



UNIVERSIDAD
DE MÁLAGA



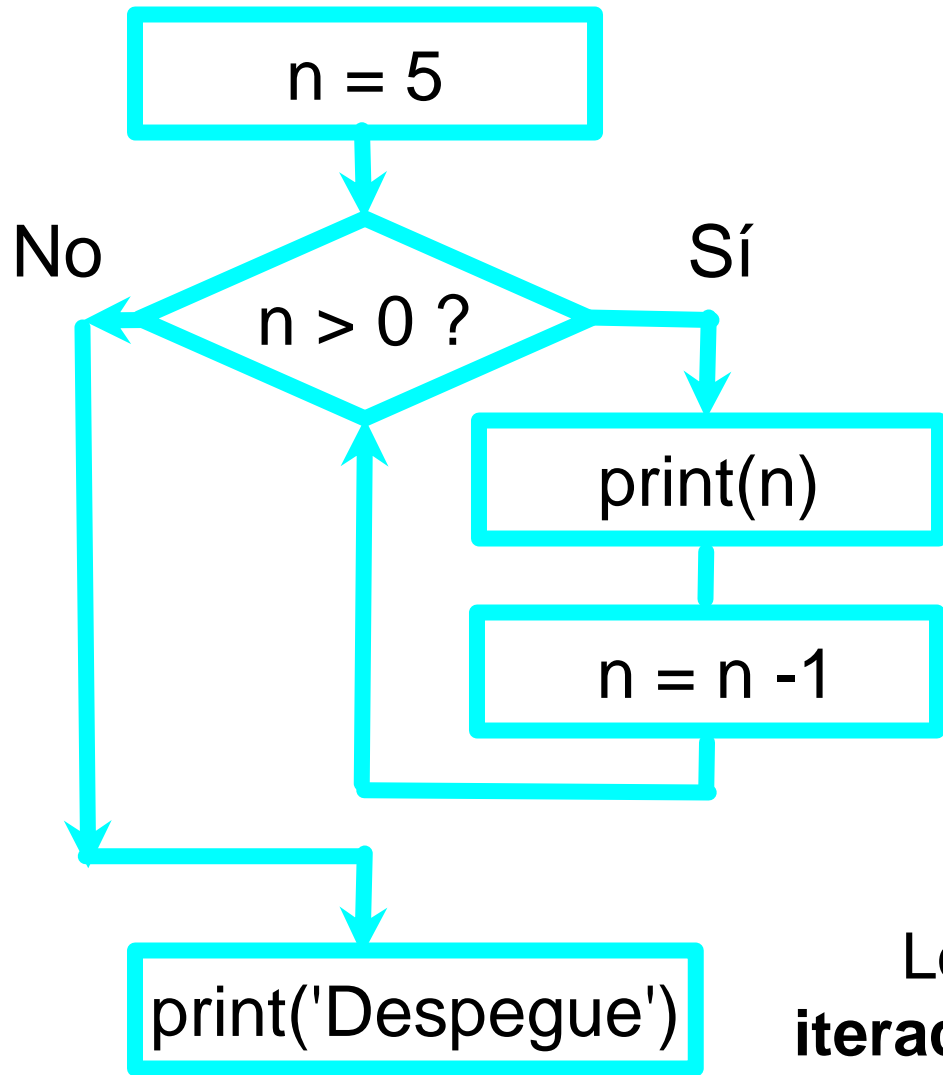
Bucles

Unidad 5

Python para principiantes
dgm@uma.es



Pasos Repetidos



Programa:

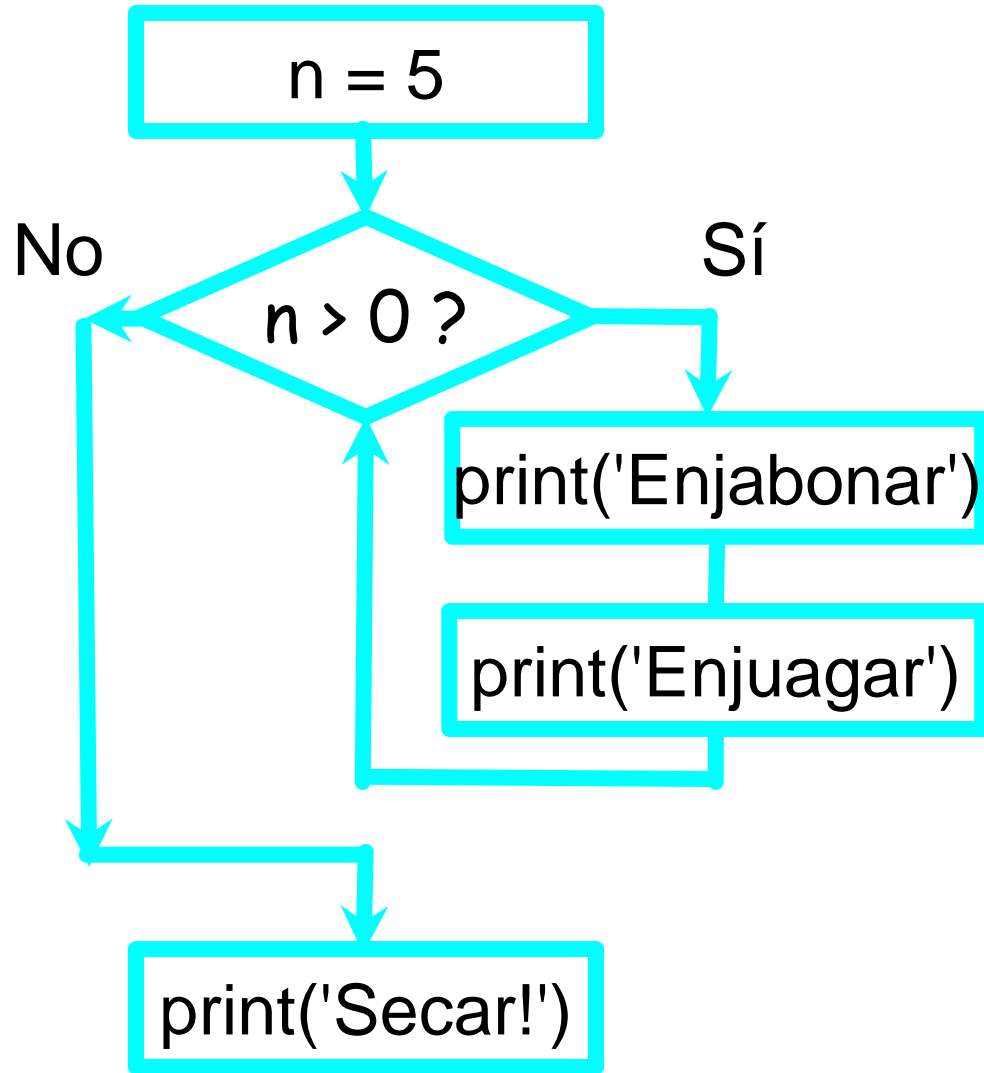
```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Despegue!')
print(n)
```

Salida:

5
4
3
2
1
Despegue!
0

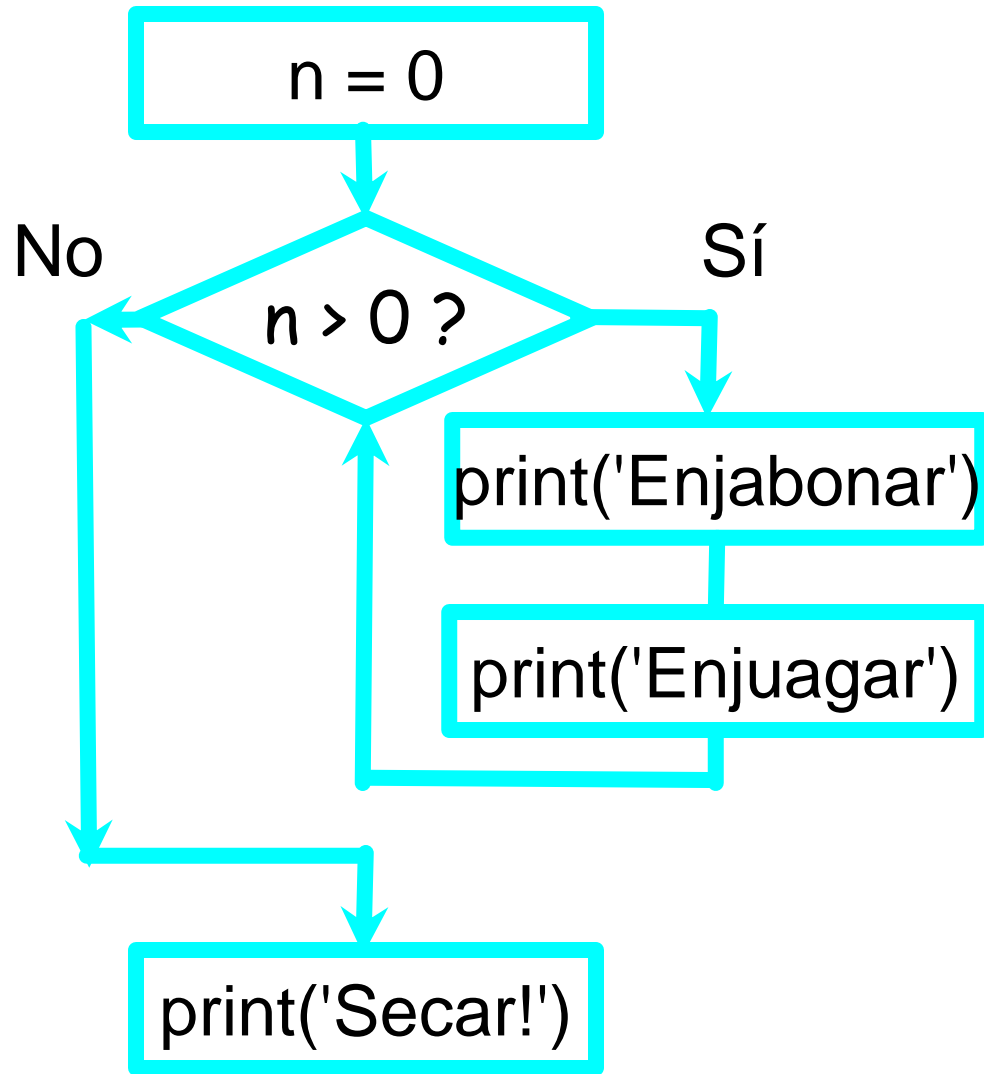
Los bucles (pasos repetidos) tienen **variables de iteración** que **cambian** cada vez que se ejecuta el bucle (iteración). Frecuentemente estas variables de iteración forman una **secuencia de números**.

