Tema 5 - Estructuras de datos sencillas

November 6, 2020

1 Introducción

- Una estructura de datos es un conjunto de datos referenciados por una sola variable (una variable que puede guardar varios datos a la vez)
- Array: variable que me permite guardar múltiples datos del mismo tipo (en general muchos, muchos). En Python tenemos algo más o menos equivalente (y mejor) que son las listas y tuplas
- Registro: variable que me permite guardar algunos datos de tipos distintos. En Python tenemos diccionarios y objetos

2 Listas

- Una colección finita, ordenada, heterogénea y mutable de valores
- finita: que tiene un número máximo de elementos posibles (que depende del ordenador)
- ordenada: el orden de los elementos importa y además puedo usar su posición para acceder a cada uno de ellos
- heterogénea: puede guardar datos de tipos distintos (no como los arrays de otros lenguajes)
- mutable: el valor de sus elementos puede cambiar (también el número de elementos, puedo añadir y quitar elementos de las listas)
- En comparación un string sería una colección finita, ordenada, homogéna e inmutable de letras
- Para declarar una lista: `lista = [elemento1, elemento2, elemento3]`

[1]:
```
# Creo una lista de 4 elementos enteros
lista1 = [1, 2, 3, 4]
# Otra de 5 elementos de tipos distintos
lista2 = [1, "hola", 3.54, False, 3 + 2j]
# Puedo usar el operador + para concatenar listas
lista3 = lista1 + lista2
print(lista1)
print(lista2)
print(lista3)
# Para acceder a un elemento de una lista, uso su índice
# el primero es el 0, puedo usar negativos también
print("El primer elemento de la lista", lista1, "es", lista1[0])
```

```
print("El 2º elemento contando por el final de", lista1, "es", lista1[-2])
# Cambio el valor del primer elemento (las listas son mutables)
lista1[0] = "adios"
print("La lista es ahora:", lista1)
# Puedo usar operadores de troceamiento como en las cadenas
print("Elementos de", lista3, "entre el 0 y el 4", lista3[0:4])
# Puedo usar operadores de troceamiento para cambiar partes de la lista
# Cambio dos elementos por uno solo
lista1[1:3] = ["Pepe"]
print("Ahora la lista es", lista1)
# Cambio 3 elementos por 4
lista3[2:5] = [5, 6, 7, 8]
print("Ahora la lista es", lista3)
```

```
[1, 2, 3, 4]
[1, 'hola', 3.54, False, (3+2j)]
[1, 2, 3, 4, 1, 'hola', 3.54, False, (3+2j)]
El primer elemento de la lista [1, 2, 3, 4] es 1
El 2º elemento contando por el final de [1, 2, 3, 4] es 3
La lista es ahora: ['adios', 2, 3, 4]
Elementos de [1, 2, 3, 4, 1, 'hola', 3.54, False, (3+2j)] entre el 0 y el 4 [1,
2, 3, 4]
Ahora la lista es ['adios', 'Pepe', 4]
Ahora la lista es [1, 2, 5, 6, 7, 8, 'hola', 3.54, False, (3+2j)]
```

2.1 Formas de crear listas

- Usando literales: directamente decimos el valor que va tener cada elemento de la lista (sólo útil para listas pequeñas)
- Usando variables: copio el valor de la variable en el elemento de la lista. No estoy metiendo la variable en la lista. Si la variable cambia, la lista no cambia y viceversa.
- Usando otras listas y el operador `+`: se copian los valores de las otras listas en la nueva lista, pero no son la misma lista. Si una cambia la otra no
- Usando otras listas y el operador `*`: me permite repetir partes de listas

[2]:
```
# Usando literales
lista1 = [1, 2, 3, 4, 5, 6]
# Usando variables
a, b, c = 1, 2, 4
lista2 = [a, b, c]
print("lista2 es ", lista2)
# Si la variable a cambia, lista2 no cambia y viceversa
# Esto se debe a que las variables de tipos básicos son inmutables
a = 8
print("lista2 es ", lista2, "no cambia")
# Usando otras listas
# Lo que hace el operador + es crear copias de lista1 y lista2,
# no trabaja directamente con ellas
```

```
lista3 = lista1 + lista2
print("lista3 es ", lista3)
# Intercambio valores entre dos posiciones de lista1
lista1[0], lista1[1] = lista1[1], lista1[0]
print("lista1 es ", lista1)
# lista3 no varía
print("lista3 es ", lista3, "no cambia")
# Usando el operador *
lista4 = lista2 * 2 + ["hola", "adios"] * 5
print("lista4 es ", lista4)
```

```
lista2 es  [1, 2, 4]
lista2 es  [1, 2, 4] no cambia
lista3 es  [1, 2, 3, 4, 5, 6, 1, 2, 4]
lista1 es  [2, 1, 3, 4, 5, 6]
lista3 es  [1, 2, 3, 4, 5, 6, 1, 2, 4] no cambia
lista4 es  [1, 2, 4, 1, 2, 4, 'hola', 'adios', 'hola', 'adios', 'hola', 'adios',
'hola', 'adios', 'hola', 'adios']
```

