

# Estructuras cíclicas I

## Las estructuras cíclicas mientras y hacer mientras

Jonatan Gómez Perdomo, Ph.D.

[jgomezpe@unal.edu.co](mailto:jgomezpe@unal.edu.co)

Arles Rodríguez, Ph.D.

[aerodriguezp@unal.edu.co](mailto:aerodriguezp@unal.edu.co)

Camilo Cubides, Ph.D.(c)

[eccubidesg@unal.edu.co](mailto:eccubidesg@unal.edu.co)

Grupo de investigación en vida artificial – Research Group on Artificial Life – (Alife)

Departamento de Ingeniería de Sistemas e Industrial

Facultad de Ingeniería

Universidad Nacional de Colombia



# Agenda

- 1 La estructura de control de ciclos mientras (while)
- 2 La estructura de control de ciclos hacer-mientras (do)
  - Uso de variables bandera
- 3 Forzando la terminación de un ciclo
- 4 Teorema fundamental de la programación estructurada



# El ciclo mientras (while) I

El ciclo **mientras** (while) permite ejecutar un bloque de instrucciones mientras que una expresión booleana dada se cumpla, es decir, mientras su evaluación dé como resultado verdadero.

La expresión booleana se denomina **condición de parada** y siempre se evalúa antes de ejecutar el bloque de instrucciones; tras esto se pueden presentar dos casos:

- Si la condición no se cumple, el bloque no se ejecuta.
- Si la condición se cumple, el bloque se ejecuta, después de lo cual la instrucción vuelve a empezar, es decir, la condición se vuelve a evaluar.



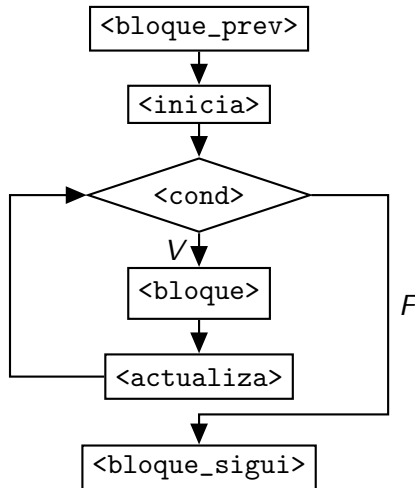
# El ciclo mientras (while) II

En el caso en que la condición se evalúe la primera vez como falsa, el bloque de instrucciones no será ejecutado, lo cual quiere decir que el número de repeticiones o iteraciones de este bloque será cero (0). Si la condición siempre evalúa a verdadero, la instrucción se ejecutará indefinidamente, es decir, un número infinito de veces.



# El ciclo mientras (while) III

Un ciclo **mientras** (while) se puede representar gráficamente mediante un diagrama de flujo de la siguiente manera.



# El ciclo mientras (while) IV

Un esquema textual que en Python representa un ciclo mientras (while) es la que se da en el siguiente fragmento de código.

```
<bloque_prev>  
<inicia>  
while(<cond>):  
    <bloque>  
    <actualiza>  
<bloque_sigui>
```

No olvidar que el ámbito (alcance) del ciclo mientras es lo que esté indentado una (1) vez debajo de la verificación de la condición de parada.

En la expresión (<cond>) los paréntesis son opcionales.



# El ciclo mientras (while) V

Las partes del ciclo mientras (while) son:

- El fragmento <bloque\_prev> es el bloque de instrucciones previas que han sido ejecutadas antes del ciclo.
- El fragmento <inicia> es el bloque de instrucciones donde se inicializan las variables que intervienen en la condición de parada.
- El fragmento <cond> es la condición de parada que se evalúa cada vez que se inicia o se reinicia el ciclo.



# El ciclo mientras (while) VI

- El fragmento <bloque> es el bloque de instrucciones principal del ciclo que se ejecuta mientras la condición se cumpla.
- El fragmento <actualiza> es el bloque que se utiliza para actualizar las variables que son utilizadas para evaluar la condición de parada cuando se intenta reiniciar el ciclo.
- El fragmento <bloque\_sigui> es el bloque de instrucciones que se ejecutan después de terminar de ejecutar el ciclo.





