

Clase 5:

Estructuras de control:

Por casos e iteradas

The logo for Stata, featuring the word "STATA" in a bold, blue, sans-serif font.The logo for Python, featuring the two interlocking snakes (one blue, one yellow) followed by the word "python" in a gray, lowercase, sans-serif font.

Contenido

1. Estructuras de control por casos
 - a) Operadores de comparación y lógicos
 - b) Estructura de control por casos: if, else if, else
 - c) Ejercicio con datos
2. Estructuras de control iteradas
 - a) while
 - b) for
 - c) Anidación

1. Condicionales

STATA



 python



a. Operadores de
comparación y
lógicos

a. Operadores de comparación y lógicos - Python

Operador	Descripción
>	Mayor que. <code>True</code> si el operando de la izquierda es estrictamente mayor que el de la derecha; <code>False</code> en caso contrario.
>=	Mayor o igual que. <code>True</code> si el operando de la izquierda es mayor o igual que el de la derecha; <code>False</code> en caso contrario.
<	Menor que. <code>True</code> si el operando de la izquierda es estrictamente menor que el de la derecha; <code>False</code> en caso contrario.
<=	Menor o igual que. <code>True</code> si el operando de la izquierda es menor o igual que el de la derecha; <code>False</code> en caso contrario.
==	Igual. <code>True</code> si el operando de la izquierda es igual que el de la derecha; <code>False</code> en caso contrario.
!=	Distinto. <code>True</code> si los operandos son distintos; <code>False</code> en caso contrario.

Operadores de comparación

Tomado de:

<https://j2logo.com/python/tutorial/operadores-en-python/>

a. Operadores de comparación y lógicos - Python

Operación	Resultado	Descripción
<code>a or b</code>	Si <code>a</code> se evalúa a falso, entonces devuelve <code>b</code> , si no devuelve <code>a</code>	Solo se evalúa el segundo operando si el primero es falso
<code>a and b</code>	Si <code>a</code> se evalúa a falso, entonces devuelve <code>a</code> , si no devuelve <code>b</code>	Solo se evalúa el segundo operando si el primero es verdadero
<code>not a</code>	Si <code>a</code> se evalúa a falso, entonces devuelve <code>True</code> , si no devuelve <code>False</code>	Tiene menos prioridad que otros operadores no booleanos

Operadores lógicos

Tomado de:
<https://j2logo.com/python/tutorial/operadores-en-python/>

a. Operadores de comparación y lógicos - Python

```
: x=8
: y=9
: x + y
: 
: x<y
: 
: x>=y
: 
: not(x<3)
: 
: not(y>14 or y<10)
: 
: not((not (x < 3)) & (not(y>14 or y<10)))
: 
: resultado =not((not (x < 3)) & (not(y>14 or y<10)))
: resultado
```

- Usando la consola de Spyder

a. Operadores de comparación y lógicos - R

<code><</code>	less than
<code><=</code>	less than or equal to
<code>></code>	greater than
<code>>=</code>	greater than or equal to
<code>==</code>	equal
<code>!=</code>	not equal
<code>!x</code>	not x (negation)
<code>x y</code>	x OR y
<code>x & y</code>	x AND y
<code>xor(x, y)</code>	exclusive OR (either in x or y, but not in both)
<code>isTRUE(x)</code>	truth test for x

- Tomado de R for Data Science
(<https://r4ds.had.co.nz>)

a. Operadores de comparación y lógicos - R

```
> x=8
> y=9
> x + y
[1] 
> x<y
[1] 
> x>=y
[1] 
> !(x < 3)
[1] 
> !(y>14 | y<10)
[1] 
> !(!(x < 3) & !(y>14 | y<10))
[1] 
> |
```

- Usando la consola de RStudio

a. Operadores de comparación y lógicos - Stata

[M-2] `op_logical` — Logical operators
([View complete PDF manual entry](#))

Syntax

<code>a == b</code>	true if <i>a</i> equals <i>b</i>
<code>a != b</code>	true if <i>a</i> not equal to <i>b</i>
<code>a > b</code>	true if <i>a</i> greater than <i>b</i>
<code>a >= b</code>	true if <i>a</i> greater than or equal to <i>b</i>
<code>a < b</code>	true if <i>a</i> less than <i>b</i>
<code>a <= b</code>	true if <i>a</i> less than or equal to <i>b</i>
<code>!a</code>	logical negation; true if <i>a</i> ==0 and false otherwise
<code>a & b</code>	true if <i>a</i> !=0 and <i>b</i> !=0
<code>a b</code>	true if <i>a</i> !=0 or <i>b</i> !=0
<code>a && b</code>	synonym for <i>a & b</i>
<code>a b</code>	synonym for <i>a b</i>

- Documentación de Stata

a. Operadores de comparación y lógicos - MATA

```
. mata
:      x=8
:      y=9
:      x+y
17
:      x<y
1
:      x>=y
0
:      !(!(x < 3) & !(y>14 | y<10))
1
: end
```

- Usando la sintaxis de MATA (lenguaje interno de STATA)

a. Operadores de comparación y lógicos - Stata

```
. scalar x=8  
  
. scalar y=9  
  
. display x<y  
1  
  
. display x>=y  
0  
  
. display !(!(x < 3) & !(y>14 | y<10))  
1
```

- Usando escalares

a. Operadores de comparación y lógicos - Stata

```
. global x=8  
  
. global y=9  
  
. display $x<$y  
1  
  
. display $x>=$y  
0  
  
. display !(!($x < 3) & !($y>14 | $y<10))  
1
```

- Usando macros globales

a. Operadores de comparación y lógicos - Stata

```
. local x=8  
  
. local y=9  
  
. display `x' + `y'  
17  
  
. display `x'<`y'  
1  
  
. display `x'>=`y'  
0  
  
. display !(!(`x' < 3) & !(`y'>14 | `y'<10))  
1
```

- Usando macros locales

a. Operadores de comparación y lógicos - Stata

```
. mata
:      x=8
:      y=9
:      x+y
17
:      x<y
1
:      x>=y
0
:      !!(x < 3) & !(y>14 | y<10))
1
: end
```

```
. global x=8
. global y=9
. display $x<$y
1
. display $x>=$y
0
. display !!(($x < 3) & !($y>14 | $y<10))
1
```

```
. local x=8
. local y=9
. display `x' + `y'
17
. display `x'<`y'
1
. display `x'>=`y'
0
. display !!(`x' < 3) & !(`y'>14 | `y'<10))
1
```

```
. scalar x=8
. scalar y=9
. display x<y
1
. display x>=y
0
. display !!(x < 3) & !(y>14 | y<10))
1
```

a. Operadores de comparación y lógicos

```
: x=8
: y=9
: x + y
: 17
: x<y
: True
: x>=y
: False
: not(x<3)
: True
: not(y>14 or y<10)
: False
: not((not (x < 3)) & (not(y>14 or y<10)))
: True
```

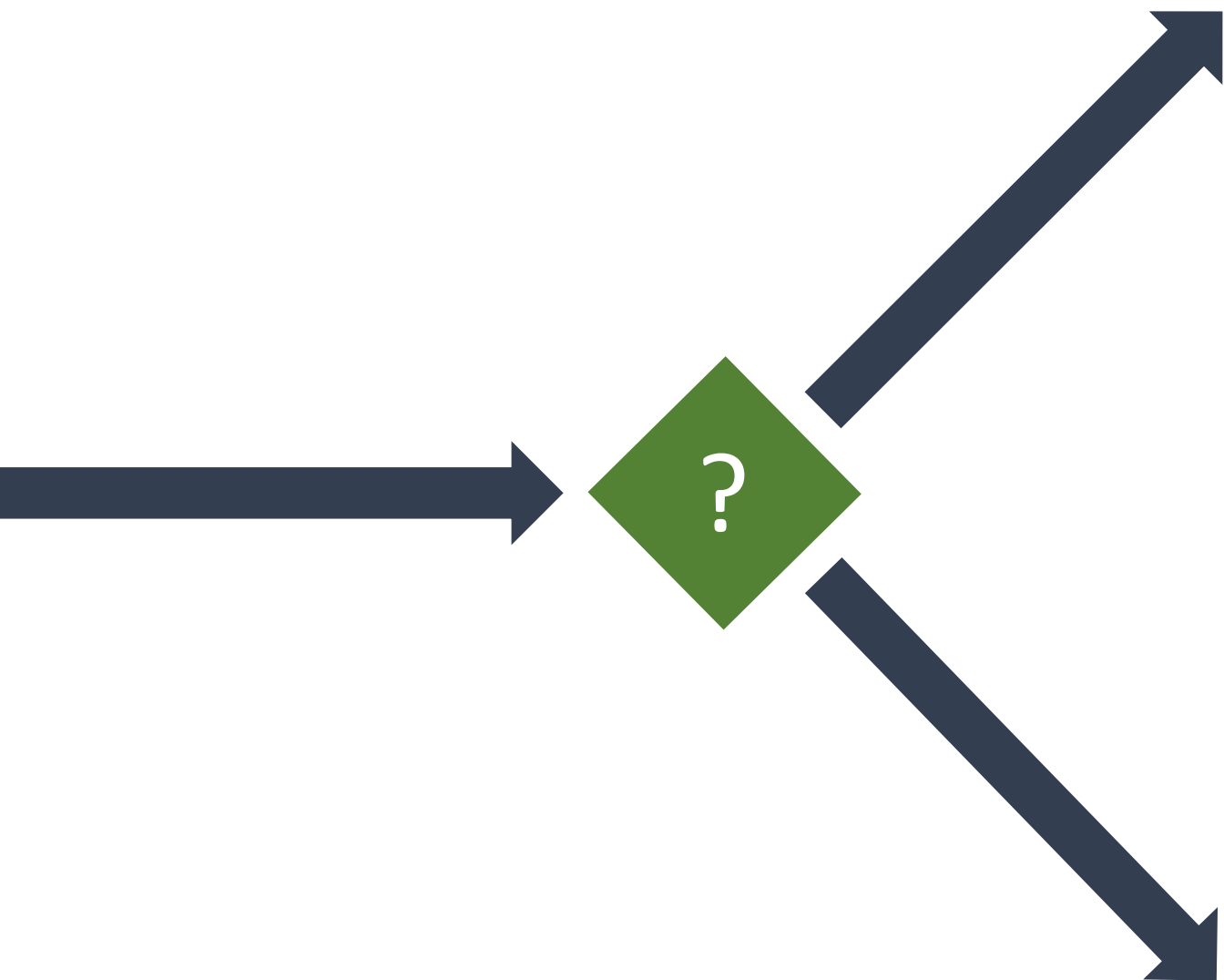


```
> x=8
> y=9
> x + y
[1] 17
> x<y
[1] TRUE
> x>=y
[1] FALSE
> !(x < 3)
[1] TRUE
> !(y>14 | y<10)
[1] FALSE
> !(!(x < 3) & !(y>14 | y<10))
[1] TRUE
> |
```



```
. local x=8
. local y=9
. display `x' + `y'
17
. display `x'<`y'
1
. display `x'>=`y'
0
. display !(!(`x' < 3) & !(`y'>14 | `y'<10))
1
```





b. Estructura de control por casos

b. Estructura de control por casos - Stata

```
global x=8
global y=9
if $x < $y{
    display $x + $y
}
else if $x > $y{
    display $x - $y
}
else {
    display $y - $x
}
```

Do-file

```
. global x=8

. global y=9

. if $x < $y{
.     display $x + $y
. }

. else if $x > $y{
.     display $x - $y
. }

. else {
.     display $y - $x
. }
```

Ventana de comandos - resultados

- Permiten realizar pruebas lógicas con cadenas de texto y números.
- De acuerdo con el resultado el condicional ejecuta algunas instrucciones. Un proceso de verificación caso por caso
- Cuando una de las sentencias es verdadera no se prueban las siguientes.
- Los casos deben ser excluyentes, por construcción.

b. Estructura de control por casos - R

The diagram illustrates the execution of an R script and the resulting command window output, with annotations explaining control flow structures.