2.2 Funciones y operadores que se pueden usar con listas

- `len(lista)`: devuelve la longitud de la lista
- `print(lista)`: imprime la lista entre corchetes
- `del(lista)`: borra todos los elementos de la lista y también la variable. No merece la pena en general porque Python ya borra de la memoria las variables que no se necesitan.
- `del(lista[posicion])`: borra el elemento de esa posición de la lista (con índices negativos se empieza contando desde el final)
- operador `in` y `not in`: me dicen si un elemento está en una lista o no

[3]:
```
print("la longitud de lista4 es", len(lista4))
print("lista1 es", lista1)
# Borramos el primer elemento de lista 1
del(lista1[0])
print("Al borrar el primer elemento, lista1 es", lista1)
# Borramos el segundo por el final
del(lista1[-2])
print("Al borrar el segundo elemento por el final, lista1 es", lista1)
print("¿Hay un 4 en la lista1?", 4 in lista1)
```

```
la longitud de lista4 es 16
lista1 es [2, 1, 3, 4, 5, 6]
Al borrar el primer elemento, lista1 es [1, 3, 4, 5, 6]
Al borrar el segundo elemento por el final, lista1 es [1, 3, 4, 6]
¿Hay un 4 en la lista1? True
```

2.3 Métodos de lista

Un método es similar a una función, pero si una función se invoca como `funcion(variable)`, los métodos se invocan con `variable.metodo()` (veremos más en los temas 6 y 7)

- `append(elemento)`: añade un elemento al final de la lista
- `insert(posición, elemento)`: inserta un elemento en la posición que le digo moviendo el resto hacia el final. Si la posición no existe lo inserta al final. Puedo usar negativos: `insert(-1, elemento)` es equivalente a `append(elemento)`
- `extend(lista2)`: añade los elementos de lista2 a lista1. Es equivalente a poner `lista1 = lista1 + lista2`, pero extend es más rápido
- `index(elemento)`: devuelve la primera posición de ese elemento en la lista o ERROR si no está en la lista (debería comprobar antes que el elemento está)
- `index(elemento, inicio, final)`: devuelve la posición de ese elemento entre el inicio y el final (no incluido)
- `count(elemento)`: devuelve cuántas veces el elemento está en la lista
- `clear()`: borra los elementos de la lista y la pone a la lista vacía (con la función `del()` también se borraba la variable)
- `remove(elemento)`: borra la primera aparición del elemento en la lista. Error si no está en la lista
- `pop()`: borra y DEVUELVE el último elemento de la lista. Debería siempre asignar el resultado de `pop()` a una variable para guardar ese elemento.
- `pop(posicion)`: borra y DEVUELVE el elemento de esa posición (la diferencia con `del(lista[posicion])` es que con `del` el elemento se pierde)
- `reverse()`: da la vuelta a la lista
- `sort()`: ordena la lista de menor a mayor. Los elementos de la lista deben ser compatibles (no se pueden ordenar listas con números y string)