# Ejemplo 1 del ciclo mientras (while) I

## Ejemplo

Para el siguiente fragmento de código que contiene un ciclo mientras (while)

```
<bloque_prev>  
i = 0  
while(i <= 6):  
    print(i)  
    i = i + 1  
<bloque_sigui>
```



# Ejemplo 1 del ciclo mientras (while) II

## Ejemplo (continuación)

se tiene que en el fragmento del código:

- <inicia> corresponde a la instrucción  
`i = 0`
- <cond> corresponde a la instrucción  
`i <= 6`
- <bloque> corresponde a la instrucción  
`print(i)`
- <actualiza> corresponde a la instrucción  
`i = i + 1`



# Ejemplo 1 del ciclo mientras (while) III

## Ejemplo (continuación)

cuando se ejecuta este ciclo, lo que se obtiene en la consola de salida es el texto que se presenta en el cuadro a la derecha

```
<bloque_prev>  
i = 0  
while(i <= 6):  
    print(i)  
    i = i + 1  
<bloque_sigui>
```

```
...  
0  
1  
2  
3  
4  
5  
6  
...
```



# Ejemplo 1 del ciclo mientras (while) IV

## Ejemplo (continuación)

en este caso la salida que se produce es la anterior porque el bloque

```
print(i)
```

se ejecuta siete veces, variando  $i$  desde 0 hasta cuando  $i$  toma el valor 7, que hace que la condición se evalúe falso y por lo tanto termina el ciclo; obsérvese que la variable termina el ciclo con valor  $i = 7$ , pero este valor no se imprime pues para este caso la condición se evalúa falso.



## Ejemplo 2 del ciclo mientras (while) I

### Ejemplo

Para el siguiente fragmento de código que contiene un ciclo while

```
<bloque_prev>
i = 2                # inicializa a i en 2
j = 25              # inicializa a j en 25
while i < j:         # mientras i sea menor a j
    print(i, j, sep = ", ")    # imprime los valores de i y j
    i *= 2            # i = i*2; i se multiplica por 2 en cada paso
    j += 10           # j = j + 10; se incrementa de 10 en 10
print("the end.")    # se ejecuta al terminar el ciclo
print(i, j, sep = ", ")    # valores finales de i y j
<bloque_sigui>
```



## Ejemplo 2 del ciclo mientras (while) II

### Ejemplo (continuación)

las variables  $i$  y  $j$  se inicializan con los valores 2 y 100 respectivamente, luego se verifica que 2 sea menor estrictamente que 100, a continuación se imprime el valor de la variable  $i$  seguido por una coma y espacio (", "), seguido por el valor de la variable  $j$ , seguido de un salto de línea; a continuación se multiplica la variable  $i$  por 2 y a la variable  $j$  se le suma 10. Esto se realiza hasta que el valor de la variable  $i$  sea mayor o igual a el valor de la variable  $j$ .



## Ejemplo 2 del ciclo mientras (while) III

### Ejemplo (continuación)

El resultado de la ejecución de este ciclo mostrado en la consola de salida es el texto que se presenta en el cuadro a la derecha.

```
<bloque_prev>
i = 2
j = 25
while i < j:
    print(i, j, sep = ", ")
    i *= 2
    j += 10
print("the end.")
print(i, j, sep = ", ")
<bloque_sigui>
```

```
...
...
2, 25
4, 35
8, 45
16, 55
32, 65
64, 75
the end.
128, 85
...
...
```

## Ejemplo 2 del ciclo mientras (while) IV

### Ejemplo (continuación)

Obsérvese que las variables  $i$  y  $j$  terminan el ciclo con los valores 128 y 85, y como 128 no es menor que 85, entonces el ciclo se para y se sale de éste, por esta razón no se imprimen estos valores dentro del ciclo, sólo se imprimen luego de salir del ciclo.





# El mínimo número positivo de la máquina I

## Ejemplo (el mínimo número positivo de la máquina)

Dado que los números reales que son representables en un computador son finitos, entonces es posible hablar del menor número positivo representable en la máquina, es decir el número

$$x_{\min} = \min \{x : (x \text{ es un número de máquina}) \wedge (x > 0)\}$$

Para encontrar dicho número hay un algoritmo muy sencillo que permite encontrar el valor.



# El mínimo número positivo de la máquina II

## Ejemplo (continuación)

El algoritmo consiste en calcular los términos de una progresión geométrica que inicia con el término  $x_0 = 1$  y para la cual los términos siguientes se calculan utilizando la razón de la progresión  $r = 1/2$ , es decir,  $x_{n+1} = \frac{x_n}{2}$ , esto se realiza mientras cada nuevo término es positivo.



