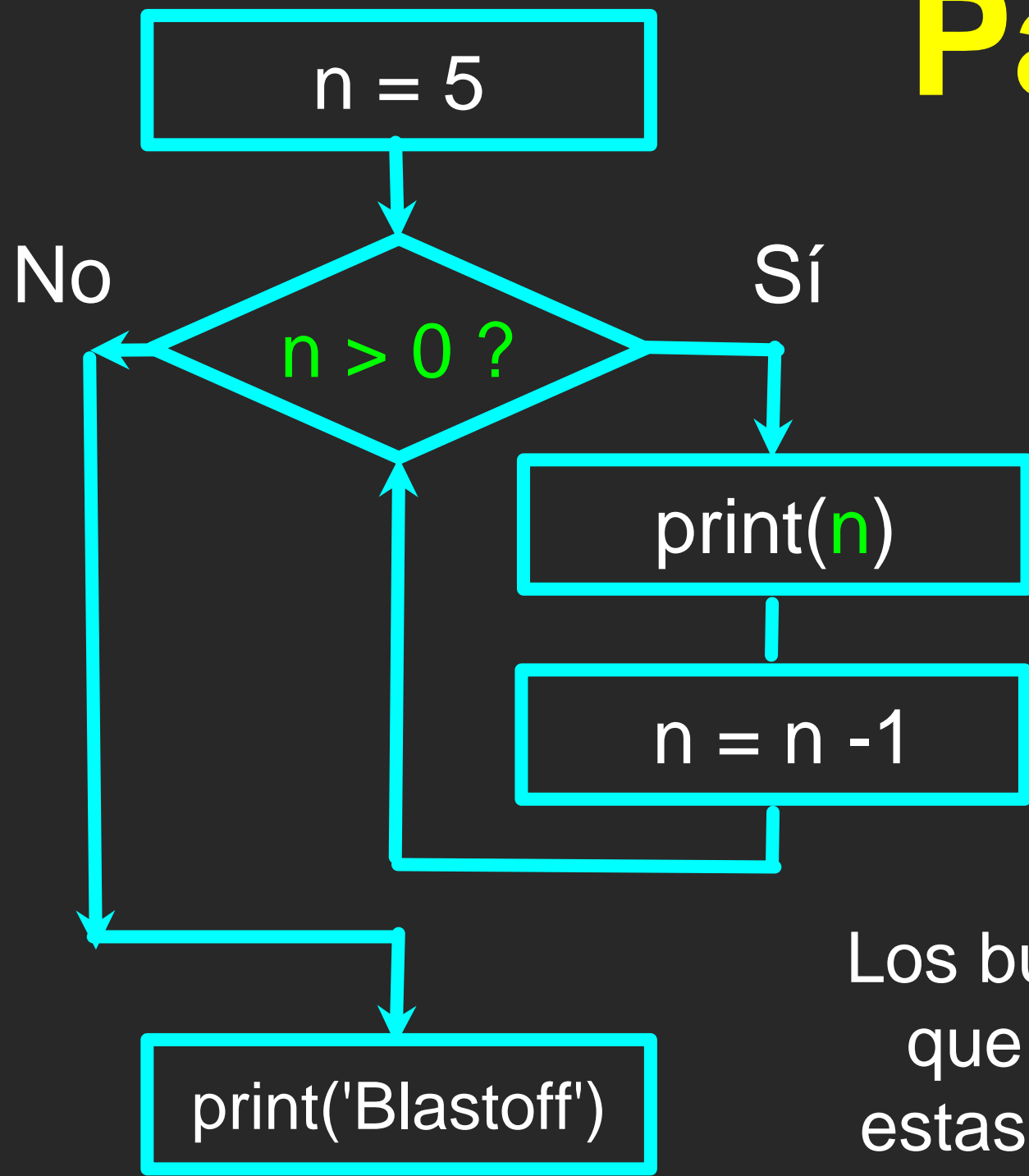


Bucles e Iteración

Pasos Repetidos



Programa:

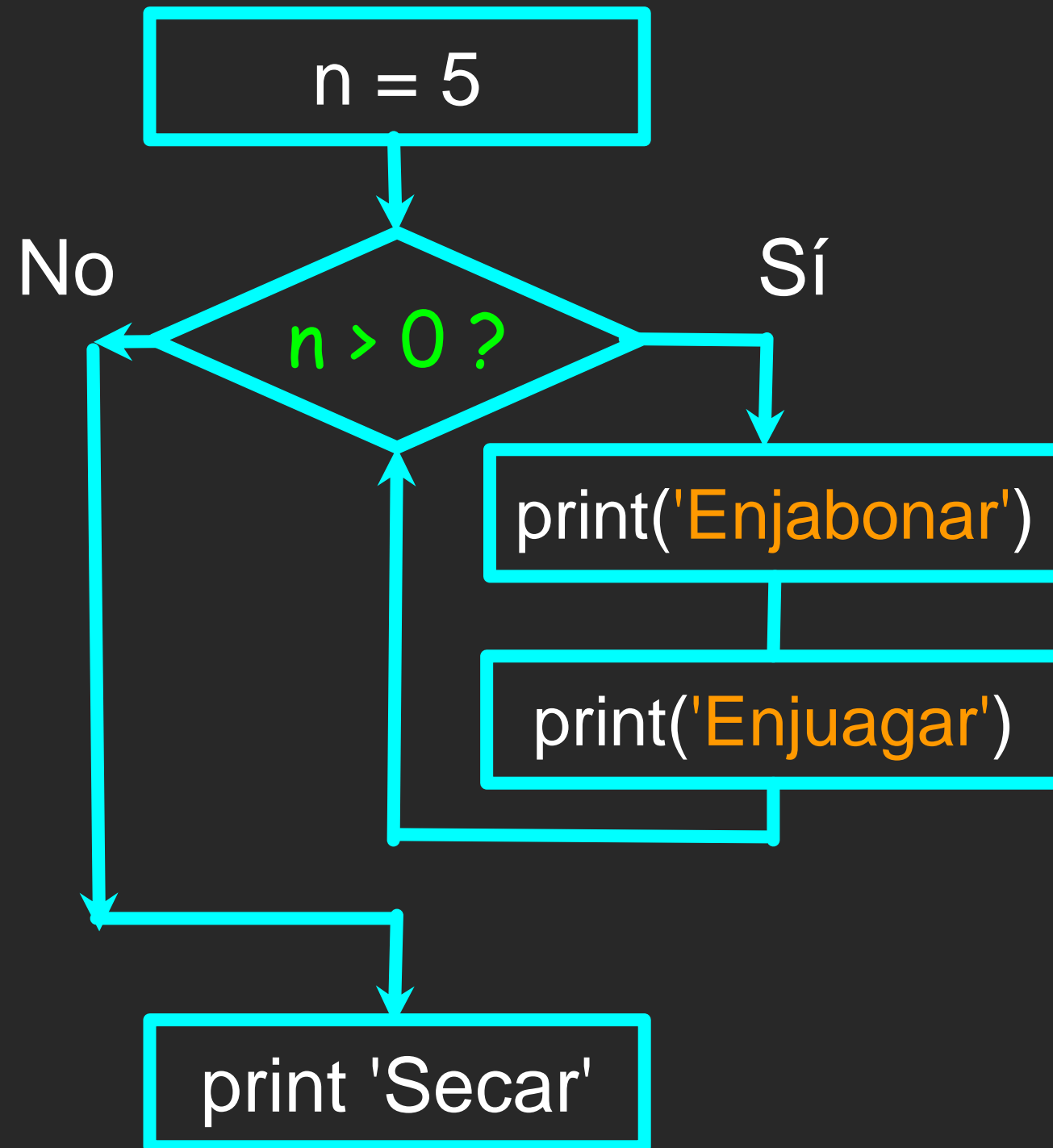
```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff')
print(n)
```