Un bucle infinito



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

¿Qué está mal en este bucle?



Otro bucle

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

¿Qué hace este bucle?

Rompiendo un buce

La sentencia **break** finaliza el bucle actual y "salta" a la sentencia inmediatamente posterior al mismo

Es como comprobar la condición del bucle en cualquier parte del cuerpo del bucle

```
while True:
    line = input('> ')
    if line == 'hecho' :
        break
    print(line)
print('Hecho!')
```

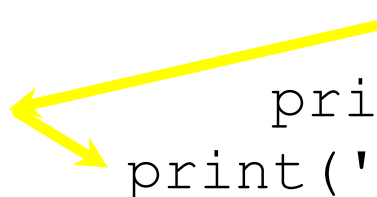
```
> hola
hola
> fin
fin
> hecho
Hecho!
```

Rompiendo un buce

La sentencia **break** finaliza el bucle actual y "salta" a la sentencia inmediatamente posterior al mismo

Es como comprobar la condición del bucle en cualquier parte del cuerpo del bucle

```
while True:
    line = input('> ')
    if line == 'hecho' :
        break
    print(line)
    print('Hecho!')
```

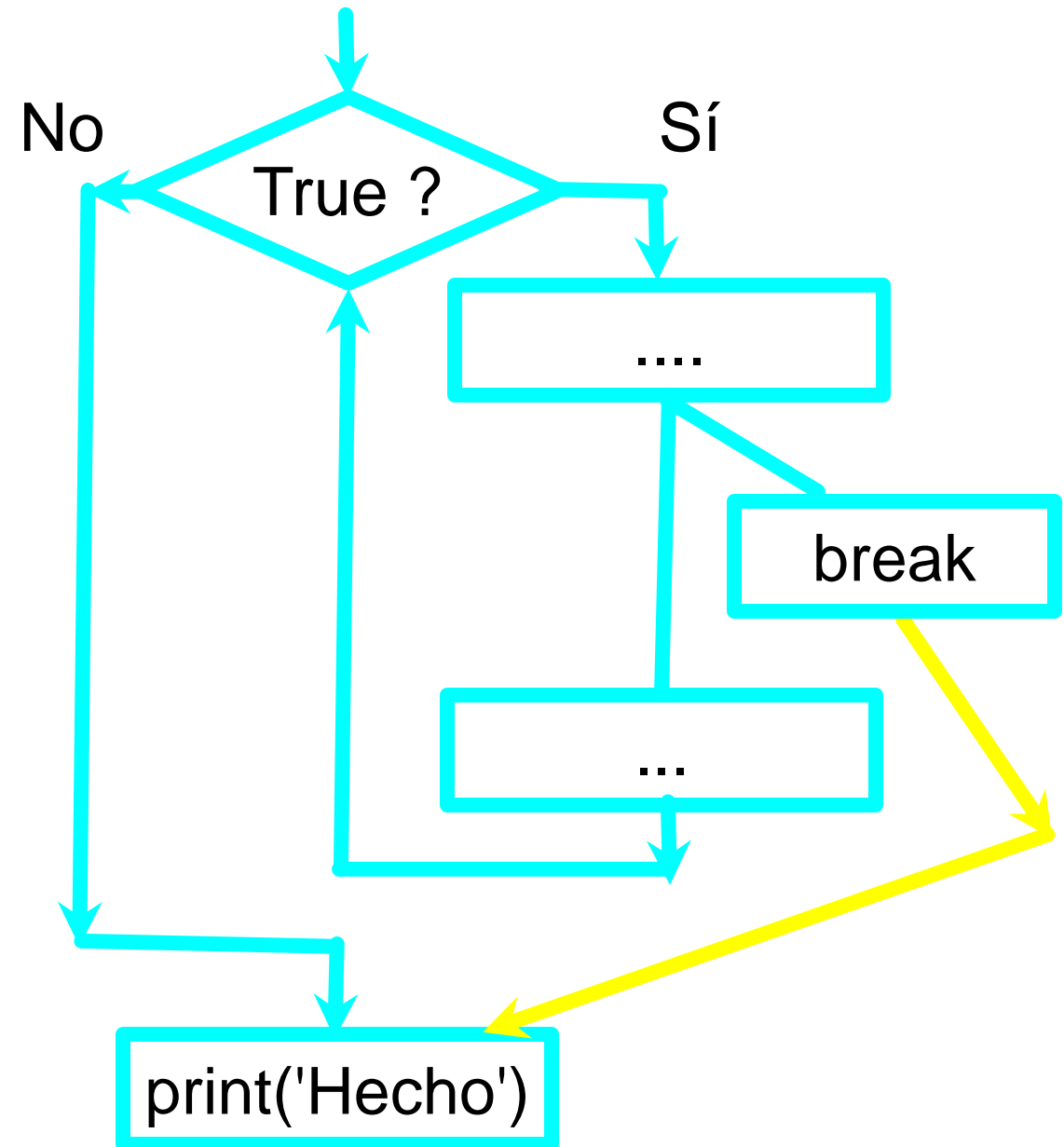


```
> hola
hola
> fin
fin
> hecho
Hecho!
```

```
while True:
    line = input('> ')
    if line == 'hecho' :
        break
    print(line)
    print('Hecho!')
```