[4]:
```
lista1 = [1, 2, 3, 4, 5]
# Inserto un elemento al final
lista1.append(23)
print("lista1 es", lista1)
# Se puede hacer también de esta forma, pero es mucho menos eficiente
lista1 = lista1 + [23]
print("lista1 es", lista1)
# Inserto un elemento en la posición que quiera
lista1.insert(2, "hola")
print("lista1 es", lista1)
# Añado una lista a otra
lista2 = [True, 22.2, 34]
lista1.extend(lista2)
print("lista1 es", lista1)
# Ojo, no usar append en este caso porque añade la segunda lista como un único␣
 ↪elemento
lista1.append(lista2)
print("lista1 es", lista1)
# Posición de 'hola' en la lista
if "hola" in lista1:
```

```
    print("La posición de 'hola' en la lista es", lista1.index("hola"))
# Inserto un 3 en la posición 12
lista1.insert(12, 3)
print("lista1 es", lista1)
# Posición del 3 entre la 5 y la 16
print("¿En qué posición está el 3 después de la 5 y antes de la 16?"
      , lista1.index(3,5, 16))
print("¿Cuantos 23 hay en la lista?", lista1.count(23))
# Borramos una lista
lista2 = ['a', 'b', 'c', 'd']
print("lista2 es", lista2)
lista2.clear()
print("Al hacer clear lista2 se vacía:" , lista2)
# Borramos 'hola' de la lista
lista1.remove("hola")
print("lista1 es", lista1)
# Quitamos el último elemento de la lista y lo guardamos en la variable
a = lista1.pop()
print("lista1 es", lista1)
print("el último elmento de lista1 era", a)
# Le damos la vuelta
lista1.reverse()
print("lista1 es", lista1)
# ordenando listas (todos los números y boolean son compatibles)
lista2 = [3, 6, 21.1, 1, True, 4, 3, 1]
lista2.sort()
print("lista2 es", lista1)
```

```
lista1 es [1, 2, 3, 4, 5, 23]
lista1 es [1, 2, 3, 4, 5, 23, 23]
lista1 es [1, 2, 'hola', 3, 4, 5, 23, 23]
lista1 es [1, 2, 'hola', 3, 4, 5, 23, 23, True, 22.2, 34]
lista1 es [1, 2, 'hola', 3, 4, 5, 23, 23, True, 22.2, 34, [True, 22.2, 34]]
La posición de 'hola' en la lista es 2
lista1 es [1, 2, 'hola', 3, 4, 5, 23, 23, True, 22.2, 34, [True, 22.2, 34], 3]
¿En qué posición está el 3 después de la 5 y antes de la 16? 12
¿Cuantos 23 hay en la lista? 2
lista2 es ['a', 'b', 'c', 'd']
Al hacer clear lista2 se vacía: []
lista1 es [1, 2, 3, 4, 5, 23, 23, True, 22.2, 34, [True, 22.2, 34], 3]
lista1 es [1, 2, 3, 4, 5, 23, 23, True, 22.2, 34, [True, 22.2, 34]]
el último elmento de lista1 era 3
lista1 es [[True, 22.2, 34], 34, 22.2, True, 23, 23, 5, 4, 3, 2, 1]
lista2 es [[True, 22.2, 34], 34, 22.2, True, 23, 23, 5, 4, 3, 2, 1]
```

2.4 Comparación de listas

- Podemos utilizar `==` y `!=` para saber si dos listas son iguales o no

- También puedo usar `>` `>=` `<` `<=` Estos operadores comparan elemento a elemento hasta que pueden responder. Primero miran el primero, si hay respuesta no miran el segundo

[5]:
```
lista1 = [1, 2, 3]
lista2 = [1, 2, 3]
print("¿Son lista1 y lista2 iguales?", lista1 == lista2)
lista3 = [1, 3]
# lista1 es menor que lista3 porque el primer elemento diferente que tienen
# es el segundo y ahí 2 < 3
print("¿Es lista1 mayor que lista3?", lista1 > lista3)
# Son iguales, así que no pueden ser menors
print("¿Es lista1 menor que lista2?", lista1 < lista2)
# Puedo comparar listas heterogéneas siempre que en la msima posición de cada␣
 ↪lista
# el tipo del dato sea el mismo
# Para str mira el código ASCII (A = 65, B = 66, etc.) (a = 97, b = 98, etc)
lista4 = [1, "hola", True]
lista5 = [1, "adios", False]
print("¿Es lista4 mayor que lista5?", lista4 > lista5)
```

```
¿Son lista1 y lista2 iguales? True
¿Es lista1 mayor que lista3? False
¿Es lista1 menor que lista2? False
¿Es lista4 mayor que lista5? True
```

2.5 Copia de listas

- Si asigno una lista a otra con el operador `=` no se copian sino que son la misma lista (esto una característica de los elementos mutables). Nunca se debe asignar una lista a otra
- La función `id()` me dice la posición de memoria de una variable, si dos variables tienen la misma posición de memoria es que son la misma variable, es decir, el mismo dato (o conjunto de datos) con dos nombres distintos
- También puedo usar el operador de identidad `is` que me dice si dos variables son en realidad la misma
- Varias formas de copiar listas
    - uso el método `copy()`
    - usando bucles sobre una lista vacía
    - usando operadores de troceamiento
    - usando el método `extend()` sobre una lista vacía

