Ciclos: Ejemplo "Buscar una letra en un string"

Tenemos la función `buscar` especificada de la siguiente manera:

```
def buscar(letra:str, texto:str) -> bool:
    '''
    Requiere: len(letra)==1
    Devuelve: True sii letra aparece en texto.
    '''
```

Recordemos que "sii" es una abreviatura para "si y sólo si". Equivalentemente, la cláusula Devuelve podría escribirse: "`True` si `letra` aparece en `texto`; `False` en caso contrario".

1. Una solución posible

Hay varios algoritmos para resolver el problema planteado. Veamos una primera idea:

1. Inicializar una variable booleana `encontré` con valor `False`.
2. Recorrer el texto, carácter a carácter.
3. Si algún carácter es igual a la letra buscada, poner la variable `encontré` en `True`.
4. Al llegar al final del texto, devolver el valor de la variable `encontré`.

Un programa que implementa este algoritmo en Python podría ser así:

```
1  def buscar(letra:str, texto:str) -> bool:
2      '''
3      Requiere: len(letra)==1
4      Devuelve: True sii letra aparece en texto.
5      '''
6      encontre:bool = False
7      i:int = 0
8      # (A)
9      while i < len(texto):
10         # (B)
11         if texto[i] == letra:
12             encontre = True
13         i = i + 1
14         # (C)
15     # (D)
16     return encontre
```

A continuación, vamos a demostrar que este programa termina y hace lo esperado.

1.1. Terminación

- Antes del ciclo, la variable `i` se inicializa en 0 (línea 7).
- En cada ejecución del cuerpo del ciclo, `i` se incrementa en 1 (línea 13).
- La variable `texto` no se modifica en el cuerpo del ciclo, por lo cual `len(texto)` arroja siempre el mismo valor.
- Entonces, es inevitable que en algún momento, `i` llegue al valor de `len(texto)`.
- En ese momento, la condición `i<len(texto)` será `False`, por lo que el ciclo terminará. □

1.2. Correctitud

Para entender mejor el código, hagamos un seguimiento manual de la ejecución para un caso particular, `letra='a'`, `texto='casa'`. La siguiente tabla muestra los valores de las variables `i` y `encontre` cada vez que pasamos por alguno de los cuatro puntos de interés (A), (B), (C) y (D):

Lugar	`i`	`encontré`
(A)	0	`False`
(B)	0	`False`
(C)	1	`False`
(B)	1	`False`
(C)	2	`True`
(B)	2	`True`
(C)	3	`True`
(B)	3	`True`
(C)	4	`True`
(D)	4	`True`

Notar que en la segunda vez ejecución del cuerpo del ciclo, la variable `encontré` pasa a valer `True` porque el carácter actual es `'a'`, y después ya nunca vuelve a valer `False`.

Viendo esta tabla, podemos afirmar dos cosas que son verdaderas en cada línea:

- La variable `i` se encuentra entre 0 y `len(texto)` inclusive. Es decir: $0 \leq$ `i` $\leq$ `len(texto)`.
- La variable `encontré` vale `True` sii `letra` aparece en `texto` antes de la posición `i`.

Así, nuestro predicado invariante será:

$\mathcal{I}$: **0** $\leq$ **`i`** $\leq$ **`len(texto)`**, y **`encontré` vale `True` sii `letra` aparece en `texto` antes de la posición `i`**

Notar que proponemos este $\mathcal{I}$ para todo valor posible de `letra` y `texto`; ya no para los valores particulares del ejemplo anterior (`'a'`, `'casa'`).

