Guía de Ejercicios 6: Listas

Objetivos:

- Ejercitar el uso del tipo de datos lista y sus operaciones básicas, disponibles en la mayoría de los lenguajes de programación.
- Comprender la asignación y el pasaje de argumentos de tipos mutables e inmutables.
- Incorporar el uso prudente de iteradores, entendiendo sus ventajas y desventajas respecto de la instrucción `while`.

Importante: Usar solamente las operaciones de listas vistas en clase, y a medida que cada enunciado lo indique. Python ofrece *muchísimas* operaciones para el procesamiento de listas: `max`, `sum`, `join`, `split`, `map`, `insert`, etc. Por ahora es importante ir despacio, y ganar práctica en la manipulación de listas usando solo operaciones básicas. Una vez ganadas estas intuiciones, después todo lo demás se aprende rápido.

Ejercicio 1. Hacer un seguimiento a mano, instrucción a instrucción, de la variable `lista` en el siguiente programa, y determinar qué valor tiene al terminar.

```
1
2  lista:list[str] = ['w', 'l', 'g', 'x', 'r', 't', 't', 'm', 'u']
3  lista.pop()
4  lista.append('x')
5  if 'w' in lista:
6      lista[0] = 'a'
7  elif 'g' in lista:
8      lista[0] = 'b'
9  else:
10     lista[0] = 'c'
11 i:int=0
12 while i<len(lista):
13     if lista[i]=='x':
14         lista[i] = 'o'
15     i = i + 1
16 lista[5] = 'i'
```

Ejercicio 2. Escribir el encabezado y el docstring de cada una de las siguientes funciones, describiendo qué hace y especificando lo que requiere y lo que devuelve.

(a)

```
1  def _________(n:int) -> _________:
2      '''
3      ...COMPLETAR...
4      '''
5      r:list[str] = []
6      while n > 0:
7          r.append(str(n))
8          n = n - 1
9      r.append('¡Despegue!')
10     return r
```

(b)

```
def ________(a:________) -> ________:
    '''
    ...COMPLETAR...
    '''
    r:int = 0
    i:int = 0
    while i < len(a):
        r = r + round(a[i])
        i = i + 1
    return r
```

(c)

```
def ________(a:________) -> ________:
    '''
    ...COMPLETAR...
    '''
    r:list[int] = []
    i:int = len(a)-1
    while i >= 0:
        r.append(a[i])
        i = i - 1
    return r
```

Ejercicio 3. Especificar, programar y verificar funciones para resolver los siguientes problemas. En los ciclos usar `while`; no usar iteradores (`for`) todavía. Para la verificación, se pide hacer testing de unidad, demostrar que los ciclos terminan, escribir predicados invariantes y usarlos para mostrar que el código hace lo pedido.

(a) Dada una lista no vacía de `float`, encontrar su máximo elemento. Por ejemplo, para la lista `[1.0, 12.7, 1.0, 8.8, 12.7, 3.0]`, debería devolverse `12.7`. No usar `max`.

(b) Dada una lista de strings, construir un string que sea la concatenación de todos sus elementos. Por ejemplo, para la lista `['ab','c','','def']`, se debe devolver `'abcdef'`.

(c) Dada una lista de enteros, determinar si está ordenada en forma estrictamente creciente. Por ejemplo, `[1,4,6]` y `[]` están ordenadas, pero `[6,4,1]` y `[4,6,6]` no lo están. Notar que una lista vacía se considera ordenada.

(d) Dado un número natural `n`, construir una lista con los primeros `n` números naturales impares, ordenados de menor a mayor. Por ejemplo, para `n=3`, la lista esperada es `[1,3,5]`.

(e) Dada una lista de enteros `ls` y un entero `n`, construir una nueva lista, resultante de sumarle `n` a cada elemento de `ls`. Por ejemplo, para `ls=[1,9,7,7]` y `n=10`, se debe devolver la lista `[11,19,17,17]`.

(f) Dada una lista de strings, contar la cantidad total de caracteres. Por ejemplo, para `['tor', 'c','ua','to']` el resultado esperado es 8.