# El mínimo número positivo de la máquina III

## Ejemplo (continuación)

La codificación en Python de una función constante que permite hallar el mínimo número positivo representable en la máquina junto con su programa principal es

```
def min_maquina():  
    Xo = 1.0  
    Xi = Xo / 2.0  
    while Xi > 0.0:  
        Xo = Xi  
        Xi = Xo / 2.0  
    return Xo  
  
print("El mínimo número positivo", end = " ")  
print("en esta máquina es:", min_maquina())
```

# El mínimo número positivo de la máquina IV

## Ejemplo (continuación)

Si se ejecuta el anterior programa, el resultado que se obtiene es el siguiente

El mínimo número positivo en esta máquina es: 5e-324

como para calcular cada término de la progresión geométrica se necesita únicamente el término anterior, entonces son necesarias sólo dos variables, las cuales se utilizarán de la siguiente manera:

- La variable X0 representa el término  $x_n$  y se inicializa con el valor  $x_0 = 1$ .
- La variable Xi representa el término  $x_{n+1}$  y se inicializa con el valor  $X_0 / 2.0$ .

# El mínimo número positivo de la máquina V

## Ejemplo (continuación)

- Ahora, dentro del ciclo la variable  $X_0$  jugará el rol del término  $x_{n+1}$  mediante la asignación  $X_0 = X_i$ .
- A la variable  $X_i$  se le asigna el siguiente término calculado de la progresión y es asignado mediante la expresión  $X_i = X_0 / 2.0$ .
- Las dos últimas rutinas descritas se realizan mientras el valor de la variable  $X_i$  sea mayor a 0.0.
- El ciclo en algún momento deja de ejecutarse ya que el conjunto de los números de máquina es finito, la progresión geométrica es decreciente y está acotada inferiormente por 0.
- Finalmente, se retorna el valor almacenado en la variable  $X_0$  pues fue el último término de la progresión que era distinto de 0.



# Problemas

## Problemas

- 1 Diseñe un algoritmo que involucre un ciclo y que nunca ingrese al ciclo.
- 2 Diseñe un algoritmo que involucre un ciclo y que se ejecute indefinidamente.
- 3 Diseñe un algoritmo que pida un valor entero, y que siga leyendo valores enteros mientras que alguno de esos valores no represente el código ASCII de una letra mayúscula del alfabeto inglés.



# Agenda

- 1 La estructura de control de ciclos mientras (while)
- 2 La estructura de control de ciclos hacer-mientras (do)**
  - Uso de variables bandera
- 3 Forzando la terminación de un ciclo
- 4 Teorema fundamental de la programación estructurada



# El ciclo hacer-mientras (do) I

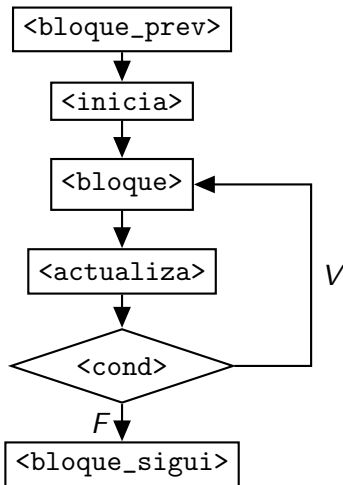
Existe otra estructura cíclica en programación, ésta se conoce como un ciclo **hacer-mientras** (do). Esta estructura es casi equivalente a la estructura mientras (while), ya que usualmente se utiliza cuando con seguridad y de forma anticipada se sabe que se hará al menos una evaluación del bloque principal del ciclo. En esta estructura cíclica la verificación de la condición de parada se realiza al final del ciclo.





# El ciclo hacer-mientras (do) II

Un ciclo **hacer-mientras** (do) se puede representar gráficamente mediante un diagrama de flujo de la siguiente manera.