Sigamos ahora el siguiente razonamiento, que nos lleva desde el principio hasta el final de la ejecución de la función `buscar`:

- $\mathcal{I}$ es verdadero en el punto (A), donde `i` vale 0 y `encontré` vale `False`. Esto tiene sentido porque antes de la primera posición de `texto` claramente no encontramos nada.
- Supongamos que $\mathcal{I}$ es verdadero en el punto (B). Si llamamos ϕ al valor que tiene `i` en ese momento, las siguientes afirmaciones son verdaderas:
  - `i` $= \phi <$ `len(texto)` (porque la condición del `while` dio verdadera)
  - `encontré` vale `True` sii `letra` aparece en `texto` antes de la posición ϕ

  En el cuerpo del ciclo, a `encontré` se le asigna `True` si `texto[`ϕ`]` es igual a `letra` (líneas 11 y 12); además, `i` se incrementa en 1 (línea 13) en cualquier caso. En consecuencia, al terminar el cuerpo del ciclo, en el punto (C), valen las siguientes afirmaciones:
  - `i` $= \phi + 1$
  - `encontré` vale `True` sii `letra` aparece en `texto` antes de la posición $\phi + 1$, o equivalentemente, antes de la posición `i`.

  Como sabíamos que $\phi <$ `len(texto)`, podemos afirmar que $\phi + 1 \leq$ `len(texto)`. O equivalentemente, `i` $\leq$ `len(texto)`. Entonces, podemos afirmar que en (C) vuelve a valer $\mathcal{I}$.
- De esta manera, sabemos que en cada iteración del ciclo, $\mathcal{I}$ empieza y termina siendo verdadero. Después de (C) se evalúa la condición `i<len(texto)`, lo cual no modifica el valor de ninguna variable. Así, mientras la condición resulte verdadera, a lo largo de sucesivas iteraciones tendremos que $\mathcal{I}$ vale en (B), vale en (C), vale en (B), vale en (C), vale en (B), vale en (C), ...

- En algún momento el ciclo termina porque la condición resulta falsa. Dado que evaluar la condición `i<len(texto)` no modifica el valor de las variables, $\mathcal{I}$ sigue siendo verdadero al llegar al punto (D), y es fácil ver que `i` vale `len(texto)`. Entonces, en este momento estamos en condiciones de afirmar que **encontré vale `True` sii `letra` aparece en `texto` (antes de la posición `len(texto)`)**. Y eso es lo que la función debe devolver, según su especificación. □

2. Otra solución posible

Veamos ahora un segundo algoritmo para resolver el problema planteado en la especificación de `buscar`:

1. Recorrer el texto, carácter a carácter.
2. Si algún carácter es igual a la letra buscada, frenar y devolver `True`.
3. Al llegar al final del texto, devolver `False`.

Una virtud de este algoritmo es que se detiene cuando encuentra la letra buscada, ahorrando así algo de trabajo. Podríamos implementarlo en Python con el siguiente programa:

```
1  def buscar(letra:str, texto:str) -> bool:
2      '''
3      Requiere: len(letra)==1
4      Devuelve: True sii letra aparece en texto.
5      '''
6      i:int = 0
7      # (A)
8      while i<len(texto) and texto[i]!=letra:
9          # (B)
10         i = i + 1
11         # (C)
12     # (D)
13     return i < len(texto)
```

A continuación, vamos a mostrar que este programa termina y hace lo esperado.

2.1. Terminación

- Antes del ciclo, la variable `i` se inicializa en 0 (línea 6).
- En cada ejecución del cuerpo del ciclo, `i` se incrementa en 1 (línea 10).
- La variable `texto` no se modifica en el cuerpo del ciclo, por lo cual `len(texto)` arroja siempre el mismo valor.
- Entonces, en algún momento `i` llegará al valor de `len(texto)`, siempre y cuando el ciclo no termine debido al segundo término de la condición (`texto[i]!=letra`).
- En ese momento, la condición `i<len(texto)` será `False`, por lo que el ciclo terminará. □

Notar que el segundo término de la condición del ciclo, `texto[i]!=letra`, puede cortar la ejecución *antes* de llegar al final del texto. Cuando eso ocurra, obviamente el ciclo terminará. Por eso, nos enfocamos en garantizar que el ciclo terminará también en el caso de que nunca se encuentre la letra buscada.

