

# Introducción a la Programación Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2023

Programación Imperativa: Estructuras de Control

# Variables en imperativo

- ▶ Nombre asociado a un espacio de memoria.
- ▶ La variable puede tomar distintos valores a lo largo de la ejecución del programa.
- ▶ En Python se **declaran** dando su nombre (y opcionalmente su tipo):  
`x: int` # x es una variable de tipo int.  
`c: chr` # c es una variable de tipo char.
- ▶ Programación imperativa:
  - ▶ Conjunto de variables.
  - ▶ Instrucciones que van cambiando sus valores.
  - ▶ Los valores finales, deberían resolver el problema.

# Programa en imperativo

- ▶ Colección de tipos, funciones y procedimientos.
- ▶ Su evaluación consiste en ejecutar una por una las instrucciones del bloque.
- ▶ El orden entre las instrucciones es importante:
  - ▶ Siempre de arriba hacia abajo.

# Instrucciones

- ▶ Asignacion
- ▶ Condicional (if ... else ...)
- ▶ Ciclos (while ...)
- ▶ Procedimientos  
(funciones que no devuelven valores pero modifican sus argumentos)
- ▶ Retorno de control (con un valor, return)

# Asignación

Semántica de la asignación:

- Sea  $e$  una expresión cuya evaluación no modifica el estado

$\#estado\ a$

$v = e;$

$\#vale\ v == e@a \wedge z_1 = z_1@a \wedge \dots \wedge z_k = z_k@a$

# Asignación

Semántica de la asignación:

- Sea  $e$  una expresión cuya evaluación no modifica el estado

$\#estado\ a$

$v = e;$

$\#vare\ v == e@a \wedge z_1 = z_1@a \wedge \dots \wedge z_k = z_k@a$

donde  $z_1, \dots, z_k$  son todas las variables del programa en cuestión distintas a  $v$  que están definidas hasta ese momento.

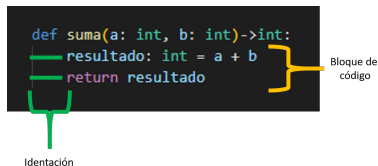
- Las otras variables se supone que no cambian así que, por convención, no hace falta decir nada.
- Si la expresión  $e$  es la invocación a una función que recibe parámetros por referencia, puede haber más cambios, pero al menos sabemos que:

$\#vare\ v == e@a$

está en la *poscondición* o *asegura* de la asignación.

# Identación

- ▶ La indentación es a un lenguaje de programación, lo que la sangría al lenguaje humano escrito.
- ▶ En ciertos lenguajes de programación, la indentación determina la presencia de un bloque de instrucciones (Python es uno de ellos).
- ▶ En otros lenguajes, un bloque puede determinarse de otra manera: por ejemplo encerrándolo entre llaves { }.



```
def suma(a: int, b: int)->int:
    resultado: int = a + b
    return resultado
```

Identación

Bloque de código

# Alcance, ámbito o scope de las variables

- ▶ El alcance de una variable, se refiere al ámbito o espacio donde una variable es reconocida.
- ▶ Una variable sólo será válida dentro del bloque (función/procedimiento) donde fue declarada. Al terminar el bloque, la variable se destruye. Estas variables se denominan **variables locales**.
- ▶ Las variables declaradas fuera de todo bloque son conocidas como **variables globales** y cualquier bloque puede acceder a ella y modificarla.



# Variables locales

Un ejemplo con Python

- ▶ `x` sólo está definida dentro del bloque de instrucciones del procedimiento `ejemploLocalScope`.
- ▶ El intento de acceder a `x` fuera del procedimiento termina en un error en tiempo de ejecución.
- ▶ Aunque el IDE ya nos lo había advertido.

```
def ejemploLocalScope():  
    x: int = 19  
    print("x: " + str(x))  
  
ejemploLocalScope()  
print("x: " + str(x))
```

Variable Local

Imprimirá: x: 19

NameError: name 'x' is not defined

```
ejemplo_scope.py > ...  
1 def ejemploLocalScope():  
2     x: int = 19  
3     print("x: " +  
4  
5  
6 ejemploLocalScope()  
7 print("x: " + str(x))
```

"x" is not defined Pylance([reportUndefinedVariable](#))  
(function) x: Any  
[View Problem \(Alt+F8\)](#) [Quick Fix... \(Ctrl+.\)](#)

# Variables globales

Un ejemplo con Python

- ▶ `x` sólo está definida de manera global.
- ▶ Cualquier bloque puede acceder a ella.

```
def ejemploGlobalScope():  
    print("x: " + str(x))  
  
def sumarEnElGlobal():  
    global x  
    x = x + 120  
    print("x: " + str(x))  
  
x: int = 20  
ejemploGlobalScope(),  
sumarEnElGlobal(),  
ejemploGlobalScope(),  
print("x: " + str(x))
```

Imprimirá:

- Primera vez: x: 20
- Segunda vez: x: 140

En Python, explícitamente hay que referenciar a la variable global para modificarla.