Script (Left Panel):

```
11 x=8
12 y=9
13 if (x < y){
14   x + y
15 } else if (x > y) {
16   x - y
17 } else {
18   y - x
19 }
```

Ventana de comandos (Right Panel):

```
> x=8
> y=9
> if (x < y){
+   x + y
+ } else if (x > y) {
+   x - y
+ } else {
+   y - x
+ }
[1] 17
>
```

Annotations:

- Operación lógica en paréntesis (Points to `(x < y)` in the command window)
- Inicio y fin con llaves (Points to the opening and closing curly braces of the `if` statement in the command window)
- else* o *else if* en la misma línea de la llave de fin anterior (Points to the `} else if` line in the command window)
- Resultado solo al final (Points to the output `[1] 17` in the command window)
- ¿En cuál parte de la estructura de control se generó este resultado? (Points to the `else` block in the script)

b. Estructura de control por casos - Python

```
22 x=8
23 y=9
24 if x < y :
25     print(x + y)
26 elif x > y :
27     print(x - y)
28 else:
29     print(y - x)
30
```

Script

```
In [165]: x=8
...: y=9
...: if x < y :
...:     print(x + y)
...: elif x > y :
...:     print(x - y)
...: else:
...:     print(y - x)
17
```

Ventana de comandos

Sin paréntesis y sin llaves.
Solo dos puntos (:)

A diferencia de R, en Python
se debe utilizar función
print() para mostrar el
resultado

-Jerarquía de línea:

La tabulación indica pertenencia a la estructura de control

Salto de línea sin tabulación marca el fin de la estructura de control

b. Estructura de control por casos - Stata

```
42 global x "¡hola"
43 global y "mundo!"
44 if "$x" == "mundo!" {
45     display "$y $x"
46 }
47 else if "$y" == "mundo!"{
48     display "$x $y"
49 }
50 else {
51     display "Hello world!"
52 }
```

Do-file

```
. global x "¡hola"
. global y "mundo!"
. if "$x" == "mundo!" {
.     display "$y $x"
. }
. else if "$y" == "mundo!"{
.     display "$x $y"
. }
. else {
.     display "Hello world!"
. }
```

Ventana de resultados

Se debe colocar entre paréntesis la macro usada como valor de entrada para la verificación en el condicional. Esto es señalar que la macro es un carácter. Comparar caracteres con caracteres.

- Ahora con cadenas de caracteres

Inicio y fin con llaves:
La tabulación para guía visual del código (buena práctica)

b. Estructura de control por casos - R

The diagram illustrates the R IDE interface with two main windows: 'Script' and 'Ventana de comandos' (Command Window). The 'Script' window contains the following R code:

```
x="¡Hola"  
y="mundo!"  
if (x == "mundo!") {  
  mensaje<-paste(y,x, sep = " ")  
  print(mensaje)  
} else if (y == "mundo!") {  
  mensaje<-paste(x,y, sep = " ")  
  print(mensaje)  
} else {  
  print("¡Hola mundo!")  
}
```

An orange arrow points from the 'Script' window to the 'Ventana de comandos' window, which displays the execution output:

```
> x="¡Hola"  
> y="mundo!"  
> if (x == "mundo!") {  
+   mensaje<-paste(y,x, sep = " ")  
+   print(mensaje)  
+ } else if (y == "mundo!") {  
+   mensaje<-paste(x,y, sep = " ")  
+   print(mensaje)  
+ } else {  
+   print("¡Hola mundo!")  
+ }  
[1] "¡Hola mundo!"  
> |
```

Annotations with arrows point to specific parts of the code:

- An arrow points from the text "Objeto definido como caracteres" to the variable `x` in the first line of the code.
- An arrow points from the text "Uso de la función *paste* para concatenar" to the `paste` function call in the first branch of the `if` statement.

Labels at the bottom identify the windows: "Script" and "Ventana de comandos".

b. Estructura de control por casos - Python

```
31 x="¡Hola"
32 y="mundo!"
33 if x == "mundo!":
34     print(y+" "+x)
35 elif y == "mundo!":
36     print(x+" "+y)
37 else:
38     print("¡Hola mundo!")
```

Script

```
In [166]: x="¡Hola"
...: y="mundo!"
...: if x == "mundo!":
...:     print(y+" "+x)
...: elif y == "mundo!":
...:     print(x+" "+y)
...: else:
...:     print("¡Hola mundo!")
¡Hola mundo!
```

Ventana de comandos

Concatenación directa de
caracteres con (+).
Similar al uso de escalares con
Stata

¡Ejercicio!

Tome las “variables” en el lenguaje que desee:

Andres = “Enemigo mío”

Julian = “Amigo mío”

Y construya una estructura if-else que muestre en pantalla si Andrés y Julián son amigos o enemigos.

Note la regla:

El amigo de mi amigo es mi amigo

El amigo de mi enemigo es mi enemigo

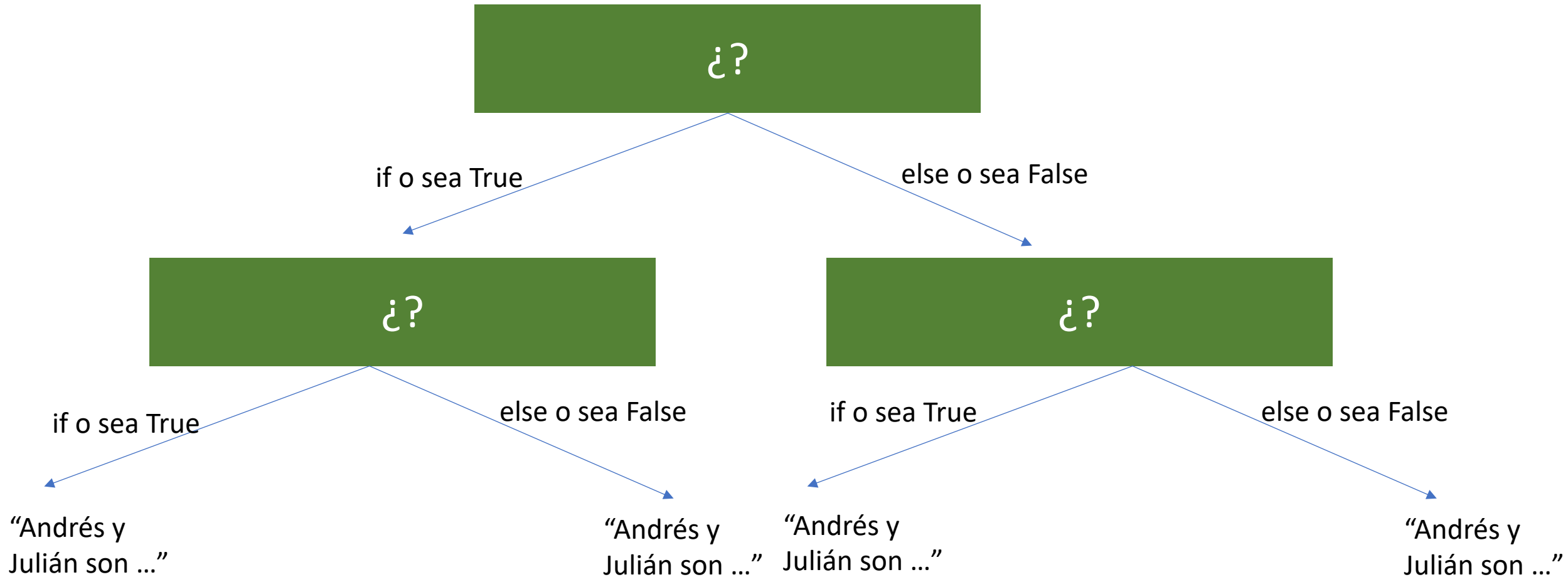
El enemigo de mi amigo es mi amigo

El enemigo de mi enemigo es mi amigo

Pista: haga primero una pregunta y adentro la otra

Dibuje primero el arbolito

Note la regla:



2. Estructuras de control iteradas

STATA



 python

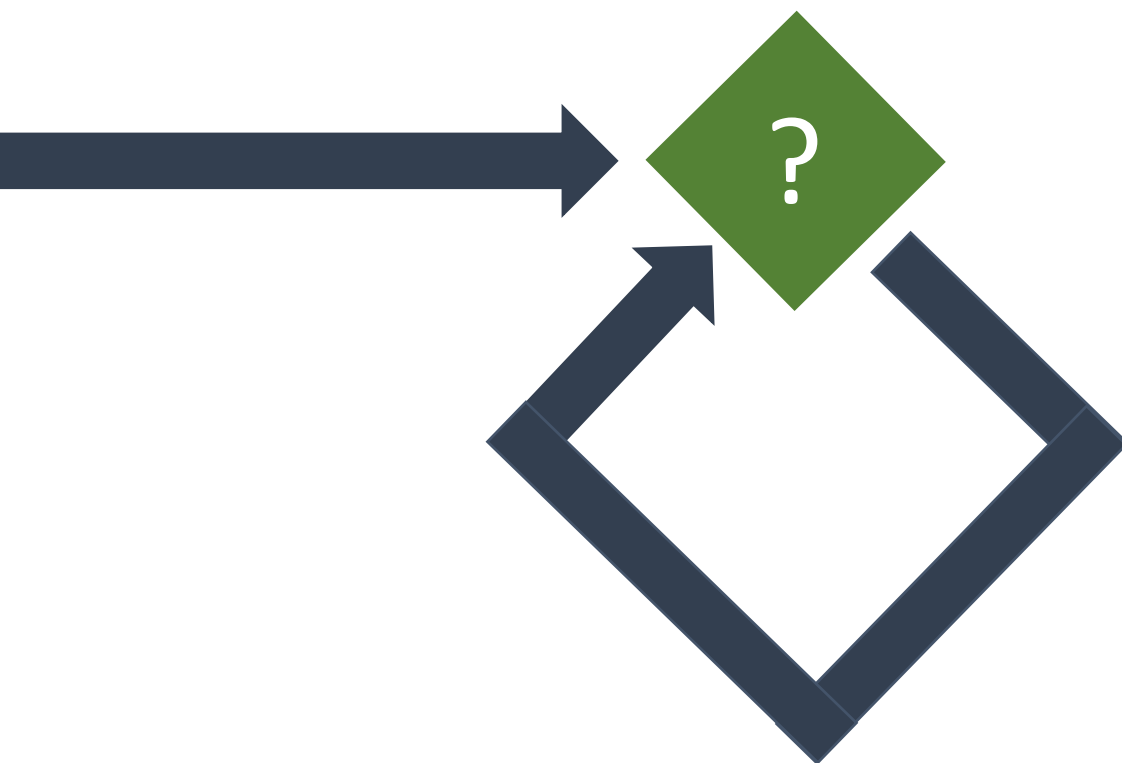
Principales estructuras de control iteradas (ciclos iterados)

While

- Comúnmente usado en simulaciones.
- Ejecución de instrucciones mientras el resultado de una operación lógica sea “Verdadero”. La ejecución se detiene cuando el resultado de la operación es “Falso”
- La ejecución de controla a través de patrones de números.
- Cuando el control está mal definido puede haber un ciclo sin fin.

