Listas

Clase #12

IIC1103 – Introducción a la Programación

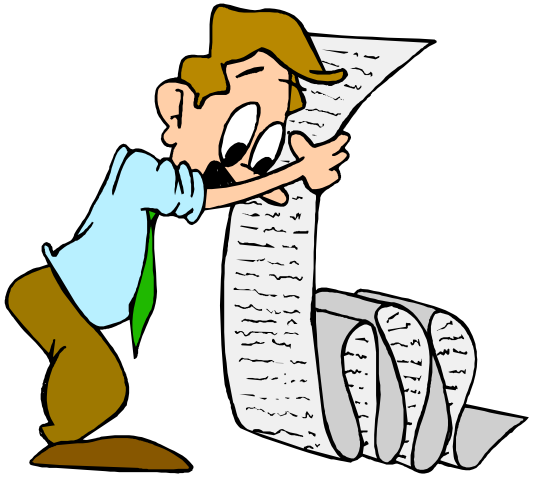
Marcos Sepúlveda (marcos@ing.puc.cl)

Veremos hoy …

- Listas
- Funciones y operaciones con listas
- Copias de listas
- Strings y listas
- Ejercicios

Listas – motivación

- De forma natural siempre hacemos listas, por ejemplo:
  - Una lista para ir al supermercado
  - Una lista de amigos que invitar a una fiesta
  - Una lista de cursos de este semestre



Listas – motivación

- Supermercado
  - Fideos
  - Arroz
  - Zapallo
  - Aceite
  - Queso
  - Sal

- Películas Favoritas
  - Gladiador
  - Blade Runner
  - Reto al destino
  - El secreto de tus ojos

Listas – motivación

- Las listas anteriores en Python se especifican del siguiente modo:

```
super = ["Fideos", "Arroz", "Zapallo", "Aceite", "Queso", "Sal"]

peliculas = ["Gladiador", "Blade Runner", "Reto al destino", "El
secreto de tus ojos"]
```

Listas – en Python

- Una lista en Python se define como una colección “ordenada” de valores.
  - Se representa como una lista de valores separados por comas entre corchetes.

- Observaciones
  - Una lista puede contener cualquier tipo de objetos (datos), pero por lo general todos tienen el mismo tipo.
  - El tipo asociado a una lista en Python se conoce como **`list`**.
  - Una lista puede no contener valores (lista vacía).

- Links:
  - https://docs.python.org/3/tutorial/introduction.html#lists
  - https://docs.python.org/3/tutorial/datastructures.html?highlight=lists#more-on-lists

Ejemplos

- **`lista1 = [1,2,3,4]`**
  - **`lista1`** tiene 4 elementos: 1, 2, 3 y 4
- **`lista2 = ["hola y chao"]`**
  - **`lista2`** tiene un solo elemento: "hola y chao”
- **`lista3 = ["agua", ["aceite", [2]], [1,2,3]]`**
  - **`lista3`** tiene 3 elementos: "agua", ["aceite",[2]], y [1,2,3]
- **`lista4 = []`**
  - **`lista4`** no tiene elementos, es una lista vacía
- **`lista5 = [lista1, lista2, lista3, lista4]`**
  - **`lista5`** es una lista compuesta de 4 listas.

Listas – índices

- De manera análoga a un String, los elementos de una lista pueden ser referenciados a través de índices.

```
L   =   [ "hola",   ["mani"],   "juan perez cotapo",   "vamos" ]
            0           1                 2                3
```

- Así se tiene:
  - `L[0]` → `"hola"`
  - `L[1]` → `["mani"]`
  - `L[2]` → `"juan perez cotapo"`
  - `L[3]` → `"vamos"`

Sublistas

- De manera análoga a los strings, se pueden obtener sublistas.

```
L   =   [ "hola",   ["mani"],   "juan perez cotapo",   "vamos" ]
            0           1                2                 3
```

- Así se tiene:
  - `L[2:4]` → `["juan perez cotapo", "vamos"]`
  - `L[2:]` → `["juan perez cotapo", "vamos"]`
  - `L[:2]` → `["hola", ["mani"]]`
  - `L[0:2]` → `["hola", ["mani"]]`
  - `L[1:3]` → `[["mani"], "juan perez cotapo"]`
  - `L[:]` → `["hola", ["mani"], "juan perez cotapo", "vamos"]`

Listas – largo y concatenación

- `L = ["hola", ["mani"], "juan perez cotapo", "vamos"]`

- También está definida la función **`len`** para listas.
  - `len(L)` → `4`

- Y es posible concatenar listas:
  - `lista1 = [1,2,3,4,5]`
  - `lista2 = [6,7,8,9,0]`
  - `lista1 + lista2` → `[1,2,3,4,5,6,7,8,9,0]`

Listas son mutables

- A diferencia de los strings, las listas son **mutables**:

- Se puede cambiar valores:

```
cubos = [1, 8, 27, 65, 125] #ups!
print(cubos)
cubos[3] = 64
print(cubos)
```

```
>>>
[1, 8, 27, 65, 125]
[1, 8, 27, 64, 125]
```

- Y se puede agregar nuevos elementos al final de la lista, utilizando el método **`append()`**:

```
cubos.append(216)
cubos.append(7 ** 3)
print(cubos)
```

```
>>>
[1, 8, 27, 64, 125, 216, 343]
```

Ejercicio 1

- Escriba una función en Python de nombre **`ListaPrimos`**, que reciba como parámetro un entero **`n`** y retorne una lista con los **`n`** primeros números primos.