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



Terminando una iteración con continue

La sentencia **continue** finaliza la iteración actual y salta al inicio del bucle para comenzar la próxima iteración

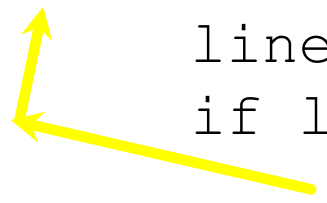
```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'hecho' :
        break
    print(line)
print('Hecho!')
```

```
> hola
hola
> # ignora esto
> imprime esto!
imprime esto!
> hecho
Hecho!
```


Terminando una iteración con continue

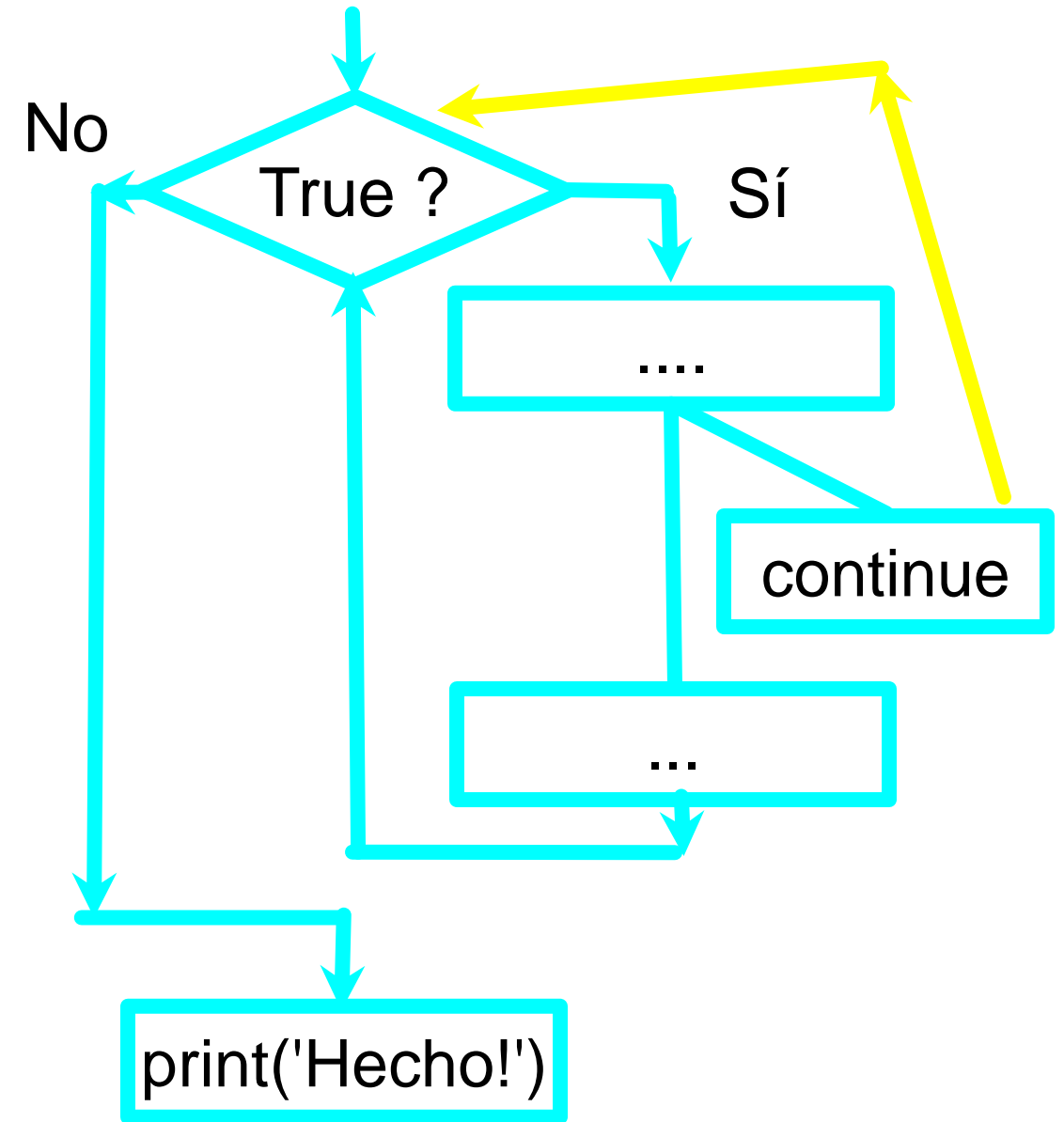
La sentencia **continue** finaliza la iteración actual y salta al inicio del bucle para comenzar la próxima iteración