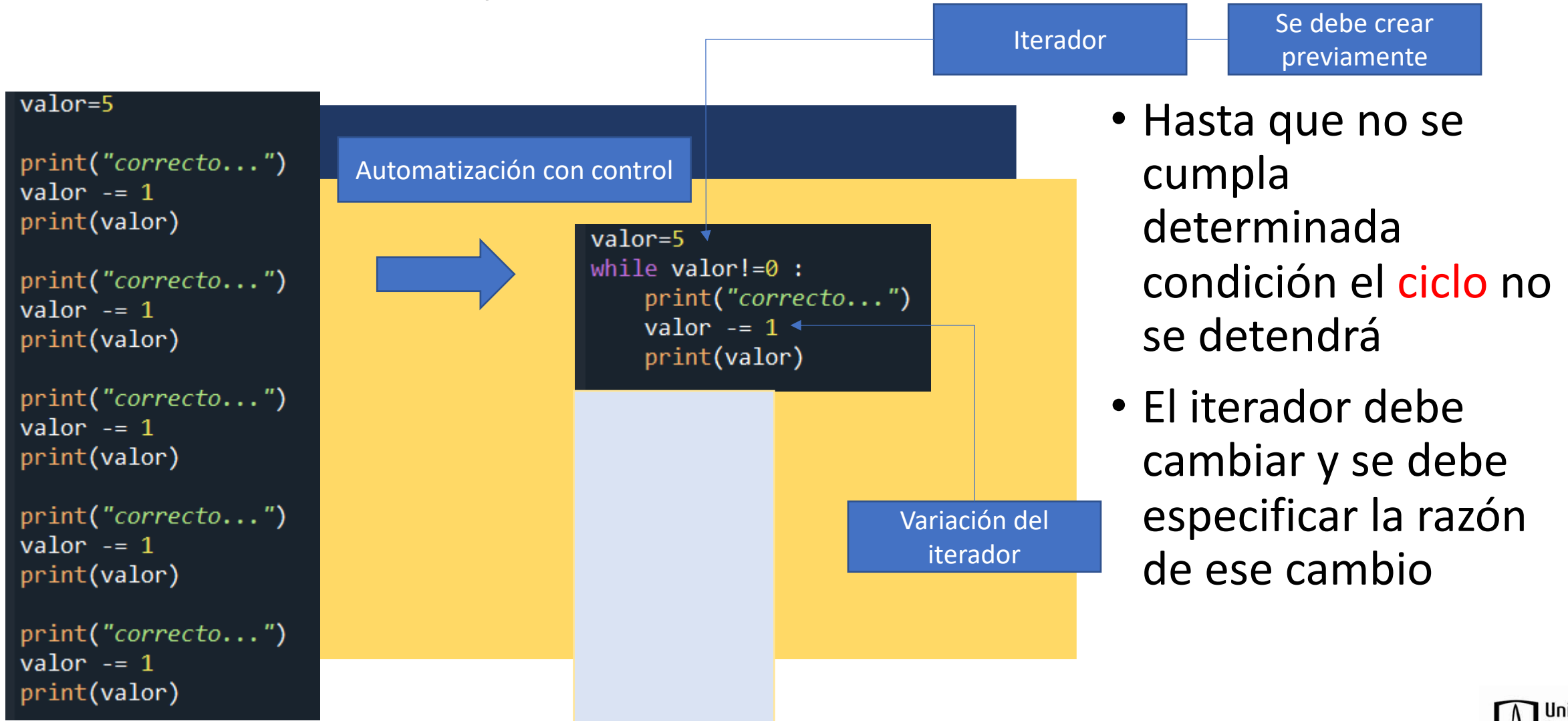
For

- Muy usado en procesamiento de datos.
- Ejecución de instrucciones sobre un número definido de elementos, que pueden ser números o cadenas de caracteres.
- Es preferible que los elementos sobre los que se itera tengan algún patrón.



a. while

a. While – Python



- Hasta que no se cumpla determinada condición el **ciclo** no se detendrá
- El iterador debe cambiar y se debe especificar la razón de ese cambio

a. While – Python

```
valor=5

print("correcto...")
valor -= 1
print(valor)

print("correcto...")
valor -= 1
print(valor)

print("correcto...")
valor -= 1
print(valor)

print("correcto...")
valor -= 1
print(valor)

print("correcto...")
valor -= 1
print(valor)
```



```
valor=5
while valor!=0 :
    print("correcto...")
    valor -= 1
    print(valor)
```

Sin paréntesis y sin llaves. Solo dos puntos (:)

Escritura sintética con operador aritmético

Para sumar 1 utilizar el operador +=

-Jerarquía de línea:
La tabulación indica pertenencia al ciclo iterado.
Salto de línea sin tabulación marca el fin de la estructura de control

a. While – R

```
valor=5
while (valor!=0){
  print("correcto...")
  valor=valor-1
  print(valor)
}
```

Operación lógica en
paréntesis

Sintaxis recursiva
estándar

Inicio y fin con llaves:
La tabulación para guía
visual del código (buena
práctica)

```
> valor=5
> while (valor!=0){
+   print("correcto...")
+   valor=valor-1
+   print(valor)
+ }
[1] "correcto..."
[1] 4
[1] "correcto..."
[1] 3
[1] "correcto..."
[1] 2
[1] "correcto..."
[1] 1
[1] "correcto..."
[1] 0
> |
```


a. While – Stata

Con While, el iterador (o local de control) se crea previamente

Operación lógica sin paréntesis

```
local valor=5
while `valor' != 0 {
    display "correcto..."
    local --valor
    display `valor'
}
```

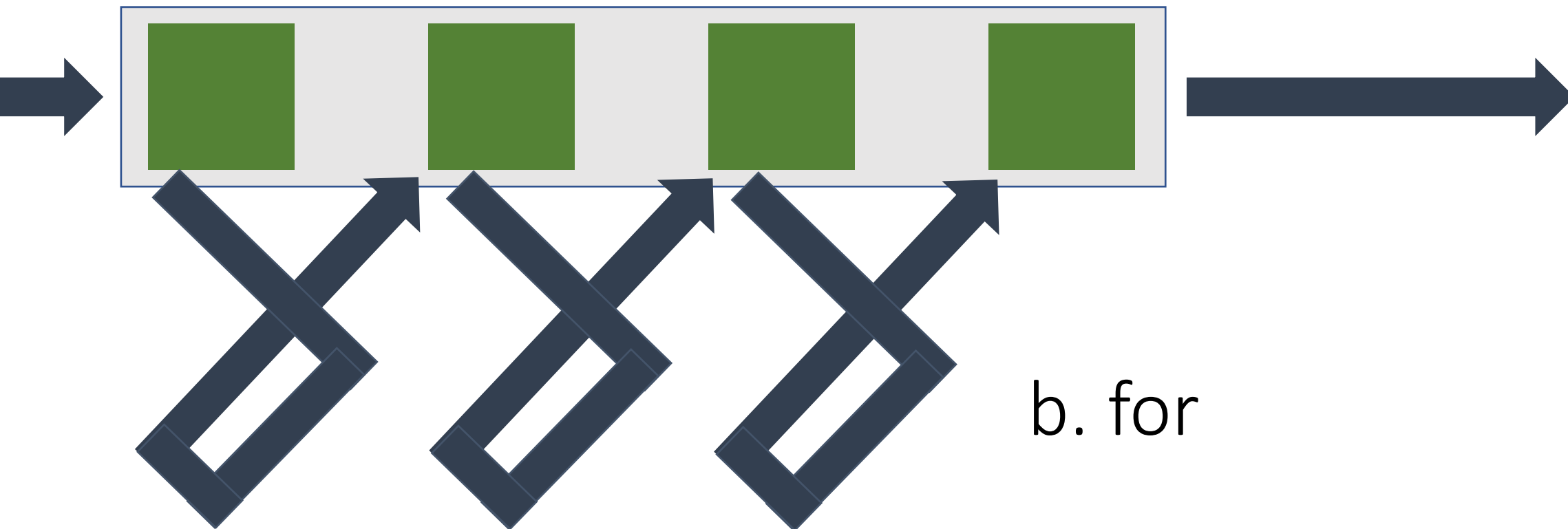
```
correcto...
4
correcto...
3
correcto...
2
correcto...
1
correcto...
0
```

Se necesita una macro para controlar la ejecución del ciclo iterado.

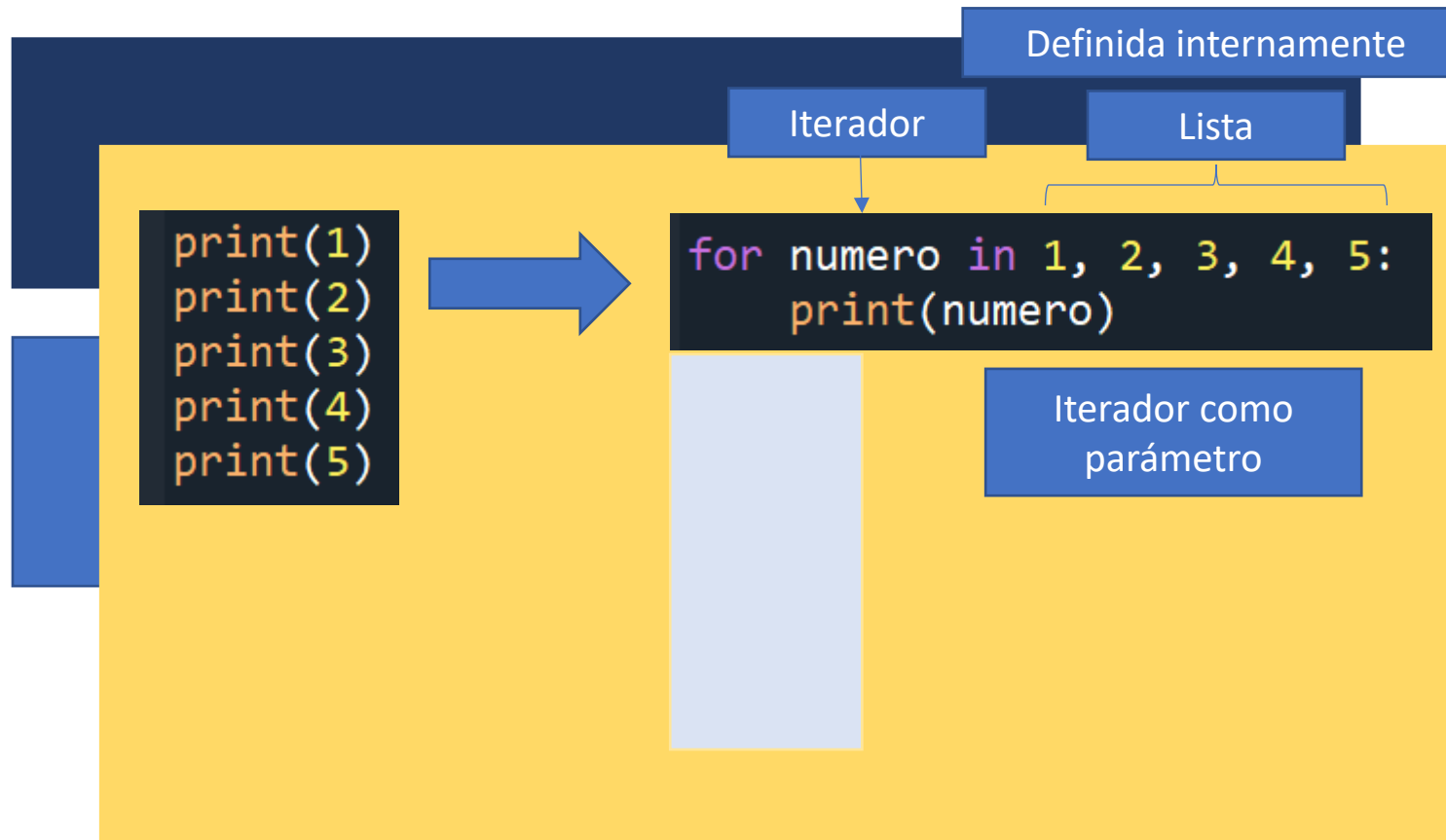
Por la notación de la macro, el uso del iterador es muy notorio, permite ver fácilmente cómo se controla la ejecución del ciclo y corregir errores.