[6]:
```
lista1 = [1, 2, 3]
lista2 = lista1
lista2[0] = 9
print(lista1)
print(id(lista1))
print(id(lista2))
print("¿Son la lista1 y la lista2 la misma?", lista1 is lista2)
# Copia usando copy
```

```
lista2 = lista1.copy()
print("Copio usando el método copy()")
print(id(lista1))
print(id(lista2))
print("¿Son lista1 y lista2 iguales?", lista1 == lista2)
print("¿Son la lista1 y la lista2 la misma?", lista1 is lista2)
# Copia con un bucle for
lista2 = []
for elemento in lista1:
    lista2.append(elemento)
print("Copio usando un bucle")
print(id(lista1))
print(id(lista2))
print("¿Son lista1 y lista2 iguales?", lista1 == lista2)
print("¿Son la lista1 y la lista2 la misma?", lista1 is lista2)
# Copia con operadores de troceamiento
# [:] son todos los elementos de una lista
lista2 = lista1[:]
print("Copio usando troceamiento")
print(id(lista1))
print(id(lista2))
print("¿Son lista1 y lista2 iguales?", lista1 == lista2)
print("¿Son la lista1 y la lista2 la misma?", lista1 is lista2)
# Copia usando extend
lista2 = []
lista2.extend(lista1)
print(id(lista1))
print(id(lista2))
print("¿Son lista1 y lista2 iguales?", lista1 == lista2)
print("¿Son la lista1 y la lista2 la misma?", lista1 is lista2)
```

```
[9, 2, 3]
103501496
103501496
¿Son la lista1 y la lista2 la misma? True
Copio usando el método copy()
103501496
101011824
¿Son lista1 y lista2 iguales? True
¿Son la lista1 y la lista2 la misma? False
Copio usando un bucle
103501496
103501616
¿Son lista1 y lista2 iguales? True
¿Son la lista1 y la lista2 la misma? False
Copio usando troceamiento
103501496
```

```
103501536
¿Son lista1 y lista2 iguales? True
¿Son la lista1 y la lista2 la misma? False
103501496
103501616
¿Son lista1 y lista2 iguales? True
¿Son la lista1 y la lista2 la misma? False
```

2.6 Bucles y listas

Generalmente se utilizan bucles `for`, aunque según el caso es mejor usar un bucle u otro. Recomendaciones:

- Si lo que quiero es recorrer una lista entera pero no cambiar nada de ella, uso un bucle `for-each`
- Si quiero recorrer una lista entera y cambiar algo de ella uso un `for` con `range`
- Si quiero recorrer solo parcialmente una lista, uso un `while`

[7]:
```python
# Primer ejemplo: sumar los elementos de una lista
# Como quiero recorrer la lista entera y no voy a cambiar nada, uso un for␣
 ↪normal
suma = 0
lista1 = [2, 3, 5, 6]
for elemento in lista1:
    # En cada iteración se copia un elemento de la lista en la variable␣
 ↪elemento,
    # trabajo con la copia, no con el original
    # Aunque yo cambiase la variable elemento, el dato de la lista no cambiaría
    suma += elemento
print(suma)

# Segundo ejemplo: cambiar impares por pares
# Como quiero recorrer la lista entera y cambiar valores, uso un for con range
# posicion tomará los valores 0, 1, etc.
for posicion in range(len(lista1)):
    # Si el elemento de esa posición es impar
    if lista1[posicion] % 2 != 0:
        # Le sumo 1 a ese elemento
        lista1[posicion] = lista1[posicion] + 1
print(lista1)

# Tercer ejemplo: buscar si hay algún elemento mayor que 2 en la lista
# No sé si voy a tener que recorrer toda la lista porque si encuentro
# el elemento ya no tengo por qué seguir buscando, así que uso un while
encontrado = False
posicion = 0
# Sigo buscando si no lo he encontrado y si me quedan elementos
while not encontrado and posicion < len(lista1):
```

```
    if lista1[posicion] > 2:
        encontrado = True
    else:
        posicion = posicion + 1
if encontrado:
    print("Hay un elemento mayor que 2 en la posición", posicion)
else:
    print("No hay elementos mayores que 2")
```

```
16
[2, 4, 6, 6]
Hay un elemento mayor que 2 en la posición 1
```

3 Tuplas

- Una secuencia ordenada, finita, heterogénea e **inmutable** de datos
- No puedo cambiar el valor de sus elementos ni añadir o quitar ninguno
- Es una especie de lista de costantes
- `tupla = (dato1, dato2, dato3)`
- Los paréntesis son opcionales
- Formas de crear tuplas:
    - Usando literales
    - Usando variables: el valor de la variable se va a copiar en la tupla. Si la variable cambia, la tupla no.
    - Usando otra tupla y el operador `+`: se crea una copia de cada una de las tuplas y se guarda en esa variable
    - Usando otra tupla y el operador `*`: se crea una copia de cada una de las tuplas y se guarda en esa variable
    - Usando una lista y haciendo un casting a tupla, con la función `tuple(lista)` (para pasar de tupla a lista se usa la función `list(tupla)`