Resultado:

5
4
3
2
1
¡Blastoff!
0

Los bucles (pasos repetidos) tienen **variables de iteración** que cambian cada vez a través del bucle. A menudo, estas **variables de iteración** atraviesan una secuencia de números.

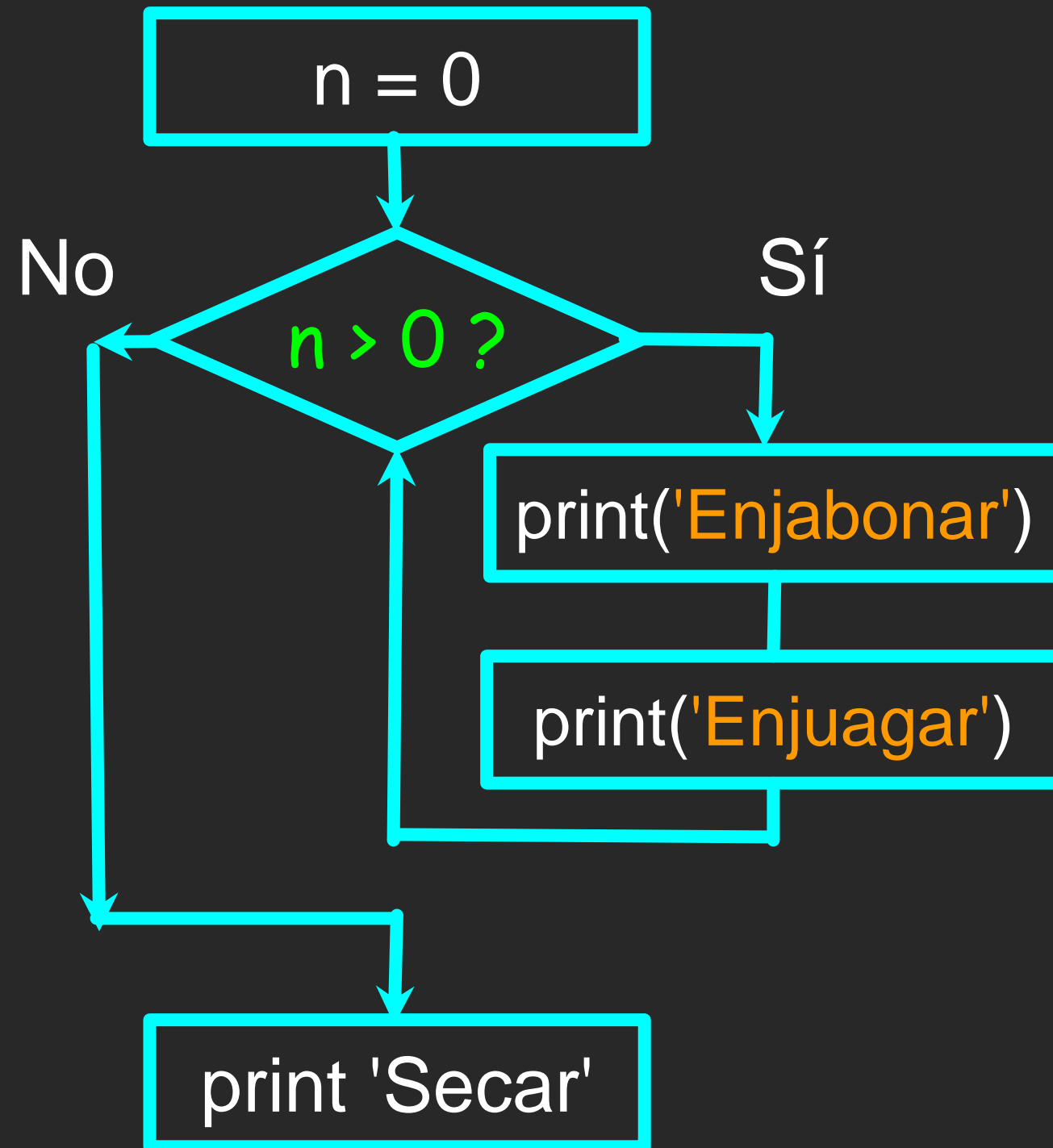
Un Bucle Infinito



```
n = 5
while n > 0 :
    print('Enjabonar')
    print('Enjuagar')
    print('Secar')
```

¿Qué es lo que está mal en este bucle?

Otro Bucle



```
n = 0
while n > 0 :
    print('Enjabonar')
    print('Enjuagar')
    print('Secar! ')
```

¿Qué es lo que está haciendo este bucle?

