**FUNDAMENTOS DE COMPUTACIÓN**
GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
Universidad de Cantabria

**Práctica 9 (laboratorio)**
25 de noviembre del 2019

# Listas vs diccionarios

## Listas: apodar, mutar y clonar

- Los valores/datos (números, cadenas, listas, etc.) se guardan en objetos.
  1. Objetos inmutables (no se pueden modificar/mutar) como enteros, cadenas o tuplas.
  2. Objetos mutables (se pueden modificar/mutar) como listas y diccionarios.
- Las variables son etiquetas para referenciar a los objetos y, son independientes de ellos.
- Asignando un valor a una variable se crea un objeto para almacenar el valor.
- Todo objeto tiene un identificador `id`.

```
>>> id(2019)
4330880944
>>> n = 2019; id(n)
4330880944
>>> L1 = [2,4,8,16]
>>> id(L1)
4331182024
```

```
>>> L = [0] + L1; L
[0, 2, 4, 8, 16]
>>> id(L)
4331182408
>>> L2 = L1
>>> L2
[2, 4, 8, 16]
```

```
>>> id(L2)
4331182024
>>> L1[0] = 1
>>> L2
[1, 4, 8, 16]
>>> id(L2)
4331182024
```

- `L` y `L1` son distintos objetos, distinto identificador. Sin embargo, `L2` es un objeto idéntico a `L1`. Se dice que `L2` es un apodo (un alias) de `L1`.
- En la última columna L1 se ha transformado(mutado) a [1, 2, 4, 8, 16].

A continuación mostramos otras instrucciones con o sobre listas:

1. L.append(x) Añade el elemento $x$ al final de la lista $L$.
2. L.insert(i,x) Inserta el elemento $x$ en la posición $i$. El primer argumento es el índice del elemento delante del cual se insertará.
3. L.remove(x) Elimina el elemento $x$ de la lista $L$. Produce un error si no existe tal elemento.
4. del(L[i]) Permite eliminar el elemento $L[i]$ de la lista $L$ o varios elementos a la vez, e incluso la misma lista.

L= [2, 1, 3, 6, 3, 7]

- L.append(0) $---> $ se muta a $L = [2, 1, 3, 6, 3, 7, 0]$
- L.remove(3) $---> $ se muta a $[2, 1, 6, 3, 7, 0]$
- del(L[2]) $----> $ se muta a $[2, 1, 3, 7, 0]$
- L.insert(1,100) $--> $ se muta a $[2, 100, 1, 3, 7, 0]$
- del(L[1:3]) $---> $ se muta a $[2, 3, 7, 0]$

```
>>> M1 = [x**2 for x in [0,1,2,3]]
>>> M1
[0, 1, 4, 9]
>>> id(M1)
4320805448
>>> M2 = M1 + [16]
 M2
[0, 1, 4, 9, 16]
```

```
>>> id(M2)
4320815496
>>> M1.append(16); M1
[0, 1, 4, 9, 16]
>>> id(M1)
4320805448
>>> M1 == M2
True
```

- Observar la diferencia M1.append(x), con concatenar listas : M1+[x],
- Podemos construir listas por comprensión, como la lista M1 o como matriz_identitad.py:

```
def matriz_identidad(n):
    In = [ [0 for x in range(n)] for x in range(n)]
    for i in range(n):
        In[i][i] == 1
    return In
```

Dadas dos listas L1 y L2, queremos eliminar los elementos de L1 que están en L2:

```
def quitar_comunes1(L1, L2):
    for x in L1:
        if x in L2:
            L1.remove(x)
```

```
def quitar_comunes1(L1, L2):
     L1c = L1[:]
    for x in L1c:
        if x in L2:
            L1.remove(x)
```

Ejecutando los dos programas:

```
>>> quitar_comunes1([1,2,3,4], [1,2,5,6])
[2, 3, 4]
quitar_comunes2([1,2,3,4], [1,2,5,6])
[3, 4]
```

La lista `L1c = L1[:]` es una clonación de la lista `L1`.