- Por ejemplo:

```
lista = ListaPrimos(5)
print(lista)
```

```
>>>
[2, 3, 5, 7, 11]
```

Ejercicio 1 – solución

- Escriba una función de nombre **`ListaPrimos`**, que reciba como parámetro un entero **`n`** y retorne una lista con los **`n`** primeros números primos.

```
def EsPrimo(n):
    esPrimo = True
    i = 2
    while i <= n/2 and esPrimo:
        if n%i == 0:
            esPrimo = False
        i += 1
    return esPrimo

def ListaPrimos(n):
    primos = []
    i = 2
    while len(primos) < n:
        if EsPrimo(i):
            primos.append(i)
        i += 1
    return primos

lista = ListaPrimos(5)
print(lista)
```

```
>>>
[2, 3, 5, 7, 11]
```

Listas – métodos

- **`lista.append(x)`**
  - Añade elemento **`x`** al final de la lista.
- **`lista.extend(L)`**
  - Extiende la lista añadiendo todos los elementos de la lista **`L`**.
- **`lista.insert(i,x)`**
  - Inserta elemento **`x`** en la posición **`i`**.
  - **`lista.insert(0,x)`** inserta al principio de la lista, y **`lista.insert(len(lista),x)`** es equivalente a **`lista.append(x)`**.
- **`lista.remove(x)`**
  - Elimina el primer elemento de la lista cuyo valor es **`x`**. Indica un error si no existe tal elemento.
- **`lista.pop(i)`**
  - Retorna el elemento en la posición **`i`** de la lista, y lo elimina de ella.
  - Si no se especifica ningún índice, **`a.pop()`** retorna y elimina el último elemento de la lista.
- **`lista.clear()`**
  - Elimina todos los elementos de la lista.

Listas – métodos

- **`lista.index(x)`**
  - Retorna el índice del primer elemento en la lista cuyo valor es **`x`**. Indica un error si no existe tal elemento.
- **`lista.count(x)`**
  - Retorna el número de veces que **`x`** aparece en la lista.
- **`lista.sort()`**
  - Reordena los elementos de la lista en orden ascendente.
- **`lista.reverse()`**
  - Invierte los elementos de la lista.
- **`lista.copy()`**
  - Devuelve una copia de la lista. Equivalente a **`lista[:]`**.

Ejemplos – métodos lista

```
lista = [66.25, 333, 333, 1, 1234.5]
print(lista)

lista.insert(2, -1)
print(lista)

lista.append(333)
print(lista)

lista.extend([1,2,3])
print(lista)

print(lista.count(333), lista.count(66.25), lista.count('x'))

print(lista.index(333))

lista.remove(333)
print(lista)

lista.reverse()
print(lista)

lista.sort()
print(lista)

print(lista.pop())
print(lista)

lista2 = lista.copy()
lista.clear()
print(lista)
print(lista2)
```

```
>>>
[66.25, 333, 333, 1, 1234.5]
[66.25, 333, -1, 333, 1, 1234.5]
[66.25, 333, -1, 333, 1, 1234.5, 333]
[66.25, 333, -1, 333, 1, 1234.5, 333, 1, 2, 3]
3 1 0
1
[66.25, -1, 333, 1, 1234.5, 333, 1, 2, 3]
[3, 2, 1, 333, 1234.5, 1, 333, -1, 66.25]
[-1, 1, 1, 2, 3, 66.25, 333, 333, 1234.5]
1234.5
[-1, 1, 1, 2, 3, 66.25, 333, 333]
[]
[-1, 1, 1, 2, 3, 66.25, 333, 333]
```

Copia de listas

- Se debe usar `[:]` o el método **`copy()`**:

```
lista1 = [1, 2, 3, 4, 5]

# no crea una copia
lista2 = lista1

# crea una copia
lista3 = lista1[:]
lista4 = lista1.copy()

lista1.append(6)

print("lista1:", lista1)
print("lista2:", lista2)
print("lista3:", lista3)
print("lista4:", lista4)
```

```
>>>
lista1: [1, 2, 3, 4, 5, 6]
lista2: [1, 2, 3, 4, 5, 6]
lista3: [1, 2, 3, 4, 5]
lista4: [1, 2, 3, 4, 5]
```

