Clase 5:
Estructuras de control:
Por casos e iteradas

Universidad de los Andes
Facultad de Economía

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

a. Operadores de comparación y lógicos - Python

Operador	Descripción
>	Mayor que. `True` si el operando de la izquierda es estrictamente mayor que el de la derecha; `False` en caso contrario.
>=	Mayor o igual que. `True` si el operando de la izquierda es mayor o igual que el de la derecha; `False` en caso contrario.
<	Menor que. `True` si el operando de la izquierda es estrictamente menor que el de la derecha; `False` en caso contrario.
<=	Menor o igual que. `True` si el operando de la izquierda es menor o igual que el de la derecha; `False` en caso contrario.
==	Igual. `True` si el operando de la izquierda es igual que el de la derecha; `False` en caso contrario.
!=	Distinto. `True` si los operandos son distintos; `False` 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
`a or b`	Si `a` se evalúa a falso, entonces devuelve `b`, si no devuelve `a`	Solo se evalúa el segundo operando si el primero es falso
`a and b`	Si `a` se evalúa a falso, entonces devuelve `a`, si no devuelve `b`	Solo se evalúa el segundo operando si el primero es verdadero
`not a`	Si `a` se evalúa a falso, entonces devuelve `True`, si no devuelve `False`	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
: 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

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

- Usando la consola de Spyder

a. Operadores de comparación y lógicos - Python

La negación de una conjunción es la separación de las negaciones.
La negación de una disyunción es la conjunción de las negaciones.

```
not (p and q) == (not p) or (not q)
not (p or q) == (not p) and (not q)
```

$$\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$$
$$\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$$

- Leyes de Morgan

Fuentes:

https://edube.org/learn/programming-essentials-in-python-part-1-spanish/operaciones-l-oacute-gicas-y-de-bits-en-python-and-or-not-1

https://es.wikipedia.org/wiki/Leyes_de_De_Morgan

a. Operadores de comparación y lógicos - R

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

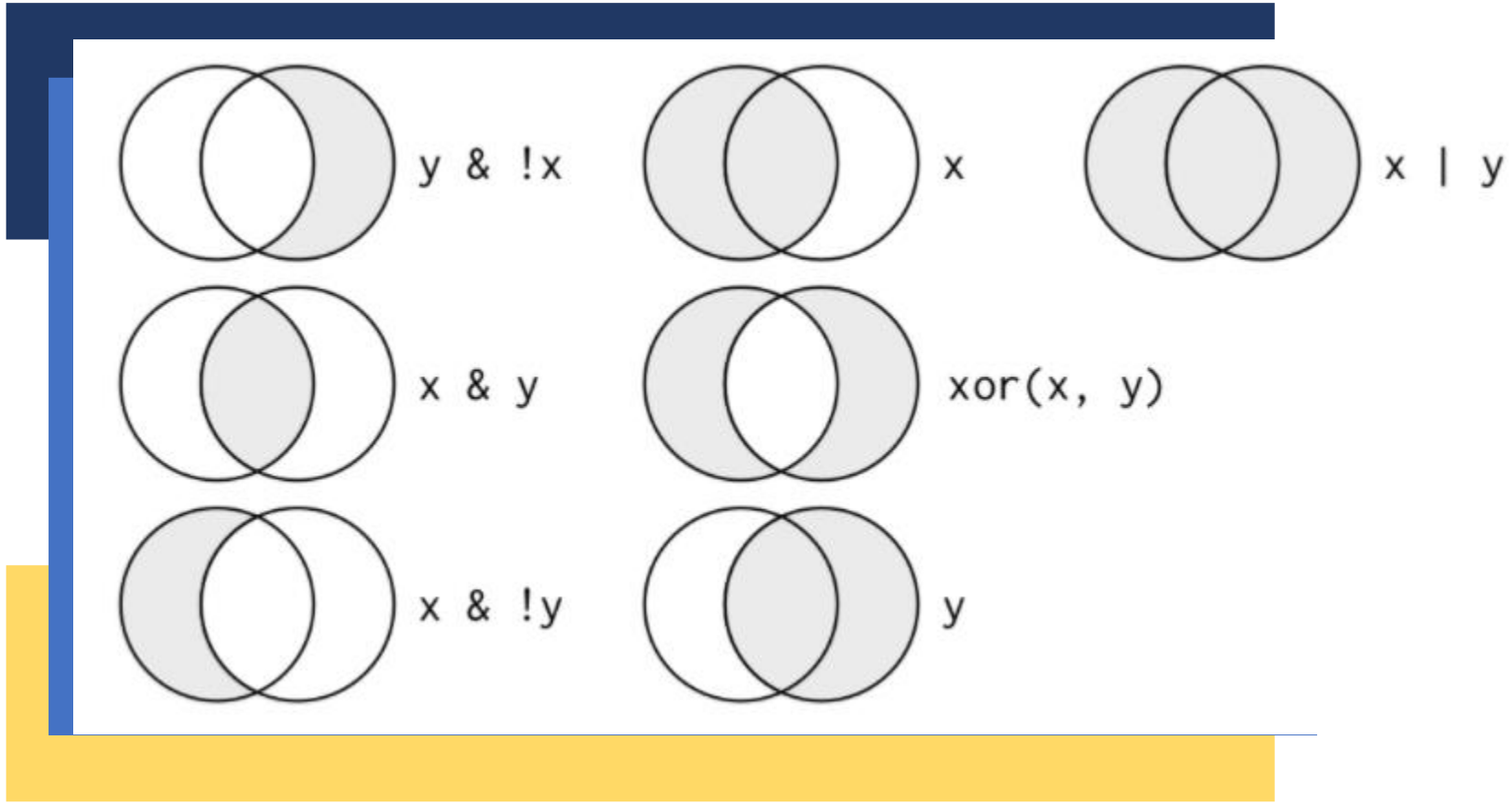
- Tomado de R for Data Science (https://r4ds.had.co.nz)

a. Operadores de comparación y lógicos - R

```
> x=8
> y=9
> x + y
[1] 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
>
```

- Usando la consola de RStudio

a. Operadores de comparación y lógicos - R



- x: círculo derecho
- Y: círculo izquierdo

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

a. Operadores de comparación y lógicos - Stata

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

Syntax

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

- 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
```

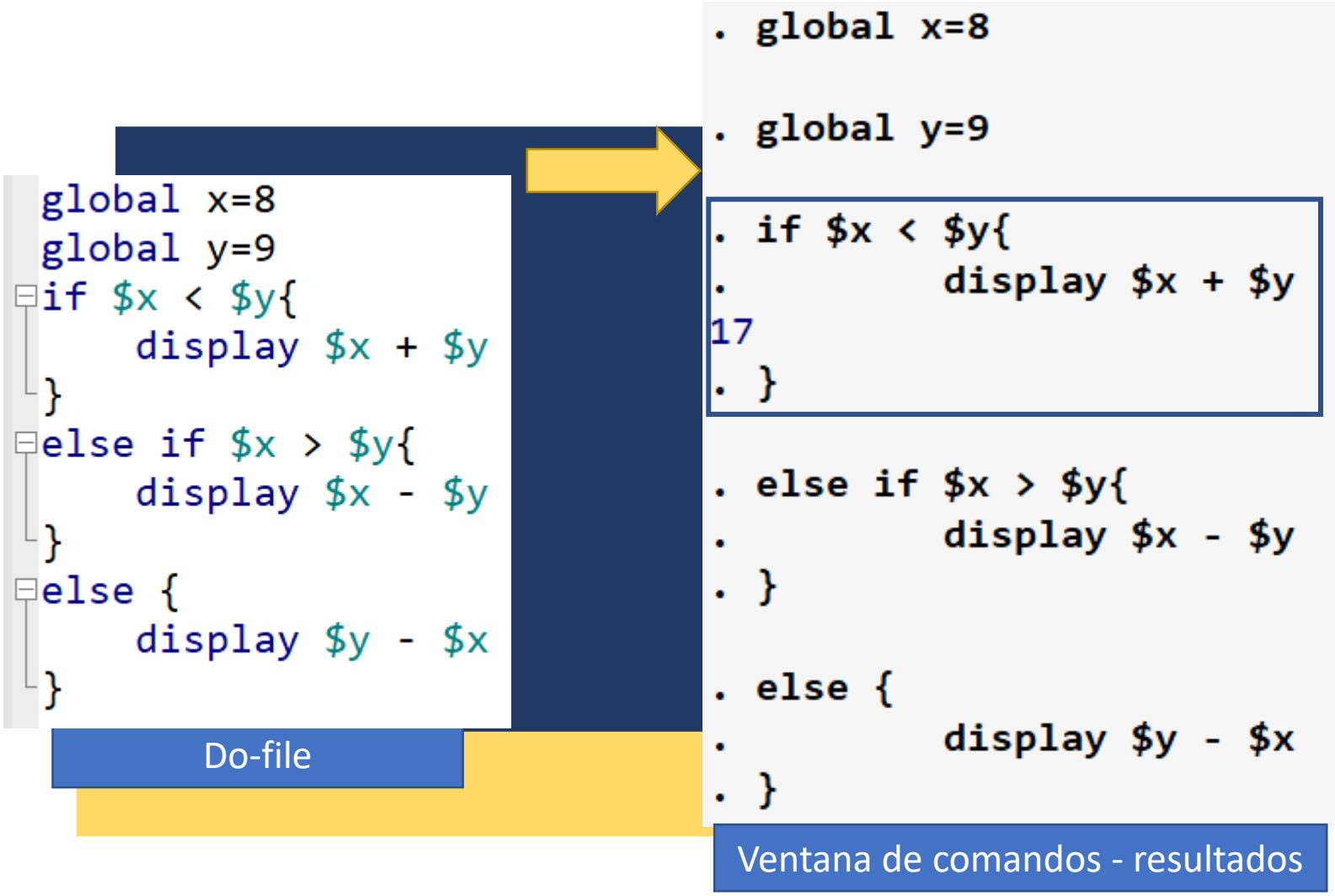
a. Operadores de comparación y lógicos