## Diccionarios

Queremos almacenar la información de los clientes de una compañía de seguros de vehículos (nombre, fecha del carnet de conducir, número de poliza, etc.).

```
list_nombre = ['Boole', 'Neumann', 'Torres', 'Turing']
list_fecha = [ '2/1987', '5/2011', '4/1765', '10/1959]
list_poliza = [ 20.83, 36.8, 211.334,  981.45]
```

- Una lista para cada pieza de información.
- Todas las listas deben tener la misma longitud.
- La información almacenada en listas en el mismo índice.
- Actualizar y recuperar la información del cliente:

```
def info_cliente(cliente, list_nombre, list_fecha, list_poliza):
     i = list_nombre.index(cliente)
     fecha = list_fecha[i]
     poliza = list_poliza[i]
     return (fecha, poliza)
```

```
>>> A = [1, 'H', 3.5, (True, list)]
>>> A.index('H')
1
```

### Problemas

- Desordenado para realizar un seguimiento, si tiene un montón de información diferente.
- Debe mantener muchas listas, pasarlas como argumentos y siempre debe indexar usando enteros.

Alternativamente, podríamos implementar esto en una sólo lista `clientes`:

```
clientes = [('Boole',[2/1987',20.83]),('Neumann',['5/2011', 36.8]),
('Torres', ['4/1765', 211.334]), ('Turing',['10/1959',981.45])
```

- Donde cada elemento de una lista es una tupla con el formato (nombre, información).
- Para obtener información sobre el nombre necesitaríamos averiguar el índice y luego usar `clientes [1]`
  - **Problema:** Muy lento para encontrar el índice, si la lista es muy grande

Un diccionario es una asignación entre un conjunto de índices (claves) y un conjunto de valores. Cada clave se asigna a un valor. La asociación de una clave y un valor se denomina par `clave : valor` o un elemento.

- Un diccionario es una secuencia de elementos del tipo `clave : valor` encerrados entre llaves y separados por comas:

  `{clave0: valor0, clave1: valor1,.....}`

- Los diccionarios son de la clase `<class 'dict'>`
- Para extraer un valor asociado a una `clave`: `nombre_diccionario[clave]`

```
>>> clientes = {'Boole': ['2/1987',20.83] , \
'Neumann' : ['5/2011', 36.8], 'Torres' :  ['4/1765', 211.334], \
 'Turing' : ['10/1959',981.45]}
>>> type(clientes)
<class 'dict'>
>>> clientes['Torres']
['4/1765', 211.334]
>>> id(clientes)
4302132928
```

- Las claves deben ser objetos inmutables, pero los valores pueden ser cualquier objeto. Las claves deben ser únicas, pero no necesariamente los valores.
- Los diccionarios son objetos **mutables**, podemos modificar y añadir elementos: `clientes[clave] = valor`
- No hay ningún orden para las claves y los valores.
- Para saber si una clave esta en el diccionario: `'Torres' in clientes`
- Para eliminar un elemento del diccionario: `del(clientes['Boole'])`

```
>>> clientes['Gauss'] = ['11/1578', 78.6]
>>> clientes['Turing'] = ['10/1959',100.2]
>>> 'Torres' in clientes
True
>>> del(clientes['Boole'])
>>> clientes
{'Neumann': ['5/2011', 36.8], 'Torres': ['4/1765', 211.334],
'Turing': ['10/1959', 100.2], 'Gauss': ['11/1578', 78.6]}
>>> id(clientes)
4302132928
```

A continuación mostramos otras instrucciones con o sobre diccionarios $D$:

- len(D) devuelve el número de elementos.
- D.keys() y D.values() devuelve las claves y los valores (respectivamente).
- D.items() devuelve los elementos (clave, valor).

**Los tres métodos son iterables**

```
>>> D = {'uno':'one', 'dos':'two', 'tres':'three'}
>>> D.keys(); D.values()
dict_keys(['uno', 'dos', 'tres'])
dict_values(['one', 'two', 'three'])
>>> D.items()
dict_items([('uno', 'one'), ('dos', 'two'), ('tres', 'three')])
>>> for x in D.items():
...  print(x)
...
('uno', 'one')
('dos', 'two')
('tres', 'three')
```

Ahora, queremos contar cuántas veces aparece cada letra en una cadena, programa denominado `histograma`.

