Estructuras Cíclicas II

Para

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Carlos Andres Sierra, M.Sc.
casierrav@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Agenda

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Los ciclos para (for) como iteradores de colecciones
4. La colección Rango (range)
5. Las colecciones range y los ciclos para (for)

La estructura de control de ciclos para (for)

Existe otra estructura cíclica que es de uso frecuente en programación, el ciclo para (`for`). Esta estructura de control tiene dos propositos primordiales que no siempre son soportados por todo lenguaje de programación:

1. Como una forma compacta de escribir un ciclo mientras (`while`).
2. Para *iterar* sobre los elementos de una colección de elementos.

Esta estructura es usualmente utilizada cuando se conocen los valores inicial y final de la variable que es utilizada en la condición de parada.

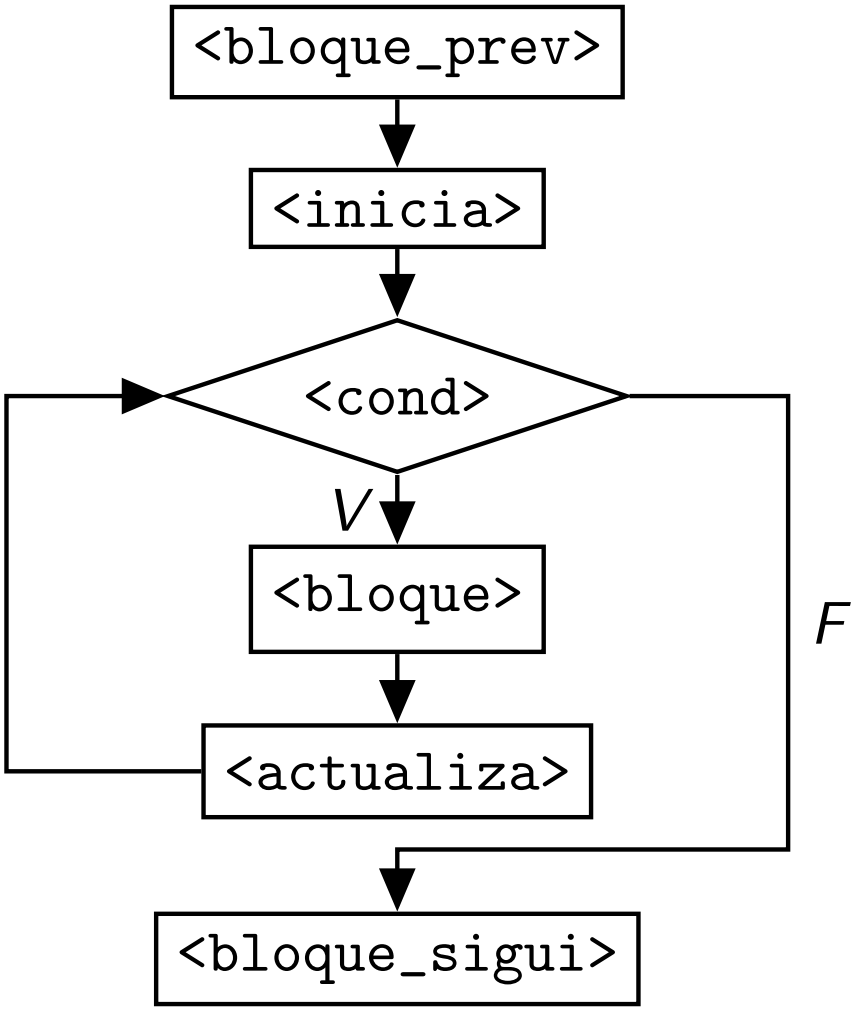
Agenda

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Los ciclos para (for) como iteradores de colecciones
4. La colección Rango (range)
5. Las colecciones range y los ciclos para (for)

El ciclo para como versión del ciclo mientras I

En muchos lenguajes (lamentablemente en Python no es de esta manera), el ciclo para (`for`) se usa para escribir de manera compacta un ciclo mientras (`while`). Dada la representación gráfica mediante un diagrama de flujo de un ciclo mientras (`while`), la cual es la siguiente.