```
: 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
>
```

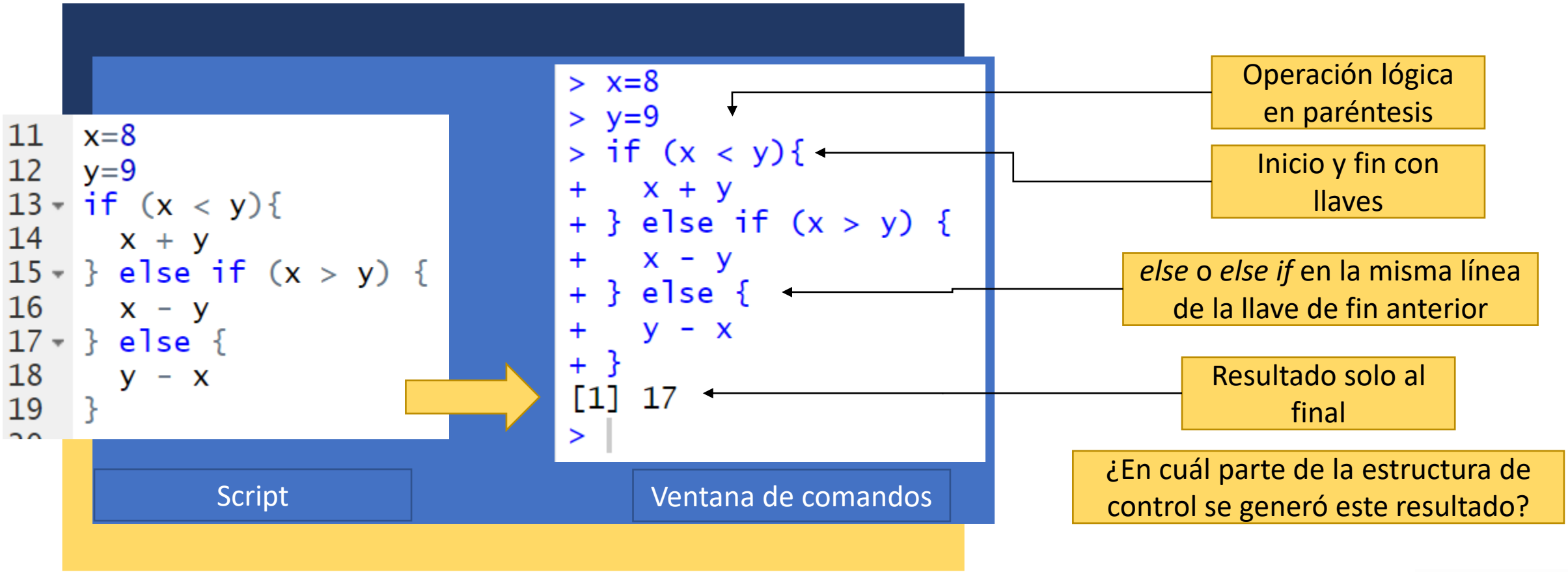
```
. local x=8

. 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 - Stata

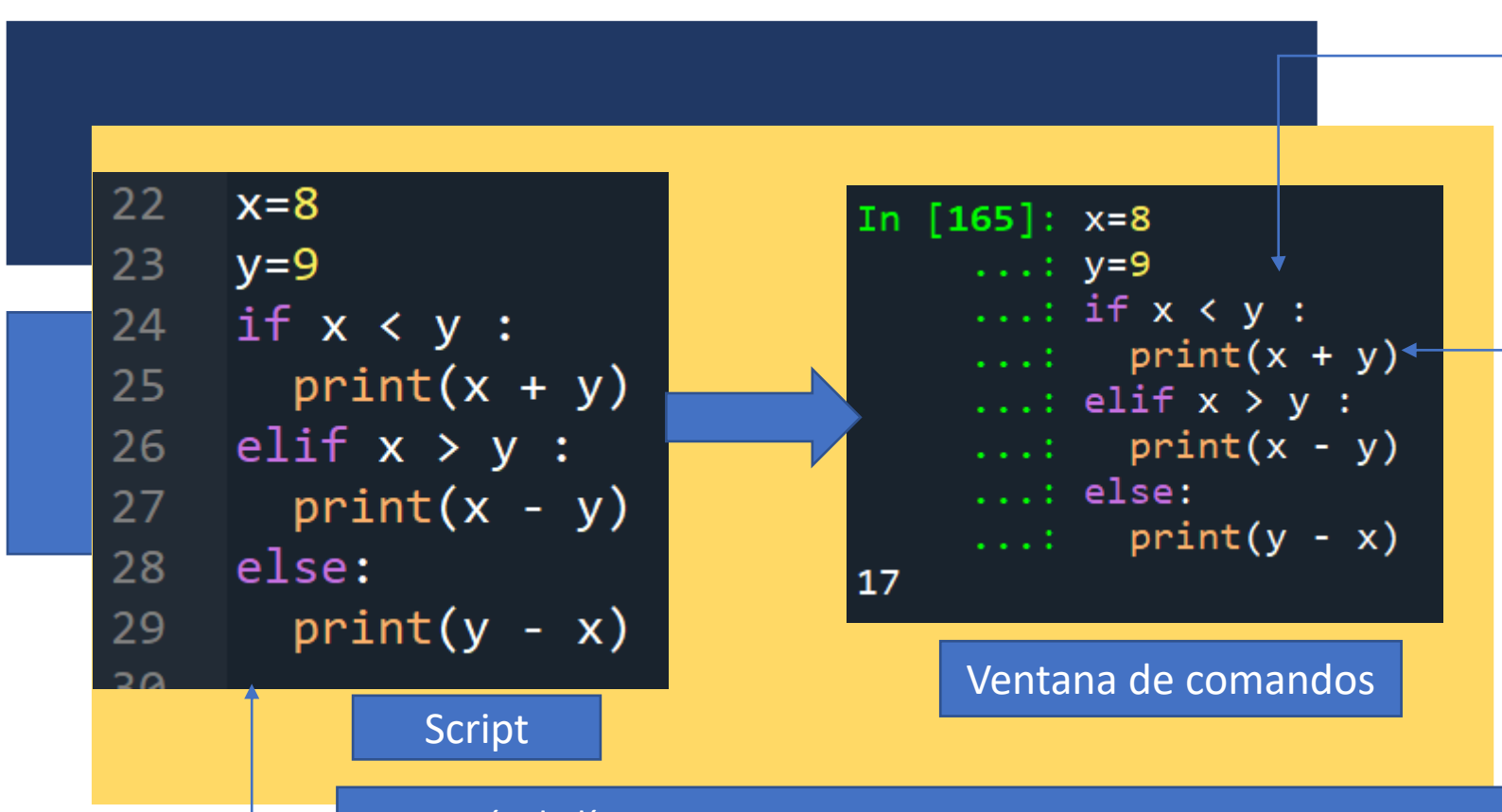


- Permiten realizar pruebas lógicas con cadenas de texto y números.
- De acuerdo con el resultado el condicional ejecuta algunas instrucciones. Un proceso de verificación caso por caso
- Cuando una de las sentencias es verdadera no se prueban las siguientes.
- Los casos deben ser excluyentes, por construcción.

b. Estructura de control por casos - R



b. Estructura de control por casos - Python



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"
¡hola mundo!
. }

. 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

```r
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!")
}
```

Script

```
> 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!"
>
```

Ventana de comandos

Objeto definido como caracteres

Uso de la función *paste* para concatenar

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

c. Ejercicio con datos: Exploración

- En la página del curso están los siguientes elementos:
  - Archivo de datos separados por comas (.csv) de la población municipal de Colombia del censo de 2005
  - Códigos de programación:
    - Stata do-file
    - R script (creado en R Studio)
    - Python script (creado en Spyder)
  - Todos los códigos ejecutan las mismas tareas, que en resumen son:
    - Importar la base de datos
    - Calcular estadísticas descriptivas básicas para todos los municipios y por grupos según un criterio de tamaño por habitantes y con un programa que utiliza una estructura de control condicional.
    - Crear una variable de identifica a los municipios grandes.

c. Ejercicio con datos: Comentar un script o do-file

Usted debe comentar el código analizando la escritura y ejecución de cada línea, teniendo en cuenta todos los elementos que se especifican en ella. Idealmente apóyese en la documentación del Software y en foros de ayuda (StackOverflow y Statalist). Google es su mejor amigo.

Puede orientarse con las siguientes preguntas:

- ¿Qué le dice cada elemento del comando al software para ejecutar adecuadamente la operación? (la semántica del código)
- ¿Cuáles son las diferencias de escritura entre los lenguajes para ejecutar esta misma operación? En otras palabras, ¿cuáles son las diferencias de sintaxis? Puede tener en cuenta:
  - Paréntesis de diferentes tipos o su ausencia. ¿Cómo se marca el inicio y el fin de una estructura de control?
  - Espacios o saltos de línea. El orden de los elementos del comando. e.g. En Stata siempre se empieza con el nombre de un comando.
  - En R y Python se usan librerías. ¿Cuál es su papel en este algoritmo?
  - ¿Cómo se llama en cada lenguaje a una variable de una base de datos (data frame)?
- ¿Qué hace cada software sin necesidad de ser especificado directamente a través del código? En otras palabras (¿Qué es lo que se hace por defecto? Para esto es útil:
  - El comando de importación de datos en cada lenguaje es especialmente interesante para notar estas diferencias. ¿Por qué es necesario especificar más ese comando en un lenguaje y en otro no?
  - Preguntarse cuándo es necesario usar una función o no.
  - Probar cambios en la especificación de los comandos, ejecutarlos y buscar alguna explicación si hubo un resultado distinto o hay un mensaje de error.

c. Ejercicio con datos – Stata

```
clear all
cd "Coloque aquí la dirección de la carpeta donde están los datos"

import delimited pob2005.csv, stringcols(1 3 ) encoding("utf-8")  clear
describe
codebook p2005
sum p2005
sum p2005, detaile

local tipo_filtro "otro"

if "`tipo_filtro'"=="Grandes" | "`tipo_filtro'"=="Todos"   {
  display "Población de municipios grandes"
  summarize p2005 if p2005>100000 & !missing(p2005)
}
else if "`tipo_filtro'"=="Pequeños" | "`tipo_filtro'"=="Todos" {
  display "Población de municipios pequeños"
  sum p2005 if p2005<=100000
}
else if "`tipo_filtro'"=="Todos" {
  display "Población de todos los municipios"
  sum p2005
}
else {
  display "Seleccione un tipo de tabla válido"
}

gen grandes_directo=p2005>100000
replace grandes_directo=. if missing(p2005)

gen grandes_if=1 if p2005>100000
replace grandes_if=0 if p2005<=100000

gen grandes_funcion=cond(p2005>100000,1,0)
```

- Do-file sin comentar

c. Ejercicio con datos – Stata - Comentarios

```
** Miguel Andrés Garzón Ramírez
** Universidad de los Andes - Facultad de Economía
** 3 de mayo de 2021
* Estadísticas descriptivas por grupos de municipios
cd "C:\Users\magse\Universidad de los Andes\Alfredo Eleazar Orozco Quesada - Curso Programación para datos
EdContinua\Clases\Implementación\Clase 5 - Condicionales y ciclos iterados" // Esta es la dirección de la carpeta donde
están los datos"

import delimited pob2005.csv, stringcols(1 3 ) encoding("utf-8")  clear // Stata importa como variable numérica columnas
que contienen solo caracteres numéricos. Como las variables 1 y 3 (dp y dpmp) son códigos DIVIPOLA del DANE pueden
iniciar con cero, se debe especificar que estas variables sean importadas como caradenas de caracteres con la opción
stringcols(). Al importar los datos los nombres de los municipios tuvieron problemas con las palabras con tildes. Para
solucionarlo se especificó que los datos tienen condificación UTF-8, que se usaba en la decada de los 2000
* La unidad de observación de esta base de datos es el municipio. Solo hay información para el año 2005.
describe // describe todas las variables de la base de datos
codebook p2005 // Estadísticas descriptivas amplias de la variable población. Si no especifica el nombre de una o mas
variables lo hace para todas las variables de la base de datos
* La variable p2005 no tiene valores faltantes, todos los municipios tienen dato de población. Esto facilita los cálculos.
sum p2005 // Estadísticas descriptivas básicas de la variable población. Si no especifica el nombre de una o mas
variables lo hace para todas las variables de la base de datos
sum p2005, detail // Estadísticas descriptivas detalladas de la variable población. Si no especifica el nombre de una o
mas variables lo hace para todas las variables de la base de datos. Ver cocumentación del comando summarize

