TEMA 4

Listas, Tuplas y Diccionarios

Curso de Python

Alfredo Sánchez Sánchez
@alfsan04
www.educoteca.com

ÍNDICE

Listas

Imagina que queremos almacenar una cantidad algo grande de variables, por ejemplo, los nombres de un grupo de personas. Podríamos guardarlos como *nombre1 = ... , nombre2 = …. , nombre3 = …* o bien podríamos buscar si hay algún tipo de elemento que nos permita almacenar una serie de datos juntos bajo un mismo nombre.

Existe una estructura para almacenar un conjunto de datos, llamada **lista**, vamos a aprender a usarla.

Crear una lista es muy sencillo. Sólo tenemos que darle un nombre e indicar a Python que dicho nombre se corresponde con una lista. Podemos declararla con contenido...

```
nombres = ["Pedro", "Juan", "Ana"]
```

… o declararla vacía...

```
nombres = []
```

… en cuyo caso tendremos que introducir el contenido de la misma posteriormente.

Una vez hemos declarado la lista podemos introducir nuevos elementos en la lista de diversas maneras, de momento vamos a usar una función de las listas que introduce un nuevo elemento al final de la misma:

```
nombres = ["Pedro", "Juan", "Ana"]
nombres.append("Maria")
```

El comando *append* sitúa el nuevo elemento tras los ya presentes en la lista. Si no hubiese ninguno sería el nuevo elemento sería el primero. Para incluirlo hay que hacerlo nombrando la lista y un punto previo a *append*. Veamos el código anterior ejecutado por terminal:

```
>>> nombres = ["Pedro", "Juan", "Ana"]
>>> nombres.append ("Maria")
>>> print (nombres)
['Pedro', 'Juan', 'Ana', 'Maria']
>>>
```

Si observas, el nombre introducido en la lista (*"Maria"*) aparece en último lugar al imprimir la lista.

A la acción de modificar una lista cambiando alguno o todos sus elementos se le denomina **mutar una lista**.

Una vez que tenemos una lista creada podemos acceder a cualquiera de los elementos que contiene indicando el índice que ocupa, siendo el 0 el primer elemento, el 1 el segundo y así sucesivamente:

```
nombres = ["Pedro", "Juan", "Ana", "Maria"]
print(nombres[1])
```

```
>>> nombres = ["Pedro", "Juan", "Ana"]
>>> nombres.append ("Maria")
>>> print (nombres)
['Pedro', 'Juan', 'Ana', 'Maria']
>>> print(nombres[1])
Juan
>>>
```

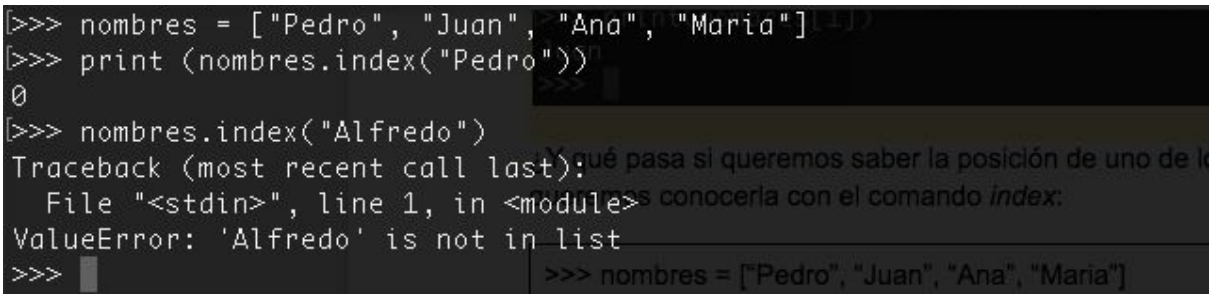
¿Y qué pasa si queremos saber la posición de uno de los nombres? Basta con indicar que queremos conocerla con el comando index:

```
nombres = ["Pedro", "Juan", "Ana", "Maria"]
print(nombres.index("Pedro"))
```

```
>>> nombres = ["Pedro", "Juan", "Ana", "Maria"]
>>> print (nombres.index("Pedro"))
0
>>>
```

Trabajando por terminal o IDLE bastaría con poner *nombres.index("Pedro")*, no necesitamos imprimirlo para que muestre el valor que ocupa el elemento "Pedro".

Si intentamos mostrar la posición de un elemento que no está en la lista obtendremos un error que nos indica que el elemento no está en la lista.