Resta 1 al local valor en cada iteración

Para sumar 1 al local valor usar la sintaxis `local ++valor`

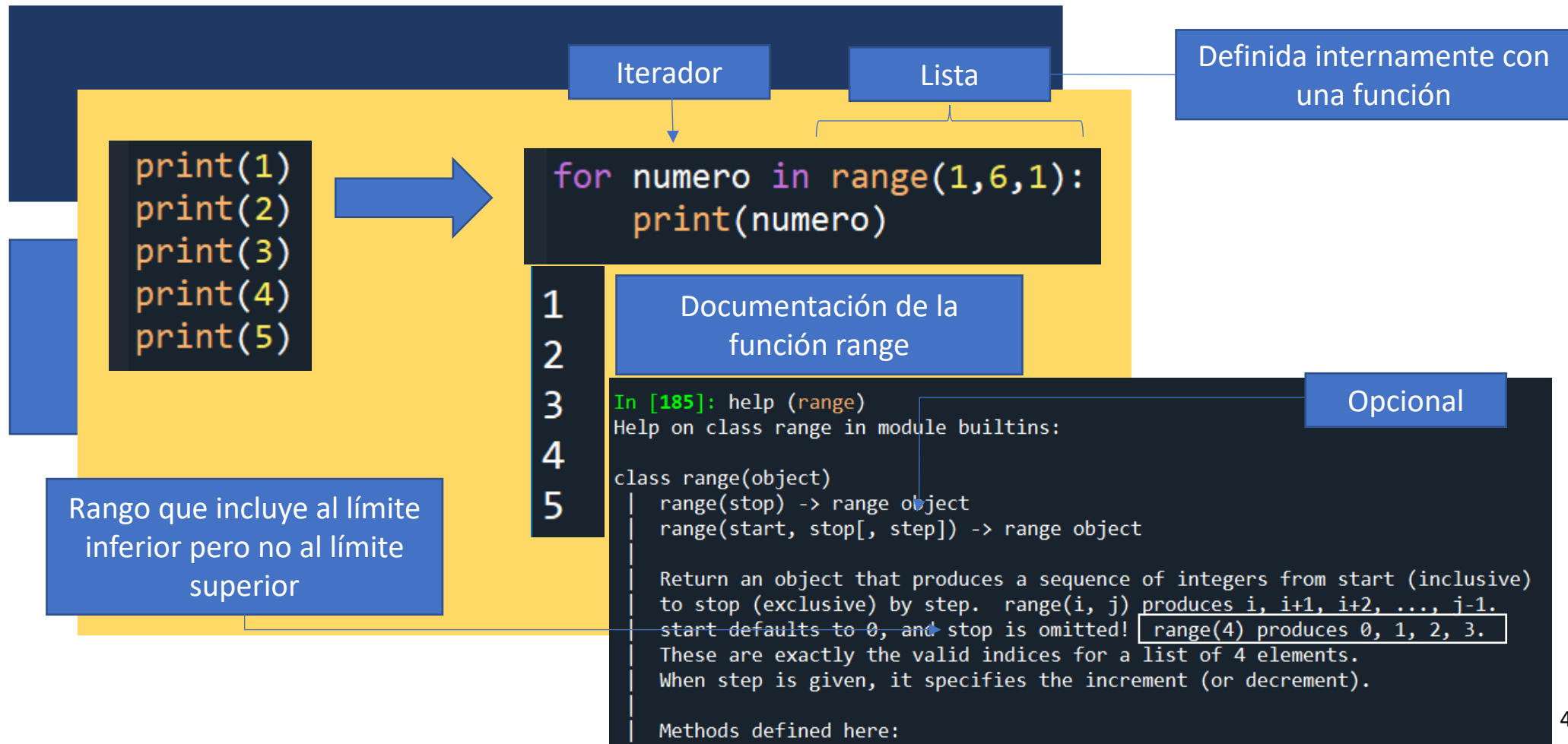


b. For con números – Python

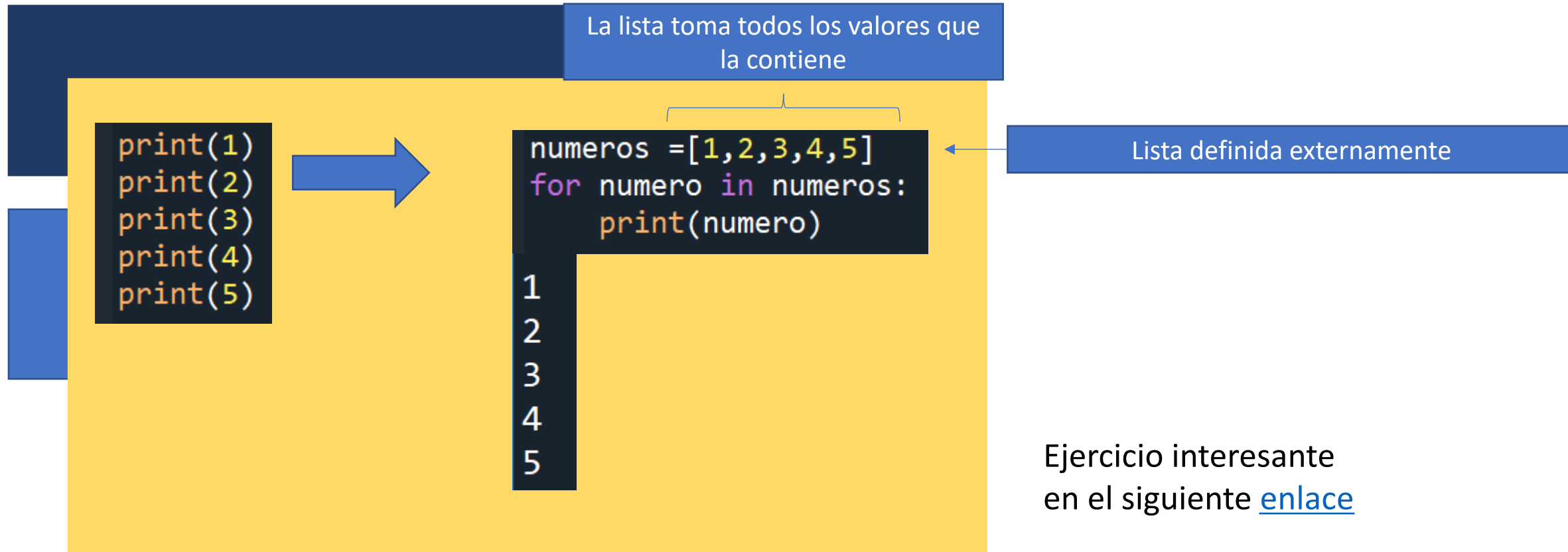


- Es un ciclo que recorre una lista, pueden ser textos o números.
- Aquí la lista está definida de forma interna, esto es, en la sintaxis del comando `for`.
- El iterador puede tener cualquier nombre con caracteres, como una letra o una palabra.

b. For con números – Python



b. For con números – Python



b. For con números – R

```
print(1)  
print(2)  
print(3)  
print(4)  
print(5)
```



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

Control de iteración en
paréntesis

Lista definida internamente con un
patrón numérico

Inicio y fin con llaves:
La tabulación para guía
visual del código (buena
práctica)

b. For con números – R

```
print(1)
print(3)
print(8)
print(9)
print(10)
print(15)
```

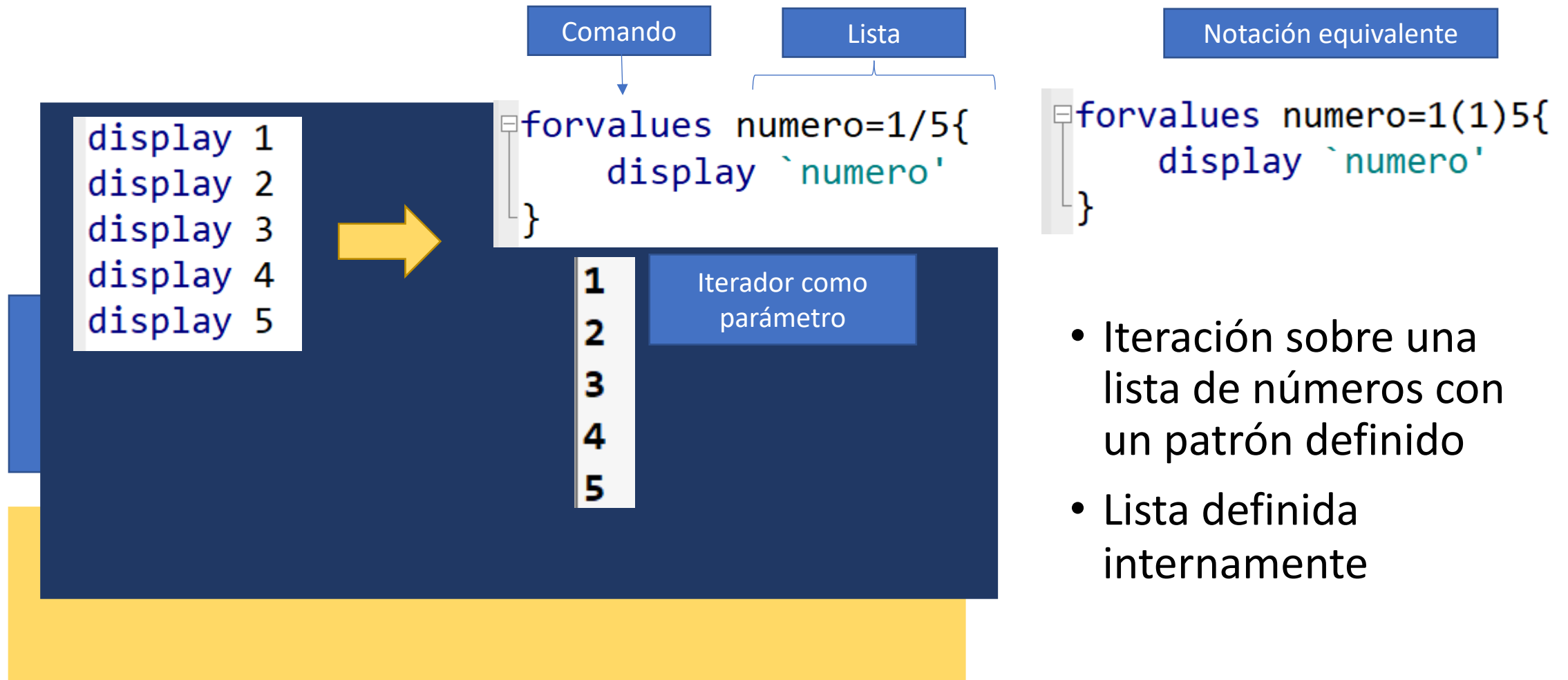


```
numeros=c(1,3,8,9,10,15)
for (i in numeros ){
  print(i)
}
```