[1]:
```
#Tupla con literales
tupla1 = (1, 2, 3, 4, 5, "hola")
print("tupla1 es", tupla1)
# Se puede hacer sin paréntesis, pero no se recomienda
tupla2 = 1, 2, 5, 6
print("tupla2 es", tupla2)
# Una tupla de un solo elemento (tengo que poner la coma)
tupla3 = (1,)
print("tupla3 es", tupla3)
# Usando variables
a, b, c = 3, 66, False
tupla4 = (a, b, c)
print("tupla4 es", tupla4)
a = 13
print("tupla4 es", tupla4, "no cambia")
```

```python
# Usando el operador +
tupla5 = tupla4 + tupla3
print("tupla5 es", tupla5)
# Usando el operador *
tupla6 = tupla3 * 2 + tupla1 * 4
print("tupla6 es", tupla6)
# Usando una lista
lista1 = []
for i in range(5):
    lista1.append(i)
tupla7 = tuple(lista1)
print("lista1 es", lista1)
print("tupla7 es", tupla7)
# Puedo sumar una tupla a sí misma, pero no debo hacerlo porque realmente estoy
# creando otra variable
tupla8 = (1, 2, 3)
print("El id de tupla8 es", id(tupla8))
tupla8 = tupla8 + (5, 6, 7)
print("Ahora el id es", id(tupla8))
print("tupla8 es", tupla8)
# No puedo cambiar los datos, esto daría error
# tupla8[0] = 7
# Esto ya no es una tupla, estoy cambiando el tipo de la variable,
# no debería hacerlo nunca
tupla8 = "hola"
```

```
tupla1 es (1, 2, 3, 4, 5, 'hola')
tupla2 es (1, 2, 5, 6)
tupla3 es (1,)
tupla4 es (3, 66, False)
tupla4 es (3, 66, False) no cambia
tupla5 es (3, 66, False, 1)
tupla6 es (1, 1, 1, 2, 3, 4, 5, 'hola', 1, 2, 3, 4, 5, 'hola', 1, 2, 3, 4, 5,
'hola', 1, 2, 3, 4, 5, 'hola')
lista1 es [0, 1, 2, 3, 4]
tupla7 es (0, 1, 2, 3, 4)
El id de tupla8 es 97097128
Ahora el id es 97178488
tupla8 es (1, 2, 3, 5, 6, 7)
```

3.1 ¿Por qué usamos tuplas?

- Porque quiero marcar que algo no va a cambiar y quiero asegurarme de ello (tiene sentido cuando usamos funciones)
- Porque me lo pide alguien (otro programa, una función): porque necesita algo inmutable
- Porque son más ¿rápidas? y ocupan algo menos de memoria

3.2 Operadores, funciones y métodos con tuplas

- `print(tupla)`: imprime la tupla
- `type(tupla)`: me dice que el tipo de la variable es una tupla
- `id(tupla)`: me devuelve la dirección de memoria en la que está guardada la tupla
- `len(tupla)`: me devuelve la longitud de la tupla
- `del(tupla)`: borro la tupla y la variable. No puedo borrar un solo elemento, solo la tupla entera
- `in` y `not in`: para saber si algo está en la tupla o no
- `.count(elemento)`: cuenta cuántas veces aparece un elemento en la tupla
- `.index(elemento)`: devuelve la primera posición del elemento en la tupla o salta un Error si no está (debería usar el operador `in` antes para asegurarme de que está)
- `.index(elemento, inicio, fin)`: solamente lo busca entre inicio y fin (sin incluir fin)
- Operadores de troceamiento `[inicio:fin]`

[8]:
```
print("Elementos de tupla7 entre el 1 y el 3 (no incluido):", tupla7[1:3])
```

```
Elementos de tupla7 entre el 1 y el 3 (no incluido): (1, 2)
```

4 Empaquetado y desempaquetado

- Empaquetar significa guardar varios valores en una tupla/lista
- Desempaquetar es guardar los valores de una tupla o lista en variables (tiene sentido cuando trabajemos con funciones). Cada uno de los elementos de la tupla se copia en una variable. Necesito tantas variables como elementos tenga la tupla