```

Comentarios sobre la semántica o la sintaxis

Comentarios sobre los datos o los resultados de los análisis

```
14
15  *** Inicio del programa para seleccionar el grupo para el cuál se calcularán estadísticas descriptivas
16  *El criterio para definir los grupos es que un municipio es grande si tiene mas de 100000 habitantes, es pequeño si tiene
    100000 habitantes o menos.
17
18  local tipo_filtro "otro"  // Elemento (macro) que permite reemplazar más adelante el grupo de municipios para el cual se
    quiere hacer las estadísticas descriptivas
19
20  if "`tipo_filtro'"=="Grandes" | "`tipo_filtro'"=="Todos"   { // Paso 1, verificar si se quiere hacer estadísticas
    descriptivas solo para municipios grandes
21    display "Población de municipios grandes"
22    summarize p2005 if p2005>100000 & !missing(p2005) // Es necesario especificar que no tome los valores faltantes en el
    filtro porque estos se consideran números muy grandes en una operación de comparación. La idea es que si se crea una
    variable con base en los valores de otra, los valores faltantes de la variable inicial se conserven en la nueva variable.
    Si se incluye o no este filtro adicional <!missing(p2005)> no cambia el resultado, pero es la forma más precisa de
    escribirlo. Si se utiliza este código para otras variables que sí tengan valores faltantes puede ser útil. Siempre se
    deben tener en cuenta.
23  }
24  else if "`tipo_filtro'"=="Pequeños" | "`tipo_filtro'"=="Todos" { // Paso 2, si no se quería hacer las estadisticas de los
    municipios grandes, ahora se verifica si se quiere hacer estadísticas descriptivas solo para municipios pequeños
25    display "Población de municipios pequeños"
26    sum p2005 if p2005<=100000
27  }
28  else if "`tipo_filtro'"=="Todos" { // Paso 3, si no se quería hacer las estadisticas de los municipios pequeño, ahora se
    verifica si se quiere hacer estadísticas descriptivas para todos los municipios
29    display "Población de todos los municipios"
30    sum p2005
31  }
32  else { // Paso 4, si nada de lo anterior fue verdadero entonces se ingresó un valor no válido para este programa y esta
    es la verificación de salida.
33    display "Seleccione un tipo de tabla válido"
34  }
```

c. Ejercicio con datos – Stata - Comentarios

```
35
36  *** Creación de una variable de identifique a los municipios grandes.
37  *En la variable habrán dos posibles valores, 0 o 1. 1 es un municipio grande según el criterio definido previamente.
38
39  * Método 1 - Método directo
40  gen grandes_directo=p2005>100000 // Aquí se aprovecha el resultado de una operación lógica, ya que Stata toma la
    respuesta "Verdadero" como el número 1 y la respuesta "Falso" como el número 0. Si la variable tuviera valores faltantes
    la nueva variable no los tendría, ya que se camuflarían en la respuesta "Falso", por esto es necesaria la siguiente línea
    de código
41  replace grandes_directo=. if missing(p2005)  // "." es la notación en Stata para un valor faltante en una variable
    númerica. Para una variable de caracter es "" (vacio entre comillas)
42
43  * Método 2 - Método por casos - La más común entre los usuarios de Stata
44  gen grandes_if=1 if p2005>100000  // Los municipios grandes
45  replace grandes_if=0 if p2005<=100000 & missing(p2005) // Los municipios pequeños conservando los valores faltantes
46
47  * Método 3 - Método con la función "cond" - Esta es la forma más parecida a como se hace en otros lenguajes, pero es la
    menos común entre los usuarios de Stata
48  gen grandes_funcion=cond(p2005>100000,1,0) // La gran ventaja es que conserva los valores faltantes de la variable base
```

c. Ejercicio con datos – R

```r
#install.packages("dplyr")
#install.packages("readr")
#install.packages("Hmisc")
#install.packages("tidyr")
#install.packages("summarytools")

library(summarytools)
library(Hmisc)
library(readr)
library(dplyr)
library(tidyr)

setwd("C:/Users/magse/Universidad de los Andes/Alfredo Eleazar Orozco Quesada - Curso Programación para datos EdCo

pob2005<-read_csv("pob2005.csv")
describe(pob2005)

tipo_filtro<-"Todos"
descr(pob2005$p2005,stats=c("min","max","sd","mean","n.valid"))
if(tipo_filtro=="Grandes" | tipo_filtro=="Todos"  ) {
  print("Población de municipios grandes")
  descr(pob2005$p2005[which(pob2005$p2005>100000)],stats=c("min","max","sd","mean","n.valid"))
} else if(tipo_filtro=="Pequeños" | tipo_filtro=="Todos" ) {
  print("Población de municipios pequeños")
  descr(pob2005$p2005[which(pob2005$p2005<=100000)],stats=c("min","max","sd","mean","n.valid"))
} else if (tipo_filtro=="Todos") {
  print("Población de todos los municipios")
  #Complete esta línea con un código que permita calcular las estadisticas descriptivas para todos los municipios
} else {
  print("Seleccione tipo de tabla")
}

pob2005$grandes_directo<-pob2005$p2005>100000
pob2005$grandes_funcion<-ifelse(pob2005$p2005>100000,1,0)

pob2005$grandes_directo<-NULL
pob2005$grandes_funcion<-NULL
```

Instalación de librerías

Habilitación de librerías

Cargar datos

Programa

Crear variable

c. Ejercicio con datos – Python

```python
#pip install pandas
#pip install numpy

import pandas as pd
import numpy as np

%cd "C:\Users\magse\Universidad de los Andes\Alfredo Eleazar Orozco Quesada - Curso Programación para datos EdContinu

pob2005=pd.read_csv("pob2005.csv", dtype={'DP': str, 'DPMP': str,'p2005': int})
pob2005.describe()
pob2005.head()
pob2005.mean()

tipo_filtro="Todos"

if tipo_filtro=="Grandes":
    print("Población de municipios grandes:")
    print(pob2005[pob2005.p2005>100000].agg({"p2005": ["count", "mean", "std", "min", "max",]}))
elif tipo_filtro=="Pequeños":
    print("Población de municipios pequeños:")
    print(pob2005[pob2005.p2005<=100000].agg({"p2005": ["count", "mean", "std", "min", "max",]}))
elif tipo_filtro=="Todos":
    print("Población de municipios pequeños:")
    #Complete esta línea con un código que permita calcular las estadísticas descriptivas para todos los municipios
else:
    print("Seleccione tipo de tabla")

pob2005['grandes'] = np.where(pob2005['p2005']>100000, 1, 0)
# Busque otro método para crear esta misma variable
```

Instalación de librerías

Habilitación de librerías

Cargar datos

Programa

Crear variable

Otros métodos de creación de variables con condicionales en el siguiente enlace.

c. Ejercicio con datos – Python

```python
if tipo_filtro=="Grandes":
    print("Población de municipios grandes:", pob2005[pob2005.p2005>100000].agg({"p2005": ["count", "mean", "std", "min", "max",]}))
elif tipo_filtro=="Pequeños":
    print("Población de municipios pequeños:", pob2005[pob2005.p2005<=100000].agg({"p2005": ["count", "mean", "std", "min", "max",]}))
elif tipo_filtro=="Todos":
        #Complete esta línea con un código que permita calcular las estadísticas descriptivas para todos los municipios
else:
    print("Seleccione tipo de tabla")
```

Para tener menos líneas de código se puede escribir así

¿Cuál es la diferencia de sintaxis con la diapositiva anterior? La semántica es equivalente

2. Estructuras de control iteradas

Principales estructuras de control iteradas (ciclos iterados)