Romper un Bucle

- El enunciado **break** (romper) termina el bucle actual y salta al enunciado que le sigue inmediatamente al bucle
- Es como una prueba de bucle que puede suceder en cualquier lado en el cuerpo del bucle

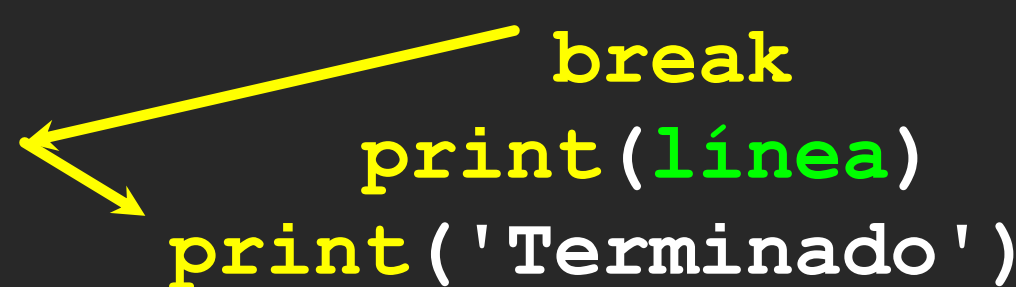
```
while True:
    línea = input('> ')
    if línea == 'terminado':
        break
    print(línea)
print('terminado')
```

```
> hola
hola
> finished
finalizado
> done
terminado
```

Romper un Bucle

- El enunciado **break** (romper) termina el bucle actual y salta al enunciado que le sigue inmediatamente al bucle
- Es como una prueba de bucle que puede suceder en cualquier lado en el cuerpo del bucle

```
while True:
    línea = input('> ')
    if línea == 'terminado'
:
    break
    print(línea)
    print('Terminado')
```



```
> hola
hola
> finished
finalizado
> done
terminado
```

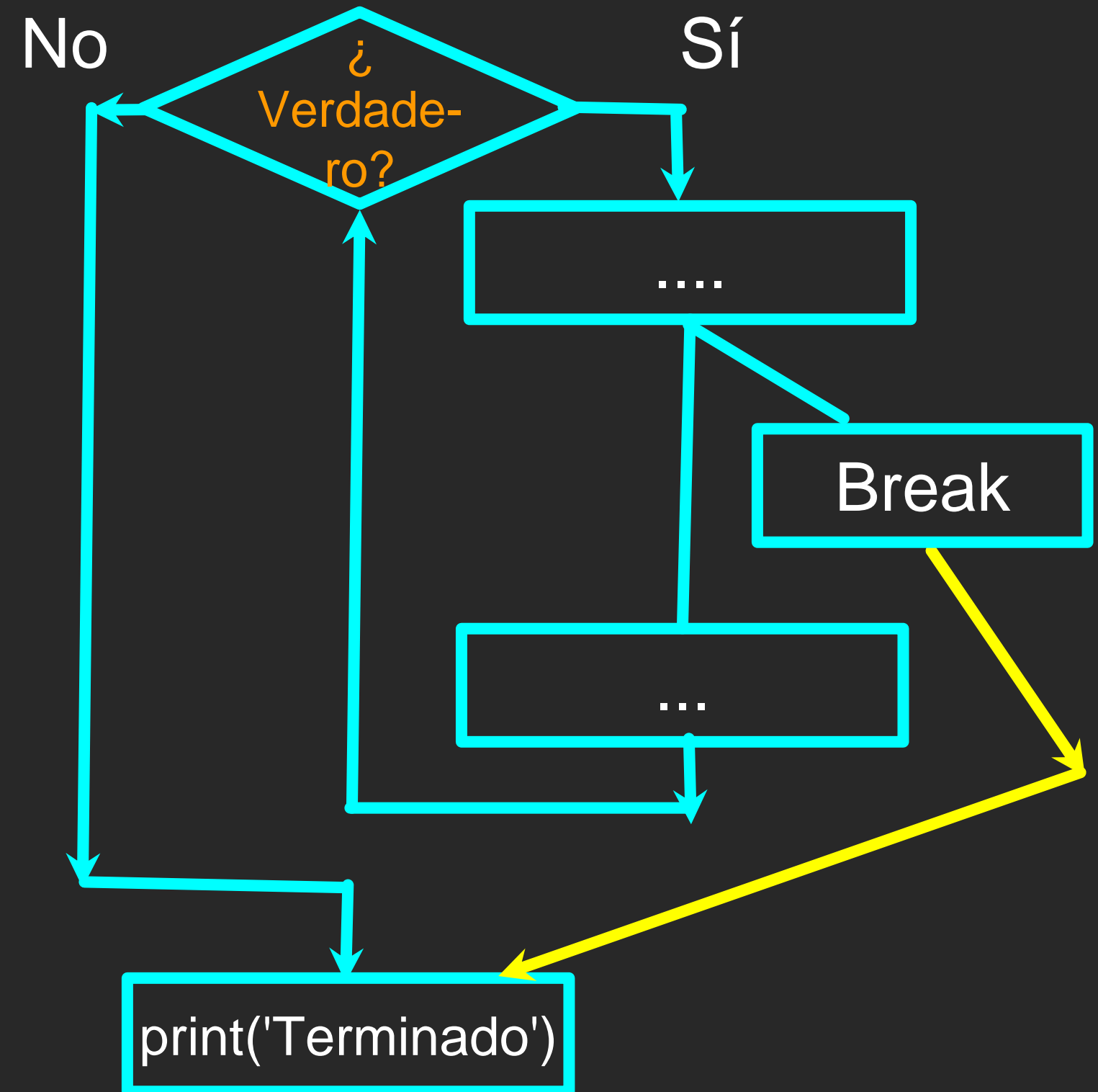
```

while True:
    línea = input('> ')
    if línea == 'terminado'
:
        break
    print(línea)
    print('terminado')

```



[http://en.wikipedia.org/wiki/Transporter_\(Star_Trek\)](http://en.wikipedia.org/wiki/Transporter_(Star_Trek))