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'hecho' :
        break
    print(line)
print('Hecho!')
```



```
> hola
hola
> # ignora esto
> imprime esto!
Imprime esto!
> hecho
Hecho!
```

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'hecho' :
        break
    print(line)
print('Hecho!')
```



Bucles Indefinidos

Los **bucles while** tienen un número de iteraciones indefinidas porque se ejecutan hasta que una condición lógica se vuelve falsa (**False**)

En los bucles anteriores era bastante fácil ver si terminarían o se convertirían en bucles "infinitos"

A veces es más difícil estar seguro de cuándo terminará un bucle

Bucles Definidos

- Iterando sobre un conjunto de elementos...

Bucles Definidos

- Frecuentemente tenemos una lista de elementos – por ejemplo, las líneas en un fichero – un **conjunto finito** de "cosas"
- En Python podemos escribir un bucle que itere sobre cada uno de estos elementos utilizando la construcción **for**
- Estos bucles se denominan “**bucles definidos**” porque ejecutan un número exacto de iteraciones
- Podemos decir que los “bucles definidos iteran sobre los miembros de un conjunto”

Un bucle definido simple

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

5

4

3

2

1

Despegue!

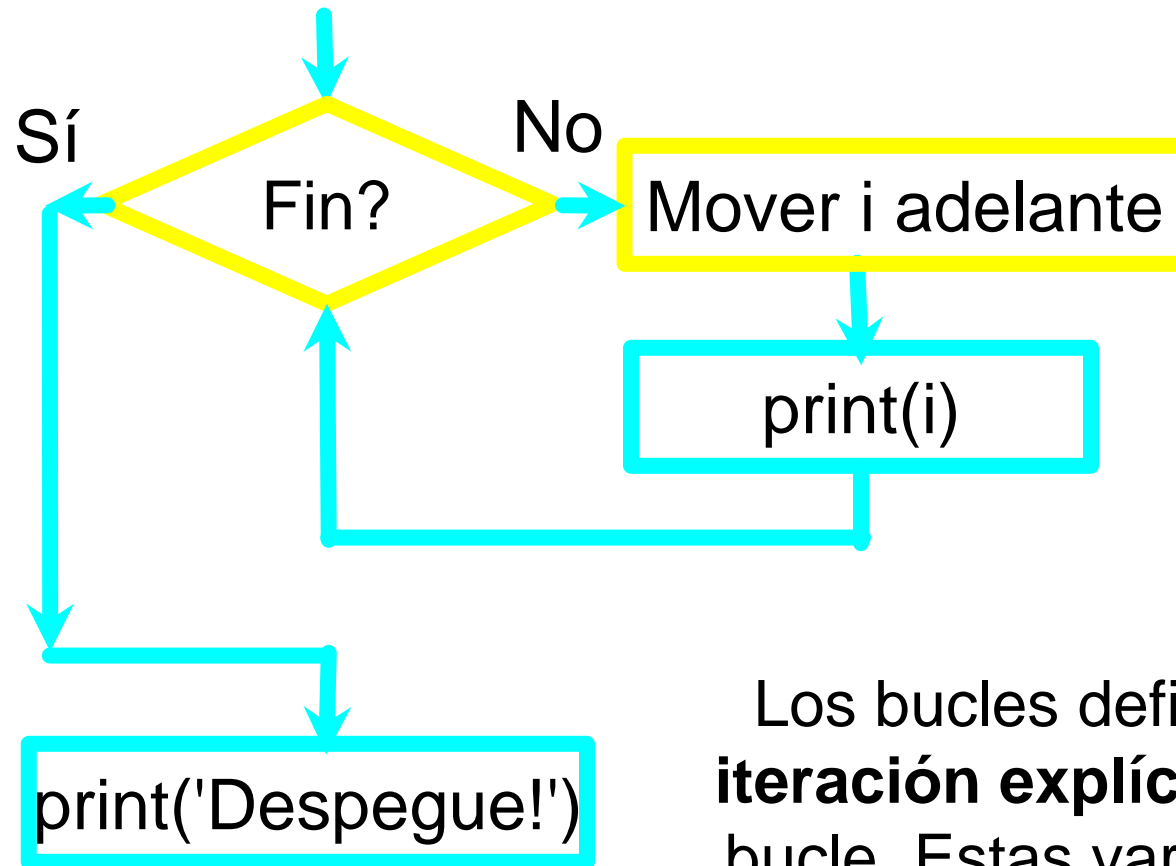
Un bucle definido con strings

```
friends = ['Joseph', 'Glenn', 'Sally']  
for friend in friends :  
    print('Feliz Año Nuevo:', friend)  
print('Hecho!')
```

Feliz Año Nuevo: Joseph
Feliz Año Nuevo : Glenn
Feliz Año Nuevo : Sally

Hecho!

Un bucle definido simple



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

5
4
3
2
1
Despegue!