1. Podríamos crear la lista **letras** con las 27 variables, una para cada letra del alfabeto. Luego, recorremos la cadena y, para cada carácter que este en la cadena, incrementamos el contador correspondiente.
2. También podríamos crear un diccionario con caracteres como claves y contadores como los valores correspondientes. La primera vez que vea un carácter, agregará un elemento al diccionario. Después de eso, incrementaría el valor de un elemento existente.

---

```
1 def histograma(c):
2     d = {}
3    for x in c:
4         if x not in d:
5             d[x] = 1
6         else:
7             d[x] += 1
8     return d
```

---

Cada una de estas opciones realiza el mismo cálculo, pero cada una de ellas implementa ese cálculo de una manera diferente.

Usando listas tenemos que recorrer la cadena 27 veces, con diccionarios solamente una.

## Ejercicios

1. Queremos contar cuántas veces aparece cada letra minúsculas en una cadena. Podríamos crear la lista **letras** con las 27 variables, una para cada letra del alfabeto. Luego, recorremos la cadena y, para cada carácter que este en la cadena, incrementamos el contador correspondiente.

   - Construir una función `histograma` que tenga como argumento una cadena y devuelva una lista, donde los elementos son tuplas **(letra, numero)**, donde **numero** es el número de veces que aparece la **letra** en la cadena.

     ```
     >>> histograma('abracadabra')
     [('a', 5), ('b', 2), ('c', 1), ('d', 1), ('r', 2)]
     >>> histograma('en un lugar de la mancha, de donde nombre no quiero acordarme')
     [('a', 6), ('b', 1), ('c', 2), ('d', 5), ('e', 7), ('g', 1), ('h', 1), ('i', 1),
     ('l', 2), ('m', 3), ('n', 6), ('o', 5), ('q', 1), ('r', 5), ('u', 3)]
     ```

   - Escribir una función `letras_mas_frecuentes` que tenga como argumento una lista como la salida de la función anterior `histograma` y devuelva las letras más frecuentes, como en el ejemplo:

     ```
     >> his = histograma('abracadabra')
     >>> his
     [('a', 5), ('b', 2), ('c', 1), ('d', 1), ('r', 2)]
     >>> letras_mas_frecuentes(his)
     (['a'], 5)
     >>> his = histograma('aabbrtyuitn')
     >>> his
     [('a', 2), ('b', 2), ('i', 1), ('n', 1), ('r', 1), ('t', 2), ('u', 1), ('y', 1)]
     >>> letras_mas_frecuentes(his)
     (['a', 'b', 't'], 2)
     ```

2. Uno de los métodos más importantes para ordenar listas fue diseñado en 1945 por John Von Neumann, el creador del ordenador. Diseña una función `ordena_mezclando` que tenga como argumento dos listas ordenadas cuyos elementos son datos numéricos, y devuelva una única lista con todos sus elementos ordenados. Para ello deberéis implementar la siguiente estratégia;

   Supongamos que tenemos dos pilas de cartas (las dos listas) con la cara por arriba. Cada pila está ordenada tal que la carta más pequeña es la superior. Queremos mezclar las dos pilas para obtener una única pila ordenada.

   - Elegimos la más pequeña entre de las dos cartas superiores de cada pila;
   - la quitamos de esa pila y la ponemos en una nueva pila con la cara por abajo;
   - repetimos este procedimiento hasta que no haya cartas en alguna de las pilas;
   - entonces sólo tenemos que coger el resto de las cartas de la otra pila y ponerlas boca abajo en la nueva pila.

```
>>> ordena_mezclando([-1,9,9,21],[3.1,7,23])
[-1,3.1,7,9,9,21,23]
```

3. Podemos representar una matriz de $n$ filas por $m$ columnas como una lista $n$ elementos formados por listas com $m$ elementos.

```
>>> A=[[1,2,3],[4,5,6]]
>>> A
[[1, 2, 3], [4, 5, 6]]
>>> cero=[[0 for i in range(3)] for i in range(2)]
>>> cero
[[0, 0, 0], [0, 0, 0]]
```

   Escribe un programa que tenga como entrada dos matrices del mismo tamaño y devuelva la matriz suma.