Finalizar una Iteración con Continue

El enunciado **continue** (continuar) termina la iteración actual y salta a la parte superior del bucle y comienza la siguiente iteración

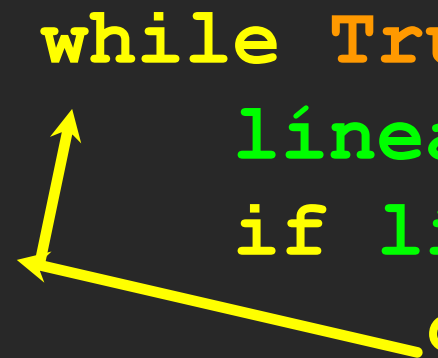
```
while True:
    line = input('> ')
    if line[0] == '#':
        continue
    if line == 'terminado':
        break
    print(line)
print('Terminado')
```

```
> hola
Hola
> # no imprimir esto
> Imprimir esto
imprimir esto
> terminado
Terminado
```


Finalizar una Iteración con Continue

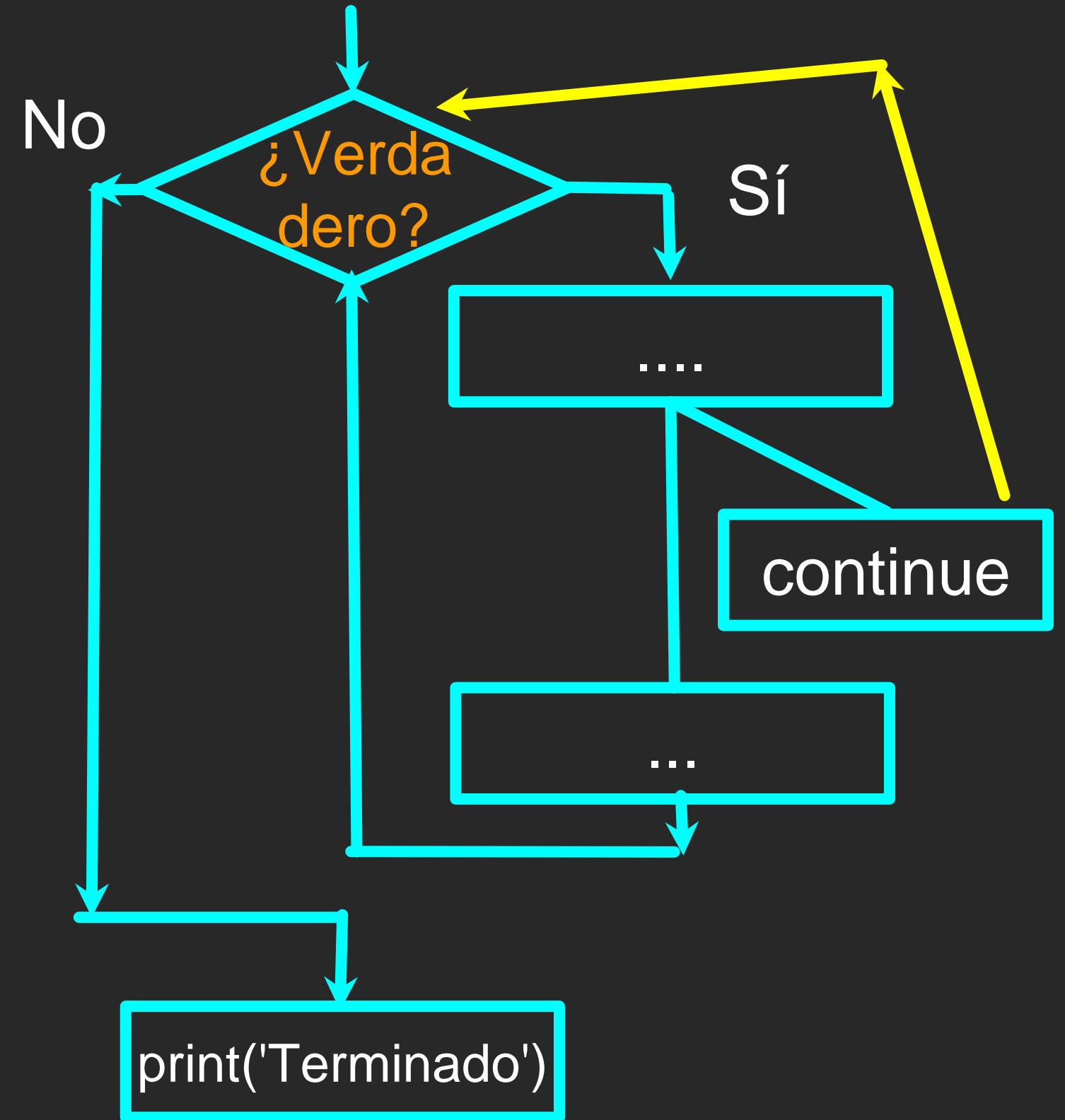
El enunciado **continue** (continuar) termina la **iteración actual** y salta a la **parte superior del bucle** y comienza la siguiente iteración

```
while True:
    línea = input('> ')
    if línea[0] == '#' :
        continue
    if línea == 'terminado' :
        break
    print(línea)
print('Terminado')
```



```
> hola
hola
> # no imprimir esto
> Imprimir esto
imprimir esto
> terminado
Terminado
```

```
while True:
    línea = raw_input('> ')
    if línea[0] == '#' :
        continue
    if línea == 'terminado' :
        break
    print(línea)
print('Terminado')
```



Bucles Indefinidos

- Los bucles while se llaman “bucles indefinidos” porque continúan hasta que una condición lógica se vuelve **False (Falsa)**
- Los bucles que hemos visto hasta ahora son bastante fáciles de examinar para determinar si terminarán o si serán “bucles infinitos”
- A veces, es más difícil saber con seguridad si un bucle terminará

Bucles Definidos

Bucles Definidos

- Con bastante frecuencia tenemos una **lista** de los ítems de las **líneas en un archivo**, es decir un **conjunto finito** de cosas
- Podemos escribir un bucle para ejecutar el bucle una vez para cada uno de los ítems de un conjunto utilizando la secuencia **for** de Python
- Estos bucles se denominan “**bucles definidos**” porque se ejecutan una cantidad exacta de veces
- Decimos que los “**bucles definidos iteran a través de los miembros de un conjunto**”

