Guía de Ejercicios 4: Ciclos

Objetivos:

- Ejercitar la escritura de ciclos (`while`) como herramienta para repetir la ejecución de segmentos de código.
- Familiarizarse con la demostración de correctitud de ciclos.

Importante: Todavía no se debe usar iteradores (`for`, por ejemplo) ni recursión algorítmica. Esos temas serán presentados más adelante en la materia. Por ahora es fundamental usar solamente los elementos de programación vistos en clase.

Ejercicio 1.

(a) Especificar e implementar una función `tablaF2C` que construya y devuelva un string con una tabla de conversión de grados Fahrenheit a grados Celsius. La tabla debe mostrar solo la parte entera de la conversión, empezar en 0°F y terminar en 120°F, tomando intervalos de 10°F. De esta manera, ejecutar `print(tablaF2C())` debería mostrar por pantalla algo así:

```
0 F = -18 C
10 F = -12 C
...
120 F = 49 C
```

Importante: ¡No reescribir el código que realiza la conversión dentro del ciclo! En cambio, reusar la función `f2c` definida en el Ejercicio 2 de la Guía 2.

(b) Escribir un predicado invariante que describa la evolución de las variables desde el principio hasta el final de la ejecución del ciclo.

(c) Justificar por qué el ciclo efectivamente termina y sirve para que la función haga lo esperado. (Para esto, relacionar el predicado invariante del ciclo y la especificación de la función.)

Ejercicio 2. Sea el problema de determinar si una palabra se trata de un palíndromo (que se lee igual de izquierda a derecha y al revés). Por ejemplo, `'reconocer'` es un palíndromo, pero `'desconocer'` no lo es.

(a) Escribir una **especificación** de una función `es_palindromo`, describiendo sus parámetros, sus requerimientos y el valor de retorno.

(b) Armar un conjunto adecuado de **casos de test** para la función `es_palindromo`.

(c) **Implementar** la función `es_palindromo`, respetando la especificación planteada, y usando este algoritmo: comparar la primera letra con la última, luego la segunda con la penúltima, etc. Si no se encuentran diferencias, devolver verdadero; en caso contrario, devolver falso. (Sugerencia: utilizar la función `len(x)` para obtener la cantidad de letras de un string `x`, y el operador `x[i]` para acceder a su `i`-ésimo carácter.)

(d) Escribir un **predicado invariante** que describa la evolución de las variables desde el principio hasta el final de la ejecución del ciclo.

(e) Justificar por qué el ciclo efectivamente termina y sirve para que la función haga lo esperado. (Para esto, relacionar el predicado invariante del ciclo con la especificación de la función.)

(f) **Verificar** que la función se ejecute correctamente para los casos de test elegidos.

Ejercicio 3. Un número $n \in \mathbb{N}$ es `primo` si y solo si tiene solo dos divisores positivos distintos: 1 y n. Sea el problema de determinar si un número entero positivo es primo o no.

(a) Escribir una **especificación** de una función `es_primo`, describiendo sus parámetros, sus requerimientos y el valor de retorno.

(b) Armar un conjunto adecuado de **casos de test** para la función `es_primo`.

(c) **Implementar** la función `es_primo`, respetando la especificación planteada.

(d) Escribir un **predicado invariante** que describa la evolución de las variables desde el principio hasta el final de la ejecución del ciclo.

(e) Justificar por qué el ciclo efectivamente termina y sirve para que la función haga lo esperado. (Para esto, relacionar el predicado invariante del ciclo con la especificación de la función.)

(f) **Verificar** que la función se ejecute correctamente para los casos de test elegidos.

Ejercicio 4. Para cada una de las funciones especificadas en el Ejercicio 3 de la Guía 2:

(a) Implementar la función, respetando la especificación planteada.

(b) Si hay ciclos involucrados, escribir un predicado invariante que describa la evolución de las variables desde el principio hasta el final de la ejecución del ciclo.

(c) Justificar por qué el ciclo efectivamente termina y sirve para que la función haga lo esperado. (Para esto, relacionar el predicado invariante del ciclo con la especificación de la función.)

(d) Verificar que la función se ejecute correctamente para los casos de test elegidos.