Variable Global

# Variables locales

## Particularidades de Python: bloque del IF

- ▶ Los conceptos de variables locales y globales trascienden a los lenguajes.
- ▶ De hecho, además de otros tipos de scope, habrá scope a nivel de funciones, procedimientos, clases, packages, etc.
- ▶ Y no todos los lenguajes tendrán siempre el mismo comportamiento con respecto a estos temas.

```
#include <iostream>

using namespace std;

int main()
{
    int x = 10;
    if(x > 5){
        int y = 6;
    } else {
        int y = 12;
    }
    cout<<y;

    return 0;
}
```

Y tiene un scope local al bloque del IF (Ejemplo en C++)

```
x: int = 10
if(x > 5):
    y: int = 6
else:
    y: int = 12

print("y = " + str(y))
```

En Python el bloque mínimo es a nivel función. Los bloques dentro de un IF o un WHILE están dentro del anterior.

y = 6

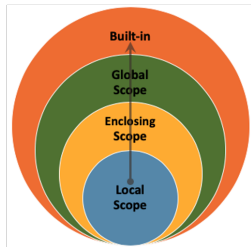
```
main.cpp: In function 'int main()':
main.cpp:21:12: error: 'y' was not declared in this scope
 21 |     cout<<y;
    |         ^
```

# A alcance de Variables en Python

Python distingue 4 niveles de visibilidad o alcance

- ▶ Local: corresponde al ámbito de una función.
- ▶ No local o Enclosed: no está en el ámbito local, pero aparece en una función que reside dentro de otra función.
- ▶ Global: declarada en el cuerpo principal del programa, fuera de cualquier función.
- ▶ Integrado o Built-in: son todas las declaraciones propias de Python (por ejemplo: def, print, etc)

```
def outer():  
    enclosed: int = 1  
  
    def inner():  
        local: int = 2  
        print("INNER: variableGlobal declarada fuera de todo: ", variableGlobal)  
        print("INNER: enclosed declarada en outer: ", enclosed)  
        print("INNER: local declarada en inner: ", local)  
  
    inner()  
  
    print("OUTER: variableGlobal declarada fuera de todo: ", variableGlobal)  
    print("OUTER: enclosed declarada en outer: ", enclosed)  
    print("OUTER: local declarada en inner: ", local)  
  
variableGlobal: int = 3  
outer()  
print("GLOBAL: variableGlobal declarada fuera de todo: ", variableGlobal)  
print("GLOBAL: enclosed declarada en outer: ", enclosed)  
print("GLOBAL: local declarada en inner: ", local)
```



# Alcance de Variables en Python

La referencias también tienen su scope:

- ▶ Al pasar un parámetro por referencia, esta referencia vivirá dentro del scope de la función
- ▶ Analicemos este caso:
  - ▶ En la primer instrucción, `y` toma las referencias de `x` (en este scope, las referencias de `y` se 'pierden')
  - ▶ Al modificar `y`, se está modificando el valor de `x`
  - ▶ Al salir de la función, `y` nunca cambió

```
4  ✓ def duplicar(x: list, y: list):  
5      y = x  
6      y *= 2  
7
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

ANTES:

x: ['a', 'b', 'c']

y: ['d', 'e']

DESPUES:

x: ['a', 'b', 'c', 'a', 'b', 'c']

y: ['d', 'e']

# Condicionales

```
if (B):  
    uno  
else:  
    dos
```

- ▶ B Tiene que ser una expresión booleana. Se llama **guarda**.
- ▶ *uno* y *dos* son bloques de instrucciones.
- ▶ Pensemos el condicional desde la transformación de estados...

# Condicionales

Pensemos el condicional desde la transformación de estados

- ▶ Todo el condicional tiene su precondition y su postcondition:  $P_{if}$  y  $Q_{if}$
- ▶ Cada bloque de instrucciones, también tiene sus condiciones y postcondiciones.