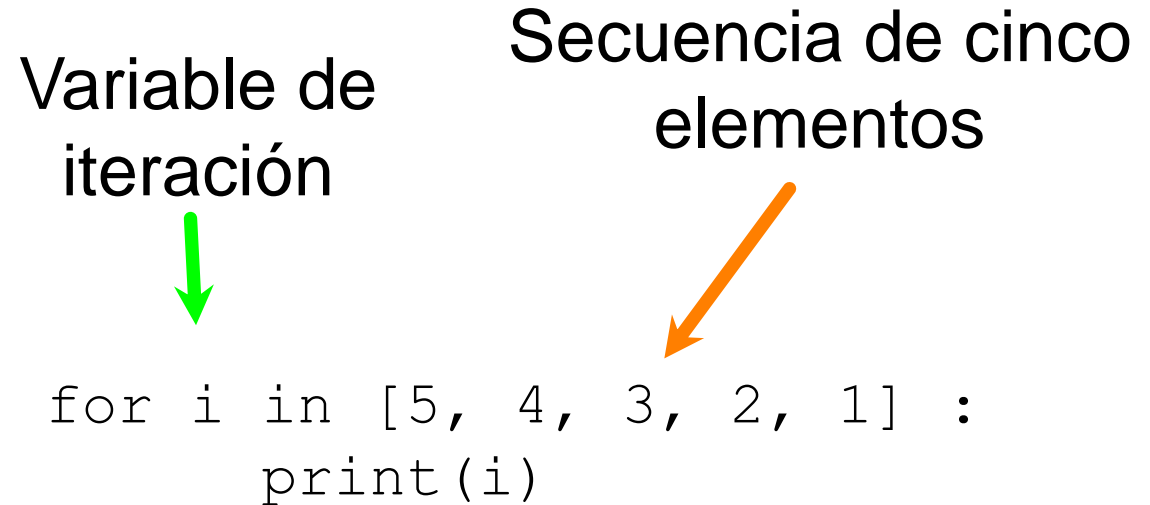
Los bucles definidos (**bucles for**) tiene **variables de iteración explícitas** que cambian en cada iteración del bucle. Estas variables de iteración se mueven a través de la **secuencia o conjunto**.

Mirando in...

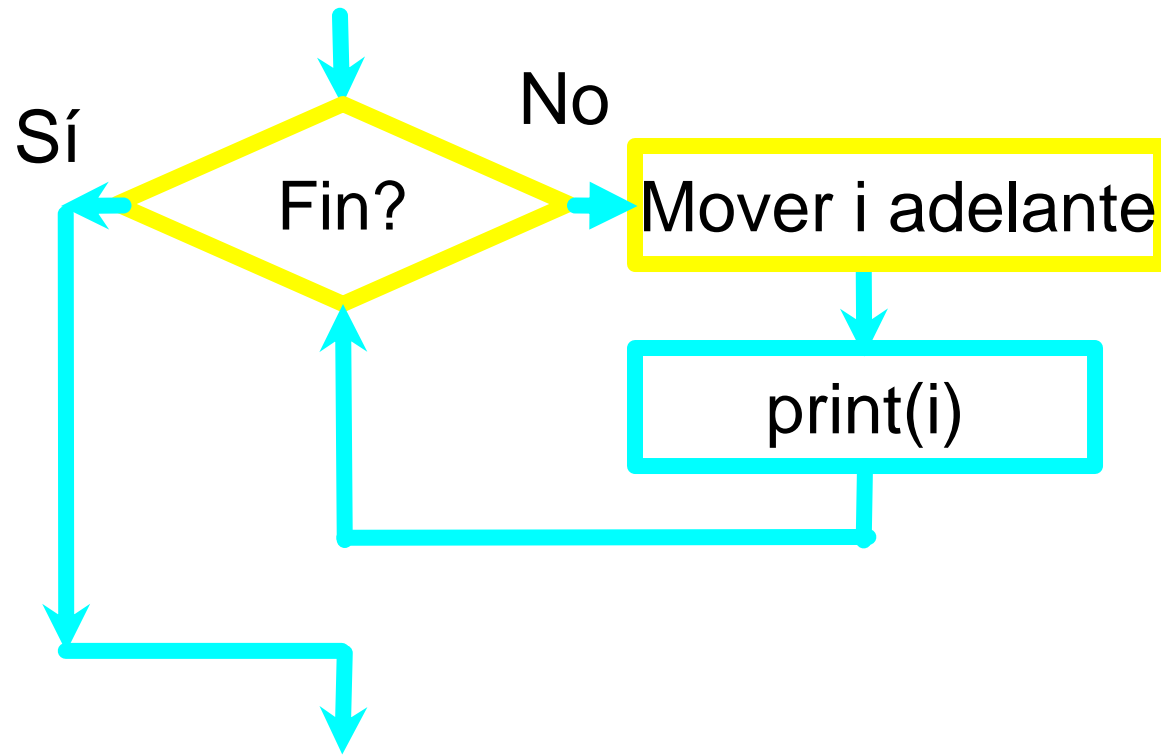
- La variable de iteración “**itera**” a través de la secuencia (conjunto ordenado)
- El bloque (cuerpo) de código es **ejecutado** una vez **para cada valor** en (in) la secuencia
- La variable de iteración se mueve a través de todos los valores en (in) la secuencia