Un Bucle Definido Simple

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff')
```

5

4

3

2

1

Blastoff

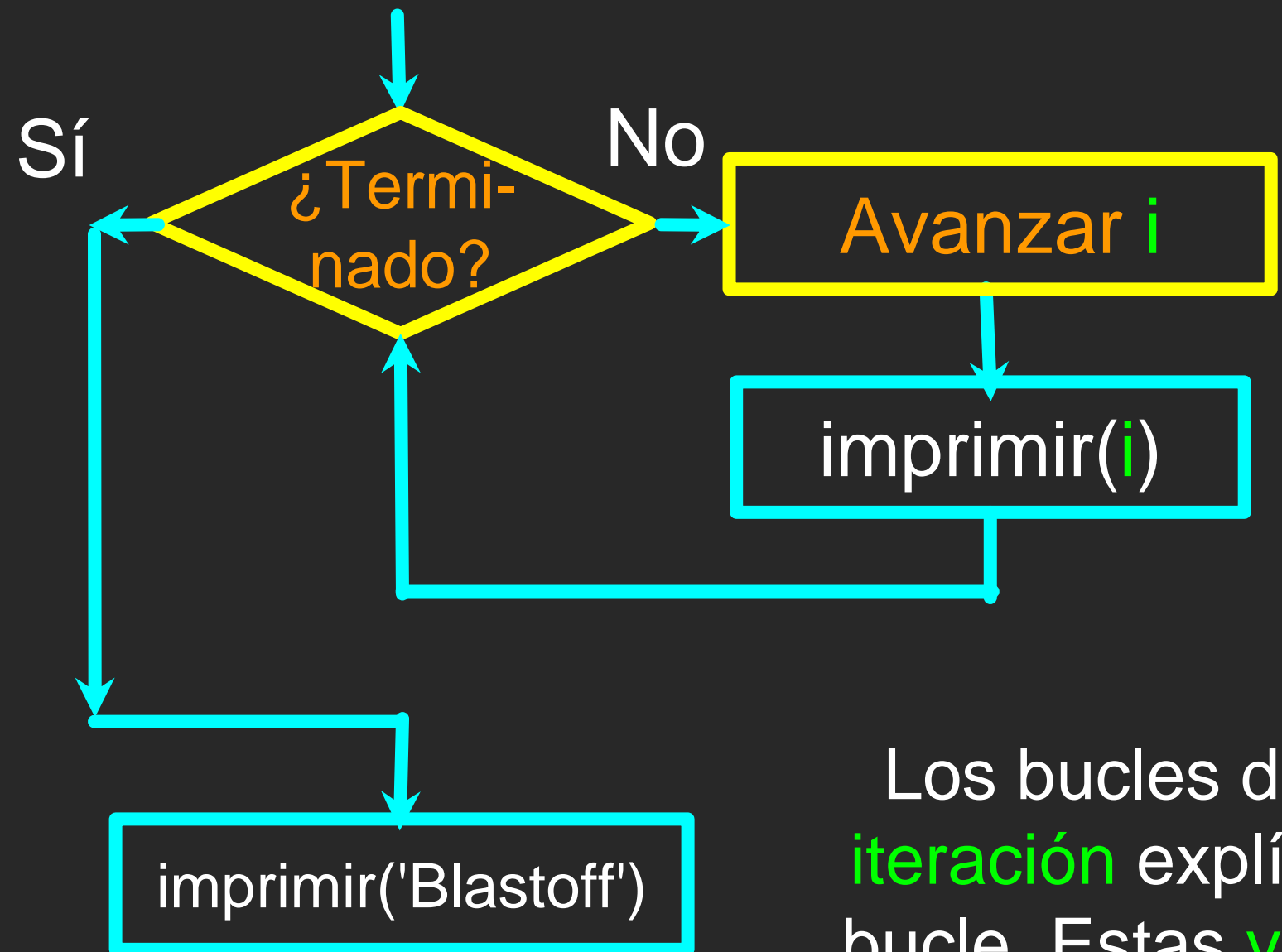
Un Bucle Definido con Cadenas

```
amigos = ['Joseph', 'Glenn', 'Sally']  
for amigos in amigos :  
    print('Feliz año nuevo:', amigo)  
print('Terminado')
```

Feliz año nuevo: Joseph
Feliz año nuevo: Glenn
Feliz año nuevo: Sally

¡Terminado!

Un Bucle Definido Simple