El ciclo para como versión del ciclo mientras II



El ciclo para como versión del ciclo mientras III

Y la sintaxis general de un ciclo mientras (`while`) en los lenguajes de programación C++ o Java

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

El ciclo para como versión del ciclo mientras IV

Un esquema textual que en C++ o Java representa dicho ciclo mientras (while) usando un ciclo para (for) es el que se da en el siguiente fragmento de código.

```
<bloque_prev>
for(<inicia>; <cond>; <actualiza>){
    <bloque>
}
<bloque_sigui>
```

El ciclo para como versión del ciclo mientras V

Ejemplo (La suma de los primeros n números naturales)

Las dos siguientes funciones en C++ o Java permiten calcular la suma de los primeros n números naturales positivos, es decir, permiten calcular el valor de la expresión

$1 + 2 + 3 + \cdots + (n-1) + n$, que abreviadamante se escribe como $\sum_{i=1}^{n} i$

```
int suma(int n){
    int s = 0;
    int i = 1;
    while(i <= n){
        s = s + i;
        i++;
    }
    return s;
}
```

```
int suma(int n){
    int s = 0;
    for(int i = 1; i <= n; i++){
        s = s + i;
    }
    return s;
}
```

Agenda

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Los ciclos para (for) como iteradores de colecciones
4. La colección Rango (range)
5. Las colecciones range y los ciclos para (for)

Ciclos para (for) como iteradores de colecciones I

Un ciclo para (`for`) puede ser usado (y en Python es su único uso) para obtener uno a uno los elementos de una colección de elementos y poder realizar con cada uno de ellos el mismo bloque de operaciones.

Un esquema textual que en Python representa dicho ciclo para (`for`) es el que se da en el siguiente fragmento de código.

```
<bloque_prev>
for <elemento> in <coleccion>:
    <bloque>
<bloque_sigui>
```

Ciclos para (for) como iteradores de colecciones II

en donde:

- El fragmento <bloque_prev> es el bloque de instrucciones previas que han sido ejecutadas antes del ciclo.
- El fragmento <elemento> es la variable que se usa para ir recorriendo (iterando) sobre los elementos de la colección (tomará como valor cada uno de ellos en cada iteración).
- El fragmento <coleccion> es la colección de elementos que será recorrida (iterada) con el ciclo.

Ciclos para (for) como iteradores de colecciones III

- El fragmento <bloque> es el bloque de instrucciones principal del ciclo que se ejecuta con cada uno de los elementos de la colección.
- El fragmento <bloque_sigui> es el bloque de instrucciones que se ejecutan después de terminar de ejecutar el ciclo.

El ciclo para (for) se puede leer en castellano como: "*Para cada elemento en la colección realice las instrucciones en el* <bloque>".

Recorriendo una lista I

Ejemplo

Dada la lista[†] de frutas:

```
["Tomate de árbol", "Maracuyá ", "Guayaba"],
```

imprimir todos sus elementos.

Un programa que permite solucionar este problema es

```
frutas = ["Tomate de árbol", "Maracuyá ", "Guayaba"]
for f in frutas:                    # para cada elemento f
                                    # en la lista de frutas
   print(f)
```

[†]Una lista es una colección de elementos que se puede definir en Python.



Recorriendo una lista II

Ejemplo (continuación)

El resultado de ejecutar este programa será entonces:

```
...
Tomate de árbol
Maracuyá
Guayaba
...
```

Forzando la detención del recorrido una lista I

Si se requiere detener el ciclo antes de iterar sobre toda la colección de elementos se debe usar la sentencia `break`.

Ejemplo (Recorriendo una lista hasta encontrar un elemento especial)

Dada la lista de frutas

```
["Pera", "Maracuyá ", "Guayaba", "Lulo", "Granadilla"]
```

imprimir sus elementos hasta encontrar la fruta `"Guayaba"`.

Forzando la detención del recorrido una lista II

Ejemplo (continuación) (Recorriendo una lista hasta encontrar un elemento especial)

Un programa que permite solucionar este problema es

```
frutas = ["Pera", "Maracuyá ", "Guayaba", "Lulo",
          "Granadilla"]
for f in frutas:                    # para cada elemento f
                                    # en la lista de frutas
   print(f)
   if f == "Guayaba":
      break
```

Forzando la detención del recorrido una lista III

Ejemplo (continuación)

El resultado de ejecutar este programa será entonces:

```
...
Pera
Maracuyá
Guayaba
...
```

Agenda

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Los ciclos para (for) como iteradores de colecciones
4. La colección Rango (range)
5. Las colecciones range y los ciclos para (for)

La colección Rango (range) I

Existen varias colecciones que se pueden iterar en Python, una de ellas es la colección Rango (`range`). Una colección Rango (`range`) es una colección de números en un intervalo (rango) semi-abierto, definido por valor inicial (el lado cerrado del intervalo), un valor final (que no se incluye en el rango, es decir el lado abierto del intervalo), y un valor de incremento/decremento usado a partir del valor inicial para determinar que valores quedan en el rango. Si no se da el valor de inicio, éste se fija en cero (0) y si no se da valor de incremento/decremento, éste se fija en uno (1).