Variable de iteración

Secuencia de cinco elementos

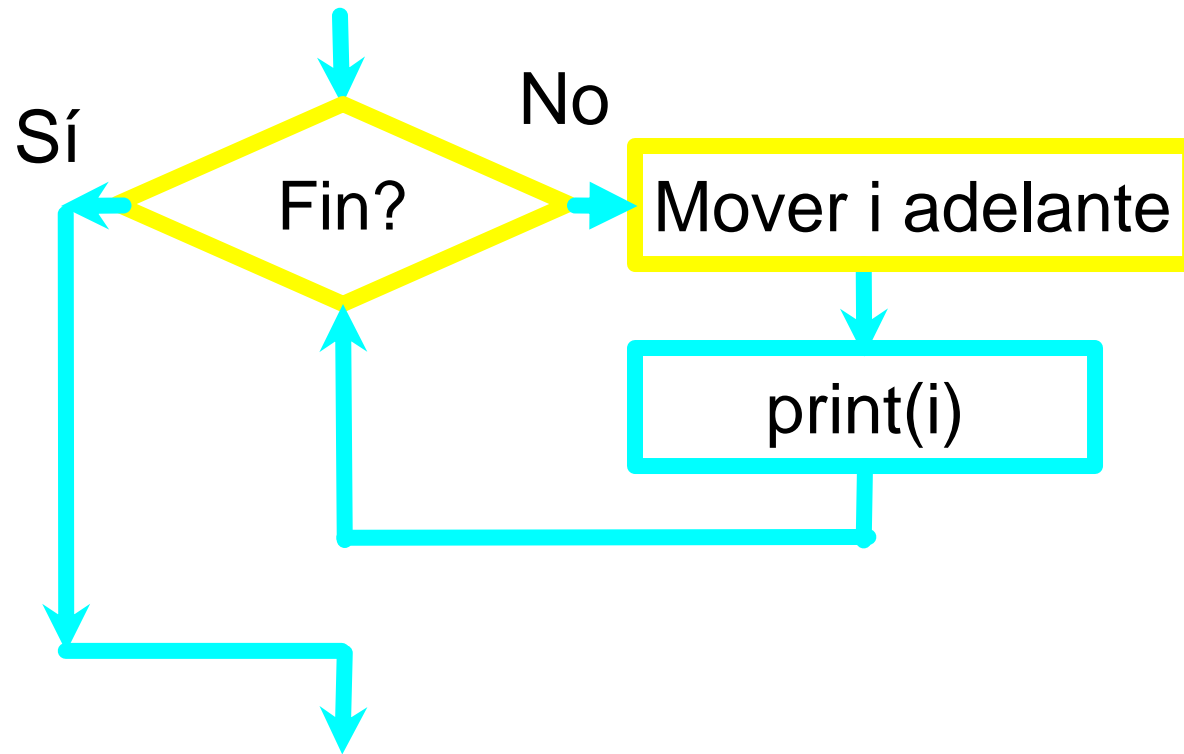


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

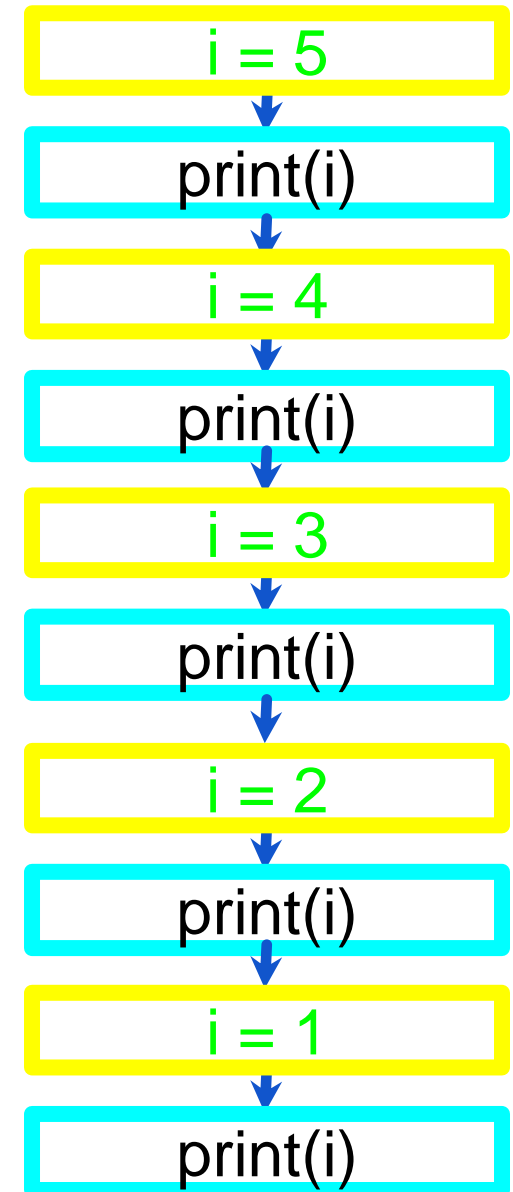


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

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



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



El idioma de los bucles: qué hacemos en los bucles

Nota: Incluso aunque estos ejemplos son simples, los patrones aplican a toda clase de bucles