Ejercicio 5. La evaluación de cortocircuito es muy útil para escribir código más simple y claro.

```
1  def tiene_x(s:str) -> bool:
2      '''
3      Requiere: nada.
4      Devuelve: True si s tiene alguna 'x'; False si no.
5      '''
6      i:int = 0
7      while i<len(s) and s[i]!='x':
8          i = i + 1
9      return not(i==len(s))
```

Hacer un seguimiento manual de la ejecución de `tiene_x('abc')`. Prestar atención a la última vez que se evalúa la condición del ciclo. ¿Qué ocurriría si la condición del ciclo fuera `s[i]!='x' and i<len(s)`?

Ejercicio 6. ¿Cuándo y por qué fallan las siguientes funciones? ¿Cómo se podrían corregir?

(a)

```
def cant_asteriscos_al_comienzo(texto:str) -> int:
    '''
    Requiere: nada.
    Devuelve: la cantidad de asteriscos consecutivos que
              hay al comienzo de texto.
    '''
    i:int = 0
    while texto[i]=='*' and i<len(texto):
        i = i + 1
    return i
```

(b)

```
def string_ascendente(texto:str) -> bool:
    '''
    Requiere: len(texto) >= 2 y texto está formado solo por los
              caracteres abcdefghijklmnñopqrstuvwxyz.
    Devuelve: True si los caracteres de texto se encuentran
              ordenados alfabéticamente; False en caso contrario.
    '''
    i:int = 0
    while texto[i]<texto[i+1] and i<len(texto)-1:
        i = i + 1
    rv:bool = (i == len(texto)-1)
    return rv
```

Ejercicio 7. Para las siguientes funciones, demostrar que el ciclo efectivamente termina para cualquier entrada válida, escribir un predicado invariante del ciclo y demostrar que la función hace lo esperado.

(a)

```
def vocales_a_mayúscula(s:str) -> str:
    '''
    Requiere: s tiene solamente letras en minúscula: abcde...xyz,
              sin tildes, diéresis ni eñes.
    Devuelve: un string que solo difiere con s en que las vocales
              aparecen en mayúscula.
    '''
    vr:str = ''
    i:int = 0
    while i<len(s):
        if s[i] in 'aeiou':
            vr = vr + s[i].upper()
        else:
            vr = vr + s[i]
        i = i + 1
    return vr
```

(b)

```python
def sumar_primos_hasta(n:int) -> int:
    '''
    Requiere: n>0
    Devuelve: la suma de los números primos entre 0 y n (inclusive)
    '''
    vr:int = 0
    i:int = 2
    while i <= n:
        if es_primo(i):
            vr = vr + i
        i = i + 1
    return vr
```

(c)

```python
def es_prefijo(s:str, t:str) -> bool:
    '''
    Requiere: len(s)<=len(t)
    Devuelve: True si en toda posición j entre 0 y len(s)-1,
              s[j]==t[j]; False en caso contrario.
    '''
    aux:bool = True
    i:int = 0
    while i<len(s) and i<len(t):
        aux = aux and s[i]==t[i]
        i = i + 1
    vr:bool = aux and i==len(s)
    return vr
```

Ejercicio 8. (Opcional) En 2006, Microsoft lanzó el reproductor portátil de música Zune para competir contra el exitoso iPod de Apple. El 1 de enero de 2009, todos los dispositivos Zune del mundo dejaron de funcionar, debido a un error de software en la función `CalculateCurrentYear` que se muestra a continuación (adaptada a Python). Encontrar el error y corregirlo.

```python
ORIGINYEAR:int = 1980

def IsLeapYear(y:int) -> bool:
    ''' Requiere: y>0
        Devuelve: True si el año y es bisiesto; False si no. '''
    return (y%4==0) and (y%100>0 or y%400==0)

def CalculateCurrentYear(days:int) -> int:
    ''' Requiere: days >= 0, la cantidad de días transcurridos
                  desde el 1 de enero de ORIGINYEAR.
        Devuelve: El año actual (ej: 2007). '''
    year:int = ORIGINYEAR
    while days > 365:
        if IsLeapYear(year):
            if days > 366:
                days = days - 366
                year = year + 1
        else:
            days = days - 365
            year = year + 1
    return year
```

Sugerencia: es difícil *emparchar* este código; es más fácil repensarlo por completo.