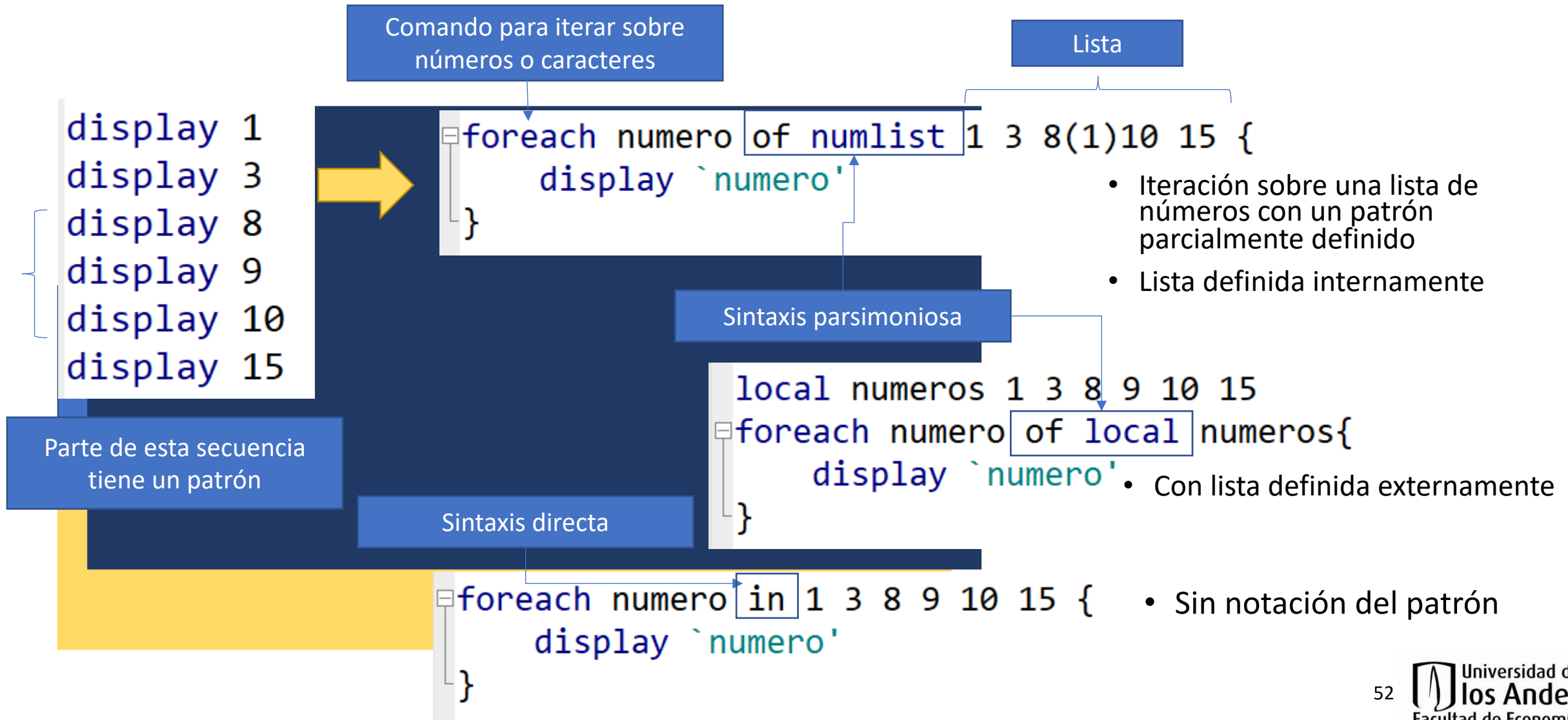
```
[1] 1
[1] 3
[1] 8
[1] 9
[1] 10
[1] 15
```

Lista definida externamente sin patrón numérico definido

b. Forvalues – Stata



b. Foreach con números – Stata



b. For con cadenas de caracteres – Python

```
print("a")  
print("n")  
print("r")  
print("t")  
print("k")  
print("o")
```

```
for letra in "a", "n", "r", "t", "k", "o" :  
    print(letra)
```

Lista definida internamente

```
letras = ["a","n","r","t","k","o"]  
for letra in letras:  
    print(letra)
```

Lista definida externamente

a
n
r
t
k
o

b. For con cadenas de caracteres – R

```
print("a")  
print("n")  
print("r")  
print("t")  
print("k")  
print("o")
```



```
letras<-c("a", "n", "r", "t", "k", "o")  
for (letra in letras) {  
  print(letra)  
}
```

```
[1] "a"  
[1] "n"  
[1] "r"  
[1] "t"  
[1] "k"  
[1] "o"
```

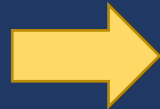
Lista definida externamente con las cadenas de caracteres

- Las listas definidas internamente en los comandos de ciclos de R solo pueden ser con patrones numéricos

R tiene una familia de comandos para realizar operaciones repetitivas a través de listas. Se denominan la familia *apply* (apply, sapply, lapply). Su estudio excede el objetivo de este curso pero puede ser consultado en la bibliografía recomendada al final de esta presentación.

b. Foreach con cadenas de caracteres – Stata

```
display "a"  
display "n"  
display "r"  
display "t"  
display "k"  
display "o"
```



```
foreach letra in a n r t k o {  
    display "`letra'"  
}
```

Sintaxis directa

```
local letras a n r t k o  
foreach letra of local letras {  
    display "`letra'"  
}
```

Sintaxis de lista macro
local

```
global letras a n r t k o  
foreach letra of global letras {  
    display "`letra'"  
}
```

Sintaxis de lista global
local

Stata cuenta con una sintaxis especial para programar operaciones repetitivas con variables de bases de datos que veremos en el módulo 4 de este curso.

b. For (Ejemplo)

Probemos:

En un grupo, conformado por María, Pedro, Juan, Alonso e Iván, se hace un registro de su orden de llegada a una sala. Un dispositivo genera un mensaje con los nombres de las personas en la sala así:

“Acaba de entrar: ” _____

Con un ciclo iterado programe los sucesivos mensajes que emite el dispositivo si los integrantes del grupo entran en el orden dicho anteriormente.

b. For – Python (Ejemplo)

Solución en clase iterando sobre la lista de los nombres

```
nombres = ['Maria', 'Pedro', 'Juan', 'Alonso', 'Iván']
```

```
for nombreActual in nombres:  
    print('Acaba de entrar: '+nombreActual)
```

```
Acaba de entrar: Maria  
Acaba de entrar: Pedro  
Acaba de entrar: Juan  
Acaba de entrar: Alonso  
Acaba de entrar: Iván
```

b. For (Variación)

Probemos:

En un grupo, conformado por María, Pedro, Juan, Alonso e Iván, cuando van entrando se van acumulando y sale el mensaje:

“Las personas en la sala son: ” _____

Con un ciclo iterado programe los sucesivos mensajes que emite el dispositivo si los integrantes del grupo entran en el orden dicho anteriormente.

b. For – Python (Ejemplo)

Solución en clase iterando sobre la lista de los nombres

```
lista = ["María", "Pedro", "Juan", "Alonso", "Iván"]  
mensaje="Las personas en la sala son: "  
  
for nombre in lista:  
    mensaje=mensaje+" "+nombre  
    print(mensaje)
```

Escritura recursiva con cadenas de caracteres sobre un objeto mutable

```
Las personas en la sala son: María  
Las personas en la sala son: María Pedro  
Las personas en la sala son: María Pedro Juan  
Las personas en la sala son: María Pedro Juan Alonso  
Las personas en la sala son: María Pedro Juan Alonso Iván
```


b. For – Python (Ejemplo) – Solución en clase iterando sobre una lista de números

0

1

2

3

4

Este índice permite programar el orden de llegada con números

```
nombres = ["María", "Pedro", "Juan", "Alonso", "Iván"]  
mensaje="Las personas en la sala son: "
```

```
for i in 0,1,2,3,4: #El orden de la lista "nombres"  
    mensaje=mensaje+" "+nombres[i]  
    print(mensaje)
```

Usando una lista interna con el mismo orden definido en la lista "nombres"

```
Las personas en la sala son: María  
Las personas en la sala son: María Pedro  
Las personas en la sala son: María Pedro Juan  
Las personas en la sala son: María Pedro Juan Alonso  
Las personas en la sala son: María Pedro Juan Alonso Iván
```

b. For – Python (Ejemplo) – Solución en clase cambiando el orden de llegada

```
nombres = ["María", "Pedro", "Juan", "Alonso", "Iván"]  
mensaje="Las personas en la sala son: "  
  
for i in 4,2,1,3,0: #Orden distinto de llegada"  
    mensaje=mensaje+" "+nombres[i]  
    print(mensaje)
```

Usando un orden distinto de llegada de personas a la sala. Se aprovecha el índice de la lista definida al inicio

```
Las personas en la sala son: Iván  
Las personas en la sala son: Iván Juan  
Las personas en la sala son: Iván Juan Pedro  
Las personas en la sala son: Iván Juan Pedro Alonso  
Las personas en la sala son: Iván Juan Pedro Alonso María
```

b. For – Python (Ejemplo) – usando la función len()

0

1

2

3

4

```
nombres = ["María", "Pedro", "Juan", "Alonso", "Iván"]
print(len(nombres))
mensaje="Las personas en la sala son: "

for i in range(len(nombres)):
    print(nombres[i])
    mensaje=mensaje+" "+nombres[i]
    print(mensaje)
```

Lista definida externamente con cadenas de caracteres

```
In [192]: help(len)
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

len(nombres) = 5
La lista es: 0,1,2,3,4
Se conserva el orden de llegada

Escritura indexada con cadenas de caracteres

```
5
María
Las personas en la sala son:  María
Pedro
Las personas en la sala son:  María Pedro
Juan
Las personas en la sala son:  María Pedro Juan
Alonso
Las personas en la sala son:  María Pedro Juan Alonso
Iván
Las personas en la sala son:  María Pedro Juan Alonso Iván
```

b. For – R (Ejemplo)

```
nombres = list("María","Pedro","Juan","Alonso","Iván")
largo<-print(length(nombres))
mensaje="Las personas en la sala son:"
```

Lista como objeto
(Lista de cadenas de caracteres)

```
for (i in nombres) {
  print(i)
  mensaje<-paste(mensaje,i, sep = " ", collapse = NULL)
  print(mensaje)
}
```

Mensaje de inicio como objeto

Concatenar caracteres (con función)
Operación recursiva

```
[1] "María"
[1] "Las personas en la sala son: María"
[1] "Pedro"
[1] "Las personas en la sala son: María Pedro"
[1] "Juan"
[1] "Las personas en la sala son: María Pedro Juan"
[1] "Alonso"
[1] "Las personas en la sala son: María Pedro Juan Alonso"
[1] "Iván"
[1] "Las personas en la sala son: María Pedro Juan Alonso Iván"
```

¿Cómo lo programaría si quisiera
iterar sobre números?

Aprovechar el índice de la lista
"nombres". Similar a Python

b. Foreach – Stata (Ejemplo)