(g) Dada una lista de strings, contar la cantidad de veces que aparece la letra `'a'`. Por ejemplo, para `['abba','acdc','bee gees','a-ha']` el resultado esperado es 5.

(h) Dado un string `txt` de longitud arbitraria y un string `sep` de longitud 1, separar `txt` en cada aparición de `sep`, construyendo así una nueva lista. Por ejemplo, para `txt='a;bb;c;;ddd;'` y `sep=';'`, la lista resultante es `['a','bb','c','','ddd','']`. No usar `split`.

Ejercicio 4. En una lista no vacía de números enteros, llamamos *mesetas* a las sublistas de números iguales que aparecen en forma consecutiva. Por ejemplo, la lista `[1,1,2,6,6,6,3,3]` contiene las mesetas `[1,1]`, `[2]`, `[6,6,6]` y `[3,3]`. Especificar, programar y verificar funciones para resolver los siguientes problemas, dada una lista de enteros:

(a) Determinar el número de la meseta más larga. Para la lista del ejemplo, la función debe devolver 6. Si hay más de una meseta de longitud máxima, devolver el número de la primera.

(b) Construir una lista de listas, conteniendo las mesetas de la lista original, en el mismo orden. Para la lista del ejemplo, la lista resultante es `[[1,1],[2],[6,6,6],[3,3]]`.

(c) Construir una lista de enteros, conteniendo la longitud de cada meseta de la lista original, en el mismo orden. Para la lista del ejemplo, se debe devolver `[2,1,3,2]`.

Slicing e iteradores en strings y listas

Ejercicio 5. Evaluación de expresiones.

(a) Dada `xs:list[int] = [10,11,12,13,14,15,16,17,18,19]`, determinar **a mano** el valor de las siguientes expresiones. Después revisar las respuestas en Python.

(I) `xs[3:7]`	(V) `xs[3:-2]`	(IX) `xs[::-1]`
(II) `xs[3:]`	(VI) `xs[-7:7]`	(X) `xs[1:8:-2]`
(III) `xs[:3]`	(VII) `xs[-5:]`	(XI) `xs[8:1:-2]`
(IV) `xs[:]`	(VIII) `xs[:-5]`	(XII) `xs[::]`

(b) Evaluar las expresiones del punto anterior, suponiendo ahora que `xs:str = 'abcdefghij'`.

Ejercicio 6. Reescribir los ciclos del Ejercicio 3, ahora usando `for` en vez de `while` para iterar listas y strings.

Ejercicio 7. `while` vs. `for`: Considerar el siguiente programa:

```
1  def buscar_v1(elem:int, ls:list[int]) -> bool:
2      '''
3      Requiere: nada.
4      Devuelve: True sii elem aparece al menos una vez en ls.
5      '''
6      r:bool = False
7      i:int = 0
8      while i < len(ls):
9          r = r or (ls[i] == elem)
10         i = i + 1
11     return r
```

```
12  def buscar_v2(elem:int, ls:list[int]) -> bool:
13      '''
14      Requiere: nada.
15      Devuelve: True sii elem aparece al menos una vez en ls.
16      '''
17      r:bool = False
18      i:int = 0
19      while i < len(ls) and not r:
20          r = ls[i] == elem
21          i = i + 1
22      return r
23
24  a:list[int] = [1,2,4,1,5]
25  print(buscar_v1(1, a), buscar_v2(1, a))
26  print(buscar_v1(3, a), buscar_v2(3, a))
27  print(buscar_v1(5, a), buscar_v2(5, a))
```

(a) Hacer un seguimiento a mano de la ejecución de las líneas 25, 26 y 27, y observar cuándo termina la ejecución del ciclo en cada caso.

(b) Reescribir las funciones `buscar_v1` y `buscar_v2` usando `for` en lugar de `while`. En una de ellas, para terminar la ejecución del ciclo en el mismo momento, será necesario usar `break` o `return` en el cuerpo del ciclo.