```
# estado  $P_{if}$ 
if (B):
    # estado  $P_{uno}$ 
    uno
    # estado  $Q_{uno}$ 
else:
    # estado  $P_{dos}$ 
    dos
    # estado  $Q_{dos}$ 
# estado  $Q_{if}$ 
#  $(B \wedge \text{estado } Q_{uno}) \vee (\neg B \wedge \text{estado } Q_{dos})$ 
# Después del IF, se cumplió B y  $Q_{uno}$  o, no se cumplió B y  $Q_{dos}$ 
```

# Condicionales

Un ejemplo con Python

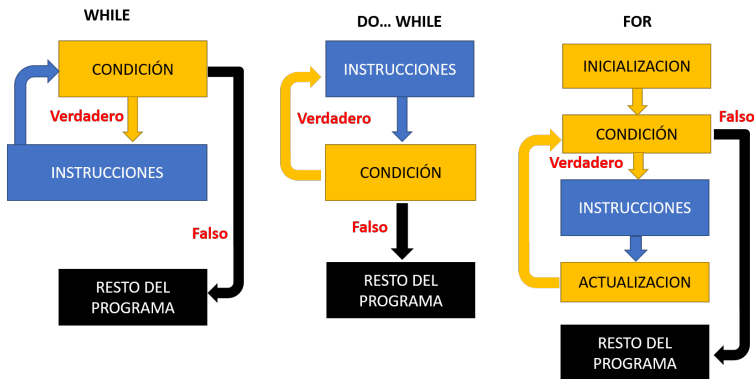
```
def elegirMayor(x: int, y: int) -> int:
    z: int
    print("x = " + str(x) + " | y = " + str(y) )
    if x > y :
        print("x es mayor que y")
        z = x
        print("(Se cumple B) -> z toma el valor de x")
    else:
        print("y es mayor o igual que x")
        z = y
        print("(No se cumple B) -> z toma el valor de y")

    return z
```



# Ciclos

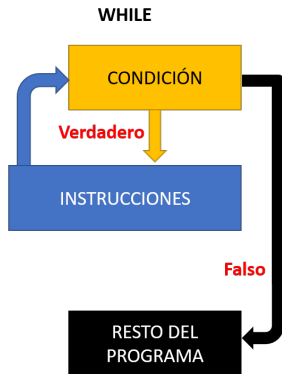
- ▶ En los lenguajes imperativos, existen estructuras de control encargadas de repetir un bloque de código mientras se cumpla una condición.
- ▶ Cada repetición suele llamarse iteración.
- ▶ Existen diferentes esquemas de iteración, los más conocidos son:
  - ▶ While, Do While, For



# While en Python

## Sintaxis del While

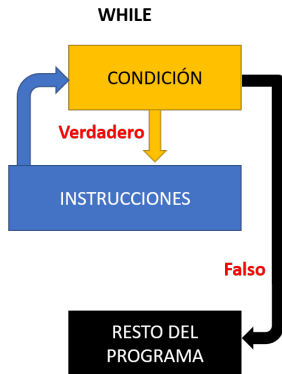
```
while(condición de finalización):  
    ...  
    ...  
    Bloque de Instrucciones...  
    Dentro del while  
    ...  
    ...  
  
Bloque de Instrucciones...  
FUERA del while
```



# While en Python

## Sintaxis del While

```
while(condición de finalización):  
    ...  
    ...  
    Bloque de Instrucciones...  
    Dentro del while  
    ...  
    ...  
  
Bloque de Instrucciones...  
FUERA del while
```



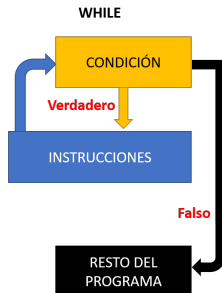
# While en Python

Un programa que muestra por pantalla el número ingresado por el usuario, hasta que el usuario ingresa 0.