Vamos a usar una lista para hacer un programa. Prueba el siguiente código guardándolo en un archivo .py

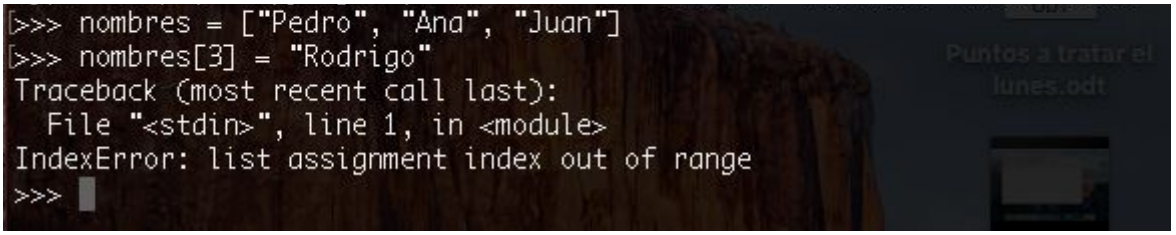
```
nombres = []
cantidad_nombres = input("¿Cuántos nombres quieres introducir en la
lista?: ")
cantidad_nombres = int(cantidad_nombres)
for numero in range(cantidad_nombres):
    numero_texto = str(numero + 1)
    nombre_elegido = input("Nombre número "+numero_texto+": ")
    nombres.append(nombre_elegido)
print(nombres)
```

¿Ya has visto cómo funciona? La mayoría de los elementos ya han sido explicados en los temas pasados (bucle for, input, int, print…). La idea es poder almacenar el número de nombres que el usuario desee. Para mejorar el script habría que añadir algún código para que el usuario sólo pueda introducir un número al principio para indicar cuántos nombres tendrá la lista, evitando así el error.

Una vez tenemos creada una lista podemos sustituir cualquiera de sus elementos por uno nuevo simplemente declarando en qué lugar queremos introducir el nuevo elemento. Si declaro un nuevo contenido para uno de los índices de la lista, éste sustituirá al contenido existente.

En el ejemplo anterior se sustituye el índice 1 (segundo elemento) de una lista por un nuevo contenido y se muestra cómo, efectivamente, ha variado.

Si no hubiese ningún contenido en ese índice (y por tanto, no existiese), Python nos dará error:



Al intentar guardar un nuevo elemento (*"Rodrigo"*) en la lista nombres obtenemos un error, pues no existe el índice o posición 3 de la misma. Sí, hay tres elementos, pero el primer elemento es el índice 0, el segundo es el índice 1 y el tercero es el índice 2.

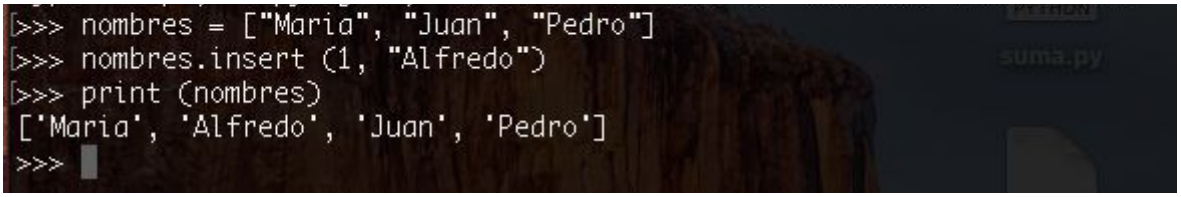
Para obtener el número de elementos que contiene una lista no tenemos más que usar el comando *len* igual que con los strings.

Podemos mutar una lista introduciendo un nuevo valor sin cambiar ninguno de los que ya contenía. Para ello podemos decirle a Python dónde queremos insertar el nuevo elemento y el programa desplazará todos los que existían de ese índice en adelante una posición.

Para insertar un valor debemos usar la función *insert* e indicar, entre paréntesis, en qué posición queremos incluir un valor y cual es el valor que queremos introducir. Al igual que con *append* se debe unir al nombre de la lista separándolos con un punto:

```
nombres = ["Maria", "Juan", "Pedro"]
nombres.insert(1, "Alfredo")
```

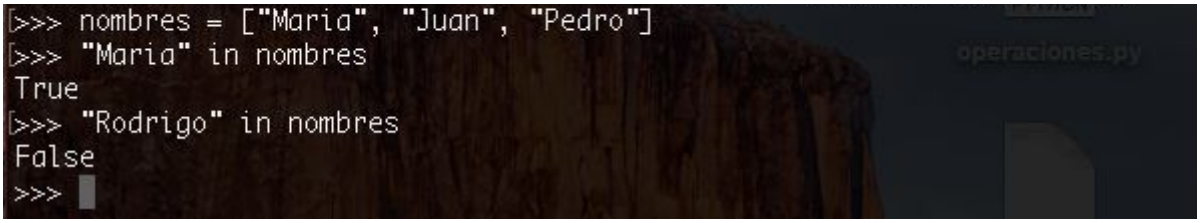
Veamos cómo funciona el código anterior:



Como ves, el índice 1 se encuentra ocupado por Alfredo pero el resto de elementos continúan en la lista.