```
local nombres María Pedro Juan Alonso Iván  
scalar mensaje="Las personas en la sala son:"
```

```
foreach nombre of local nombres{  
    display "`nombre'  
    scalar temp= "`nombre'  
    scalar mensaje=mensaje+" "+temp  
    display mensaje  
}
```

```
María  
Las personas en la sala son: María  
Pedro  
Las personas en la sala son: María Pedro  
Juan  
Las personas en la sala son: María Pedro Juan  
Alonso  
Las personas en la sala son: María Pedro Juan Alonso  
Iván  
Las personas en la sala son: María Pedro Juan Alonso Iván
```

Lista en una macro local

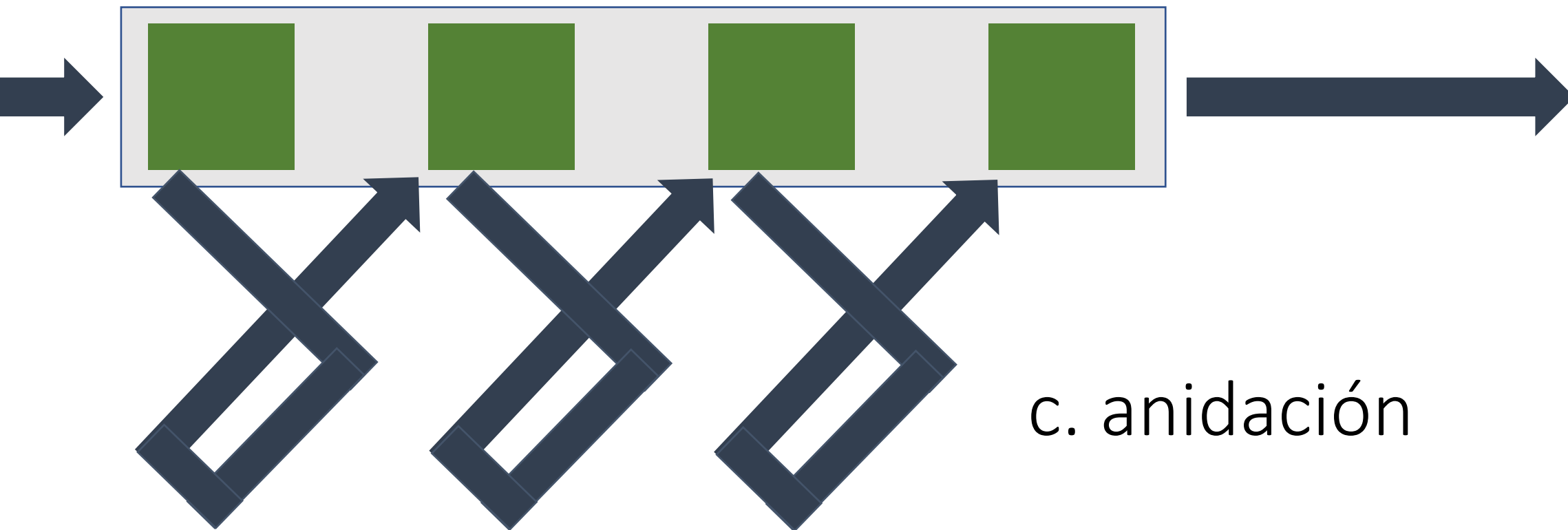
Mensaje de inicio en un elemento escalar (caracteres)

Convertir un elemento que esta dentro de una macro (iterador) a caracteres para concatenar

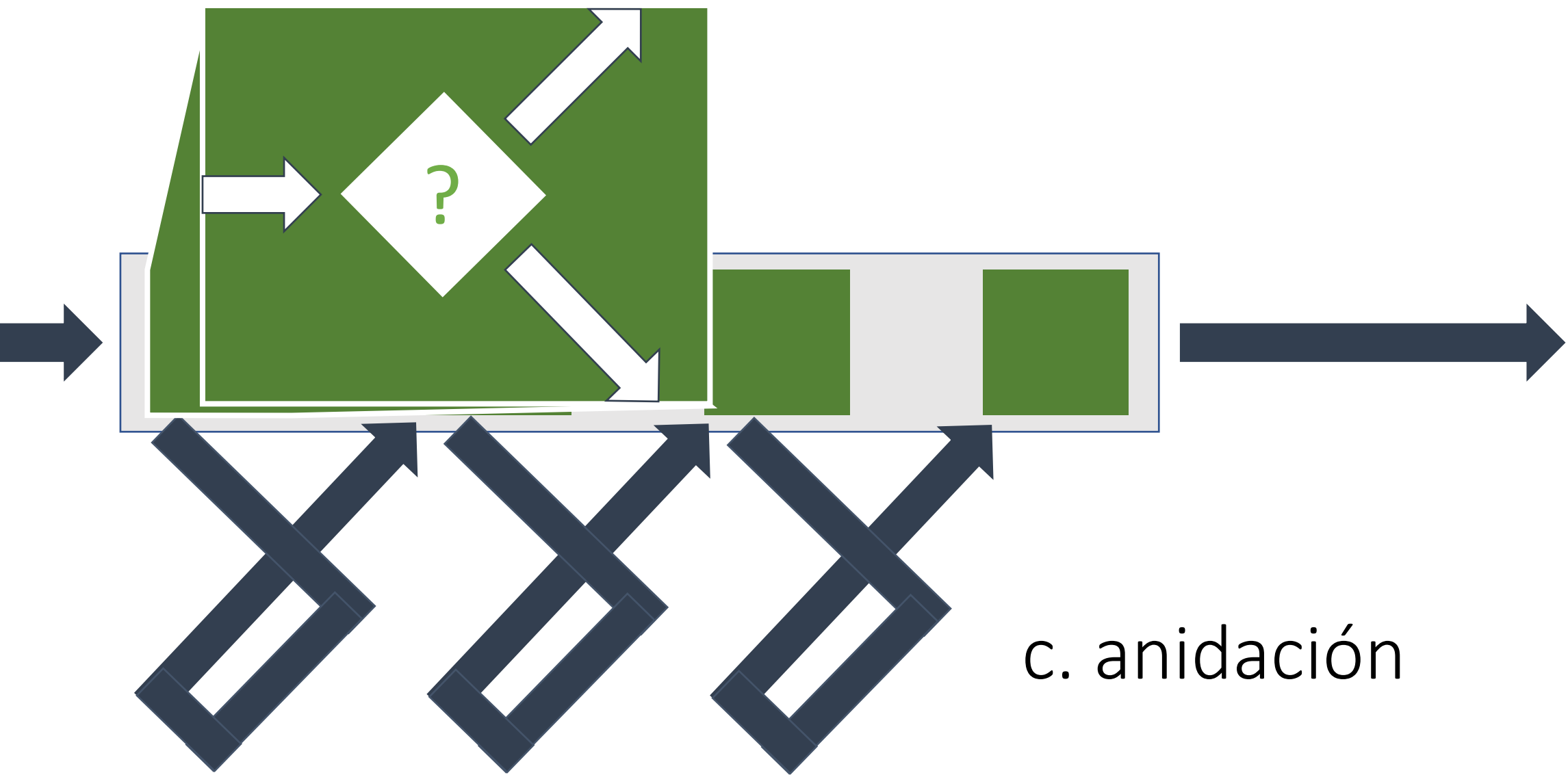
Concatenar caracteres
(similar a Python)
Operación recursiva

¿Cómo lo programaría si quisiera iterar sobre números?

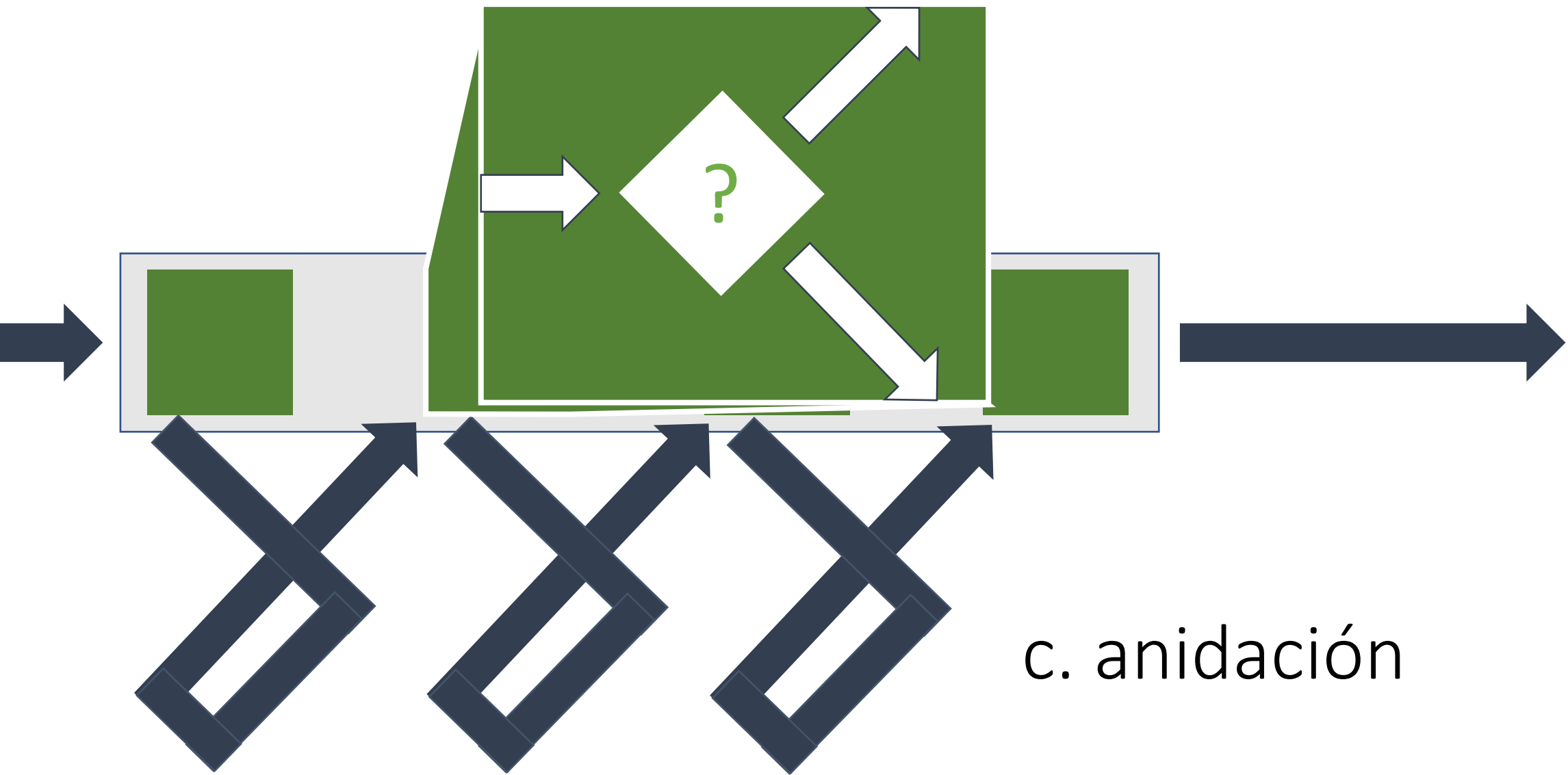
El comando tokenize puede ayudar



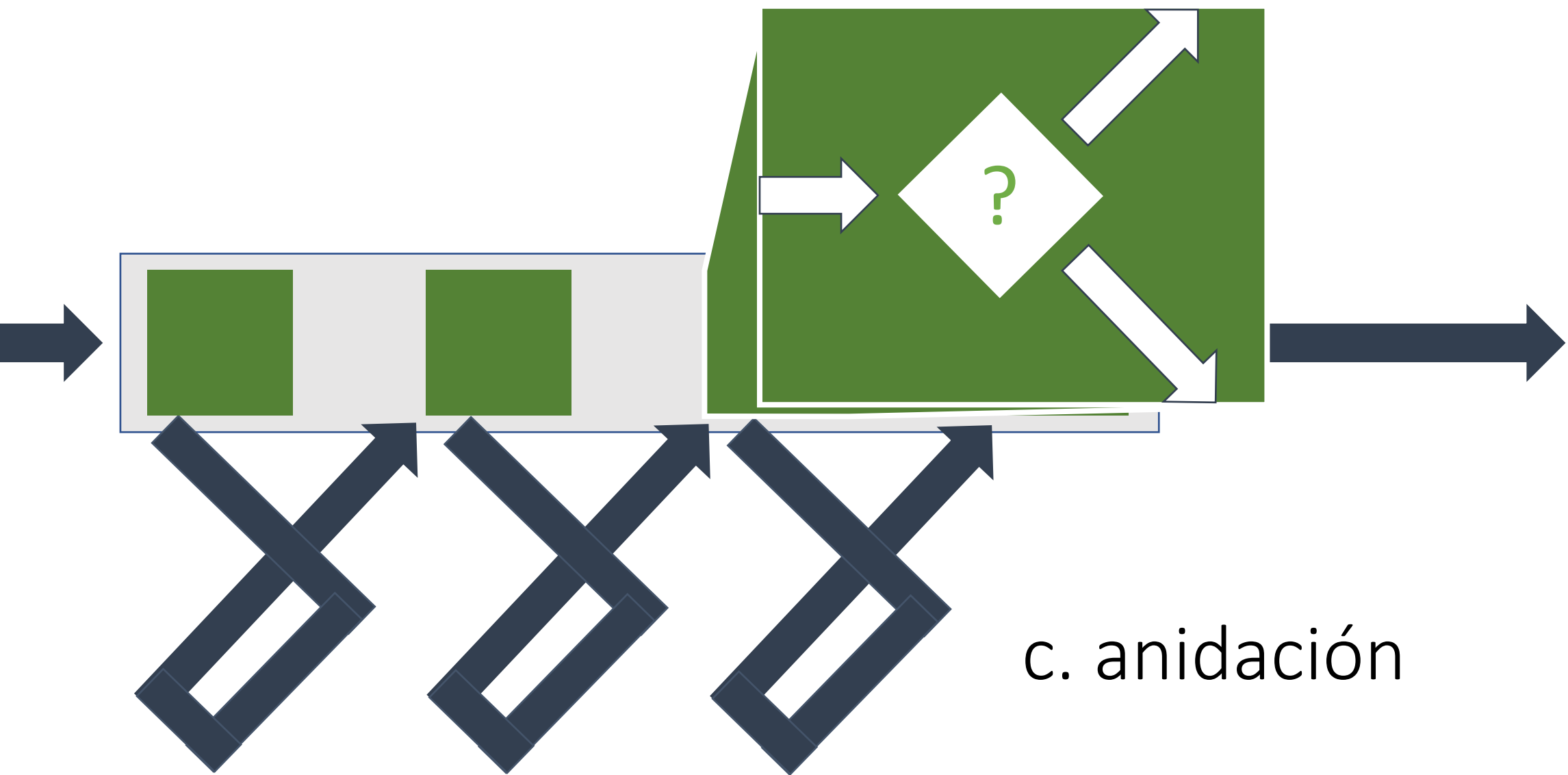
c. anidación



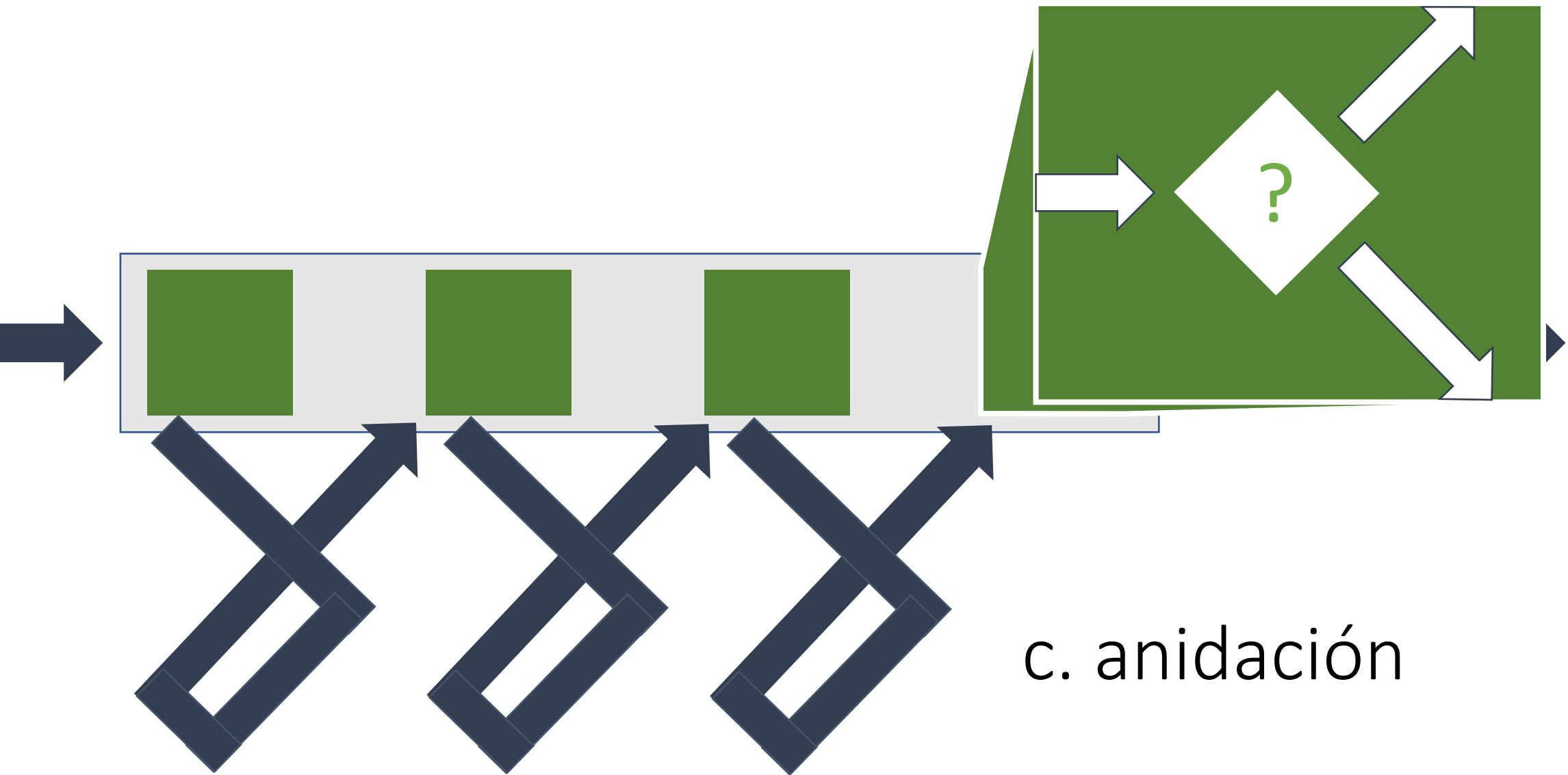
c. anidación



c. anidación



c. anidación



c. anidación

c. Anidación 1 – Python