La colección Rango (range) II

Las tres formas de crear un rango en Python se presentan a continuación:

`range(N)`: La colección de enteros $0, 1, 2, \ldots, N-1$, ésta es la colección de enteros empezando en cero (0), y llegando hasta $N-1$ (no incluye N) incrementando de uno en uno. Si N no es positivo genera el rango vacío.

`range(C,F)`: La colección de enteros $C, C+1, C+2, \ldots, F-1$, ésta es la colección de enteros empezando en C, y llegando hasta $F-1$ (no incluye F) incrementando de uno en uno. Si $F \leq C$ genera el rango vacío.

La colección Rango (range) III

`range(C,F,I)`: La colección de enteros $C, C+I, C+2I, \ldots, C+kI$, hasta el k tal que $C+(k+1)I \geq F$ si $I > 0$ o $C+(k+1)I \leq F$ si $I < 0$. Esta es la colección de enteros empezando en C, y llegando hasta el más cercano $C+kI$ a F (no incluye F) incrementando/decrementando de I en I. Dependiendo de los valores dados se puede generar el rango vacío.

Nota

Tengan en cuenta que los valores C, F, I deben ser valores enteros, es decir, $C, F, I \in \mathbb{Z}$.

La colección Rango (range) IV

Ejemplos

- El comando `range(6)` genera la secuencia `[0, 1, 2, 3, 4, 5]`.
- El comando `range(10, 21)` genera la secuencia `[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]`.
- El comando `range(-7, 8)` genera la secuencia `[-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7]`.
- El comando `range(2,11,2)` genera la secuencia `[2, 4, 6, 8, 10]`.
- El comando `range(11,20,2)` genera la secuencia `[11, 13, 15, 17, 19]`.
- El comando `range(5,-1,-1)` genera la secuencia `[5, 4, 3, 2, 1, 0]`.
- El comando `range(-2)` genera la secuencia `[ ]`.

Agenda

1. La estructura de control de ciclos para (for)
2. Los ciclos para como versiones compactas de ciclos mientras
3. Los ciclos para (for) como iteradores de colecciones
4. La colección Rango (range)
5. Las colecciones range y los ciclos para (for)

Las colecciones `range` y los ciclos para (`for`) I

Los rangos se combinan perfectamente con la instrucción para (`for`):

Se define una variable que se utilizará para recorrer cada número en el rango. En castellano se podría pensar como: "*Para cada número i en el rango dado*".

Esto se puede observar en los siguientes ejemplos:

Las colecciones range y los ciclos para (for) II

Ejemplo

El programa que se presenta a la izquierda permite imprimir de forma creciente los números enteros desde -5 hasta 5, la salida de la ejecución de este programa se muestra al lado derecho

```
for i in range(-5, 6, 1):
    print(i)
```

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

Las colecciones range y los ciclos para (for) III

Ejemplo

El programa que se presenta a la izquierda permite imprimir de forma decreciente los números enteros desde 10 hasta 1, la salida de la ejecución de este programa se muestra al lado derecho

```
for i in range(10, 0, -1):
    print(i)
```

```
...
10
9
8
7
6
5
4
3
2
1
...
```

Las colecciones range y los ciclos para (for) IV

Ejemplo

El programa que se presenta a la izquierda permite imprimir de forma creciente los múltiplos de 5 desde 0 hasta 50, la salida de la ejecución de este programa se muestra al lado derecho

```
for i in range(0, 55, 5):
    print(i)
```

```
...
0
5
10
15
20
25
30
35
40
45
50
...
```

Las colecciones range y los ciclos para (for) V

Ejemplo

El programa que se presenta a la izquierda permite imprimir de forma creciente los números enteros desde 10 hasta 20, la salida de la ejecución de este programa se muestra al lado derecho

```
for i in range(10, 21):
    print(i)
```

```
...
10
11
12
13
14
15
16
17
18
19
20
...
```

Las colecciones range y los ciclos para (for) VI

Ejemplo

El programa que se presenta a la izquierda permite imprimir de forma creciente los números enteros desde 0 hasta 10, la salida de la ejecución de este programa se muestra al lado derecho

```
for i in range(11):
    print(i)
```

```
...
0
1
2
3
4
5
6
7
8
9
10
...
```

La suma de los primeros n números naturales I

Ejemplo (La suma de los primeros n números naturales)

Dos funciones en Python, diferentes, que permiten calcular la suma de los primeros n números naturales, es decir, permiten calcular la expresión