Haciendo bucles “inteligentes”

El truco es "**saber algo**" en cada momento sobre el “proceso” completo

Cada iteración solo ve una entrada cada vez

Establecer algunas variables a **valores iniciales**

for thing in data:

Buscar algo o hacer algo **para cada entrada** de manera separada, actualizando una variable

Mirar las variables

Iterando a través de un conjunto

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

\$ python basicloop.py

Antes

9

41

12

3

74

15

Después

¿Cuál es el número más grande?

¿Cuál es el número más grande?

3

¿Cuál es el número más grande?

41

¿Cuál es el número más grande?

12

¿Cuál es el número más grande?

9

¿Cuál es el número más grande?

74

¿Cuál es el número más grande?

15

¿Cuál es el número más grande?

¿Cuál es el número más grande?

3 41 12 9 74 15

¿Cuál es el número más grande?

El más
grande por
ahora

largest_so_far

-1

¿Cuál es el número más grande?

3

largest_so_far

3

¿Cuál es el número más grande?

41

largest_so_far

41

¿Cuál es el número más grande?

12

largest_so_far

41

¿Cuál es el número más grande?

9

largest_so_far

41

¿Cuál es el número más grande?

74

largest_so_far

74

¿Cuál es el número más grande?

15

74

¿Cuál es el número más grande?

3 41 12 9 74 15

74

Encontrando el valor mayor

```
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 contiene el valor más grande que hemos **visto por el momento**. Si el número actual que estamos mirando es más grande, entonces ese será el valor más grande que hemos visto por el momento.

Más patrones de bucles...

Contando en un bucle

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

\$ 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 contador** que comienza en 0 y vamos sumando 1 en cada iteración del bucle.

Sumando en un bucle

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

```
$ python sumloop.py
```

```
Antes 0
```

```
9 9
```

```
50 41
```

```
62 12
```

```
65 3
```

```
139 74
```

```
154 15
```

```
Después 154
```

Para **totalizar** un valor en un bucle, introducimos una **variable de suma** que comienza en 0 y sumamos el valor actual a la suma en cada iteración del bucle.

Calculando la media en un bucle

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

\$ 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 25.666

Para la media combinamos los patrones de cuenta y suma y dividimos cuando el bucle acaba.

Filtrando en un bucle

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

```
$ python search1.py
Antes
Número grande 41
Número grande 74
Después
```

Utilizamos una sentencia if en el bucle para capturar/filtrar los valores que estamos buscando.

Búsqueda utilizando una variable booleana

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

```
$ python search2.py
Antes False
False 9
False 41
False 12
True 3
True 74
True 15
Después True
```

Si solo queremos saber si un valor fue encontrado, utilizamos una variable que inicializamos a False y ponemos a True en cuanto encontramos lo que estamos buscando.

Cómo encontrar el valor más pequeño

```
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

¿Cómo podemos cambiarlo para encontrar el valor más pequeño en la lista?

Encontrando el valor más pequeño

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

print('Después', smallest_so_far)
```

Cambiamos el nombre de la variable a `smallest_so_far` e intercambiamos `>` por `<`

Encontrando el valor más pequeño

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

print('Después', smallest_so_far)
```

\$ 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 a `smallest_so_far` e intercambiamos `>` por `<`

Encontrando el valor más pequeño

```
smallest = None
print('Antes')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('Después', smallest)
```

\$ python smallest.py

Antes

9 9

9 41

9 12

3 3

3 74

3 15

Después 3

Todavía tenemos una variable que es la más pequeña por el momento (smallest).

En la primera iteración, smallest vale **None**, así que podemos hacer que value sea el más pequeño por el momento (smallest).

Los operadores is , is not

```
smallest = None
print('Antes')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)

print('Después', smallest)
```

- Python tiene el **operador is**, el cual puede ser usado en expresiones lógicas
- Significa “es lo mismo que”
- Similar a, pero más fuerte que **==**
- **is not** es también un operador lógico

Resumen

- Bucles while (indefinidos)
- Bucles infinitos
- Usando break
- Usando continue
- Constantes None y variables
- Bucles for (definidos)
- Variables de iteración
- El idioma de los bucles
- Más grande o más pequeño

Ejercicio

Escribe un programa que pida números al usuario hasta que escriba 'fin'. Una vez escrito 'fin', el programa debe mostrar el número más grande y el más pequeño. Si el usuario escribe algo diferente a un número, se debe capturar con un try/except e ignorar el valor introducido.

Probar con 7, 2, bob, 10, y 4

Nota: el ejercicio debe resolverse con los visto hasta el momento. No está permitido el uso de otros elementos como listas (que se verán posteriormente)

Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Continue...

Initial Development: Charles Severance, University of Michigan School of Information

... Insert new Contributors and Translators here
Spanish Version: Daniel Garrido (dgm@uma.es)