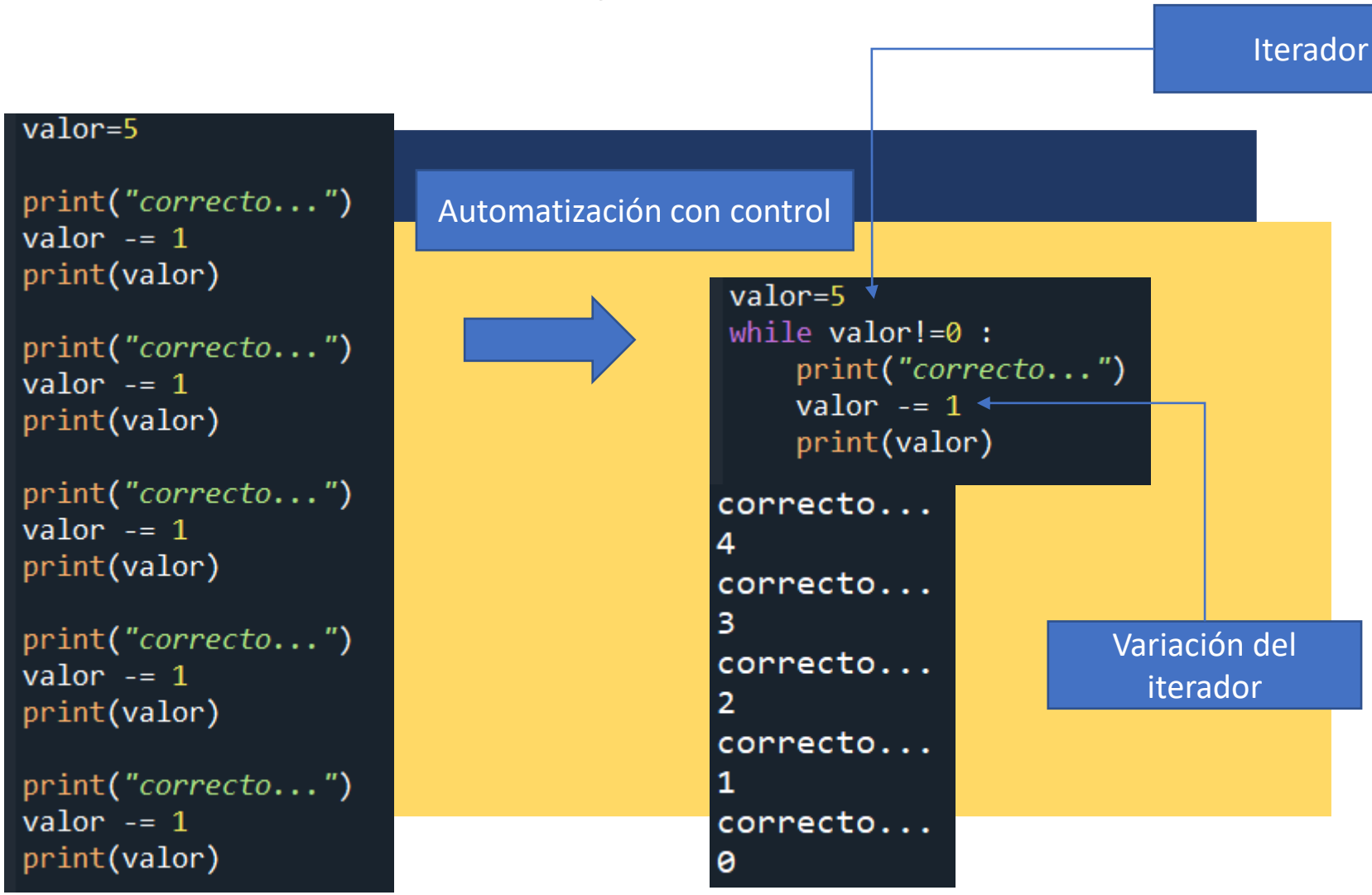
While

- 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 – 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)
```

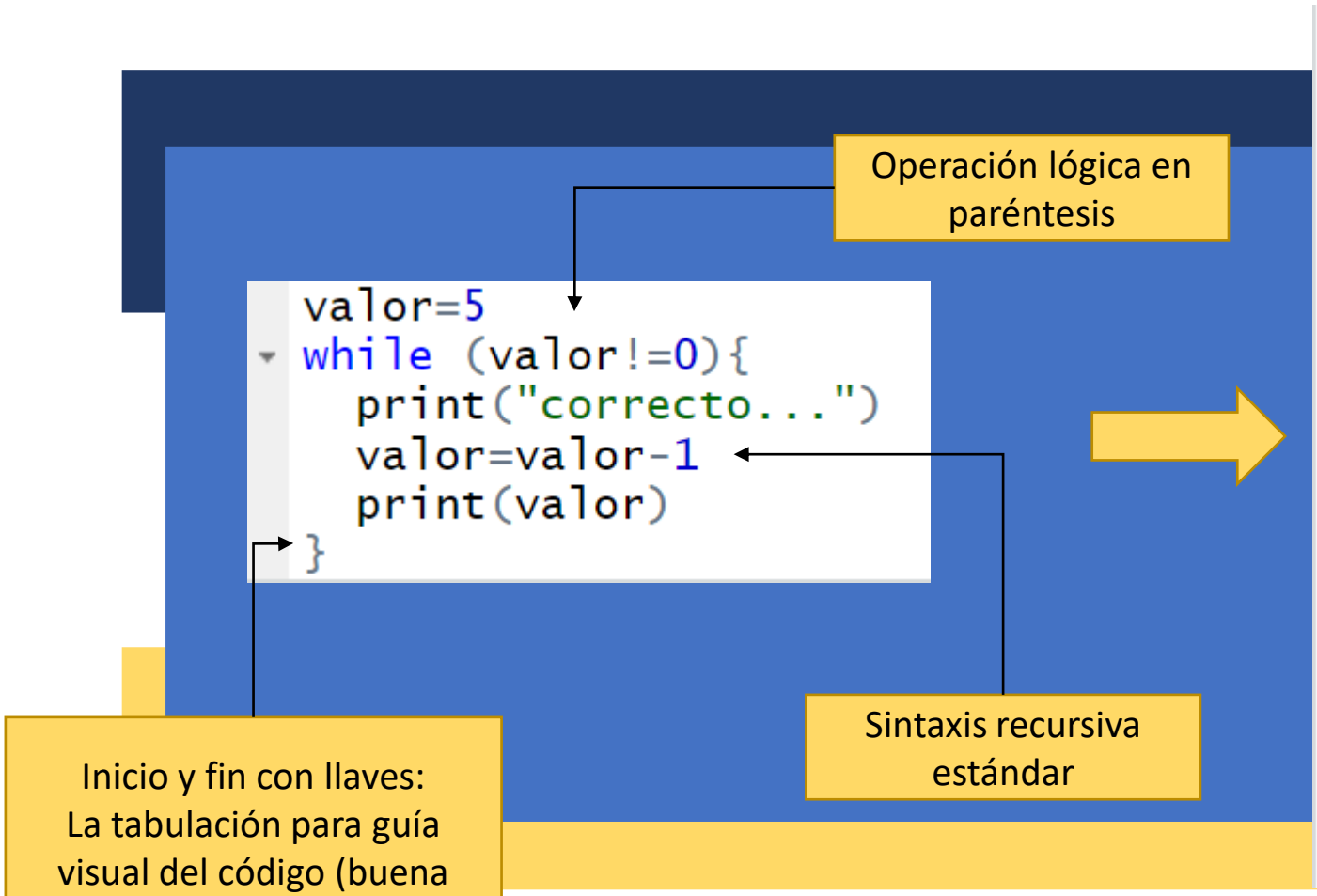
Sin paréntesis y sin llaves. Solo dos puntos (:)

Escritura recursiva 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

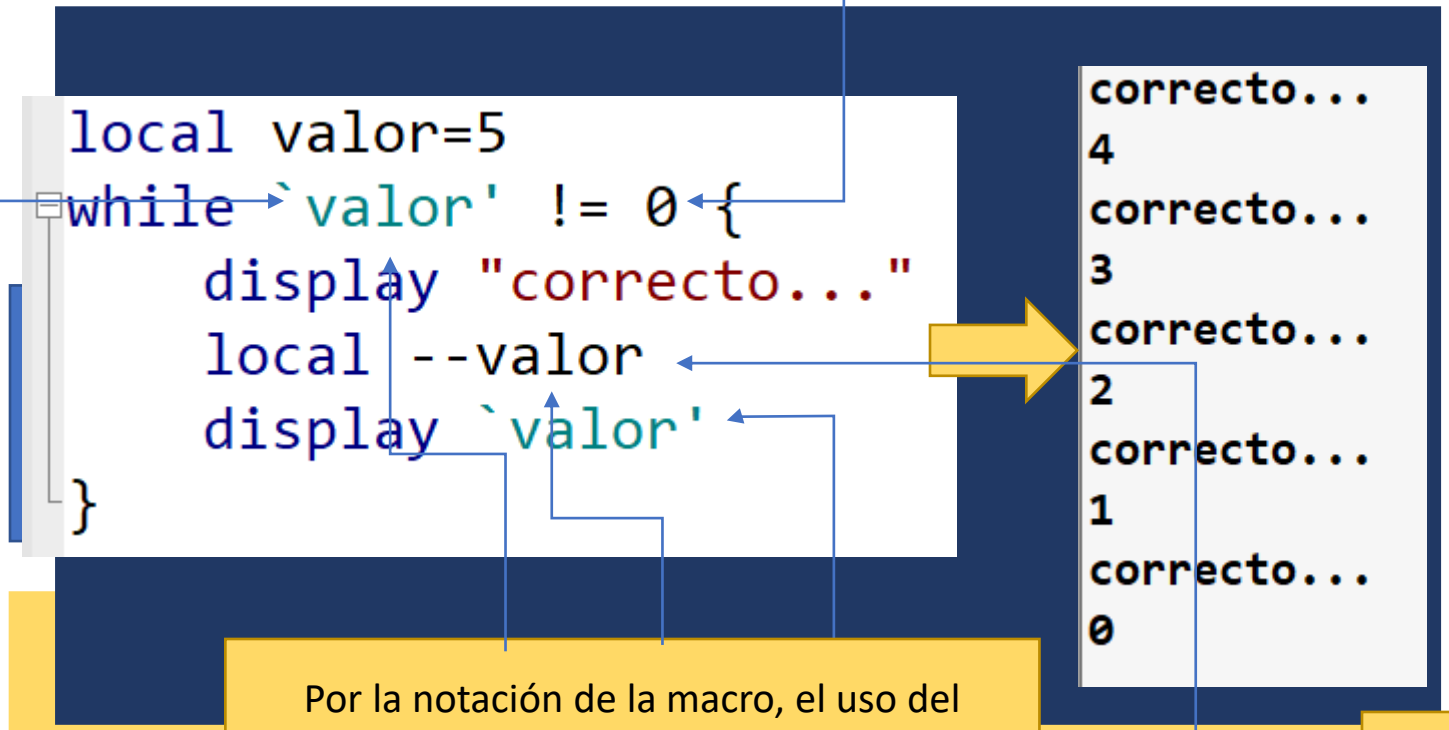