```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff')
```


5
4
3
2
1
Blastoff

Los bucles definidos (bucles for) tienen **variables de iteración** explícitas que cambian cada vez a través del bucle. Estas **variables de iteración** se mueven a través del conjunto o secuencia.

Observando a In...

- La **variable de iteración** “itera” a través de la **secuencia** (conjunto ordenado)
- El **bloque (cuerpo)** del código se ejecuta una vez para cada valor **in** de la **secuencia**
- La **variable de iteración** se mueve a través de todos los valores **in** de la **secuencia**

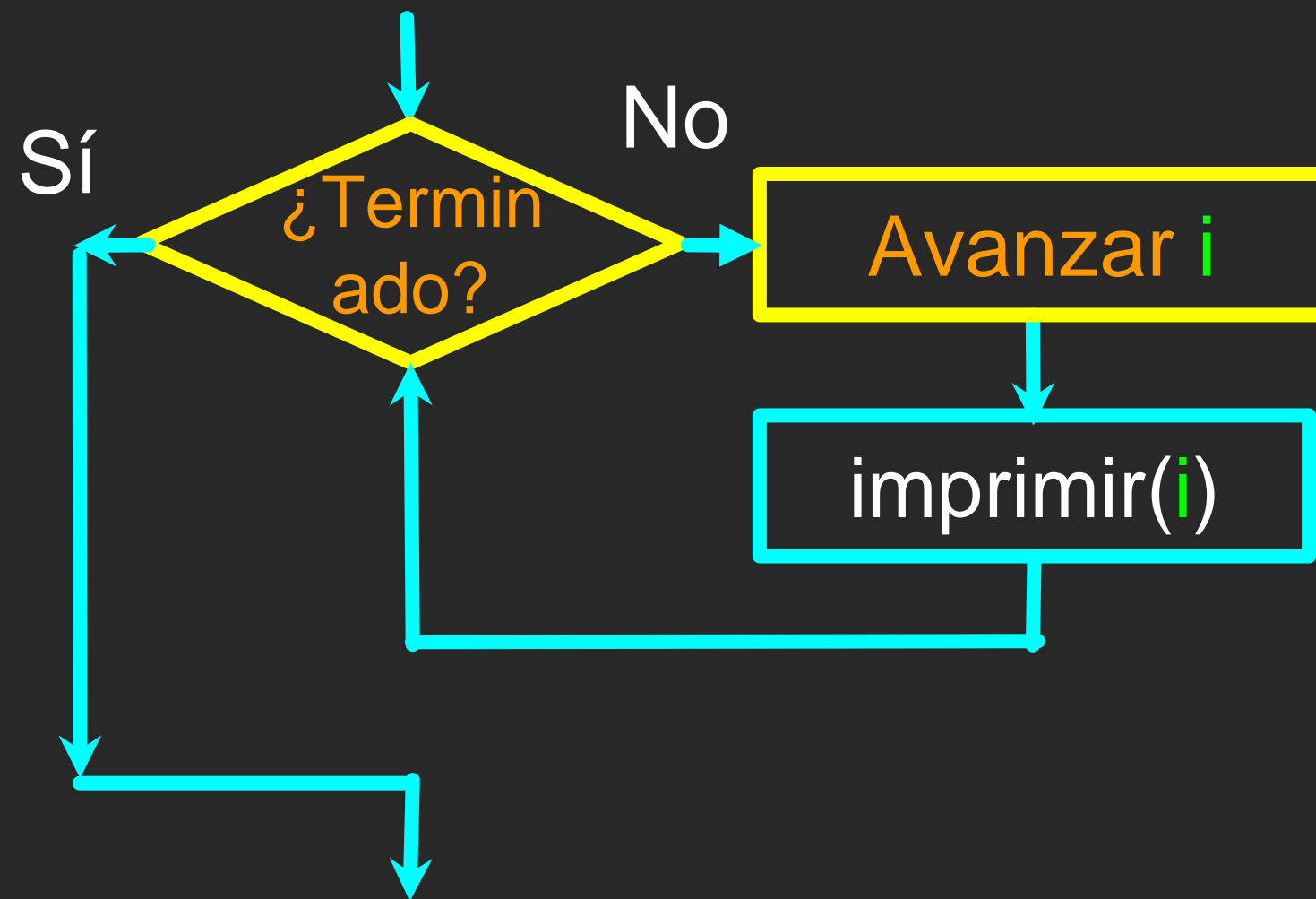
Variable de iteración



Secuencia de cinco elementos

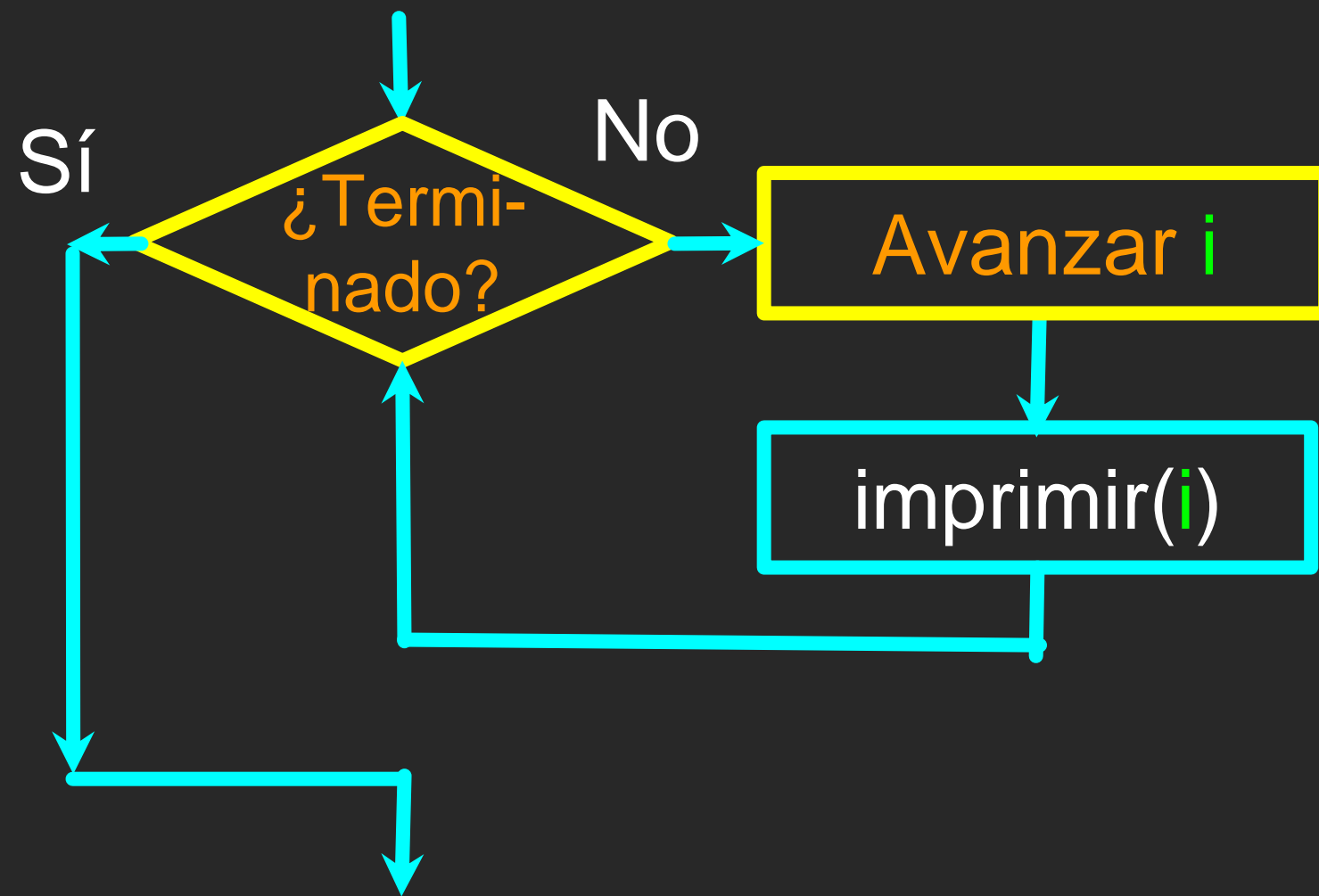


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

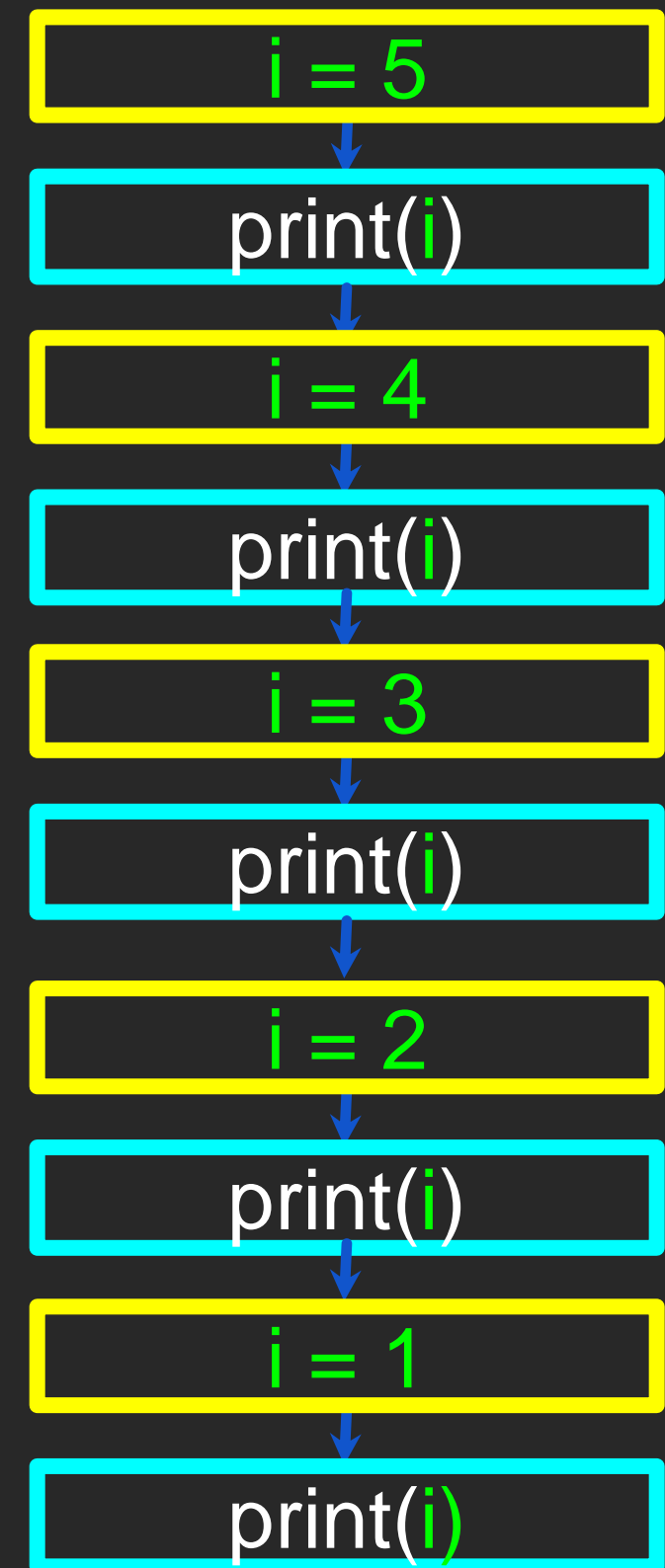


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

- La **variable de iteración** “itera” a través de la **secuencia** (conjunto ordenado)
- El **bloque (cuerpo)** del código se ejecuta una vez para cada valor **in** de la **secuencia**
- La **variable de iteración** avanza a través de todos los valores **in** de la **secuencia**