Ejercicio 9. Los números naturales pueden escribirse de muchas formas. Por ejemplo, el sistema romano de numeración usa los símbolos I, V, X, L, C, D y M, que se combinan según ciertas reglas. Así, XIV representa al 14 y MMXXII al 2022. Los números romanos continúan vigentes para nombrar los siglos, para designar olimpíadas y congresos, en la numeración de reyes y papas, y en algunos relojes, entre otros usos.
Hoy en todo el mundo se usa cotidianamente el sistema *decimal* (en base 10), que emplea diez símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. Esos diez símbolos significan cosas distintas según su posición; por eso decimos que es un sistema de numeración *posicional*. Por ejemplo, el número 137 en base 10 significa:

$$\text{1 centena + 3 decenas + 7 unidades.}$$

En el sistema decimal, una unidad es 10^0 (la base 10 elevada a la potencia 0); una decena es 10^1 (la base 10 elevada a la 1); una centena es 10^2 (la base 10 elevada a la 2); y así. Entonces, 137 es igual a:

$$1 \times 10^2 + 3 \times 10^1 + 7 \times 10^0.$$

Existen otros sistemas numéricos posicionales que usan otras cantidades de símbolos. El sistema *binario* (en base 2) tiene una importancia central en la computación, dado que es la forma natural de almacenar y operar con información en los sistemas digitales. Usa solo dos símbolos: 0 y 1. Por ejemplo, el número 137 (en decimal) se representa como `10001001` en notación binaria, porque:

$$\begin{array}{llllllll} 1 \times 2^7 & + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = \\ 1 \times 128 + 0 \times 64 + 0 \times 32 + 0 \times 16 + 1 \times 8 & + 0 \times 4 & + 0 \times 2 & + 1 \times 1 & = \\ 1 \times 128 & + 1 \times 8 & & + 1 \times 1 & = 137. \end{array}$$

Otro sistema importante en computación es el *hexadecimal* (en base 16), que usa 16 símbolos: los diez dígitos del sistema decimal y las letras A, B, C, D, E y F, las cuales representan los valores 10, 11, 12, 13, 14 y 15, respectivamente. Por ejemplo, el número 137 (en decimal) se escribe `89` en hexadecimal, porque:

$$\begin{array}{ll} 8 \times 16^1 + 9 \times 16^0 = \\ 8 \times 16 \ + 9 \times 1 \quad = 137. \end{array}$$

El sistema hexadecimal se emplea para muchas cosas, entre ellas para representar colores en HTML (`"#94AAD3"`) y direcciones IPv6 (`"FE80::91AE:819A:F02A:D280"`).

El siguiente algoritmo permite obtener la representación de un entero n en base b:

> *resultado* $\leftarrow$ string vacío
> Repetir mientras $n > 0$:
> Dividir al número n por la base b. Esto arroja un cociente c y un resto r, de forma tal que $n = c \times b + r$.
> Agregar el símbolo correspondiente a r al principio del string *resultado*.
> $n \leftarrow c$
> Retornar el string almacenado en *resultado*.

(a) Convertir manualmente los siguientes números según se indica:

(I) 1234 (decimal) a binario (base 2).

(II) 8991 (decimal) a hexadecimal (base 16).

(III) AFA (hexadecimal) a decimal.

(IV) 111001 (binario) a decimal.

(V) DF (hexadecimal) a binario.

(VI) 10101010 (binario) a hexadecimal.

(b) Especificar, programar y verificar funciones para resolver los siguientes problemas:

(I) Dados dos enteros n y b ($n \geq 0$; $2 \leq b \leq 16$), devolver un string con la representación de n en base b. Ejemplos: para $n = 26$ y $b = 2$ debe devolver `"11010"` (notación binaria); para $n = 26$ y $b = 10$ debe devolver `"26"` (notación decimal); para $n = 26$ y $b = 16$ debe devolver `"1A"` (notación hexadecimal).

(II) Dados un entero b ($2 \leq b \leq 16$) y un string con una representación en base b, calcular el valor del número representado. Ejemplos: para $b = 2$ y `"11010"` debe devolver 26; para $b = 10$ y `"26"` debe devolver `"26"`; para $b = 16$ y `"1A"` debe devolver 26.