Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 9: Ciclos

Control del flujo de ejecución de un programa

En el **paradigma de programación imperativa**, un programa es una secuencia finita de instrucciones: operaciones que transforman datos (el *estado* del programa), o bien modifican el flujo de ejecución.

Estructuras de control de flujo:

Funciones

✓

Condicionales

✓

Ciclos

(hoy)

Ciclos

while *CONDICIÓN*:
** *BLOQUE***

CONDICIÓN es una expresión de tipo `bool`.
BLOQUE es un bloque de código.

Flujo de ejecución:

1. Se evalúa *CONDICIÓN*.
2. Si es verdadera: se ejecuta *BLOQUE* y se vuelve al paso 1.
3. Si es falsa: se termina.

Ejemplo:

```
1  i:int = 10
2  while i>0:
3      print(i)
4      i = i - 1
5  print('despegue!')
```

Ciclos

while *CONDICIÓN*:
**   *BLOQUE***

CONDICIÓN es una expresión de tipo `bool`.
BLOQUE es un bloque de código.

Flujo de ejecución:

1. Se evalúa *CONDICIÓN*.
2. Si es verdadera: se ejecuta *BLOQUE* y se vuelve al paso 1.
3. Si es falsa: se termina.

Ejemplo:

```
1  def n_equis(n:int) -> str:
2      ''' Requiere: n>=0
3          Devuelve: la concatenación de n veces la letra 'x'.
4      '''
5      res:str = ''
6      i:int = 0
7      while i < n:
8          res = res + 'x'
9          i = i + 1
10     return res
```

Ciclos · Terminación

¿La ejecución de este ciclo termina?

```
1  i:int = 10
2  while i > 0:
3      print(i)
4      i = i - 1
5  print('despegue!')
```

- La variable `i` empieza valiendo 10.
- En cada ejecución del cuerpo del ciclo, `i` se decrementa en 1.
- Entonces, es inevitable que en algún momento, `i` llegue a 0.
- En ese momento, la condición `i>0` será falsa, y el ciclo terminará. ✓

Ciclos · Terminación

Ejercicio: Mostrar que el siguiente programa termina:

```
1  def n_equis(n:int) -> str:
2      ''' Requiere: n>=0
3          Devuelve: la concatenación de n veces la letra 'x'.
4      '''
5      res:str = ''
6      i:int = 0
7      while i < n:
8          res = res + 'x'
9          i = i + 1
10     return res
```

Solución: (pensarlo antes de leer!!)

- La variable `i` empieza valiendo 0.
- En cada ejecución del cuerpo del ciclo, `i` se incrementa en 1.
- Por la cláusula *Requiere* de la especificación, sabemos que $\mathtt{n} \geq 0$. Además en el cuerpo del ciclo no se modifica el valor de `n`.
- Entonces, es inevitable que en algún momento, `i` llegue al valor de `n`.
- En ese momento, la condición $\mathtt{i} < \mathtt{n}$ será falsa, y el ciclo terminará. ✓

Ciclos · Correctitud

¿Este ciclo es correcto respecto de la especificación? Es decir, ¿hace lo que queremos que haga?

```
1  def n_equis(n:int) -> str:
2      res:str = ''
3      i:int = 0
4      while i < n:
5          res = res + 'x'
6          i = i + 1
7      return res
```

Queremos poder afirmar que, al terminar la ejecución del ciclo, la variable `res` vale la concatenación de `n` letras `'x'` (como indica la especificación).

Identifiquemos qué cosas podemos afirmar que valen al principio y al final de cada vez que se ejecuta el cuerpo del ciclo:

- `i` se mueve entre 0 y `n` (inclusive); es decir: $0 \leq \texttt{i} \leq \texttt{n}$
- la variable `res` vale la concatenación de `i` letras `'x'`.

(Notar que estas cosas tal vez no valen durante toda la ejecución del cuerpo del ciclo; solo antes y después de cada iteración.)

A este conjunto de afirmaciones lo llamamos predicado invariante del ciclo (notación: $\mathcal{I}$). Es muy útil para probar la correctitud de un ciclo, y también para organizar la escritura de ciclos.

Ciclos · Correctitud

$\mathcal{I}$: $0 \leq \texttt{i} \leq \texttt{n}$, y `res` vale la concatenación de `i` letras `'x'`

```
1  def n_equis(n:int) -> str:
2      res:str = ''
3      i:int = 0
4      while i < n:
5          res = res + 'x'
6          i = i + 1
7      return res
```

Observar que:

- $\mathcal{I}$ vale al llegar al `while` por primera vez.
  - En ese momento, `i` vale 0 y `res` vale `''` (es decir, 0 letras `'x'`). ✓
- Si $\mathcal{I}$ vale al principio de una iteración cualquiera (antes de la línea 5), también valdrá al final de esa iteración (después de la línea 6).
  - Al principio, sabemos que $\texttt{i} < \texttt{n}$ (porque la condición fue verdadera). Entonces, luego de sumarle 1 en la línea 6, `i` valdrá a lo sumo `n`. ✓
  - Al principio, `res` tiene `i` letras `'x'`. Al final, eso mismo sigue valiendo, porque se agregó una `'x'` más y se incrementó `i`. ✓
- En algún momento se niega la condición y termina el ciclo. A partir de $\mathcal{I}$ (que sigue valiendo por el punto anterior) y la negación de la condición, podemos afirmar que la función devuelve lo que esperamos.
  - De $\texttt{i} \geq \texttt{n}$ y $0 \leq \texttt{i} \leq \texttt{n}$ concluimos que $\texttt{i} = \texttt{n}$. Entonces, `res` vale la concatenación de `n` letras `'x'`, tal como pide la postcondición. ✓

Ciclos · Correctitud

```
# vale ℐ
while CONDICIÓN:
    # valen ℐ y CONDICIÓN
    BLOQUE
    # vale ℐ
# valen ℐ y la negación de CONDICIÓN
```

El predicado invariante describe en forma simple qué hace el ciclo desde el principio hasta el final:

- Vale justo antes del ciclo.
- En cada iteración del ciclo:
  - Vale justo antes de ejecutar el *BLOQUE*, momento en que también vale la *CONDICIÓN*.
  - No necesariamente vale *durante* la ejecución del *BLOQUE*.
  - Vale justo después de ejecutar el *BLOQUE*.
- Vale justo después del ciclo, momento en que también vale la negación de la *CONDICIÓN*. Estas dos cosas combinadas describen el *efecto* de la ejecución del ciclo.

Ciclos · Correctitud

Ejercicio:

```
1  def sumatoria(n:int) -> int:
2      ''' Requiere: n>=0.
3          Devuelve: la suma de los enteros entre 0 y n (inclusive).
4      '''
5      res:int = 0
6      i:int = 0
7      while i < n:
8          i = i + 1
9          res = res + i
10     return res
```

1. Mostrar que el ciclo termina.
2. Elegir un predicado invariante $\mathcal{I}$, y usarlo para mostrar que cuando termina el ciclo, se cumple la postcondición de la función.

Repaso de la clase de hoy

- Ciclos: comando `while`.
  - Existen otras formas de repetir la ejecución de código.
    P.ej., Python tiene `for`, que es muy útil y veremos más adelante.
    Por ahora, trabajaremos sólo con `while`.
- Terminación de ciclos.
- Correctitud de ciclos. Predicado invariante $\mathcal{I}$.

Bibliografía complementaria: APPP2, capítulo 6. En este caso, no recomiendo leer la versión en inglés, que mezcla estos temas con otros que veremos más adelante.

Con lo visto, ya pueden resolver toda la Guía de Ejercicios 4.