```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```



Bucles Definidos

- Con bastante frecuencia tenemos una **lista** de los ítems de las **líneas en un archivo**, es decir un **conjunto finito** de cosas
- Podemos escribir un bucle para ejecutar el bucle una vez para cada uno de los ítems de un conjunto utilizando la secuencia **for** de Python
- Estos bucles se denominan “**bucles definidos**” porque se ejecutan una cantidad exacta de veces
- Decimos que los “**bucles definidos iteran a través de los miembros de un conjunto**”

Lenguajes de Bucle: Lo Que Hacemos en los Bucles

Nota: Aunque estos ejemplos sean simples, los patrones se aplican a todos los tipos de bucles

Creando Bucles “inteligentes”

El truco consiste en “conocer” algo acerca del bucle entero cuando está estancado escribiendo código que solo ve una entrada por vez

Configure algunas variables con los valores iniciales

para objeto en los datos:

Buscar o hacer algo para cada entrada por separado, que actualice una variable

Observe las variables

Iteración de un conjunto

```
print('Antes')
for objeto in [9, 41, 12, 3, 74, 15] :
    print(objeto)
print('Después')
```

\$ python basicloop.py

Antes

9

41

12

3

74

15

Después

¿Cuál es el número mayor?

¿Cuál es el número mayor?

3

¿Cuál es el número mayor?

41

¿Cuál es el número mayor?

12



¿Cuál es el número mayor?

9

¿Cuál es el número mayor?

74

¿Cuál es el número mayor?

15

¿Cuál es el número mayor?

¿Cuál es el número mayor?

3

41

12

9

74

15

¿Cuál es el número mayor?

largest_so_far

-1

¿Cuál es el número mayor?

3

largest_so_far

3

¿Cuál es el número mayor?

41

largest_so_far

41

¿Cuál es el número mayor?

12

largest_so_far

41

¿Cuál es el número mayor?

9

largest_so_far

41

¿Cuál es el número mayor?

74

largest_so_far

74

¿Cuál es el número mayor?