2.2. Correctitud

Nuevamente armamos una tabla para seguir las variables del programa durante el ciclo, para el ejemplo `letra='a'`, `texto='casa'`. En este caso, hay una única variable, `i`:

Lugar	`i`
(A)	0
(B)	0
(C)	1
(B)	1
(C)	2
(B)	2
(C)	3
(B)	3
(C)	4
(D)	4

Esta tabla no pareciera ser muy útil... Lo único que está claro es el rango de `i`: $0 \leq$ `i` $\leq$ `len(texto)`. ¿Cómo hacemos para describir el trabajo realizado durante la ejecución del ciclo? ¿Y cómo llegamos a concluir que la función devuelve lo pedido?

La clave está en el segundo término de la condición del ciclo: `texto[i]!=letra`. Significa que hay dos formas de detener la ejecución del ciclo: al llegar al final del texto, o bien al encontrar la letra buscada. En otras palabras, *avanzaremos mientras no hayamos encontrado la letra buscada*.

Con esto en mente, un posible predicado invariante sería:

$$\mathcal{I}: \mathbf{0} \leq \texttt{i} \leq \texttt{len(texto)}, \text{ y } \textbf{letra no aparece en texto antes de la posición i}$$

Repetimos ahora el razonamiento que nos lleva del principio al final de la ejecución de `buscar`:

- $\mathcal{I}$ es verdadero en el punto (A), donde `i` vale 0 y (obviamente) todavía no encontramos la letra.
- Supongamos que $\mathcal{I}$ es verdadero en el punto (B). Si llamamos ϕ al valor que tiene `i` en ese momento, las siguientes afirmaciones son verdaderas:
  - `i` $= \phi <$ `len(texto)` (porque la condición del `while` es verdadera)
  - `texto[`ϕ`]` $\neq$ `letra` (también porque la condición del `while` es verdadera)
  - `letra` no aparece en `texto` antes de la posición ϕ

  En el cuerpo del ciclo, `i` se incrementa en 1 (línea 13), por lo que pasa a valer $\phi + 1$. Juntando estas observaciones, podemos afirmar que en el punto (C), no hemos encontrado la letra antes de la posición $\phi + 1 =$ `i`. Además, $0 \leq$ `i` $\leq$ `len(texto)` (siguiendo el mismo razonamiento que en el programa anterior). En consecuencia, podemos afirmar que en (C) vuelve a valer $\mathcal{I}$.
- De esta manera, sabemos que en cada iteración del ciclo, $\mathcal{I}$ empieza y termina siendo verdadero: vale en (B), en (C), en (B), en (C), ...
- El ciclo termina cuando la condición `i<len(texto) and texto[i]!=letra` es falsa. Según la propiedad de De Morgan, esto ocurre cuando `i` $\geq$ `len(texto)`, o `texto[i]` $=$ `letra`. Así, al llegar a (D) vale $\mathcal{I}$ junto con alguna de esas dos cosas. Veamos los dos casos posibles:
  - **Supongamos que `i` $\geq$ `len(texto)`**. Como $\mathcal{I}$ afirma que `i` $\leq$ `len(texto)`, sabemos que $i =$ `len(texto)`. Como $\mathcal{I}$ también dice que `letra` no aparece en `texto` antes de `i`, podemos concluir que `letra` no aparece en `texto` (en ninguna parte). Entonces, en este caso la función debería devolver `False`, y eso es exactamente lo devuelto en la línea 13, porque la expresión `i<len(texto)` es `False`.
  - **Supongamos que `i` $<$ `len(texto)`**. Entonces, debe ser cierto `texto[i]` $=$ `letra`; o sea, que encontramos la letra. Por lo tanto, la función debería devolver `True`, y eso es exactamente lo devuelto en la línea 13, porque la expresión `i<len(texto)` es `True`.

  En conclusión, en las dos formas posibles de cortar la ejecución del `while`, la función devuelve lo indicado por la especificación. □