De la misma forma podemos eliminar un elemento desplazando todos los siguientes una posición a la izquierda con el comando *remove*. A diferencia del comando *insert*, con *remove* tengo que indicar qué elemento quiero eliminar:

También podemos comprobar si un elemento está dentro de una lista:



Esta estructura se puede usar dentro de cualquier condicional o bucle:

```
if "Maria" in nombres:
   print ("Maria está en la lista nombres")
```

Una lista puede contener cualquier tipo de valores dentro: números decimales, números enteros, strings e incluso otras listas.

Por último vamos a ver cómo **ordenamos una lista** en orden creciente o decreciente. Para ordenar una lista los valores que contiene deben ser del mismo tipo, si no Python no sabrá cómo ordenar los elementos que contiene (por ejemplo strings y números enteros).

Para ordenar una lista usaremos el comando *.sort( )* como apéndice del nombre de la lista:

```
nombres = ["María", "Pedro", "Alfredo", "Juan"]
nombres.sort()
print(nombres)
```

Veamos cómo se ejecuta el código anterior y cómo, efectivamente, ordena una lista:

```
>>> nombres = ["María", "Pedro", "Alfredo", "Juan"]
>>> print (nombres)
['María', 'Pedro', 'Alfredo', 'Juan']
>>> nombres.sort()
>>> print (nombres)
['Alfredo', 'Juan', 'María', 'Pedro']
>>>
```

En la primera impresión de la lista los nombres no están ordenados. En la segunda aparecen ordenados alfabéticamente.

Para ordenarlos de forma descendente o en alfabeto inverso debemos usar un parámetro en la función *sort( )* que no es más que activar el modo inverso:

```
nombres = ["María", "Pedro", "Alfredo", "Juan"]
nombres.sort(reverse = True)
print(nombres)
```

Podemos ver cómo se ordenan de forma inversa en la siguiente imagen:

```
>>> nombres.sort(reverse = True)
>>> print (nombres)
['Pedro', 'Maria', 'Juan', 'Alfredo']
>>>
```

De momento vamos a dejar aquí las listas, aunque seguiremos usándolas en otros temas.

Tuplas

Las tuplas son un tipo de lista un poco especial, no mutable (es decir, no se puede modificar lo que contienen). Se podría decir que son iguales que las listas pero no podemos alterar su contenido. Las tuplas se declaran con paréntesis (recuerda que en las listas se usaban corchetes).

```
nombres = ("Ana", "Maria", "Rodrigo")
```

Para leer el contenido de la tupla podemos acceder por el índice que ocupa aquello que necesitemos de la misma, recordando que el primer elemento es el 0. Se escribe igual que una lista:

```
nombres = ("Ana", "Maria", "Rodrigo")
print(nombres[1])
```

No vamos a profundizar en ellas en este tema, pero es bueno conocer su existencia pues se usan en diferentes ocasiones, especialmente cuando necesitamos hacer comparativas entre diferentes tuplas de forma que los términos de la misma sean inmutables.

Un ejemplo de lo anterior son los vectores, un vector siempre tiene un par de coordenadas (x, y) y si queremos comparar vectores necesitamos que la coordenada x sea la primera y la y la segunda, por ello se usan tuplas.

También serviría de ejemplo estructuras que almacenen una fecha (día, mes y año) o una hora (horas, minutos y segundos). Siempre tendrán el mismo formato y no queremos que cambien bajo ningún concepto de orden.

Por ahora basta con saber que la tupla es una lista que no se puede modificar.

Diccionarios

Los diccionarios son una estructura que almacena elementos a los que se les da un nombre o clave (en inglés key). De esta forma, en lugar de acceder al elemento por medio del índice que ocupa se accede a él por el nombre que tiene. Podemos definir un diccionario como un conjunto de parejas clave - valor (key - value) donde la clave es el nombre que se le da a un valor.