```
for i in range(6):  
    if i<3:  
        print("valor menor a 3")  
    else:  
        print("valor mayor o igual a 3")
```

Tabulación por jerarquía de código

Condicional interno

Ciclo principal

```
valor menor a 3  
valor menor a 3  
valor menor a 3  
valor mayor o igual a 3  
valor mayor o igual a 3  
valor mayor o igual a 3
```

- ¿Cuántas veces se ejecuta el condicional interno?

c. Anidación 1 – R

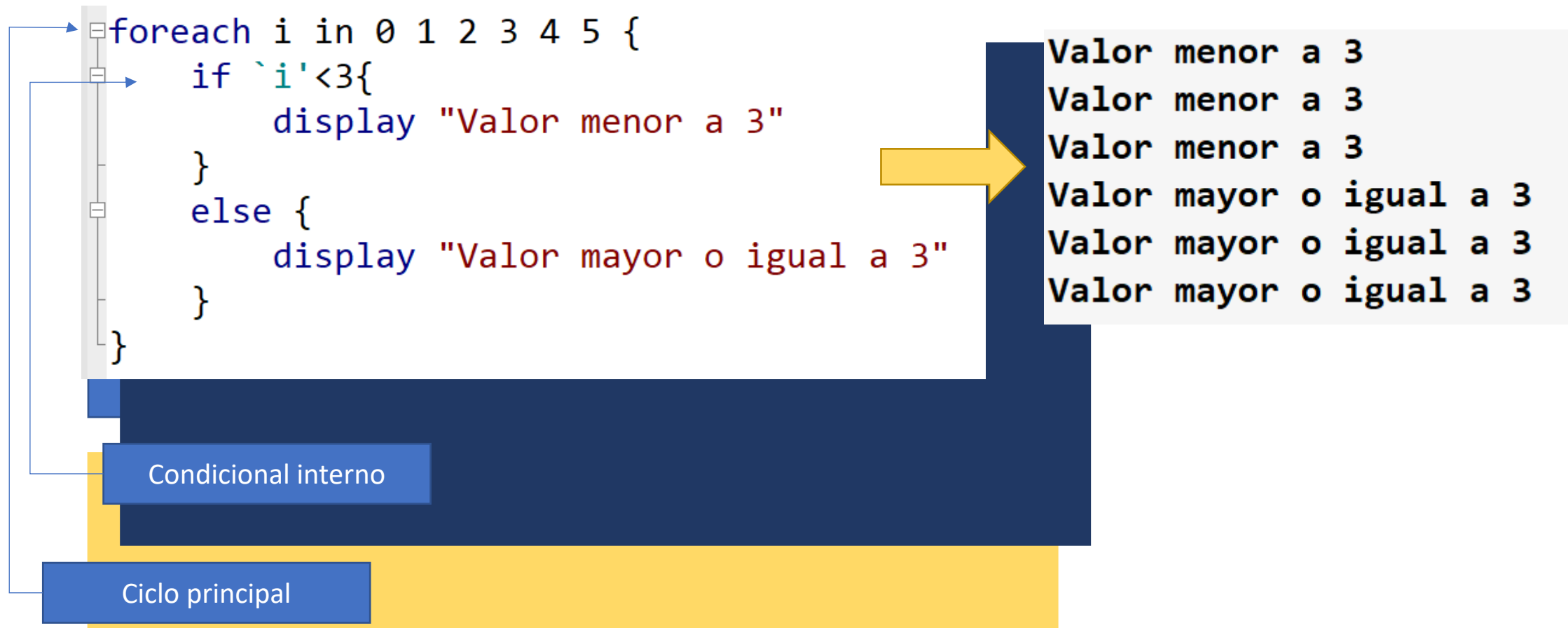
```
for (i in 0:5){  
  if (i<3){  
    print("valor menor a 3")  
  } else {  
    print("valor mayor o igual a 3")  
  }  
}
```