4. Escribe un programa de Python para generar todas las sublistas de una lista.

```
 >>> sub_listas([1,2,3,4])
[[], [1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [2], [2, 3], [2, 3, 4], [3], [3, 4], [4]]
>>> sub_listas(['a','b','c'])
[[], ['a'], ['a', 'b'], ['a', 'b', 'c'], ['b'], ['b', 'c'], ['c']]
```

5. Crea un diccionario donde la clave sea el nombre del usuario y el valor sea el teléfono. Tendrás que ir pidiendo contactos hasta el usuario diga que no quiere insertar mas. No se podrán meter nombres repetidos.

```
Introduce un nombre: Alan Turing
Introduce un telefono: 94234567
Añadido el contacto
¿Quieres salir? (S/N) N
Introduce un nombre: Alfred Boole
Introduce un telefono: 942206787
Añadido el contacto
¿Quieres salir? (S/N) S
{'Alan Turing': 94234567, 'Alfred Boole': 942206787}
```

6. Escribe una función que tenga como argumentos dos listas de la misma longitud y devuelva el diccionario de la misma longitud y con clave y valor de cada lista.

```
 >>> nombre_color = ["Negro", "Rojo", "Marron", "Amarillo"]
>>> codigo_color = ["#000000", "#FF0000", "#800000", "#FFFF00"]
>> list_dic(nombre_color,codigo_color)
{'Negro': '#000000', 'Rojo': '#FF0000', 'Marron': '#800000', 'Amarillo': '#FFFF00'}
```

7. Como en el ejercicio 1 de arriba, queremos contar cuántas veces aparece cada letra minúscula en una cadena.

   - Construir una función `histograma1` que tenga como argumento una cadena y devuelva el diccionario, donde los valores son los caracteres y las claves el número de veces que aparece el carácter en la cadena.

   ```
   >>> histograma1('abracadabra')
   [('a', 5), ('b', 2), ('r', 2), ('c', 1), ('d', 1)]
   >>> histograma1('en un lugar de la mancha, de donde nombre no quiero acordarme')
   {'e': 7, 'n': 6, ' ': 11, 'u': 3, 'l': 2, 'g': 1, 'a': 6, 'r': 5, 'd': 5, 'm': 3, 'c': 2, 'h': 1, ',
   ```

   - Escribir una función `letras_mas_frecuentes2` que tenga como argumento un diccionario como la salida de la función anterior `histograma1` y devuelva los caracteres más frecuentes, como en el ejemplo:

   ```
   >>> his =histograma1('abracadabra')
   >>> his
   {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
   >>> letras_mas_frecuentes2(his)
   (['a'], 5)
   >>> his = histograma1('aabbrtyuitn')
   >>> his
   {'a': 2, 'b': 2, 'r': 1, 't': 2, 'y': 1, 'u': 1, 'i': 1, 'n': 1}
   >>> letras_mas_frecuentes2(his)
   (['a', 'b', 't'], 2)
   >>>
   ```

8. Dado un diccionario d y una clave k, es fácil encontrar el valor correspondiente `v = d [k]`. Pero ¿y si tienes `v` y quieres encontrar `k`? Tiene dos problemas: primero, puede haber más de una clave que se asigna al valor v. Según la aplicación, puede elegir una o puede tener que hacer una lista que contenga todas ellas. Escribe una función `invertir_diccionario` que invierta un diccionario:

   ```
   >>> his = histograma1('abracadabra')
   >>> his
   {'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1}
   >>> invertir_diccionario(his)
   {5: ['a'], 2: ['b', 'r'], 1: ['c', 'd']}
   ```