Ejercicio 2

- Escriba una función en Python que permita eliminar todas las ocurrencias de un elemento **x** de una lista.

```
def Elimina(L,x):
    while x in L:
        L.remove(x)
    return L

lista = [1,0,2,0,0,3,0,0,0]
print(lista)
lista = Elimina(lista, 0)
print(lista)
```

```
>>>
[1, 0, 2, 0, 0, 3, 0, 0, 0]
[1, 2, 3]
```

String – método split

- `str.split(sep=None, maxsplit=-1)`

- Retorna una lista de las palabras contenidas en el string, utilizando **sep** como el string delimitador.
  - Se puede usar **sep** para especificar el delimitador. Si no se especifica, considera un conjunto de espacios como un solo separador y no toma en cuenta los espacios iniciales o finales.
  - Se puede usar **maxsplit** para acotar la cantidad de divisiones. La lista tendrá a lo más, **maxsplit + 1** elementos. Si no se especifica, entonces no hay límite en el número de divisiones (se hacen todas las divisiones posibles).

- Ver detalle en:
  - https://docs.python.org/3/library/stdtypes.html#str.split

String – método split

- Ejemplo

```
texto = "hola amigo Juan,   ¿qué tal?"
print(texto)

listaPalabras = texto.split()
print("texto separado con split()")
print(listaPalabras)

listaPalabras = texto.split(' ')
print("texto separado con split(' ')")
print(listaPalabras)

listaPalabras = texto.split(maxsplit=3)
print("texto separado con split() y maxsplit=3")
print(listaPalabras)
```

```
>>>
hola amigo Juan,   ¿qué tal?
texto separado con split()
['hola', 'amigo', 'Juan,', '¿qué', 'tal?']
texto separado con split(' ')
['hola', 'amigo', 'Juan,', '', '', '¿qué', 'tal?']
texto separado con split() y maxsplit=3
['hola', 'amigo', 'Juan,', '¿qué tal?']
```

String – método split

- Ejemplo

```
texto = "   hola, ¿qué tal?    "
print(texto)
print(texto.count(' '), "espacios")

listaPalabras = texto.split()
print("lista de palabras separadas con split()")
print(listaPalabras)

listaPalabras = texto.split(' ')
print("lista de palabras separadas con split(' ')")
print(listaPalabras)
```

```
>>>
   hola, ¿qué tal?
9 espacios
lista de palabras separadas con split()
['hola,', '¿qué', 'tal?']
lista de palabras separadas con split(' ')
['', '', '', 'hola,', '¿qué', 'tal?', '', '', '', '']
```

String – método split

- Ejemplo

```
texto = '''@marcos
@jgonzalez
hola amigo Juan,   ¿qué tal?'''
print(texto)
print(texto.count(' '), "espacios")

listaPalabras = texto.split()
print("lista de palabras separadas con split()")
print(listaPalabras)

listaPalabras = texto.split(' ')
print("lista de palabras separadas con split(' ')")
print(listaPalabras)

listaLineas = texto.split('\n')
print("lista de líneas separadas con split('\\n')")
print(listaLineas)
```

```
>>>
@marcos
@jgonzalez
hola amigo Juan,   ¿qué tal?
6 espacios
lista de palabras separadas con split()
['@marcos', '@jgonzalez', 'hola', 'amigo', 'Juan,', '¿qué', 'tal?']
lista de palabras separadas con split(' ')
['@marcos\n@jgonzalez\nhola', 'amigo', 'Juan,', '', '', '¿qué', 'tal?']
lista de líneas separadas con split('\n')
['@marcos', '@jgonzalez', 'hola amigo Juan,   ¿qué tal?']
```

String – método join

- **`str.join(`*`iterable`*`)`**

- Retorna un string que es la concatenación de los strings en el iterable (puede ser, por ejemplo, un **string** o una **lista de strings**), como delimitador entre los elementos se utiliza el string respecto al cual se está invocando el método.
  - Si es un **string**, se separa cada caracter por el delimitador.
  - Si es una **lista**, se separa cada elemento de la lista por el delimitador.

- Ver detalle en:
  - https://docs.python.org/3/library/stdtypes.html#str.join

String – método join

- Ejemplo

```
print("+++ Letras separadas")
delimitador = "---"
texto = delimitador.join("Hugo")
print(texto)

print("+++ Palabras separadas")
texto = delimitador.join(["Hugo", "Paco", "Luis"])
print(texto)

print("+++ Lista horizontal")
texto = ' '.join(["Hugo", "Paco", "Luis"])
print(texto)

print("+++ Lista vertical")
texto = '\n'.join(["Hugo", "Paco", "Luis"])
print(texto)
```

```
>>>
+++ Letras separadas
H---u---g---o
+++ Palabras separadas
Hugo---Paco---Luis
+++ Lista horizontal
Hugo Paco Luis
+++ Lista vertical
Hugo
Paco
Luis
```