[9]:
```
# Esto es empaquetar (crear una tupla o una lista usando variables)
a, b, c = 3, 66, False
tupla4 = (a, b, c)
# Desempaquetado, tengo que tener tantas variables como elementos
# tenga la tupla (si hay más o menos, salta un error)
# Copio cada valor de la tupla/lista en una variable
tupla9 = (1, 4, "hola", "adios")
a, b, c, d = tupla9
print("tupla9 es", tupla9)
print("a contiene tupla9[0]:", a)
print("b contiene tupla9[1]:", b)
print("c contiene tupla9[2]:", c)
print("d contiene tupla9[3]:", d)
```

```
tupla9 es (1, 4, 'hola', 'adios')
a contiene tupla9[0]: 1
b contiene tupla9[1]: 4
c contiene tupla9[2]: hola
d contiene tupla9[3]: adios
```

5 Listas anidadas

- Una lista/tupla dentro de otra lista/tupla (en otros lenguajes se conocen como matrices)

[6]:
```
# Lista anidada: es una lista que tiene dentro 3 listas
mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9, 10]]
print("la lista de listas mat es", mat)
# Con esto imprimo el 8 de la tercera matriz: del elemento 2, dime
# el elemento 1
print(mat[2][1])
# Con esto cambio el 4 por un "hola": del elemento 1, cambio el 0
mat[1][0] = 'hola'
print("la lista de listas mat es", mat)
# Esto es una lista, cuyos elementos son listas,  y un elemento de
# su primer elemento es también una lista
mat2 = [[1, 2, 3, [11, 12]], [4, 5, 6], [7, 8, 9, 10]]
print("la lista de listas mat2 es", mat2)
# Para cambiar el 12 por otra cosa
mat2[0][3][1] = "adios"
print(mat2)
# Añado un 14 a la lista. Ahora tengo una lista cuyos tres primeros
# elementos son listas y el cuarto es un único valor,
# no es una lista
mat2.append(14)
print("la lista de listas mat2 es", mat2)
# Añado un elemento a la sublista de la primera sublista
mat2[0][3].append(13)
print("la lista de listas mat2 es", mat2)
# Las funciones y métodos trabajan con la lista principal
# no entran en las sublistas
print("la longitud de mat2 es", len(mat2))
print("¿Hay un 1 en mat2?", 1 in mat2)
print("¿Hay un 1 en la primera sublista de mat2?", 1 in mat2[0])
print("¿Hay un 11 en la primera sublista de mat2?", 11 in mat2[0])
print("¿Hay un 11 en el último elemento de la primera "
      + "sublista de mat2?", 11 in mat2[0][-1])
```

```
la lista de listas mat es [[1, 2, 3], [4, 5, 6], [7, 8, 9, 10]]
8
la lista de listas mat es [[1, 2, 3], ['hola', 5, 6], [7, 8, 9, 10]]
la lista de listas mat2 es [[1, 2, 3, [11, 12]], [4, 5, 6], [7, 8, 9, 10]]
[[1, 2, 3, [11, 'adios']], [4, 5, 6], [7, 8, 9, 10]]
la lista de listas mat2 es [[1, 2, 3, [11, 'adios']], [4, 5, 6], [7, 8, 9, 10],
14]
la lista de listas mat2 es [[1, 2, 3, [11, 'adios', 13]], [4, 5, 6], [7, 8, 9,
10], 14]
la longitud de mat2 es 4
¿Hay un 1 en mat2? False
```

```
¿Hay un 1 en la primera sublista de mat2? True
¿Hay un 11 en la primera sublista de mat2? False
¿Hay un 11 en el último elemento de la primera sublista de mat2? True
```

5.1 Copia de listas anidadas

- Si utilizo el signo `=` no estoy copiando sino que las dos variables son la misma lista
- Si utilizo `copy()` o troceamiento o bucles `for` como con listas normales, las listas no son la misma pero los elementos sí lo son. A esto se le llama copia superficial porque no entra a copiar cada elemento de la sublista sino que la copia entera.
- Si queremos hacer una copia profunda:
  - Uso la librería `copy`: `import copy` luego `copy.deepcopy()`
  - Lo hago con bucles anidados (y si tengo más de un nivel de anidamiento tengo que usar recursividad)

