Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 10: Listas

Temas de la materia (en orden más o menos cronológico)

1. Problema, especificación, algoritmo y programa. Lenguaje Python. ✓
2. Tipos de datos, expresiones, variables, memoria, estado del programa. ✓
3. Funciones, especificación de funciones. Concepto de encapsulamiento. ✓
4. Tipos de errores de los programas, testing de unidad. ✓
5. Control de flujo: condicionales (`if`) y ciclos (`while`).✓
6. Listas: secuencias de elementos. Tipos de datos compuestos.
7. Lectura/escritura de archivos.
8. Representación de la información.

---

9. Complejidad algorítmica, tiempo de ejecución de un programa.
10. Problemas clásicos: ordenamiento y búsqueda.
11. Más tipos de datos compuestos: tuplas, conjuntos y diccionarios.
12. Definición de tipos propios: clases y objetos.
13. Recursión algorítmica.

Listas (`List[T]`)

Hasta ahora vimos tipos: `bool`, `int`, `float`, `str`.

Las listas son secuencias de valores de algún tipo `T`. Al igual que `str`, son un tipo compuesto, porque están formados por piezas menores.

```
[45, 657, 11, 0, 45, -303, 0, -11, 45]      es una lista de int
[3.14159, 0.0]                              es una lista de float
['Marte', 'Venus', 'Júpiter']               es una lista de str
[False, False, True, False]                 es una lista de bool
```

En Python, para usar listas en las sugerencias de tipos (*type hints*), debemos importar `List` de una biblioteca llamada `typing`:

```
from typing import List
números:List[int] = [1, 2, 3]
planetas:List[str] = ['Marte', 'Venus', 'Júpiter']
```

En todos los ejemplos que siguen, vamos a suponer que ya hicimos la importación `from typing import List` al comienzo.

Listas (`List[T]`) · Operaciones básicas

```python
# Creamos una lista con varios elementos.
mundiales:List[str] = ['Corea-Japón', 'Alemania',
                       'Sudáfrica', 'Brasil', 'Rusai']

# Leemos una posición de la lista.
print(mundiales[4])

# Sobreescribimos una posición.
mundiales[4] = 'Rusia'

# Agregamos un elemento al final.
mundiales.append('Qatar')

# Imprimimos toda la lista.
print(mundiales)
```

Dos aclaraciones importantes sobre listas en Python

1. Python permite listas heterogéneas, con elementos de distintos tipos:

```
lista_loca = [2021, False, 3.141592, 'UTDT']
print(type(lista_loca[0]))      # <class 'int'>
print(type(lista_loca[1]))      # <class 'bool'>
print(type(lista_loca[2]))      # <class 'float'>
print(type(lista_loca[3]))      # <class 'str'>
```

   Desaconsejamos **fuertemente** esta práctica, que suele llevar a errores... de todo tipo. ;-)

   Usamos siempre listas homogéneas y especificamos el tipo de sus elementos con *type hints*: `List[int]`, `List[str]`, etc.

2. Python ofrece muchísimas herramientas para manipular listas: iteradores, sublistas (*slices*), listas por comprensión, etc.

   Pero vayamos despacio: por ahora usamos solo operaciones básicas (las vistas en esta clase), y de a poco iremos viendo el resto.

Listas (`List[T]`) · Operaciones básicas

Dadas las variables `a,b` de tipo `List[T]`; `x,y,z` de tipo `T`; `i` de tipo `int`:

- `a = list()` Crea una lista de `T` vacía.
- `a = []` Crea una lista de `T` vacía.
- `a = [x,y,z]` Crea una lista de `T` no vacía.
- `len(a)` Devuelve la longitud de la lista.
- `x in a` Consulta pertenencia de un elemento en la lista.
- `a.append(x)` Agrega un elemento al final de la lista.
- `a.pop()` Elimina el elemento de la última posición y lo devuelve.
- `a[i]` Lee el valor en la `i`-ésima posición.
- `a[i] = x` Sobreescribe la `i`-ésima posición.
- `a.insert(i,x)` Inserta un elemento en la `i`-ésima posición.
- `a.pop(i)` Elimina la `i`-ésima posición y devuelve su valor.
- `a == b` Compara dos listas.
- `a + b` Concatena dos listas.
- `a * i` Concatena `i` veces seguidas la lista `a`.

Listas (`List[T]`) · Operaciones básicas

Ejemplos:

```
1  a:List[int] = [1977, 1980, 1983]
2  len(a)              # Devuelve la longitud de la lista: 3
3  a.append(2000)      # Agrega 2000 al final.
4  a.pop()             # Elimina el último elemento y lo devuelve: 2000
5
6  a[2]                # Devuelve el valor en la posición 2: 1983
7  a[2] = 1999         # Sobreescribe la posición 2.
8  a                   # Devuelve: [1977, 1980, 1999]
9
10 a.insert(2, 1983)   # Inserta 1983 en la posición 2.
11 a                   # Devuelve: [1977, 1980, 1983, 1999]
12 a.pop(3)            # Elimina la posición 3 y devuelve su valor: 1999
13 a                   # Devuelve: [1977, 1980, 1983]
14
15 1983 in a           # Consulta pertenencia: devuelve True
16
17 b:List[int] = [1999, 2002, 2005]
18 a == b     # Compara las listas: devuelve False
19 a + b      # Concatena a con b:   [1977,1980,1983,1999,2002,2005]
20 a * 2      # Concatena a 2 veces: [1977,1980,1983,1977,1980,1983]
```

¡No se dejen marear con temas de sintaxis! Lo importante es entender qué hace cada operación.