```
> valor=5
> while (valor!=0){
+   print("correcto...")
+   valor=valor-1
+   print(valor)
+ }
[1] "correcto..."
[1] 4
[1] "correcto..."
[1] 3
[1] "correcto..."
[1] 2
[1] "correcto..."
[1] 1
[1] "correcto..."
[1] 0
>
```

a. While – Stata

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

Operación lógica sin paréntesis



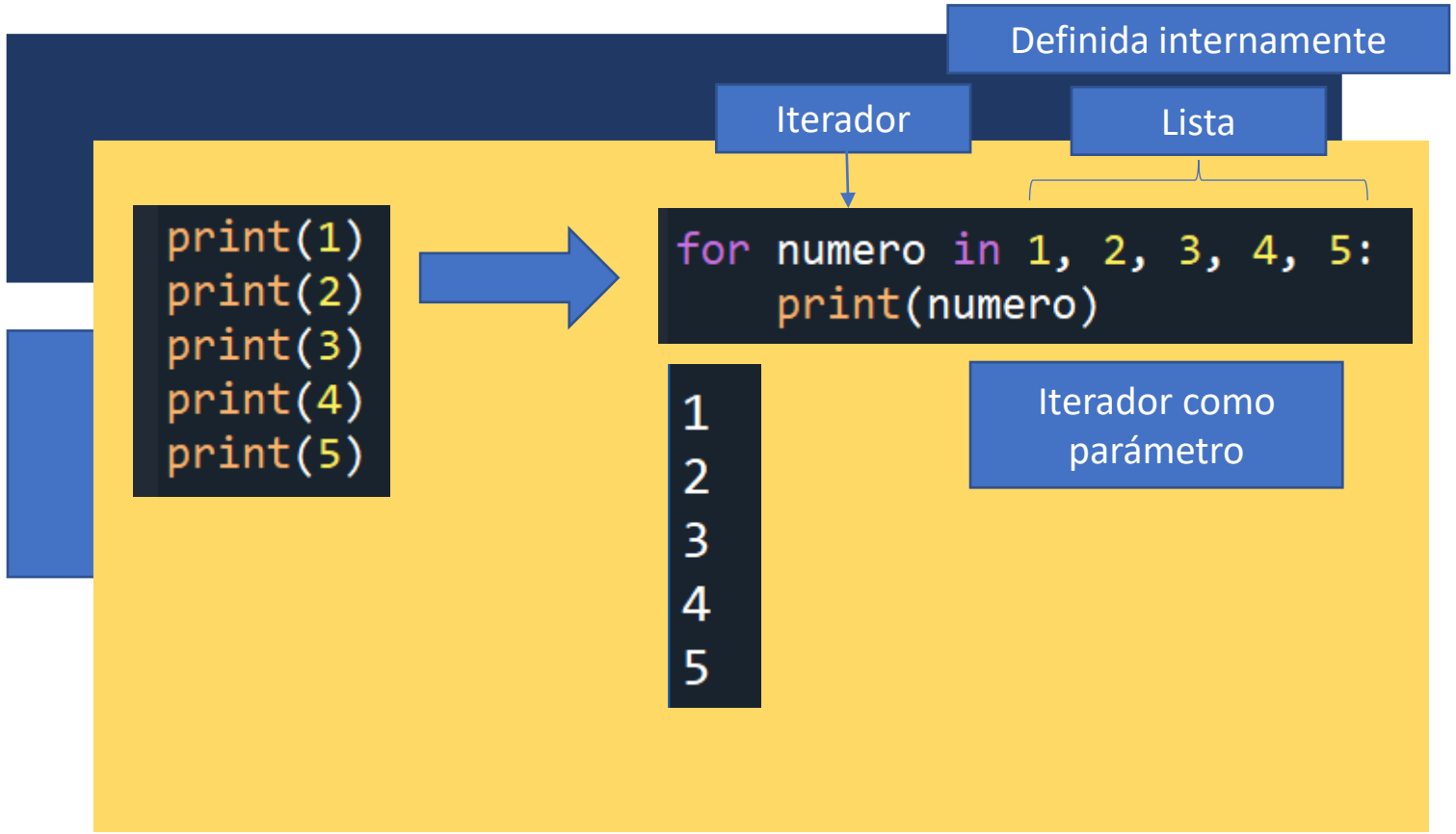
Por la notación de la macro, el uso del iterador es muy notorio, permite ver fácilmente cómo se controla la ejecución del ciclo y corregir errores.

Resta 1 al local valor en cada iteración

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

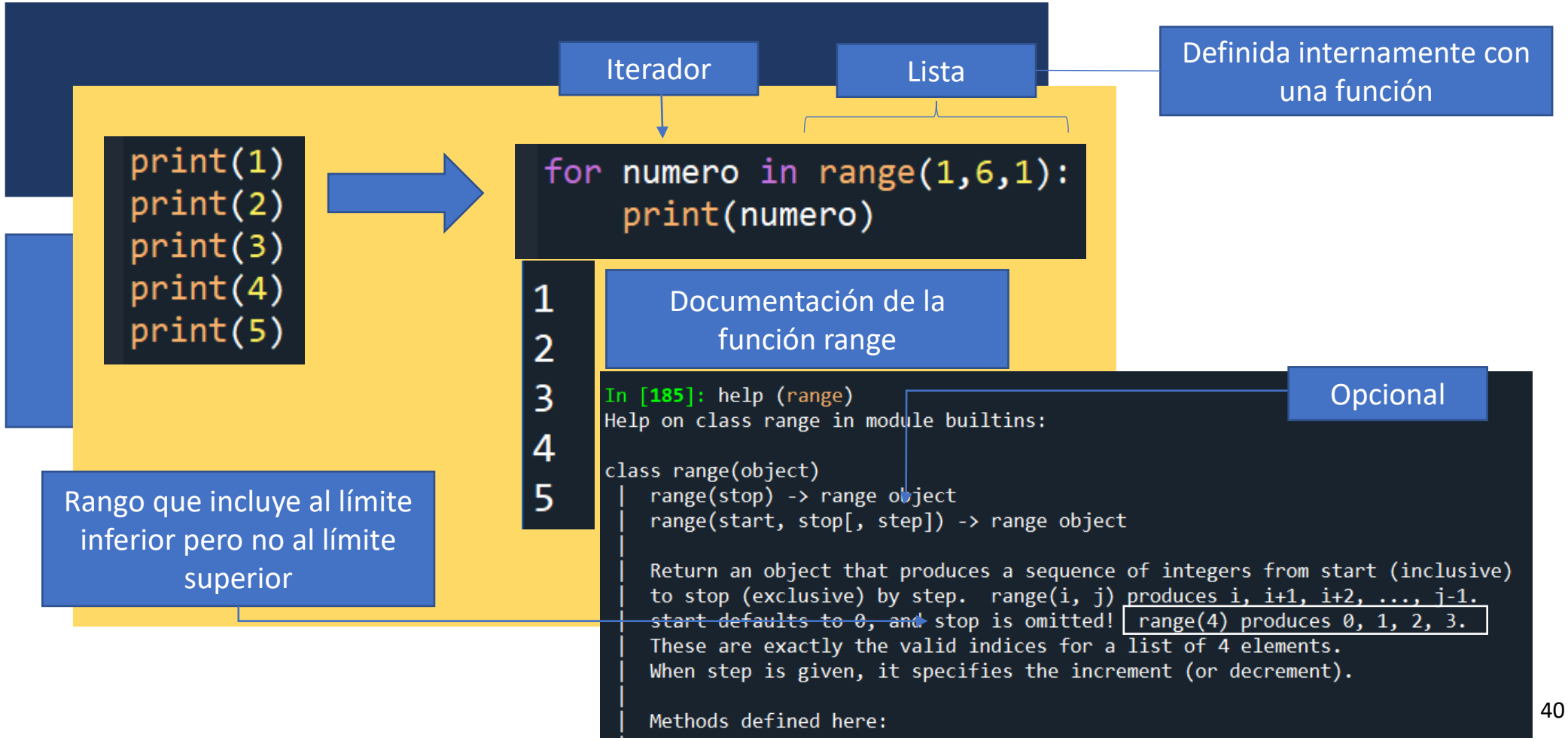
Se necesita una macro para controlar la ejecución del ciclo iterado.

b. For con números – Python

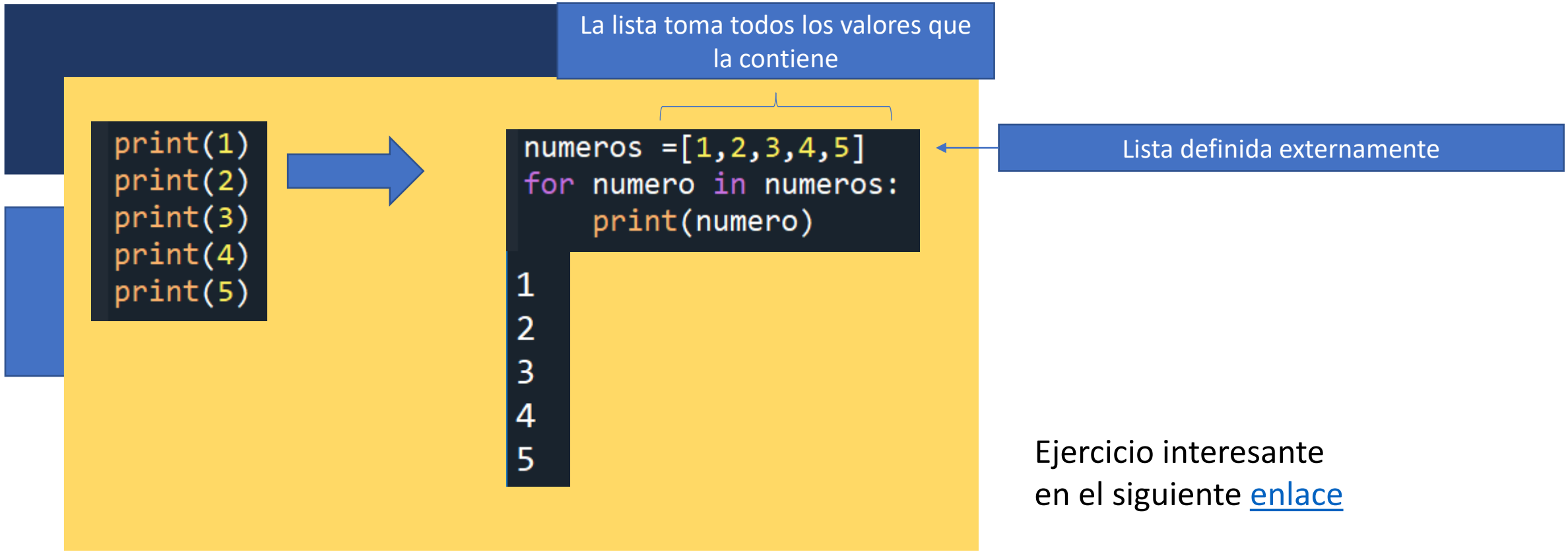


- Es un ciclo que recorre una lista, pueden ser textos o números.
- Aquí la lista está definida de forma interna, esto es, en la sintaxis del comando for.
- El iterador puede tener cualquier nombre con caracteres, como una letra o una palabra.

b. For con números – Python

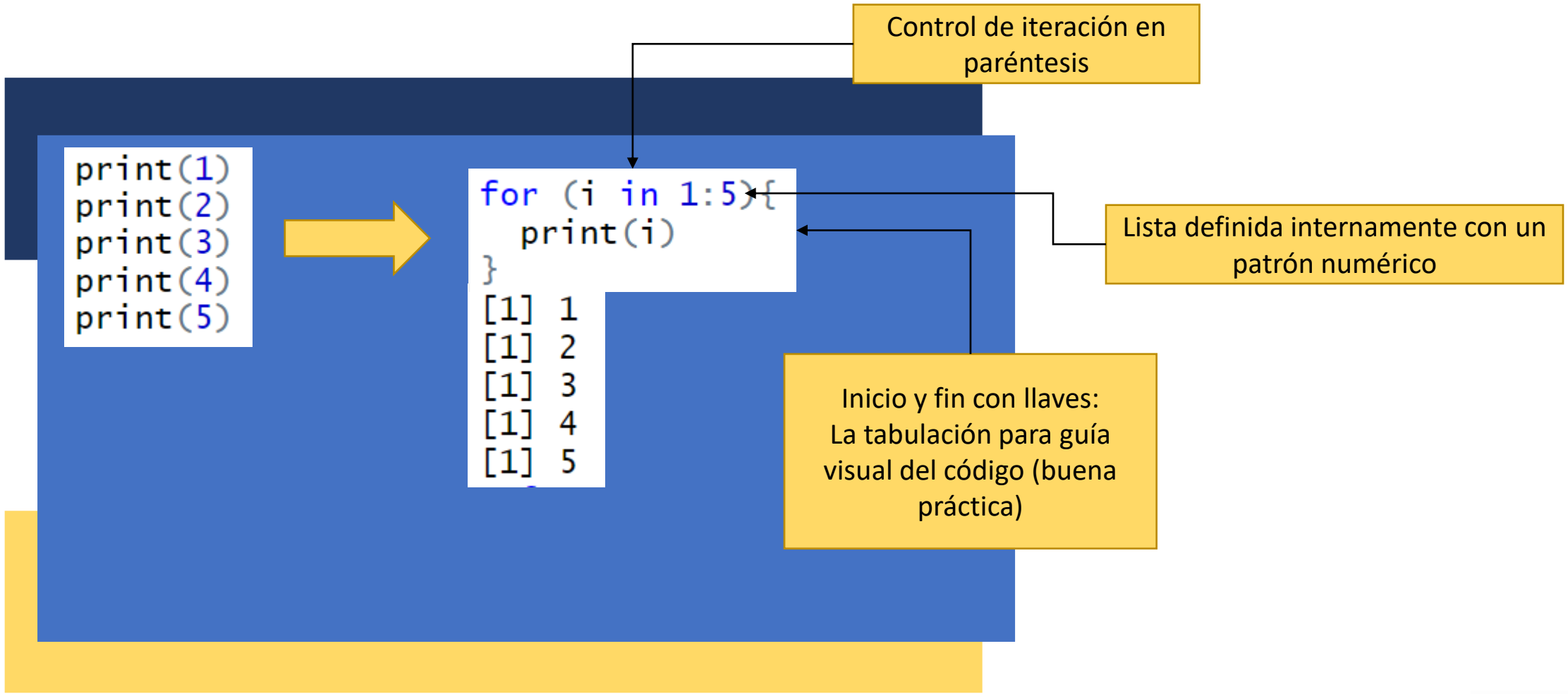


b. For con números – Python



Ejercicio interesante
en el siguiente [enlace]

b. For con números – R



b. For con números – R

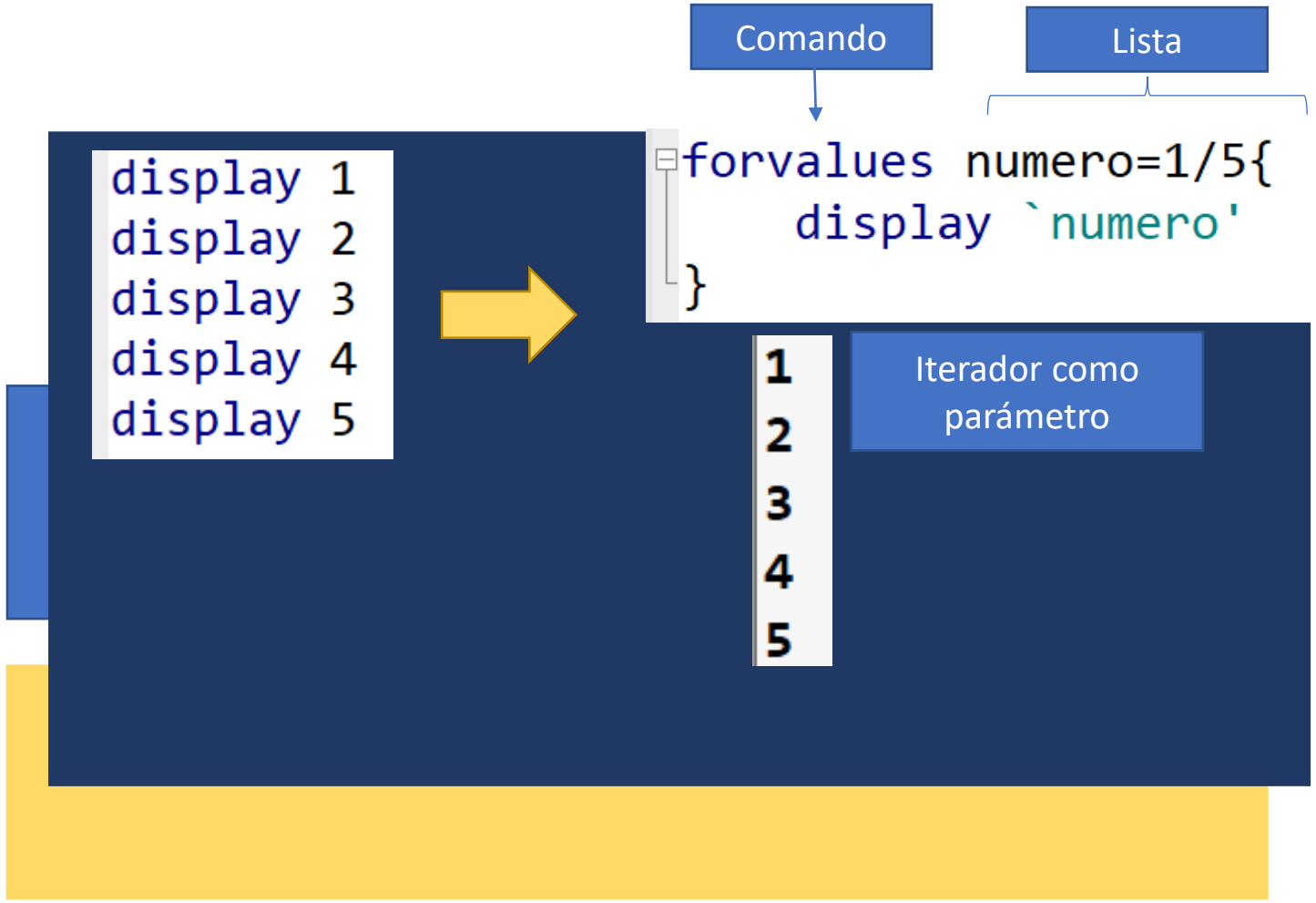
```
print(1)
print(3)
print(8)
print(9)
print(10)
print(15)
```