[10]:
```python
lista1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9, 10]]
# Ahora las dos son la misma (no debería usarlo)
lista2 = lista1
# Puedo comprobarlo con los id
print("El id de lista2 es", id(lista2))
print("El id de lista1 es", id(lista1))
# O con el operador is
print("¿Son lista1 y lista2 la misma lista?", lista1 is lista2)
# Uso copy en su lugar: esto es como hacer lista2[0] = lista1[0]
# lista2[1] = lista1[1] y así sucesivamente
lista2 = lista1.copy()
# Puedo comprobarlo con los id
print("El id de lista2 es", id(lista2))
print("El id de lista1 es", id(lista1))
# O con el operador is
print("¿Son lista1 y lista2 la misma lista?", lista1 is lista2)
# Cambio un elemento de una sublista en lista2
lista2[0][0] = "hola"
print("lista2 es", lista2)
print("lista1 es", lista1)
print("¿Son la primeras sublistas de lista1 y lista2 la misma?", lista1[0] is␣
 ↪lista2[0])
# Aunque cada una de las sublistas sean iguales, las listas
# madre no lo son, si añado algo a una, no se añade a la otra
lista2.append(13)
print("lista2 es", lista2)
print("lista1 es", lista1)

# Usamos un bucle anidado para hacer copia profunda
# Solo funciona bien si las sublistas no contienen listas a su vez
# En el próximo tema veremos cómo hacerlo independientemente del
# nivel de anidamiento
```

```
lista2 = []
for item in lista1:
    # Si el elemento no es una lista, lo añadimos directamente
    if not type(item) == list:
        lista2.append(item)
    else:
        # Si es una lista, primero añadimos una lista vacía
        lista2.append([])
        # Y ahora copiamos la sublista (-1 es el último
        # elemento de lista2)
        lista2[-1] = item.copy()

# Ahora ya no son la misma
print("lista1 es", lista1)
print("lista2 es", lista2)
lista2[0][0] = "no cambia"
print("lista1 es", lista1)
print("lista2 es", lista2)
```

```
El id de lista2 es 102110392
El id de lista1 es 102110392
¿Son lista1 y lista2 la misma lista? True
El id de lista2 es 102064616
El id de lista1 es 102110392
¿Son lista1 y lista2 la misma lista? False
lista2 es [['hola', 2, 3], [4, 5, 6], [7, 8, 9, 10]]
lista1 es [['hola', 2, 3], [4, 5, 6], [7, 8, 9, 10]]
¿Son la primeras sublistas de lista1 y lista2 la misma? True
lista2 es [['hola', 2, 3], [4, 5, 6], [7, 8, 9, 10], 13]
lista1 es [['hola', 2, 3], [4, 5, 6], [7, 8, 9, 10]]
lista1 es [['hola', 2, 3], [4, 5, 6], [7, 8, 9, 10]]
lista2 es [['hola', 2, 3], [4, 5, 6], [7, 8, 9, 10]]
lista1 es [['hola', 2, 3], [4, 5, 6], [7, 8, 9, 10]]
lista2 es [['no cambia', 2, 3], [4, 5, 6], [7, 8, 9, 10]]
```

6 Tuplas de listas

- Una tupla cuyo contenido son listas
- Dado que la tupla guarda los punteros a las listas y no las listas en sí, se pueden modificar las listas. Pero no se pueden añadir o quitar listas de la tupla.

[1]:
```
# Una tupla de listas
tupla = ([1, 2], ["hola", 3, 4])
print("La longitud del primer elemento de la tupla es", len(tupla[0]))
print("El id the tupla es", id(tupla))
# Añadimos algo al elemento 0 de la tupla (que es una lista)
tupla[0].append(5)
```

```
print("la tupla es", tupla)
print("El id the tupla es", id(tupla), "es la misma tupla")
# Cambiamos algo en el elemento 1 de la tupla
tupla[1][0] = "adios"
print("la tupla es", tupla)
# Borramos una de las listas
tupla[0].clear()
print("la tupla es", tupla)
```

```
La longitud del primer elemento de la tupla es 2
El id the tupla es 99694912
la tupla es ([1, 2, 5], ['hola', 3, 4])
El id the tupla es 99694912 es la misma tupla
la tupla es ([1, 2, 5], ['adios', 3, 4])
la tupla es ([], ['adios', 3, 4])
```

7 Diccionarios

- Es un conjunto finito, heterogéneo, mutable de datos **indexados por clave**
- La clave es generalemente un str, pero podría ser cualquier tipo inmutable (int, flotante, tupla, etc.)
- `dic = {clave: valor, clave2: valor2, clave3: valor3}`
- Para acceder uso `dic[clave]`, no puedo usar el orden en que están definidas, no hay concepto de orden en un diccionario
- Parecido a los registros de otros lenguajes. Otros nombres: tablas asociativas, tablas de hash, arrays asociativos
- Si declaro dos claves iguales, el valor de la segunda borra al vlaor de la primera
- Si intento obtener el valor de una clave que no existe, salta un error
- Para introducir nuevas claves, les doy valor: `dic[nueva_clave] = valor`
- Para borrar una clave uso `del(dic[clave])`