Listas (`List[T]`) · Ejercicios

Considerar el siguiente código y pensar las respuestas:

```
1  from typing import List
2
3  letras:List[str] = ['p', 'w', 'r']
4  letras.append('x')                    # append: agregar al final
5  longitud:int = len(letras)            # len: longitud
6  letras[longitud - 1] = 'y'            # []: leer o escribir
7  print(letras)
8
9  letras.pop(0)                         # pop: eliminar
10 letras.pop(0)                         # pop: eliminar
11 muchas_letras:List[str] = ['a', 'b'] + letras * 2
12                                       # +: concatenar
13 print(muchas_letras)
```

1. ¿Cuál es el tipo de las variables `letras` y `muchas_letras`?
2. ¿Qué imprime la línea 7?
3. ¿Qué imprime la línea 13?

Listas (`List[T]`) · Ejemplo de ciclos

```
1  from typing import List
2
3  xs:List[int] = [5, 6, 7, 8]
4
5  i:int = 0
6  while i < len(xs):
7      xs[i] = xs[i] + 100
8      i = i + 1
9
10 print(xs)
```

Listas (`List[T]`) · Ejercicios

1. Completar el código, para construir una lista con los *cuadrados* de los primeros 100 enteros positivos: 1, 4, 9, 16, ..., 10000:

```
cuadrados:List[int] = ______
i:int = ______
while i <= 100:
    cuadrados._____________
    i = i + 1
```

2. Completar el código, para imprimir los elementos *impares* de la lista cuadrados:

```
i:int = 0
while i < ______(cuadrados):
    if _______ % 2 == 1:
        print(_________________)
    i = i + 1
```

Listas (`List[T]`) · Ejemplo

```
1  def lista_de_cuadrados(n:int) -> List[int]:
2      '''
3      Requiere: n >= 0
4      Devuelve: una lista formada por los n primeros
5                enteros positivos elevados al cuadrado:
6                [1*1, 2*2, 3*3, ..., n*n].
7      '''
8      vr:List[int] = []
9      i:int = 1
10     while i <= n:
11         vr.append(i * i)
12         i = i + 1
13     return vr
```

Conjunto de ejemplos:

n	**Salida**	Criterio/comentarios
0	`[]`	lista vacía
1	`[1]`	un elemento
3	`[1, 4, 9]`	varios elementos
5	`[1, 4, 9, 16, 25]`	varios elementos

Listas (`List[T]`) · Ejemplo

Conjunto de ejemplos:

n	**Salida**	Criterio/comentarios
0	`[]`	lista vacía
1	`[1]`	un elemento
3	`[1, 4, 9]`	varios elementos
5	`[1, 4, 9, 16, 25]`	varios elementos

```
1  import unittest
2
3  class TestCantVocales(unittest.TestCase):
4
5    def test_vacío(self):
6      self.assertEqual(lista_de_cuadrados(0), [])
7
8    def test_un_elemento(self):
9      self.assertEqual(lista_de_cuadrados(1), [1])
10
11   def test_varios_elementos(self):
12     self.assertEqual(lista_de_cuadrados(3), [1,4,9])
13     self.assertEqual(lista_de_cuadrados(5), [1,4,9,16,25])
```

Listas (`List[T]`) · Ejemplo

```
1  def lista_de_cuadrados(n:int) -> List[int]:
2      '''
3      Requiere: n >= 0
4      Devuelve: una lista formada por los n primeros
5                enteros positivos elevados al cuadrado:
6                [1*1, 2*2, 3*3, ..., n*n].
7      '''
8      vr:List[int] = []
9      i:int = 1
10     while i <= n:
11         vr.append(i * i)
12         i = i + 1
13     return vr
```

Predicado invariante:

- $1 \leq \texttt{i} \leq \texttt{n} + 1$
- `vr` está formada por los primeros `i-1` enteros positivos al cuadrado.

Repaso de la clase de hoy

- Primeros pasos con el tipo lista.
- Operaciones básicas: creación, longitud, inserción, eliminación, lectura/escritura de una posición, pertenencia, comparación.

Bibliografía complementaria:

- APPP2, secciones 8.2 a 8.4, 8.6.
- HTCSP3, secciones 11.1 a 11.5 (no prestar atención al uso de `for`).

Con lo visto, ya pueden resolver hasta el Ejercicio 4 (inclusive) de la Guía de Ejercicios 5.