Guía de Ejercicios 4: Ciclos

Objetivos:

- Ejercitar la escritura de ciclos (`while`) como herramienta para repetir la ejecución de segmentos de código.
- Familiarizarse con el efecto de los ciclos en el estado del programa.

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. Sea el problema de, dado un string, calcular la cantidad de veces que contiene la letra `'a'`. Por ejemplo, `'manzana'` tiene 3, `'azar'` tiene 2 y `'linux'` tiene 0. Considerar la siguiente función para resolverlo:

```
1  def cant_a(s:str) -> int:
2      ''' Requiere: Nada
3          Devuelve: La cantidad de ocurrencias de 'a' en s.
4      '''
5      cant:int = 0
6      i:int = 0
7      while i < len(s):
8          if s[i]=='a':
9              cant = cant + 1
10         i = i + 1
11     return cant
```

(a) Demostrar que el ciclo termina para cualquier entrada válida.

(b) Hacer un seguimiento completo de la ejecución de `cant_a('alaska')`:

Luego de 0 iteraciones: `cant` vale *0*; `i` vale *0*; `i<len(s)` es *True*.
Luego de 1 iteración: `cant` vale ____; `i` vale ____; `i<len(s)` es _____.
Luego de 2 iteraciones: `cant` vale ____; `i` vale ____; `i<len(s)` es _____.
Luego de 3 iteraciones: `cant` vale ____; `i` vale ____; `i<len(s)` es _____.
Luego de 4 iteraciones: `cant` vale ____; `i` vale ____; `i<len(s)` es _____.
Luego de 5 iteraciones: `cant` vale ____; `i` vale ____; `i<len(s)` es _____.
Luego de 6 iteraciones: `cant` vale ____; `i` vale ____; `i<len(s)` es _____.

(c) Completar el siguiente predicado invariante, de manera que describa el trabajo realizado por el ciclo para cualquier valor de `s`:

Inv: `cant` vale *la cantidad de ocurrencias de 'a' en ...*.

(d) Entonces, ¿qué valor tendrá `cant` luego de `len(s)` iteraciones? Comparar con la cláusula `Devuelve` de la especificación de la función. (Deberían expresar lo mismo.)

Ejercicio 2. Sea el problema de, dado un string, devolver un nuevo string, resultante de pasar a mayúsculas todas las ocurrencias de la letra `'n'`. Por ejemplo, para `'mandarina'` el resultado sería `'maNdariNa'`. Considerar la siguiente función para resolverlo:

```
1  def mayus_n(s:str) -> str:
2      ''' Requiere: Nada
3          Devuelve: Una copia de s pero con todas las 'n' en mayúscula.
4      '''
5      res:str = ''
6      j:int = 0
7      while j < len(s):
8          if s[j]=='n':
9              res = res + s[j].upper()
10         else:
11             res = res + s[j]
12         j = j + 1
13     return res
```

(a) Demostrar que el ciclo termina para cualquier entrada válida.

(b) Hacer un seguimiento completo de la ejecución de `mayus_n('antena')`:

Luego de 0 iteraciones: `res` vale _____; `j` vale ____; `j<len(s)` es _____.
Luego de 1 iteración: `res` vale _____; `j` vale ____; `j<len(s)` es _____.
Luego de 2 iteraciones: `res` vale _____; `j` vale ____; `j<len(s)` es _____.
Luego de 3 iteraciones: `res` vale _____; `j` vale ____; `j<len(s)` es _____.
Luego de 4 iteraciones: `res` vale _____; `j` vale ____; `j<len(s)` es _____.
Luego de 5 iteraciones: `res` vale _____; `j` vale ____; `j<len(s)` es _____.
Luego de 6 iteraciones: `res` vale _____; `j` vale ____; `j<len(s)` es _____.

(c) Completar el siguiente predicado invariante, de manera que describa el trabajo realizado por el ciclo para cualquier valor de `s`:

Inv: `res` vale _____.

(d) Entonces, ¿qué valor tendrá `res` luego de `len(s)` iteraciones? Comparar con la cláusula `Devuelve` de la especificación de la función. (Deberían expresar lo mismo.)

Ejercicio 3. Para cada una de las siguientes funciones, se pide:

i. Demostrar que el ciclo termina para cualquier entrada válida.
ii. Hacer un seguimiento completo de la ejecución de `enumerar(6)`.
iii. Escribir un predicado invariante que describa el trabajo realizado por el ciclo para cualquier valor de `n`.
iv. Usar el predicado invariante para mostrar que la función cumple con la especificación.

(a)

```
1  def enumerar(n:int) -> str:
2      ''' Requiere: n>0
3          Devuelve: los números de 1 a n, separados por comas: "1,2,...,n".
4      '''
5      res:str = '1'
6      i:int = 2
7      while i<=n:
8          res = res + ',' + str(i)
9          i = i + 1
10     return res
```

(b)

```
def enumerar(n:int) -> str:
    ''' Requiere: n>0
        Devuelve: los números de 1 a n, separados por comas: "1,2,...,n".
    '''
    res:str = str(n)
    i:int = n-1
    while i>0:
        res = str(i) + ',' + res
        i = i - 1
    return res
```

(c)

```
def enumerar(n:int) -> str:
    ''' Requiere: n>0
        Devuelve: los impares de 1 a n, separados por comas: "1,3,5,...,n".
    '''
    res:str = '1'
    i:int = 3
    while i<=n:
        res = res + ',' + str(i)
        i = i + 2
    return res
```

Ejercicio 4. Sea el problema de, dado un entero n ≥ 0, calcular $\lfloor \sqrt{n} \rfloor$ (es decir, la parte entera de $\sqrt{n}$). Considerar la siguiente función para resolverlo:

```
def raiz_entera(n:int) -> int:
    ''' Requiere: n>=0
        Devuelve: La parte entera de la raíz cuadrada de n.
    '''
    i:int = 0
    while i*i <= n:
        i = i + 1
    return i-1
```

(a) Demostrar que el ciclo termina para cualquier entrada válida.