Los valores de un diccionario pueden tener cualquier forma: números, strings, listas... Veamos un diccionario sencillo:

```
pronombres = {"yo" : "I", "tú" : "You", "él" : "He"}
```

Como puedes ver, el diccionario se crea usando llaves y dentro de ellas situaremos cada pareja clave - valor con dos puntos entre la clave y el valor, utilizando una coma para separar cada pareja.

Para acceder a cada valor debemos escribir el siguiente código:

```
pronombres = {"yo" : "I", "tú" : "You", "él" : "He"}
print (pronombres["yo"])
```

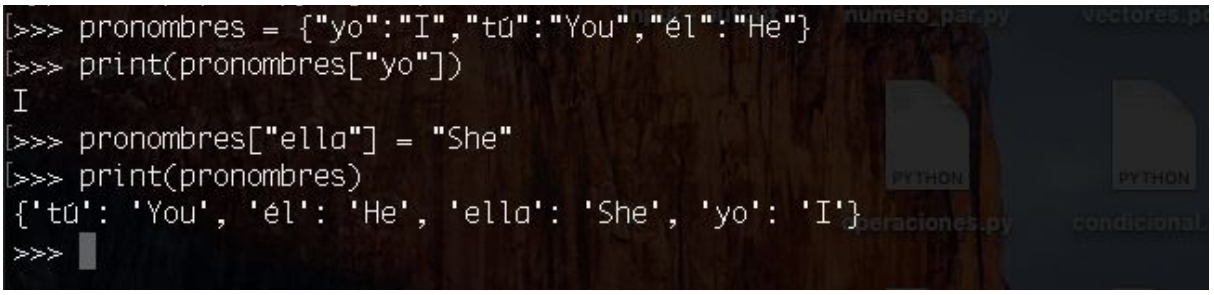
Usamos la clave del diccionario para mostrar el valor:

```
>>> pronombres = {"yo":"I","tú":"You","él":"He"}
>>> print(pronombres["yo"])
I
>>>
```

Para añadir nuevas parejas clave - valor al diccionario sólo tenemos que indicarlo de la siguiente forma:

```
pronombres["ella"] = "She"
```

Veamos cómo, con el comando anterior, la clave se ha añadido imprimiendo el diccionario:

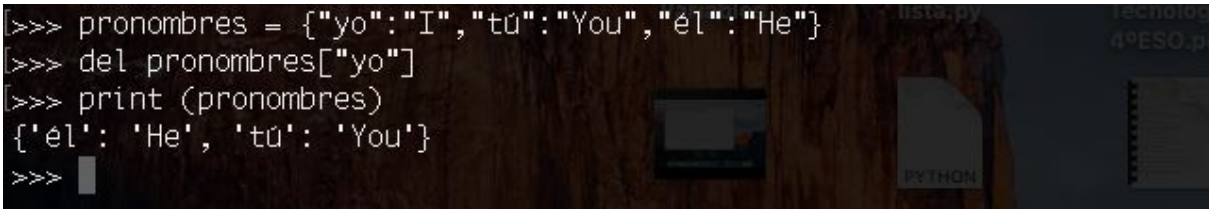


Como puedes observar se ha añadido la clave *"ella"* con el valor *"She"* aunque al imprimir el diccionario vemos que los elementos que lo componen están desordenados.

En realidad ordenar un diccionario es complicado y no tiene sentido hacerlo ya que se trata de una estructura de almacenamiento a la que accederemos mediante una clave para que Python muestre el valor asociado. La posición que ocupa la clave no nos interesa demasiado.

Así mismo, podemos eliminar cualquier clave - valor del diccionario con la siguiente instrucción:

```
pronombres = {"yo" : "I", "tú" : "You", "él" : "He"}
del pronombres["Yo"]
```



Como ves la clave *"Yo"* de la lista mostrada en la imagen se ha borrado con el comando *del*.

Recorriendo listas y diccionarios

Algo habitual es querer recorrer los elementos de una lista o diccionario para compararlos con un valor o mostrarlos. Para ello usaremos un bucle for.

Empecemos con las listas. Prueba el siguiente código guardándolo en un script y abriéndolo por terminal o en la IDLE:

```
notas = [4, 3, 2, 7, 6, 8, 5, 10, 3, 8]
suspensos = 0
aprobados = 0
for nota in notas:
   if nota < 5:
      suspensos = suspensos + 1
   elif nota >= 5:
      aprobados += 1
suspensos = str (suspensos)
aprobados = str (aprobados)
print ("Hay "+suspensos+" suspensos.")
print ("Hay "+aprobados+" aprobados.")
```

En el for se declara una variable denominada *nota* que almacenará un elemento de la lista *notas* en cada iteración. En la primera será un 4, luego será un 3, luego un 2, luego un 7 y así sucesivamente.