```
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

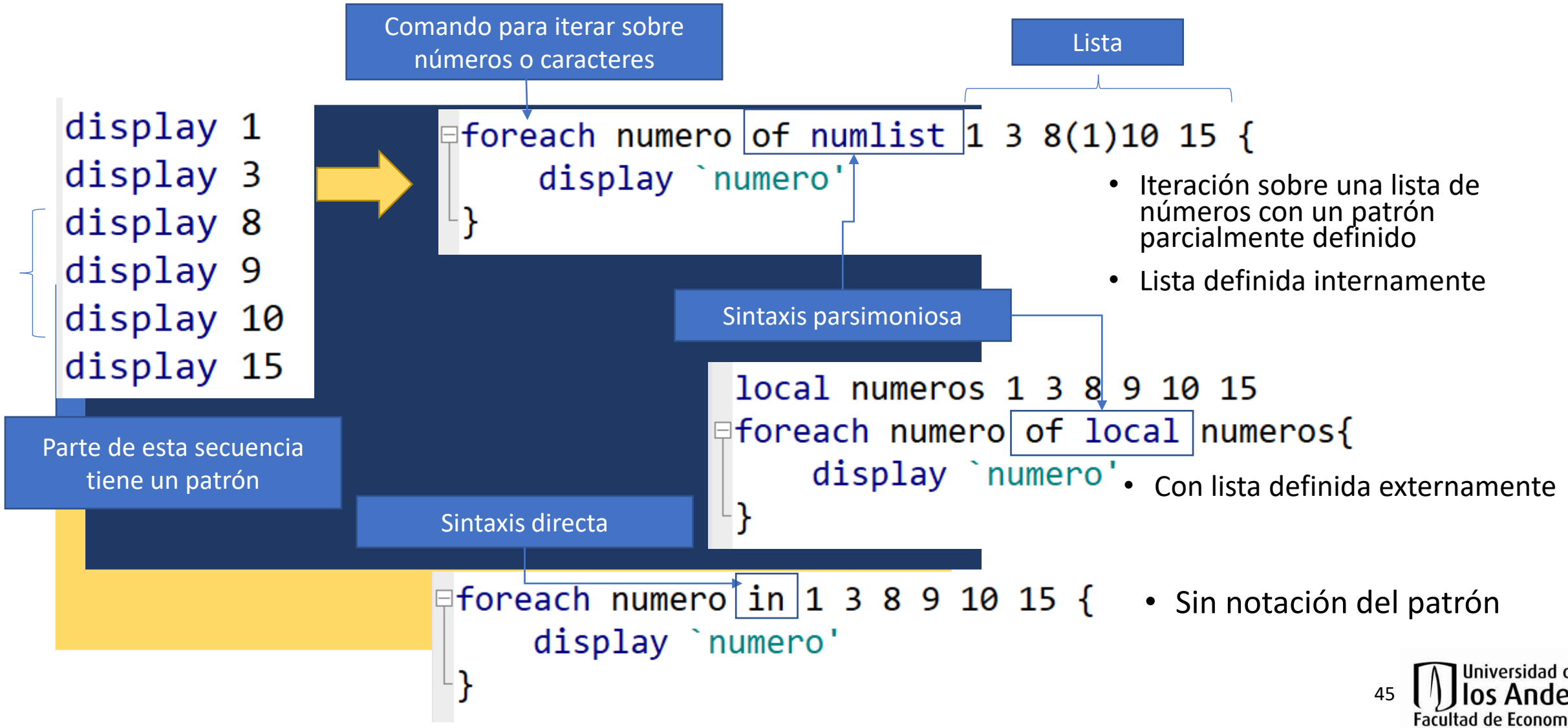


Notación equivalente

```
forvalues numero=1(1)5{
    display `numero'
}
```

- Iteración sobre una lista de números con un patrón definido
- Lista definida internamente

b. Foreach con números – Stata



- Iteración sobre una lista de números con un patrón parcialmente definido
- Lista definida internamente
- Con lista definida externamente
- Sin notación del patrón

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"
```

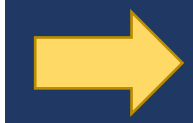
Lista definida externamente con las cadenas de caracteres

- 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 consola:

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:

“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