```
numero = int(input('Ingresa un número. 0 para terminar: '))

while(numero != 0):
    print('Usted ingresó: ', numero)
    numero = int(input('Ingresa un número. 0 para terminar: '))

print('Fin del programa.')
```

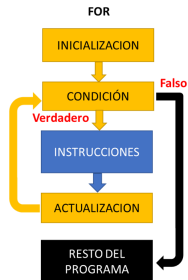


- ▶ `input`: espera que el usuario ingrese algo por teclado.
- ▶ `int(input('...'))`: convierte en `int` lo que el usuario ingresó por teclado.

# For en Python

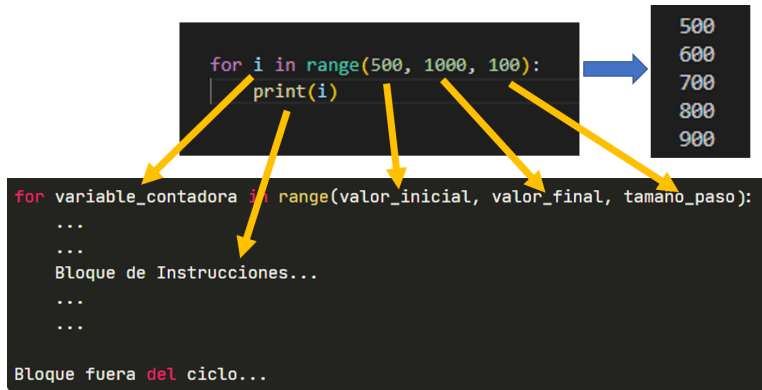
## Sintaxis del For

```
for variable_contadora in range(valor_inicial, valor_final, tamaño_paso):  
    ...  
    ...  
    Bloque de Instrucciones...  
    ...  
    ...  
Bloque fuera del ciclo...
```



# For en Python

¿Qué hace este programa?



# Interrumpiendo ciclos: Break

- ▶ La instrucción Break permite romper la ejecución de un ciclo.
- ▶ No fomentamos su uso, sólo mencionamos su existencia (por varios motivos que van más allá del alcance de la materia).
  - ▶ Su uso le quita declaratividad al código.
  - ▶ Desde el punto de vista de analizar la correctitud de un programa, traerá problemas (pero eso ya lo verán más adelante en la carrera).

```
while(True):  
    numero = int(input('Ingresa un número. 0 para terminar: '))  
    print('Usted ingresó: ', numero)  
    if(numero==0):  
        break  
  
print('Fin del programa.')
```

# Ciclos y transformación de estados...

```
while(condición de finalización):  
    ...  
    ...  
    Bloque de Instrucciones...  
    Dentro del while  
    ...  
    ...  
  
Bloque de Instrucciones...  
FUERA del while
```

- ▶ En un programa imperativo, cada instrucción transforma el estado.
- ▶ Mediante la transformación de estados, podemos hacer una ejecución simbólica del programa.



# Ciclos y transformación de estados...

```
while(condición de finalización):  
    ...  
    ...  
    Bloque de Instrucciones...  
    Dentro del while  
    ...  
    ...  
  
Bloque de Instrucciones...  
FUERA del while
```

- ▶ En un programa imperativo, cada instrucción transforma el estado.
- ▶ Mediante la transformación de estados, podemos hacer una ejecución simbólica del programa.
- ▶ ¿Cómo sería la transformación de estados de un ciclo?
  - ▶ Podemos pensar en el ciclo como una instrucción: con un estado previo y uno posterior

# Ciclos y transformación de estados...

```
while(condición de finalización):  
    ...  
    ...  
    Bloque de Instrucciones...  
    Dentro del while  
    ...  
    ...  
  
Bloque de Instrucciones...  
FUERA del while
```

- ▶ En un programa imperativo, cada instrucción transforma el estado.
- ▶ Mediante la transformación de estados, podemos hacer una ejecución simbólica del programa.
- ▶ ¿Cómo sería la transformación de estados de un ciclo?
  - ▶ Podemos pensar en el ciclo como una instrucción: con un estado previo y uno posterior
  - ▶ ¿Qué sucede dentro del ciclo? ¿Qué sucede en cada iteración?

# Ciclos y transformación de estados...

```
while(condición de finalización):  
    ...  
    ...  
    Bloque de Instrucciones...  
    Dentro del while  
    ...  
    ...  
  
Bloque de Instrucciones...  
FUERA del while
```

- ▶ En un programa imperativo, cada instrucción transforma el estado.
- ▶ Mediante la transformación de estados, podemos hacer una ejecución simbólica del programa.
- ▶ ¿Cómo sería la transformación de estados de un ciclo?
  - ▶ Podemos pensar en el ciclo como una instrucción: con un estado previo y uno posterior
  - ▶ ¿Qué sucede dentro del ciclo? ¿Qué sucede en cada iteración?
- ▶ Más adelante en la carrera, verán cómo manejar estas situaciones.