```
[1] "valor menor a 3"  
[1] "valor menor a 3"  
[1] "valor menor a 3"  
[1] "valor mayor o igual a 3"  
[1] "valor mayor o igual a 3"  
[1] "valor mayor o igual a 3"  
> |
```

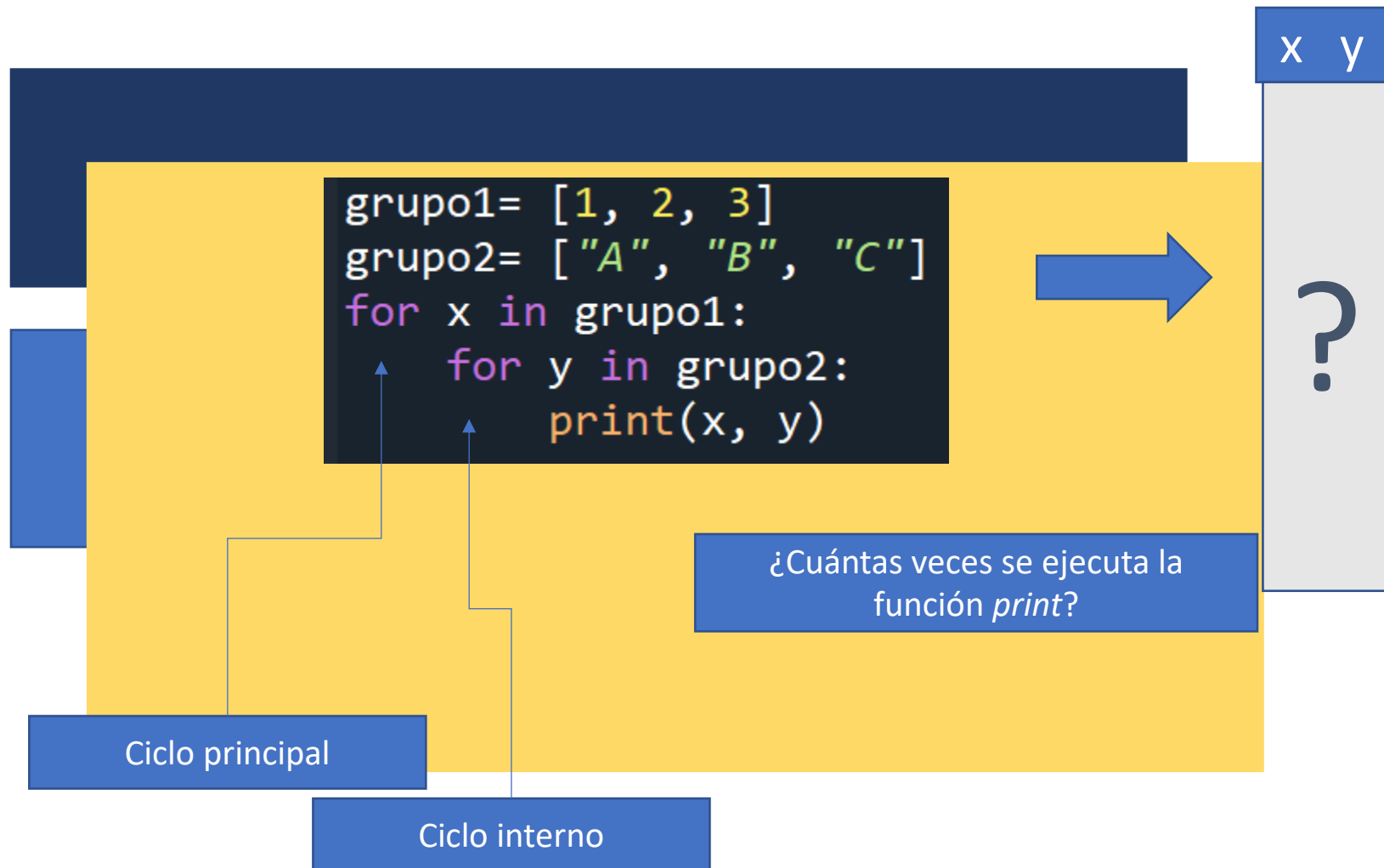
Condiciona interno

Ciclo principal

c. Anidación 1 – Stata



c. Anidación 2 – Python



- Por cada iteración en el ciclo principal hay 3 en el ciclo interno

c. Anidación 2 – R

```
grupo1= list(1, 2, 3)
grupo2= list("A", "B", "C")
for (x in grupo1){
  for (y in grupo2){
    xy<-paste(x,y, sep = " ")
    print(xy)
  }
}
```

Ciclo interno

Ciclo principal

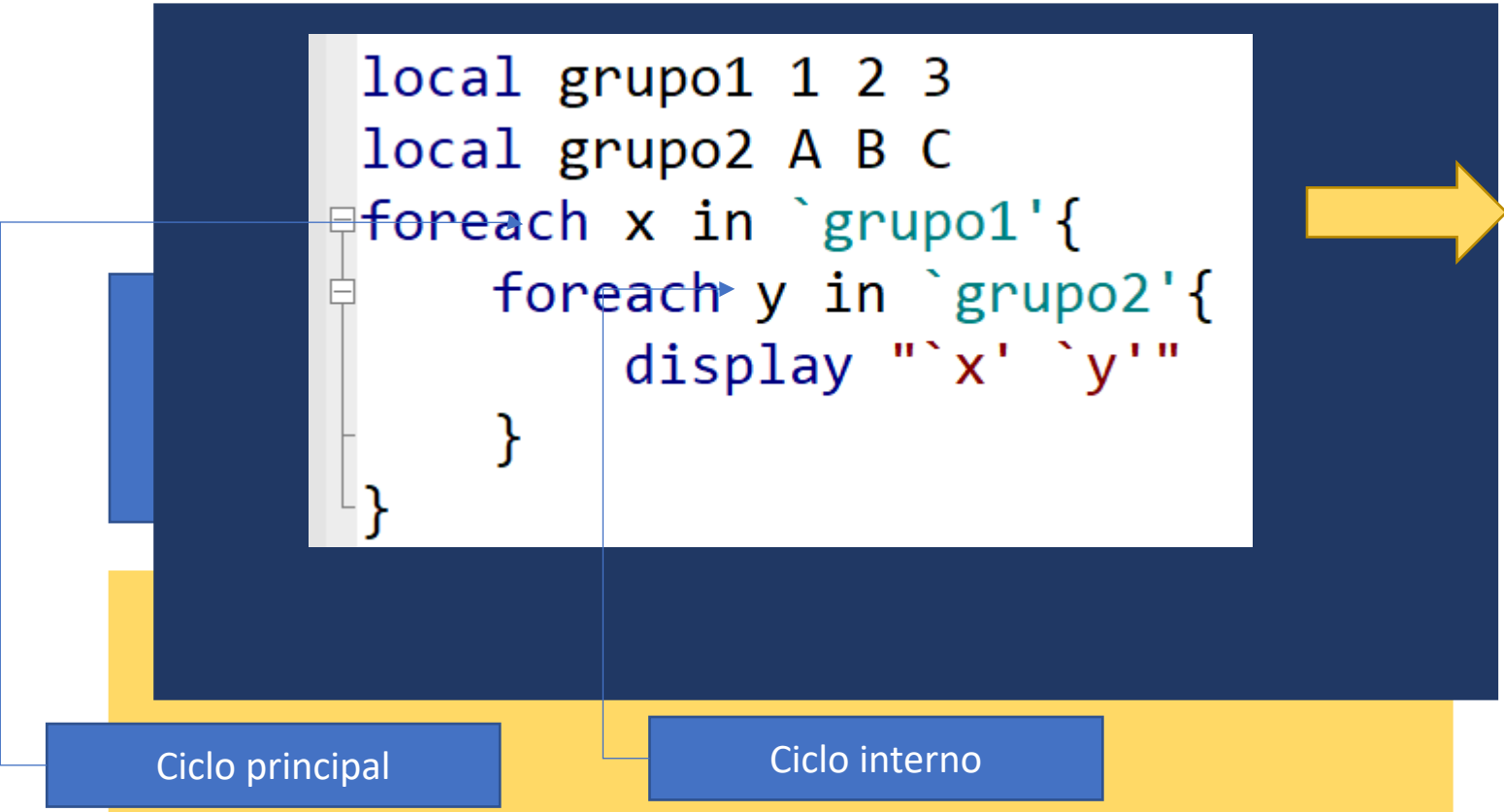
	x	y
--	---	---

[1]	"1	A"
[1]	"1	B"
[1]	"1	C"
[1]	"2	A"
[1]	"2	B"
[1]	"2	C"
[1]	"3	A"
[1]	"3	B"
[1]	"3	C"

- Por cada iteración en el ciclo principal hay 3 en el ciclo interno

c. Anidación 2 – Stata

```
local grupo1 1 2 3
local grupo2 A B C
foreach x in `grupo1'{
    foreach y in `grupo2'{
        display "`x' `y'"
    }
}
```



x	y
1	A
1	B
1	C
2	A
2	B
2	C
3	A
3	B
3	C

- Por cada iteración en el ciclo principal hay 3 en el ciclo interno

Fuentes– Python

Python Institute

- [Inicio](#) (En inglés)
- [Curso esencial de Python](#) (Interactivo y en español)
- [Ejemplos y ejercicios de estructuras de control en Python](#), del curso esencial

Documentación de la librería pandas, para procesamiento de Data Frames (En inglés)

- [Inicio](#)
- [Estadísticas descriptivas básicas](#)
- [Estadísticas descriptivas seleccionadas](#), función agg

Páginas de internet con material abierto

- [Más sobre estadísticas descriptivas en Pandas](#)
- [Operaciones lógicas en Python](#)
- [StarkOverFlow para operaciones con columnas en pandas](#)

Portal para el aprendizaje de Python j2logo (En español)

- [Inicio](#)
- [Tutoriales gratuitos](#)
- [Tutorial de operadores en Python](#)

Fuentes– R

Libros

- [R for Data Science](#)
- [R para principiantes](#)
 - [Cadenas de caracteres](#)
 - [Operadores](#)
 - [Familia *apply*](#)
 - [Estructuras de control](#)
- [Modern R with the tidyverse](#)
 - [Estadísticas descriptivas](#)
- [Introduction to R](#)

Otros recursos

- [Quick R by DataCamp](#)
- [Introduction to Tidyverse : readr, tibbles, tidyr & dplyr – CheatSheets](#)
- [Tutorial para importar datos en R \(consejos útiles\)](#)
- [R Tutorial](#)
 - [R operators](#)
- [StackOverFlow para R \(Español\) \(Inglés\)](#)

Fuentes – Stata

Repositorio del curso Taller de Stata (Incluye notas de clase y videos)

Otros recursos:

- Software Collections at IDEAS: <https://ideas.repec.org/i/c.html>
- Stack Overflow (Preguntas y respuestas sobre programación): <https://stackoverflow.com/>
- Stata documentation: <https://www.stata.com/features/documentation/>
- Stata resources for learning : <https://www.stata.com/links/resources-for-learning-stata/>
 - CheatSheets : <https://www.stata.com/bookstore/stata-cheat-sheets/>
 - Stata Tutorial: <https://data.princeton.edu/stata>
- Stata FAQ: <http://www.stata.com/support/faqs/>
- Statalist (Preguntas y respuestas sobre programación en Stata): <https://www.statalist.org/>
- The Stata Journal: <https://www.stata-journal.com/>
- UCLA guide to Stata: <http://www.ats.ucla.edu/stat/stata/>