# El ciclo hacer-mientras (do) III

Sin embargo, no existe una instrucción en Python para el ciclo hacer-mientras (do) y éste es aproximado usando un ciclo mientras (while). Un esquema textual que en Python es una aproximación al ciclo hacer-mientras (do) es la que se da en el siguiente fragmento de código, obsérvese que antes del ciclo tanto el bloque principal del ciclo como la actualización se ejecutan una vez.

```
<bloque_prev>  
<inicia>  
<bloque>  
<actualiza>  
while(<cond>):  
    <bloque>  
    <actualiza>  
<bloque_sigui>
```



# El mínimo número positivo de la máquina versión do

## Ejemplo

Un algoritmo para obtener el mínimo número positivo de la máquina que usa un ciclo hacer-mientras (do) se muestra a continuación

```
def min_maquina():  
    Xi = 1.0 # Valor inicial  
    Xo = Xi  
    Xi = Xo / 2.0  
    while(Xi > 0.0):  
        Xo = Xi  
        Xi = Xo / 2.0  
    return Xo  
  
print("El mínimo número positivo", end = " ")  
print("en esta máquina es:", min_maquina())
```

# Agenda

- 1 La estructura de control de ciclos mientras (while)
- 2 La estructura de control de ciclos hacer-mientras (do)**
  - Uso de variables bandera
- 3 Forzando la terminación de un ciclo
- 4 Teorema fundamental de la programación estructurada



# Uso de variables bandera I

Aunque con “copiar” el bloque de instrucciones interno, antes del ciclo mientras (`while`) se obtiene la funcionalidad esperada de un ciclo hacer-mientras (`do`), esta aproximación no se recomienda mucho pues se duplica código, el programa se hace más largo dificultando su lectura y comprensión y se pueden introducir errores fácilmente cuando se requiera cambiar precisamente ese bloque duplicado.



# Uso de variables bandera II

Para evitar este tipo de problemas, se puede usar una variable booleana (conocida como bandera) que garantice que se entre al ciclo al menos una vez. Un esquema textual en Python para esta aproximación al ciclo hacer-mientras (do) es la que se da en el siguiente fragmento de código, obsérvese el comportamiento de la bandera garantizando que el ciclo se ejecute al menos una vez.



# Uso de variables bandera III

```
<bloque_prev>
<inicia>
bandera = True
while(bandera or <cond>):
    bandera = False
    <bloque>
    <actualiza>
<bloque_sigui>
```

## Ejemplo

El algoritmo para el obtener el mínimo número positivo de la máquina usando la aproximación de un ciclo hacer-mientras (do) con una bandera como se muestra a continuación

# Uso de variables bandera IV

## Ejemplo (continuación)

```
def min_maquina():
    Xi = 1.0                                # Valor inicial
    bandera = True                          # Permite que se ejecute el
                                           # ciclo al menos una vez
    while(bandera or Xi > 0.0):
        bandera = False                    # Indica que ya se entro al
                                           # ciclo y que luego sólo se
                                           # debe evaluar la condición

        Xo = Xi
        Xi = Xo / 2.0
    return Xo

print("El mínimo número positivo", end = " ")
print("en esta máquina es:", min_maquina())
```



# Agenda

- 1 La estructura de control de ciclos mientras (while)
- 2 La estructura de control de ciclos hacer-mientras (do)
  - Uso de variables bandera
- 3 Forzando la terminación de un ciclo
- 4 Teorema fundamental de la programación estructurada



# Terminación forzada de un ciclo I

Cuando se construyen algoritmos que utilizan ciclos, algunos programadores requieren que dadas unas condiciones dentro del bloque de instrucciones internas del ciclo, se pare la ejecución del ciclo y se continúe con las instrucciones subsiguientes.

Este tipo de salidas forzadas, se obtiene usando la instrucción `break`. Aunque el uso de ésta no se recomienda, pues hace difícil realizarle un seguimiento al programa, pues dejan muchas variables en valores que no se pueden controlar y/o determinar apropiadamente.



# Terminación forzada de un ciclo II

Sin embargo, la instrucción `break` es utilizada por programadores ya sea por facilidad (algunos no realizan un buen diseño de los algoritmos cíclicos), a veces por velocidad de ejecución (sin verificar las condiciones de continuación o terminación del ciclo), ésta última situación es muy común cuando se usan ciclos de tipo `para` (`for`).

## Ejemplo

Desarrollar un programa que lea números enteros y los sume hasta que lea un cero (0). Un algoritmo que soluciona este problema es el que se muestra a continuación



# Terminación forzada de un ciclo III

## Ejemplo (continuación)

```
suma = 0
while True:
    dato = int(input("Ingrese un número entero " +
                    "a sumar o 0 para salir: "))
    if(dato == 0):
        break
    suma += dato
print("La suma es: " + str(suma))
```

Se puede ver que el ciclo siempre se ejecutaría (su condición siempre es verdadera) por lo que la instrucción `break` se utiliza para poder salir cuando el usuario ingresa un cero (0).