```
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)
```

```
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
```

Este índice permite programar el orden de llegada con números

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

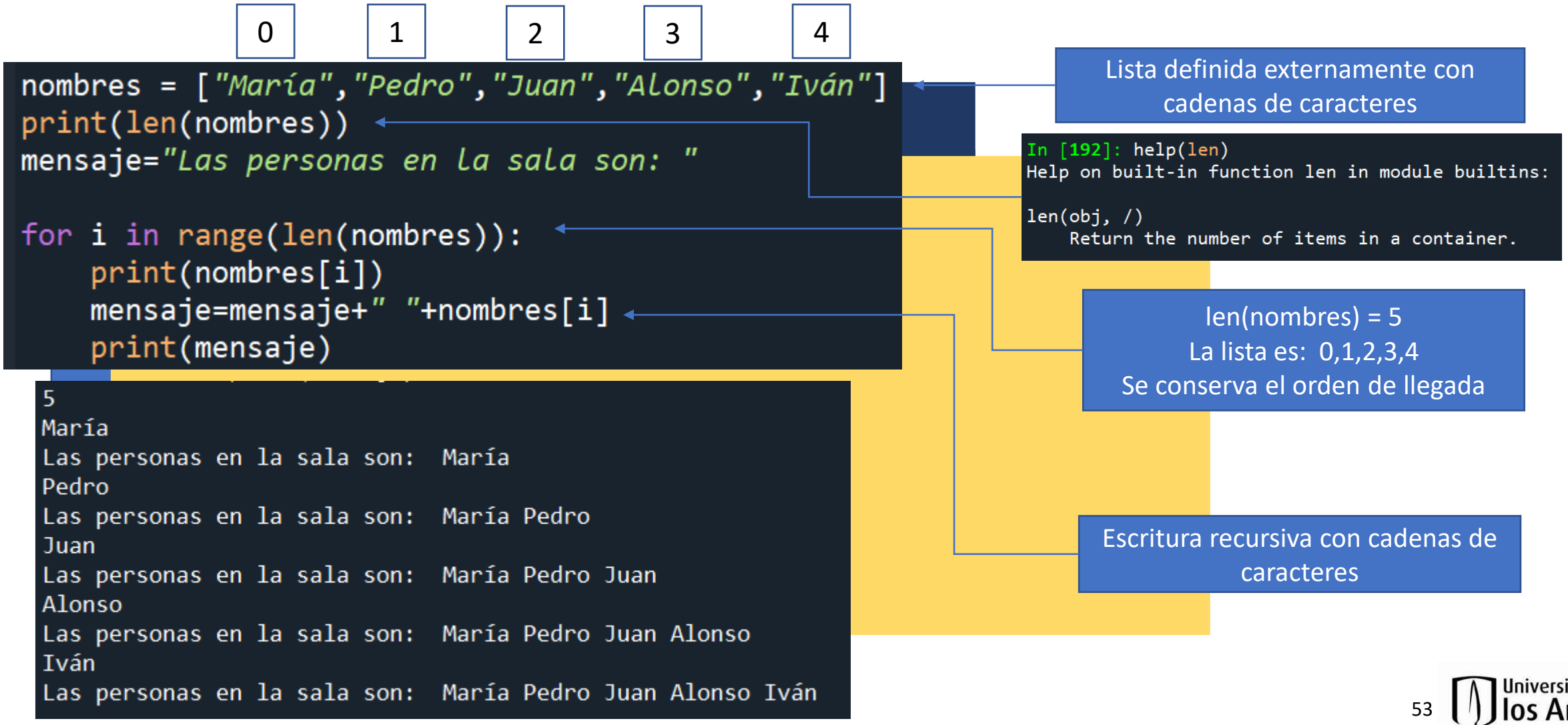
b. For – Python (Ejemplo) – Solución en clase cambiando el orden de llegada

```
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()



b. For – R (Ejemplo)

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

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

```
[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"
```

Lista como objeto
(Lista de cadenas de caracteres)

Mensaje de inicio como objeto

Concatenar caracteres (con función)
Operación recursiva

¿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)

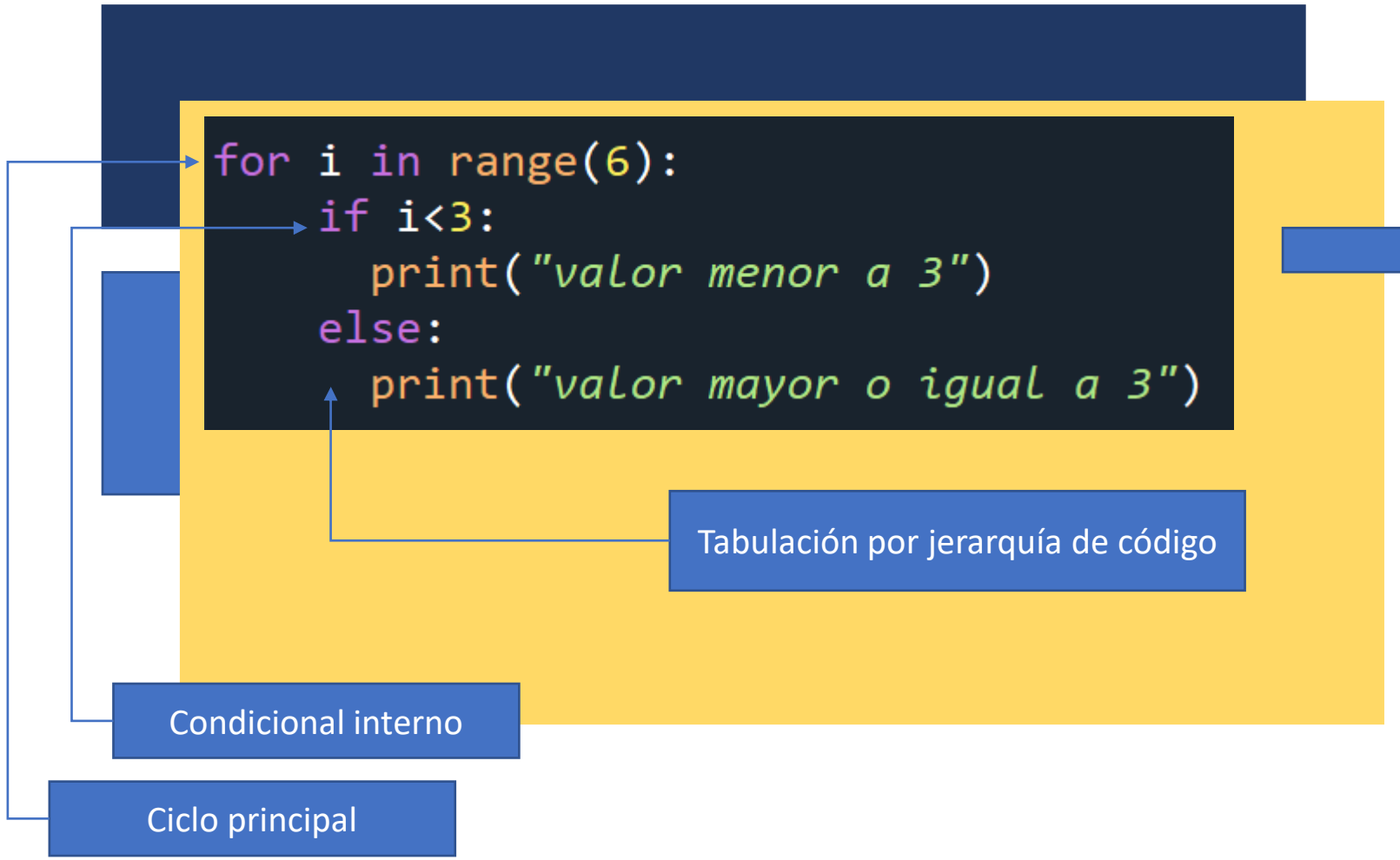
Convertir un elemento que esta dentro de una macro (iterador) a caracteres para concatenar

Concatenar caracteres
(similar a Python)
Operación recursiva

¿Cómo lo programaría si quisiera iterar sobre números?

El comando tokenize puede ayudar

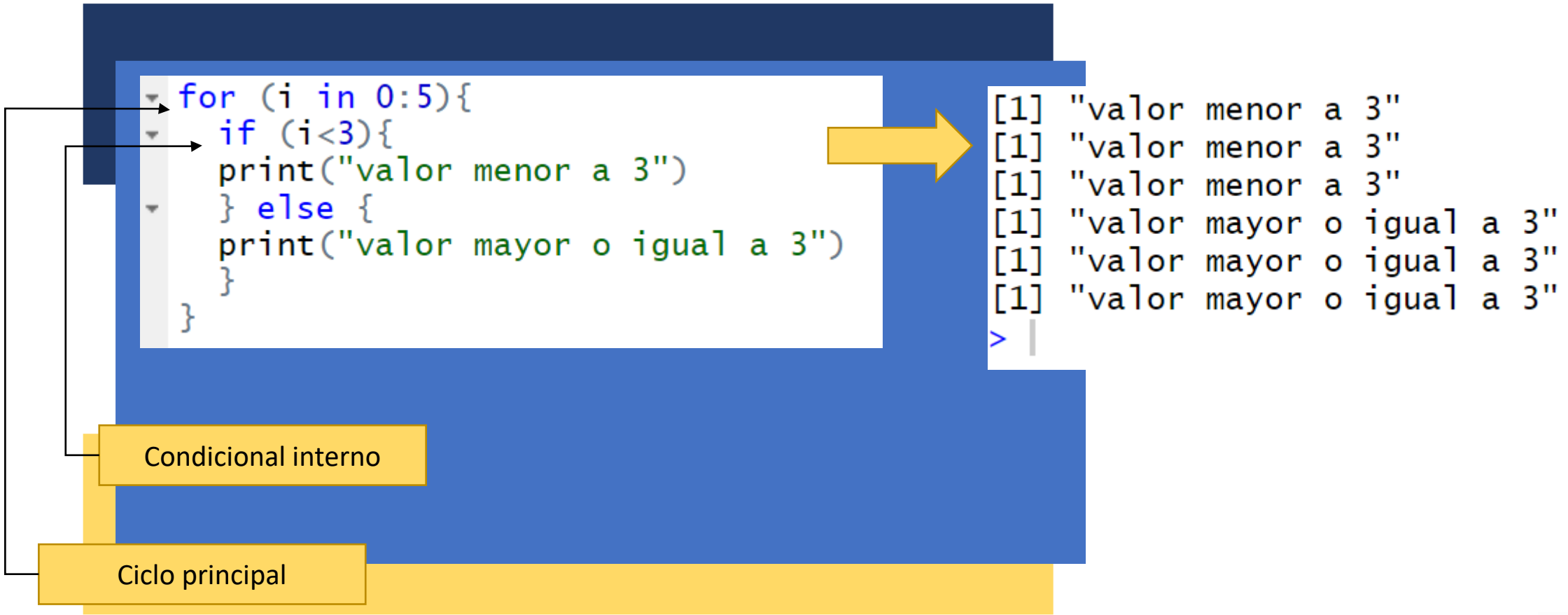
c. Anidación 1 – Python




```
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



c. Anidación 1 – Stata

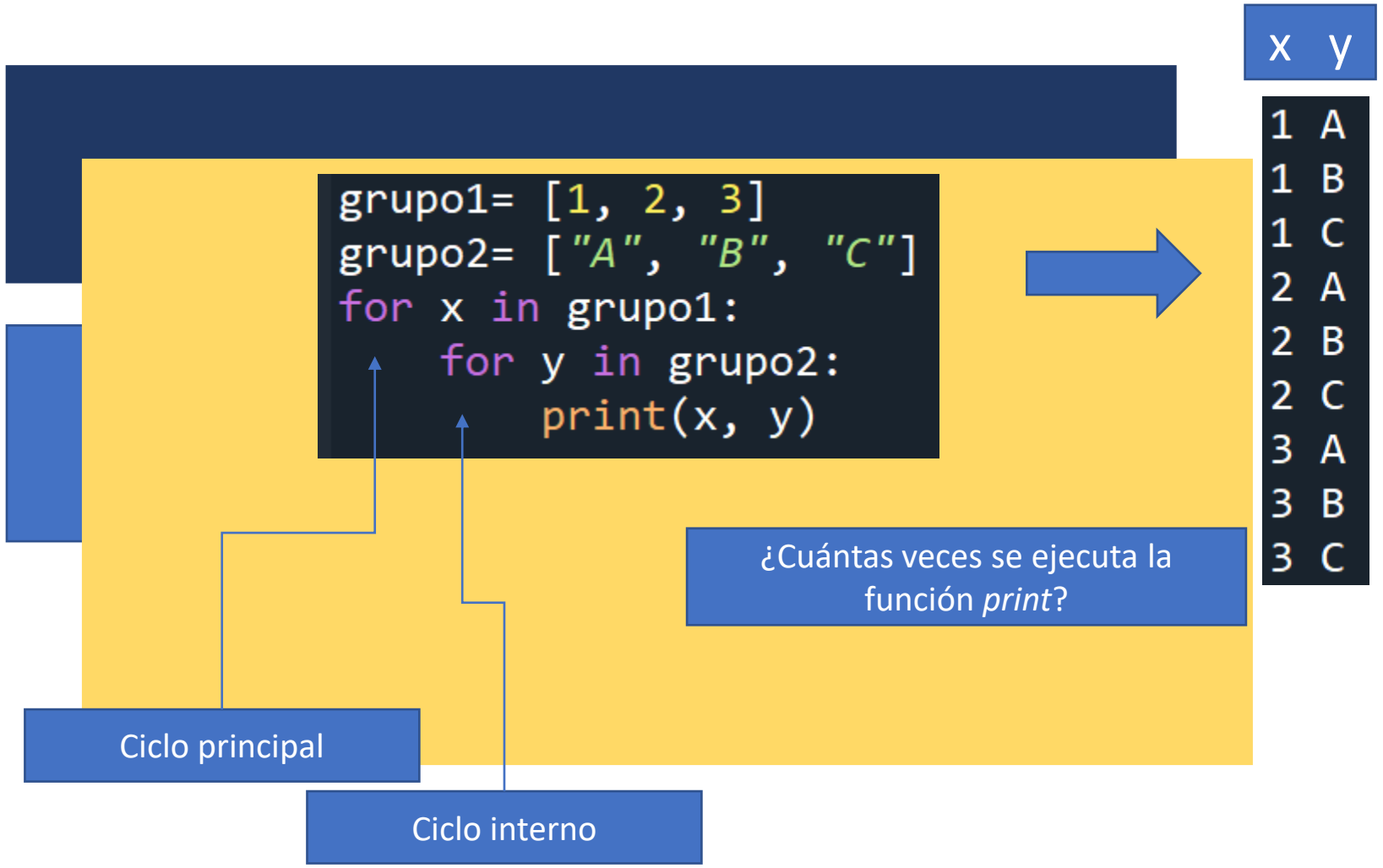
```
foreach i in 0 1 2 3 4 5 {
    if `i'<3{
        display "Valor menor a 3"
    }
    else {
        display "Valor mayor o igual a 3"
    }
}
```