Ejercicio 8. `while` vs. `for`: Reescribir la función del Ejercicio 3(c), de modo que la ejecución del ciclo corte en cuanto se sepa que la lista no está ordenada. Escribir dos versiones: una con `while` y otra con `for`.

Ejercicio 9. `while` vs. `for`:
Considerar el siguiente programa:

```
1   def primera_ocurrencia(elem:int, ls:list[int]) -> int:
2       '''
3       Requiere: ls contiene al menos una ocurrencia de elem.
4       Devuelve: el índice de la primera ocurrencia de elem en ls.
5       '''
6       i:int = 0
7       while ls[i] != elem:
8           i = i + 1
9       return i
10
11  a:list[int] = [1,2,4,1,5]
12  print(primera_ocurrencia(1, a))
13  print(primera_ocurrencia(5, a))
```

(a) Reescribirlo usando `for` en lugar de `while`.

(b) (Opcional) Reescribirlo usando `for (i,x) in enumerate(a)`. Buscar y leer la documentación de `enumerate`.

Ejercicio 10. Para las siguientes funciones, demostrar que el ciclo efectivamente termina para cualquier entrada válida, escribir un predicado invariante del ciclo y usarlo para mostrar que la función hace lo esperado.

(a)

```
def iniciales(ls:list[str]) -> list[str]:
    '''
    Requiere: Todos los strings de ls tienen longitud>0.
    Devuelve: Una lista de longitud len(ls) tal que la posición j,
              para todo j entre 0 y len(ls)-1, tiene el primer
              carácter del string ls[j].
    '''
    r:list[str] = []
    i:int = 0
    while i < len(ls):
        primer_carácter:str = ls[i][0]
        r.append(primer_carácter)
        i = i + 1
    return r
```

(b)

```
def iniciales2(ls:list[str]):
    '''
    Requiere: Todos los strings de ls tienen longitud>0.
              Llamamos LS al valor inicial de ls.
    Devuelve: Nada.
    Modifica: En ls[j], para todo j entre 0 y len(ls)-1,
              queda el primer carácter del string LS[j].
    '''
    i:int = 0
    while i < len(ls):
        primer_carácter:str = ls[i][0]
        ls[i] = primer_carácter
        i = i + 1
```

Tipos mutables e inmutables

Ejercicio 11. Considerar el siguiente código:

```
1 a:list[int] = [1,2]
2 b:list[int] = a
3 a.append(3)
4 print(len(b))
```

(a) ¿Qué valor se imprime en la última instrucción? ¿Por qué no se imprime 2, si luego de ejecutar la línea 2, b tenía 2 elementos y luego no la modificamos?

(b) ¿Qué se imprime si reemplazamos la línea 2 por `b:list[int] = list(a)`?

(c) Comparar con el siguiente código, visto en el Ejercicio 6 de la Guía 1:

```
1  a:int = 1
2  b:int = a
3  a = 2
4  print(b)
```

Es importante entender por qué el comportamiento es distinto para enteros y para listas.

Ejercicio 12. Sin ejecutarlo en Python, determinar qué imprime por pantalla el siguiente código. ¿En qué difiere la ejecución de las líneas 5 y 7?

```
1  def despedir(ls:list[str]):
2      ls.append('chau')
3
4  a:list[str] = ['hola', 'mundo']
5  despedir(a)
6  print(a)
7  despedir(list(a))
8  print(a)
```

Ejercicio 13. Sin ejecutarlo en Python, determinar qué imprime por pantalla el siguiente código.

```
1  def suma_problematica(ls:list[int]) -> int:
2      n:int = 0
3      i:int = 0
4      while i < len(ls):
5          n = n + ls[i]
6          i = i + 1
7      ls.clear()   # clear() elimina todos los elementos de la lista.
8      return n
9
10 a:list[int] = [1,2,3,4]
11 print(suma_problematica(a))
12 print(suma_problematica(a))
```

¿Qué cambia si reemplazamos la línea 7 por `ls = []`? Es importante entender la diferencia entre esas dos instrucciones.