Fíjate en las dos formas en las que se incrementan las variables suspensos y aprobados. Ambas hacen exactamente lo mismo, podemos incrementar una variable asignándole su valor + 1 o podemos incrementarla con la expresión += 1, significan exactamente lo mismo.

Una vez que probéis el código debería salir lo siguiente:

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 notas.py
Hay 4 suspensos.
Hay 6 aprobados.
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

Ahora te toca practicar a tí. Añade una instrucción para que el código anterior, además de darnos los aprobados y los suspensos, nos dé el porcentaje de aprobados y la nota media. Para la nota media tendremos que crear una nueva variable que almacene la suma de todos los elementos (cosa que podemos hacer al irse ejecutando el for) y el porcentaje de

aprobados se puede realizar al final introduciendo una nueva variable que almacene la fórmula que es necesaria para calcular el porcentaje pedido.

¿Conseguido? Perfecto.

Vamos a ver ahora cómo recorremos un diccionario:

La primera forma de recorrer un diccionario es recorriendo las claves.

```
print ("Vamos a traducir algunos pronombres al inglés:")
pronombres = {"yo" : "I", "tú" : "You", "él" : "He", "ella" : "She"}
for clave in pronombres:
   print ("La traducción de "+clave+" es: "+pronombres[clave])
```

Pruébalo guardándolo como script antes de seguir.

El código utiliza la clave del diccionario tanto para imprimirla tal cual como para buscar el valor asociado e imprimirlo.

Como habrás podido comprobar, el código anterior nos muestra cada uno de los elementos de la lista.

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 recorrer_diccion
ario.py
Vamos a traducir algunos pronombres al inglés:
La traducción de yo es: I
La traducción de tú es: You
La traducción de ella es: She
La traducción de él es: He
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

También podemos recorrer el diccionario con la instrucción *.items( )* como apéndice del nombre del diccionario. Esta instrucción nos permite almacenar tanto la clave como el valor en una variable. Veamos el mismo código anterior con esta instrucción, pruébalo guardándolo en un script:

```
print ("Vamos a traducir algunos pronombres al inglés:")
pronombres = {"yo" : "I", "tú" : "You", "él" : "He", "ella" : "She"}
for clave, valor in pronombres.items():
   print ("La traducción de "+clave+" es: "+valor)
```

Como verás, hace exáctamente lo mismo que el código anterior.

Hay muchas otras opciones con listas y diccionarios a la hora de recorrerlos y mostrarlos, pero de momento nos vamos a quedar aquí, es mejor afianzar esto antes de seguir profundizando en opciones más enrevesadas.

Instrucciones de código usadas

```
#Crear una lista
   lista = []
#Introducir un nuevo elemento en la lista
   lista.append(elemento)
#Acceder a un elemento de la lista
   lista[indice]
#Conocer la posición de un elemento de la lista
   lista.index(elemento)
#Insertar un elemento en la lista, desplazando los existentes
   lista.insert(indice,elemento)
#Borrar un elemento de una lista
   lista.remove(elemento)
#Comprobar si un elemento está en una lista
   elemento in lista
#Ordenar una lista alfabéticamente o de forma creciente
   lista.sort()
#Ordenar una lista de forma decreciente o alfabéticamente inversa
   lista.sort(reverse=True)
#Declarar una tupla
   tupla = ()
#Declarar un diccionario
   diccionario = {}
#Introducir un nuevo valor del diccionario mediante la clave
   diccionario[clave] = valor
#Borrar una clave-valor del diccionario mediante la clave
   del diccionario[clave]
#Comando para recorrer las parejas clave-valor de un diccionario
   for clave, valor in diccionario.items():
```

Reto 1

Crea un script que busque, hasta el número que el usuario indique, los números primos y los vaya imprimiendo. Para ello te doy las siguientes pistas:

- Un número primo sólo se puede dividir entre 1 y sí mismo, por lo que cualquier otra división dará decimales.
- Para saber si un número es primo basta con dividirlo entre todos los números primos menores que ese número, si el resultado de esas divisiones tiene resto (es decir, *% != 0*) ese número es primo.

Reto 2

Crea un script que te dé la traducción de un color al inglés, a petición del usuario. El script debe contener al menos 10 posibles traducciones que serán indicadas como posibilidades al usuario (naranja, rojo, amarillo…). Usa un diccionario para almacenar las traducciones.