```
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
```

Condicional interno

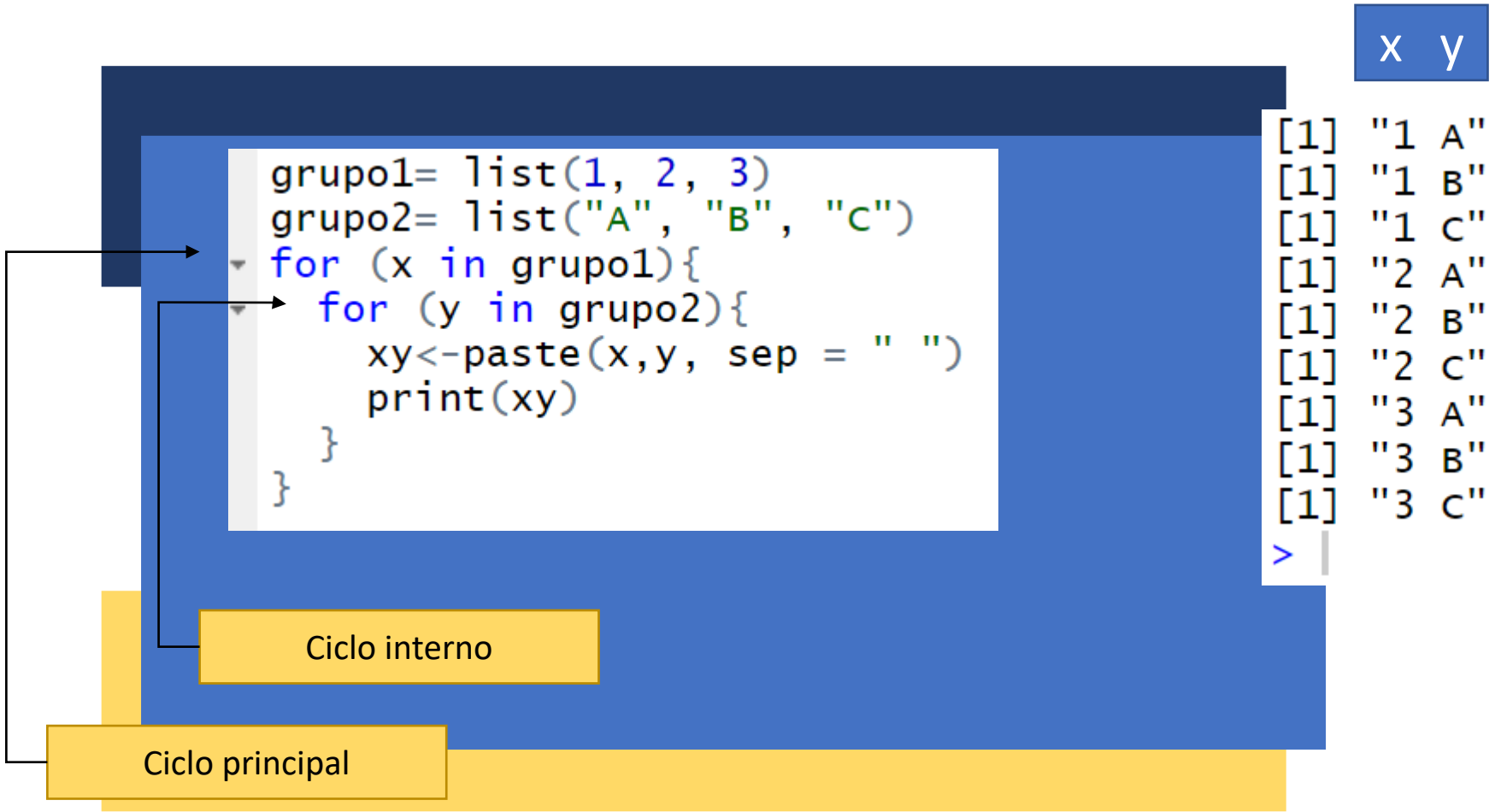
Ciclo principal

c. Anidación 2 – Python



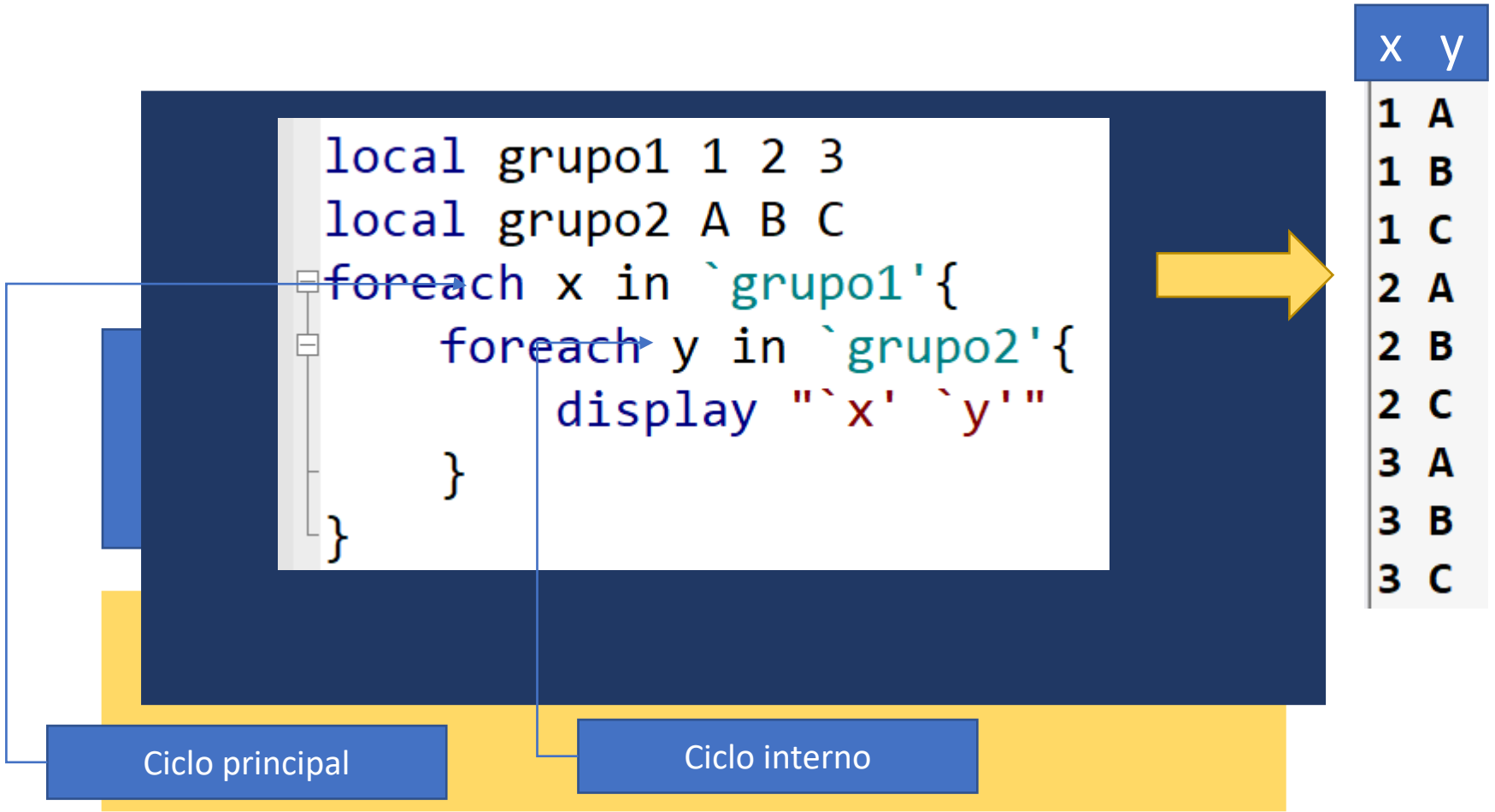
- Por cada iteración en el ciclo principal hay 3 en el ciclo interno

c. Anidación 2 – R



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

c. Anidación 2 – Stata



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/