[3]:
```
# Hay dos claves apellido, la segunda borra a la primera
dic1 = {"nombre": "Pepe", "apellido": "Pérez", "edad": 18,
        "bebe": True, "apellido": "Castro"}
print("El valor de la clave 'nombre' es", dic1["nombre"])
dic1["edad"] = 22
print("dic1 es", dic1)
print("El valor de la clave 'apellido' es", dic1["apellido"])
# Esto daría error porque esta clave no existe
# print(dic1["esta clave no existe"])
# Creo una nueva clave (ojo porque podría crear claves sin querer)
dic1["peso"] = 65
print("dic1 es", dic1)
# Para borrar una clave
del(dic1["edad"])
print("dic1 es", dic1)
# Creo un diccinario vacío
```

```
dic2 = {}
print("dic2 es", dic2)
```

```
El valor de la clave 'nombre' es Pepe
dic1 es {'nombre': 'Pepe', 'apellido': 'Castro', 'edad': 22, 'bebe': True}
El valor de la clave 'apellido' es Castro
dic1 es {'nombre': 'Pepe', 'apellido': 'Castro', 'edad': 22, 'bebe': True,
'peso': 65}
dic1 es {'nombre': 'Pepe', 'apellido': 'Castro', 'bebe': True, 'peso': 65}
dic2 es {}
```

7.1 Operadores, funciones y métodos con diccionarios

- `in` y `not in` me dicen si un diccionario contiene una clave
- `len(dic)`: devuelve el número de claves
- `del(dic[clave])`: borra esa clave
- No se puede usar `+` para unir dos diccionarios, tampoco `=` para copiarlos (mismo problema que con listas)
- No puedo usar troceamiento (slicing)
- `dic1.update(dic2)`: añade a `dic1` el diccionario `dic2` (copia superficial). Si hay claves comunes, toman el valor de `dic2`
- `.copy()`: crea una copia del diccionario (copia superficial)
- `.clear()`: borra claves y valores dejando el diccionario vacío
- `.get(clave)`: devuelve el valor de esa clave y si no existe la clave devuelve `None`. Es mejor usar `get` que `dic[clave]`
- `.keys()`: devuelve una pseudo-lista con todas las claves
- `.values()`: devuelve una pseudo-lista con todos los valores
- `.items()`: devuelve una pseudo-lista con tuplas (clave, valor)
- `.pop(clave)`: devuelve el valor de esa clave y borra la clave y el valor

[4]:
```
# Veo si u na clave está en el diccionario
print("nombre" in dic1)
print("El número de claves es", len(dic1))
dic2 = {"nombre": "Antonia", "notas": [5, 5, 7]}
dic1.update(dic2)
print("dic1 es", dic1)
print("dic2 es", dic2)
# Es copia superficial
dic2["notas"][0] = 8
print("dic1 es", dic1, "también ha cambiado")
print("uso get para una clave que no existe",
      dic1.get("clave que no existe"))
# Lista con todas las claves
claves = list(dic1.keys())
print("claves de dic1:" ,claves)
# Borro el dic2
dic2.clear()
print("dic2 ha sido borrado:", dic2)
```

```python
# Todos los valores
valores = list(dic1.values())
print("valores de dic1:", valores)
# Claves y valores
todos = list(dic1.items())
print("claves y valores de dic1", todos)
# Borra la clave nombre y devuelve su valor que se guarda en n
n = dic1.pop("nombre")
print("El valor de la clave 'nombre' de dic1 era", n)
print("dic1 es", dic1)
```

```
True
El número de claves es 4
dic1 es {'nombre': 'Antonia', 'apellido': 'Castro', 'bebe': True, 'peso': 65,
'notas': [5, 5, 7]}
dic2 es {'nombre': 'Antonia', 'notas': [5, 5, 7]}
dic1 es {'nombre': 'Antonia', 'apellido': 'Castro', 'bebe': True, 'peso': 65,
'notas': [8, 5, 7]} también ha cambiado
uso get para una clave que no existe None
claves de dic1: ['nombre', 'apellido', 'bebe', 'peso', 'notas']
dic2 ha sido borrado: {}
valores de dic1: ['Antonia', 'Castro', True, 65, [8, 5, 7]]
claves y valores de dic1 [('nombre', 'Antonia'), ('apellido', 'Castro'),
('bebe', True), ('peso', 65), ('notas', [8, 5, 7])]
El valor de la clave 'nombre' de dic1 era Antonia
dic1 es {'apellido': 'Castro', 'bebe': True, 'peso': 65, 'notas': [8, 5, 7]}
```