15

74

¿Cuál es el número mayor?

3 41 12 9 74 15

74

Para encontrar el mayor valor

```
largest_so_far = -1
print('Antes', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
        print(largest_so_far, the_num)

print('Después', largest_so_far)
```

\$ python largest.py

Antes -1

9 9

41 41

41 12

41 3

74 74

74 15

Después 74

Creamos una variable que contenga el mayor valor que se haya visto hasta ahora (largest_so_far). Si el número actual que estamos buscando es más grande, entonces será el nuevo mayor valor que se haya visto hasta ahora (largest_so_far).

Más Lenguajes de Bucle

Conteo en un Bucle

```
zork = 0
print('Antes', zork)
for objeto in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, objeto)
print('Después', zork)
```

```
$ python countloop.py
```

Antes 0

1 9

2 41

3 12

4 3

5 74

6 15

Después 6

Para **contar** cuántas veces ejecutamos un bucle, introducimos una **variable de conteo** que comience en 0 y le sumamos uno cada vez a través del bucle.

Suma en un Bucle

```
zork = 0
print('Antes', zork)
for objeto in [9, 41, 12, 3, 74, 15]:
    zork = zork + objeto
    print(zork, objeto)
print('Después', zork)
```

\$ python countloop.py

Antes 0

9 9

50 41

62 12

65 3

139 74

154 15

Después 154

Para **sumar** un **valor** que encontramos en un bucle, introducimos una **variable de suma que comience en 0** y le sumamos el **valor** a la suma cada vez a través del bucle.

Sacar el Promedio en un Bucle

```
conteo = 0
suma = 0
print('Antes', conteo, suma)
for valor in [9, 41, 12, 3, 74, 15] :
    conteo = conteo + 1
    suma = suma + valor
    print(conteo, suma, valor)
print('Después', conteo, suma, suma /
      conteo)
```

```
$ python averageloop.py
```

```
Antes 0 0
```

```
1 9 9
```

```
2 50 41
```

```
3 62 12
```

```
4 65 3
```

```
5 139 74
```

```
6 154 15
```

```
Después 6 154 25
```

Un **promedio** solo combina los patrones de **conteo** (count) y **suma** (sum) y divide cuando el bucle ha terminado.

Filtrar en un Bucle

```
print('Antes')  
for valor in [9, 41, 12, 3, 74, 15] :  
    if valor > 20:  
        print 'Mayor Número',valor  
print('Después')
```

```
$ python search1.py
```

Antes

Mayor número 41

Mayor número 74

Después

Utilizamos un enunciado hipotético “if” en el bucle para captar / filtrar los valores que estamos buscando.

Búsqueda Utilizando una Variable Booleana

```
found = False
print('Antes', found)
for valor in [9, 41, 12, 3, 74, 15] :
    if valor == 3 :
        found = True
    print(found, valor)
print('Después', found)
```

```
$ python search1.py
```

```
Antes False (Falsa)
```

```
False (Falsa) 9
```

```
False (Falsa) 41
```

```
False (Falsa) 12
```

```
True (Falsa) 3
```

```
True (Falsa) 74
```

```
True (Falsa) 15
```

```
Después True (Verdadera)
```

Si solo deseamos buscar y **saber si un valor fue hallado (found)**, utilizamos una **variable** que comience como **False** (Falsa) y se vuelva **True** (Verdadera) tan pronto como **encontramos (find)** lo que estamos buscando.

Cómo Encontrar el Menor Valor

```
mayor_hasta_ahora = -1
print('Antes', mayor_hasta_ahora)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > mayor_hasta_ahora :
        mayor_hasta_ahora = the_num
        print(mayor_hasta_ahora, the_num)

print('Después', mayor_hasta_ahora)
```

\$ python largest.py

Antes -1

9 9

41 41

41 12

41 3

74 74

74 15

Después 74

¿Cómo cambiaríamos esto para hacer que encuentre el menor valor de la lista?

Cómo Encontrar el Menor Valor

```
menor_hasta_ahora = -1
print('Antes', menor_hasta_ahora)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < menor_hasta_ahora :
        menor_hasta_ahora = the_num
        print(menor_hasta_ahora, the_num)

print('Después', menor_hasta_ahora)
```

Cambiamos el nombre de la variable por **menor valor hasta ahora** (**smallest_so_far**) y cambiamos **>** por **<**

Cómo Encontrar el Menor Valor

```
menor_hasta_ahora = -1
print('Antes', menor_hasta_ahora)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < menor_hasta_ahora :
        menor_hasta_ahora = the_num
        print(menor_hasta_ahora, the_num)

print('Después', menor_hasta_ahora)
```

\$ python smallbad.py

Antes -1

-1 9

-1 41

-1 12

-1 3

-1 74

-1 15

Después -1

Cambiamos el nombre de la variable por `menor valor hasta ahora`
(`smallest_so_far`) y cambiamos `>` por `<`

Cómo Encontrar el Menor Valor

```
menor = Ninguno
print('Antes')
for valor in [9, 41, 12, 3, 74, 15] :
    if menor is Ninguno:
        menor = valor
    elif valor < menor :
        menor = valor
    print(menor, valor)
print('Después', menor)
```

```
$ python smallest.py
```

Antes

9 9

9 41

9 12

3 3

3 74

3 15

Después 3

Aún tenemos una variable que es **menor valor (smallest)** hasta ahora. La primera vez en el bucle **menor valor** es **Ninguno**, entonces tomamos el primer **valor** como **menor valor**.

Los Operadores “is” e “is not”

```
menor = Ninguno
print('Antes')
for valor in [3, 41, 12, 9, 74, 15] :
    if menor is Ninguno:
        menor = valor
    elif valor < menor :
        menor = valor
    print menor, valor
print('Después', menor)
```

- Python tiene un operador **is** (es) que puede ser utilizado en expresiones lógicas
- Implica que “es el mismo que”
- Similar a, pero más fuerte que **==**
- **is not** (no es) también es un operador lógico

Síntesis

- Bucle While (indefinido)
- Bucles infinitos
- Uso de Break
- Uso de Continue
- Bucle For (definido)
- Variables de iteración
- Lenguajes de bucle
- Mayor o menor