$1 + 2 + 3 + \cdots + (n-1) + n$, que abreviadamante se escribe como $\sum_{i=1}^{n} i$

```
def suma(n):
    s = 0
    i = 1
    while(i <= n):
        s += i
        i += 1
    return s
```

```
def suma(n):
    s = 0
    for i in range(1, n + 1):
        s += i
    return s
```



La suma de los primeros *n* números naturales II

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

estas dos funciones son equivalentes, ya que ejecutan las mismas modificaciones de las variables, pues se tiene que el fragmento de código:

- <bloque_prev> corresponde a la instrucción

```
s = 0
```

- <inicia> corresponde a la instrucción

```
i = 1
```

- <cond> corresponde a la instrucción

```
i <= n
```

La suma de los primeros *n* números naturales III

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

- <bloque> corresponde a la instrucción

```
s += i
```

- <actualiza> corresponde a la instrucción

```
i += 1
```

- <bloque_sigui> corresponde a la instrucción

```
return s
```

La suma de los primeros *n* números naturales IV

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

En la construcción de estas funciones aparecen dos variables que tienen una connotación muy importante:

- La variable `i` juega el rol de una *variable contadora* ya que ésta permite llevar el conteo de cuantos ciclos se han efectuado.
- La variable `s` juega el rol de una *variable acumuladora* pues en ésta se acumula o almacena el valor parcial que se desea calcular utilizando el ciclo.

La suma de los primeros *n* números naturales V

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

La codificación en Python de una función que permite sumar los primeros *n* números naturales positivos junto con su programa principal es

```
def suma(n):
    s = 0
    for i in range(1, n + 1):
        s += i
    return s

def main():
    n = int(input("n? = "))
    print("La suma de los primeros n números es:", end=" ")
    print(suma(n))

main()
```


La suma de los primeros *n* números naturales VI

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

Si se ejecuta el anterior programa y como entrada se ingresa el valor `n` $= 6$ (por ejemplo el número de caras de un dado ⚀ + ⚁ + ⚂ + ⚃ + ⚄ + ⚅), el resultado que se obtiene es el siguiente

```
n? = 6
La suma de los primeros n números es: 21
```

La suma de los primeros *n* números naturales VII

Ejemplo (continuación) (La suma de los primeros *n* números naturales)

En general es fácil comprobar si el resultado que se obtiene utilizando la función `suma(n)` es correcto, pues existe una fórmula muy sencilla para calcular la suma de los primeros *n* números naturales positivos sin necesidad de realizar la suma exhaustiva. Esta fórmula es

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

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. Imprimir los números de 1 hasta un número natural n dado, cada uno con su respectivo factorial.
5. Calcular el valor de 2 elevado a la potencia n.
6. Leer un número natural n, leer otro dato de tipo real x y calcular x^n.
7. Diseñe un programa que muestre las tablas de multiplicar del 1 al 9.

Problemas varios II

Problemas

8. Diseñar una función que permita calcular una aproximación de la función exponencial alrededor de 0 para cualquier valor $x \in \mathbb{R}$, utilizando los primeros n términos de la serie de Maclaurin

$$\exp(x, n) \approx \sum_{i=0}^{n} \frac{x^i}{i!}.$$

9. Diseñar una función que permita calcular una aproximación de la función seno alrededor de 0 para cualquier valor $x \in \mathbb{R}$ (x dado en radianes), utilizando los primeros n términos de la serie de Maclaurin

$$\sin(x, n) \approx \sum_{i=0}^{n} \frac{(-1)^i x^{2i+1}}{(2i+1)!}.$$

Problemas varios III

Problemas

10. Diseñar una función que permita calcular una aproximación de la función coseno alrededor de 0 para cualquier valor $x \in \mathbb{R}$ (x dado en radianes), utilizando los primeros n términos de la serie de Maclaurin

$$\cos(x, n) \approx \sum_{i=0}^{n} \frac{(-1)^i x^{2i}}{(2i)!}.$$

11. Diseñar una función que permita calcular una aproximación de la función logaritmo natural alrededor de 0 para cualquier valor $x \in \mathbb{R}^+$, utilizando los primeros n términos de la serie de Maclaurin

$$\ln(x, n) \approx \sum_{i=0}^{n} \frac{1}{2i+1} \left(\frac{x^2 - 1}{x^2 + 1} \right)^{2i+1}.$$

Problemas varios III

Problemas

12. Diseñar una función que permita calcular una aproximación de la función arco tangente para cualquier valor $x \in [-1, 1]$, utilizando los primeros n términos de la serie de Maclaurin (al evaluar esta función el resultado que se obtiene está expresado en radianes)

$$\arctan(x, n) \approx \sum_{i=0}^{n} \frac{(-1)^i x^{2i+1}}{(2i+1)}.$$