(b) Hacer un seguimiento completo de la ejecución de `raiz_entera(10)`.

(c) Escribir un predicado invariante que describa el trabajo realizado por el ciclo.

(d) Usar el predicado invariante para mostrar que la función cumple con la especificación.

Ejercicio 5. Para cada una de las funciones especificadas en el Ejercicio 4 de la Guía 2:

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

(b) Si hay ciclos involucrados, demostrar que terminan, proponer predicados invariantes y usarlos para mostrar que las funciones hacen lo esperado.

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

Ejercicio 6. Considerar la siguiente función para averiguar si un string es prefijo de otro:

```
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
    return aux and (i==len(s))
```

(a) Demostrar que el ciclo termina para cualquier entrada válida.

(b) Hacer un seguimiento completo de la ejecución de `es_prefijo('abc', 'abcd')`, y de `es_prefijo('azc', 'abcd')`.

(c) Escribir un predicado invariante que describa el trabajo realizado por el ciclo.

(d) Usar el predicado invariante para mostrar que la función cumple con la especificación.

Ejercicio 7. 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) Demostrar que el ciclo termina, proponer un predicado invariante y usarlo para mostrar que la función hace lo esperado.

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

Ejercicio 8. 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) Demostrar que el ciclo termina, proponer un predicado invariante y usarlo para mostrar que la función hace lo esperado.

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

Ejercicio 9. A continuación se muestran especificaciones de funciones y posibles implementaciones que están incompletas. En cada caso, falta terminar de escribir un ciclo, para el cual se muestra un predicado invariante que describe su comportamiento. Se pide terminar de escribir ese ciclo, de modo que la función haga 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: Una copia de s pero con las vocales en mayúscula.
    '''
    res:str = ''
    i:int = 0
    while ______:
        # completar...
    return res
```

Inv: `res` es una copia de los primeros i caracteres de `s`, pero con las vocales en mayúscula.

Observación: en Python, un string `x` se puede pasar a mayúsculas con `x.upper()`

(b)

```
def sumar_primos_hasta(n:int) -> int:
    ''' Requiere: n>0
        Devuelve: La suma de los números primos desde 0 hasta n
                  (inclusive).
    '''
    res:int = 0
    i:int = 2
    while ______:
        # completar...
    return res
```

Inv: `res` vale la suma de los números primos desde 0 hasta $i - 1$ (inclusive).

Ejercicio 10. La **evaluación de cortocircuito** es muy útil para escribir código más simple y claro.

```
def tiene_x(s:str) -> bool:
    ''' Requiere: nada.
        Devuelve: True si s tiene alguna 'x'; False si no.
    '''
    i:int = 0
    while i<len(s) and s[i]!='x':
        i = i + 1
    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 11. Evaluación de cortocircuito. ¿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 12. Sistemas de numeración[1]

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:

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:

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

El siguiente algoritmo permite obtener la representación binaria de un número entero n:

entero_a_binario($n \in \mathbb{Z}$) $\rightarrow$ string:
    resultado $\leftarrow$ string vacío
    Repetir mientras $n > 0$:
        Dividir a n por 2.
        Esto arroja un cociente c y un resto r, tales que $n = 2 \times c + r$.
        Agregar r (0 o 1) al principio del string *resultado*.
        $n \leftarrow c$
    Retornar el string almacenado en *resultado*.

(a) Usando el algoritmo anterior, obtener a mano la representación binaria de los siguientes números decimales: 1, 2, 15, 16, 127, 128.

(b) Escribir en pseudocódigo el algoritmo de una función `binario_a_decimal`, que reciba un string representando a un número binario (solo 0s y 1s) y devuelva el número entero correspondiente en base 10. Por ejemplo, para `'1100'` debe devolver 12.

(c) Usando el algoritmo del punto anterior, obtener a mano la representación decimal de los siguientes números binarios: 1, 10, 11, 100, 11111, 100000.

(d) Implementar en Python y verificar las funciones `entero_a_binario` y `binario_a_decimal`.

[1] Este tema es muy útil para estudiar la representación de los distintos tipos de datos en la computadora, lo cual se estudia en la materia "TD2 Sistemas de Computación".

Ejercicio 13. *(Ejercicio opcional, para ilustrar la importancia de escribir ciclos entendibles.)*
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 `calculate_current_year` que se muestra a continuación (adaptada a Python). Encontrar el error y corregirlo.

```
ORIGIN_YEAR:int = 1980

def is_leap_year(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 calculate_current_year(days:int) -> int:
    ''' Requiere: days >= 0, la cantidad de días transcurridos
                  desde el 1 de enero de ORIGIN_YEAR.
        Devuelve: El año actual (ej: 2007).
    '''
    year:int = ORIGIN_YEAR
    while days > 365:
        if is_leap_year(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.