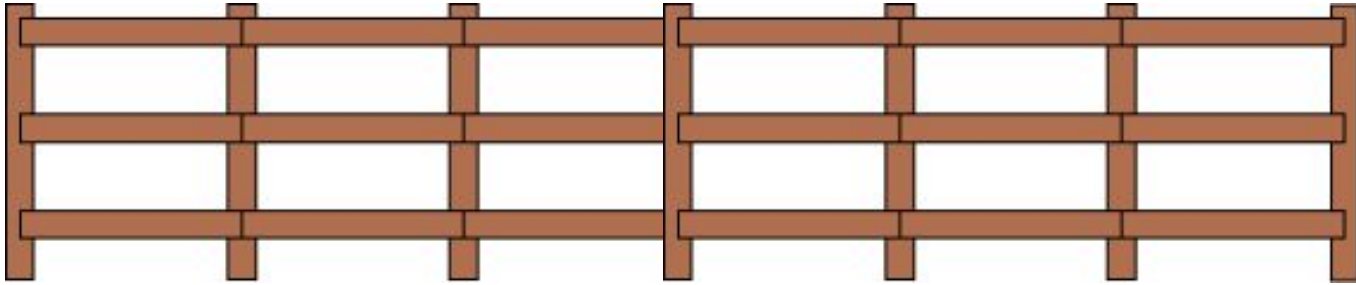
Quiero que Karel llene una fila con conos. ¿Cómo lo hago?



Línea de conos

Demostración

El problema de la cerca



¡6 segmentos de cerca, pero hay 7 postes!

Estructura de la cerca

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

Opción 1

```
1. def main():
2.     poner_cono()          # poste
3.     while frente_despejado():
4.         moverse()         # cerca
5.         poner_cono()      # poste
6.
```

Opción 2

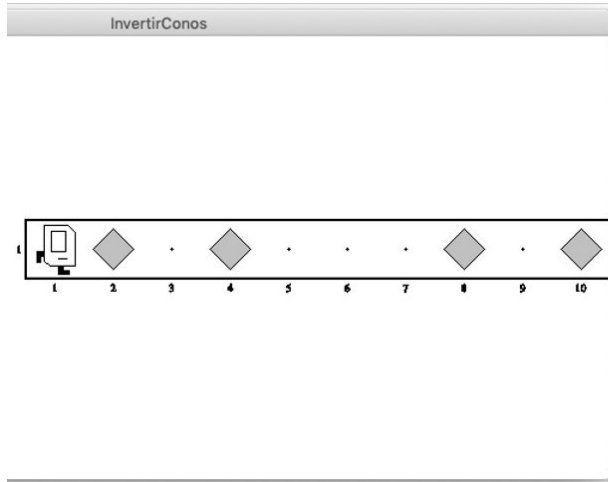
```
1. def main():
2.     while frente_despejado():
3.         poner_cono()      # poste
4.         moverse()         # cerca
5.         poner_cono()      # poste
```

Condicionales

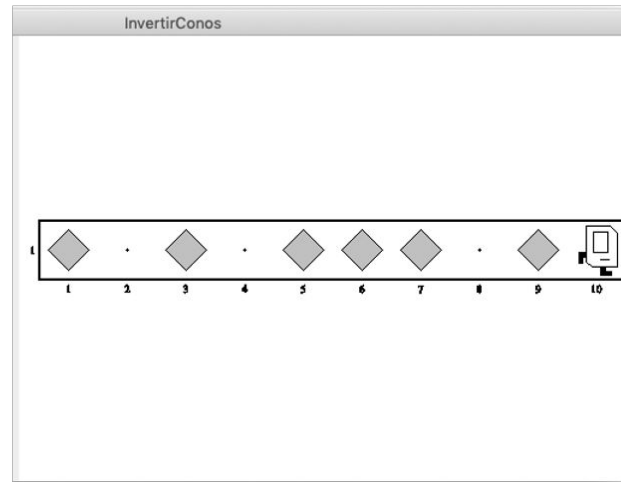
Invertir conos

Quiero que Karel invierta los conos en la fila. Si hay un cono, debería recogerlo y si no hay un cono, debería poner uno. ¿Cómo lo hago?

Antes



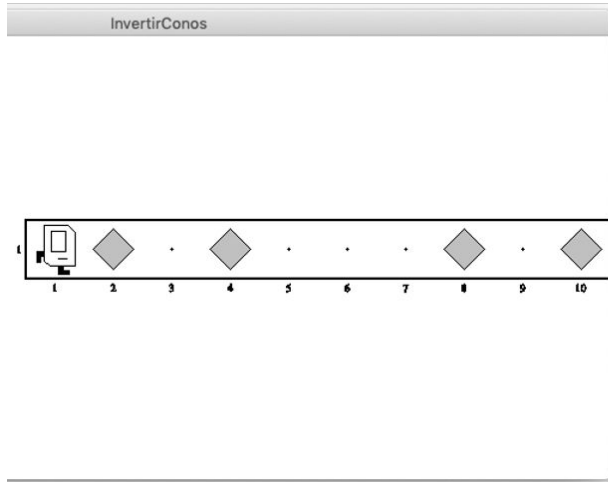
Después



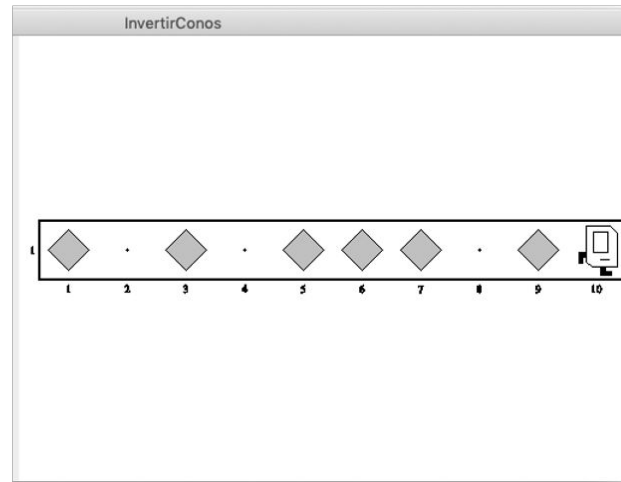
Invertir conos

Quiero que Karel invierta los conos en la fila. Si hay un cono, debería recogerlo y si no hay un cono, debería poner uno. ¿Cómo lo hago?

Antes



Después



Instrucciones condicionales

if *condición:*

instrucción

instrucción

...

Para ejecutar una instrucción
condicional, usa if

Instrucciones condicionales

if *condición:*

instrucción

instrucción

else:

instrucción

instrucción

También puedes incluir una
instrucción else:

Invertir conos

Ahora podemos decir:

```
1. def main():
2.     while frente_despejado():
3.         if conos_presentes():
4.             recoger_cono()
5.         else:
6.             poner_cono()
7.         moverse()
8.     if conos_presentes():
9.         recoger_cono()
10.    else:
11.        poner_cono()
```