# Eliminación de la instrucción `break` I

Todo programa con un ciclo mientras (`while`) que tenga una instrucción `break` se puede ajustar para que no la use, sin embargo, el proceso de eliminar dicha instrucción requiere leer y entender correctamente el papel de la instrucción `break` en el ciclo. Cuando se usa por velocidad en los ciclos para (`for`), es muy difícil (sino imposible) eliminarla.



# Eliminación de la instrucción `break` II

## Ejemplo (Eliminar la instrucción `break` del programa del ejemplo anterior)

Lo primero que se puede determinar es que se puede cambiar la condición del ciclo para que deje de ser siempre verdadera por una que dependa del valor del dato leído (que es lo que se usa para ejecutar la instrucción `break`). Lo otro importante es que se debe leer al menos una vez un dato, ya sea para no leer más datos (si el usuario ingresa 0), o para sumarlos. De esta manera se puede usar la aproximación al ciclo hacer-mientras (`do`) que usa una bandera. Por lo tanto, la eliminación del `break` produce el algoritmo que se muestra a continuación



# Eliminación de la instrucción break III

## Ejemplo (continuación)

```
dato = 0
suma = 0
bandera = True
while (bandera or dato != 0):
    bandera = False
    dato = int(input("Ingrese un número entero " +
                    "a sumar o 0 para salir: "))
    suma += dato
print("La suma es: " + str(suma))
```



# Eliminación de la instrucción break IV

## Ejemplo (continuación)

Una simplificación del programa anterior es el siguiente programa, hay que tener muy en cuenta que para hacer este tipo de mejoras hay que entender muy bien el algoritmo.

```
dato = 1
suma = 0
while (dato != 0):
    dato = int(input("Ingrese un número entero " +
                    "a sumar o 0 para salir: "))
    suma += dato
print("La suma es: " + str(suma))
```





# Agenda

- 1 La estructura de control de ciclos mientras (while)
- 2 La estructura de control de ciclos hacer-mientras (do)
  - Uso de variables bandera
- 3 Forzando la terminación de un ciclo
- 4 Teorema fundamental de la programación estructurada



# Teorema fundamental de la programación estructurada

En 1966 C. Böhm y G. Jacopini demostraron el siguiente teorema

## Teorema (Teorema fundamental de la programación estructurada)

*Un lenguaje de programación es completo en Turing siempre que tenga variables enteras no negativas, las operaciones aritméticas elementales sobre dichas variables, y que permita ejecutar enunciados en forma secuencial, incluyendo enunciados de asignación (=), selección (if) y ciclos (while).*

Lo que significa que en este momento

!ya es posible programar cualquier algoritmo que pueda ser ejecutado en un computador¡.



# Problemas varios I

## Problemas

- 1 Imprimir un listado con los números del 1 al 100 cada uno con su respectivo cuadrado.
- 2 Imprimir un listado con los números impares desde 1 hasta 999 y seguidamente otro listado con los números pares desde 2 hasta 1000.
- 3 Imprimir los números pares en forma descendente hasta 2 que son menores o iguales a un número natural  $n \geq 2$  dado.
- 4 En 2022 el país  $A$  tendrá una población de 25 millones de habitantes y el país  $B$  de 18.9 millones. Las tasas de crecimiento anual de la población serán de 2% y 3% respectivamente. Desarrollar un algoritmo para informar en que año la población del país  $B$  superará a la de  $A$ .



# Problemas varios II

## Problemas

- 5 Diseñar una función que permita calcular el épsilon de la máquina. El epsilon de máquina es el número decimal más pequeño que sumado a 1 se puede representar de manera precisa en la máquina (que no es redondeado), es decir, retorna un valor diferente de 1, éste da una idea de la precisión o número de cifras reales que pueden ser almacenadas en la máquina. La idea es realizar un ciclo en el cual se realiza la operación  $1 + \epsilon$  para potencias de 2 desde  $\epsilon = 2^0$  y continuando con potencias decrecientes de 2 ( $\epsilon = 2^{-1}, \epsilon = 2^{-2}, \epsilon = 2^{-3}, \epsilon = 2^{-4}, \dots$ ) hasta obtener que el resultado de la suma  $1 + \epsilon$  no se altere.

