Diccionarios

Unidad 4

Apunte de cátedra

Pensamiento computacional (90)
Cátedra: Camejo

.UBAXXI

Diccionarios

Las **listas** son Estructuras de Datos extremadamente flexibles. Sabemos que podemos acceder **directamente** a cualquier elemento. Como son secuencias, basta con referenciar a un elemento en una **posición** determinada y podremos dar un salto hasta él. Ahora, si precisamos un dato específico, y no sabemos dónde se encuentra, **primero debemos averiguar su posición y luego acceder a él**. En muchas ocasiones, cuando sabemos con precisión cuál es el objeto buscado se puede usar la ayuda de algún método de búsqueda sobre lista para obtener su dirección. El método, de manera muy eficiente, hará de todos modos lo que nosotros haríamos por nuestra cuenta; es decir, inspeccionar una por una las posiciones de la lista hasta detectar el objeto buscado. Esto es irremisiblemente así porque en una lista, el lugar donde se guarda un objeto no tiene nada que ver con el valor del mismo. El nombre 'ana' puede ir en cualquier lugar de la lista.

Y es fácil imaginar que serán muchas las situaciones donde la información de acceso que tengamos será por **contenido** y no por **posición**. Para facilitar este tipo de trabajos (o deberíamos decir, para acelerar más este tipo de procesos), Python provee una clase de Estructura de Datos que es ideal a tales efectos: Los **diccionarios**.

Un **diccionario** (tipo **dict**) es una Estructura que permite **mapear** un valor (**clave**) con un grupo de datos. De este modo, si precisamos acceder a un valor en particular, no necesitamos indicar su posición (de hecho, no la sabremos), sólo indicando su valor saltamos directamente a él. Es decir, empleamos el dato como **clave de acceso.**

Un **diccionario** ubica sus elementos en memoria de acuerdo al resultado que obtiene al aplicar una función de cálculo a su **clave**. De ahí su denominación de **Estructura Hashable** o con acceso **hash**; haciendo referencia al nombre que se le da a los algoritmos de cálculo de direcciones para acceso directo a la información (**Algoritmos Hash**).

Esa característica vuelve muy eficiente la localización de un determinado dato; porque no se resolverá inspeccionando una a una las posiciones de la Estructura, se obtendrá de manera directa (o casi) su posición por medio del cálculo de la función sobre el dato de búsqueda. Cuando decimos "casi" estamos haciendo referencia a que muy probablemente exista más de una **clave** que devuelva la misma posición como resultado de la función. En ese caso lo que se hace es almacenar los datos en "**contenedores**" unificados por el valor **hash**. Y eventualmente habrá una búsqueda secuencial, pero estrictamente dentro del contenedor; cosa que disminuye drásticamente, de todas formas, el tiempo de búsqueda.

La realidad es que una búsqueda secuencial (como las que hacemos en una lista) se vuelve más y más ineficiente a medida que el tamaño de la misma aumenta; mientras que el tiempo requerido para realizar la misma localización en un diccionario tiende a permanecer constante, sin importar su tamaño.

Un diccionario Python se escribe como una colección de pares (separados por ':') entre corchetes

Ejemplos:

```
a={1:1,2:4,3:6,4:8,5:10}

b={}

c={'ana':27,'juan':15,'elena':1}

d={345:('clavos',1),346:('tuercas',1),202:('manguera',2)}
```

Y para referenciar a cualquier valor vinculado a una clave sólo se requiere indicar la clave en lugar de la posición. Así:

`print(a[3])` devolverá `6`

`print(b['yo'])` devolverá `error`

`print(c['elena'])` devolverá `1`

`print(d[202],d[345][0])` devolverá `('manguera',2) clavos`

Por supuesto disponemos de **in** y **len()** para detectar si una determinada clave existe o un diccionario está vacío.

El tipo **dict** es **mutable**. Podemos agregar o quitar elementos. Para sumar un elemento no necesitamos hacer lugar como en las listas. Simplemente asignamos el par **clave:valor** y listo!
*Pero si la **clave** ya existía, el **valor** anterior es pisado*

Atención!

Las claves de un diccionario pueden ser números, booleanos, strings, tuplas y cualquier tipo de Python que NO admita cambios (inmutable). No pueden cambiar, porque si no, cambiaría el resultado de la función hash aplicada a ellas y se debería reubicar la información.
Por contraposición, los datos o valores vinculados a una clave pueden ser de cualquier tipo simple o estructurado (listas y diccionarios quedan incluídos acá)
Un diccionario admite heterogeneidad de tipos de claves (siempre que no puedan cambiar) y heterogeneidad de valores. El siguiente ejemplo es perfectamente válido:

Código	**Salida**

`#prueba de heterogeneidad`
`d={1:'yo','yo':2,'ella':(1,2,3)}`
`print(d[1],d['ella'][0],d['yo'])`
	`yo 1 2`
	`>>>`

Métodos para Diccionarios:

d.clear() → Elimina todos los elementos de d

d.pop(clave) → Remueve el par clave,valor del diccionario y devuelve valor. Si la clave no está da error

d.popitem() → Remueve y devuelve cualquier par clave,valor. Si d está vacío da error d.copy() devuelve una copia de d en otra región de memoria

d.fromkeys(secuencia) → Crea el diccionario d, tomando las claves de una secuencia d.get(clave) devuelve el valor asociado a clave

d.items() → Devuelve una lista con tuplas. Cada tupla tiene elemento 0 clave; posición 1, valor

d.keys() → Devuelve una lista con las claves de d

d.update(b) → Agrega los pares clave,valor de b a d. Si alguna clave existe actualiza su valor

Los usos típicos de un diccionario son los **mapeos de información.** Por ejemplo, las traducciones. Miremos un ejemplo empleando de manera combinada dos métodos de string (**maketrans**() y **translate**()) que permiten cambiar caracteres por otros y utilizan un diccionario para establecer el mapeo entre ellos.

Código	**Salida**

`def sacaAcen(t):`
`  con='áéíóú'`
`  sin='aeiou'`
`  #Arma el diccionario traductor`
`  traductor=t.maketrans(con,sin)`
`  '''Aplica la traducción con el`
`  Mapeo de traductor`
`  á será cambiada por a, é por e '''`
`  return t.translate(traductor)`
 	
`#PPal`
`txt=input('Ingresá un texto: ')`
`txt=sacaAcen(txt.lower())`
`print('Texto sin acentos')`
`print(txt.capitalize())`
	`Ingresá un texto: LA INFORMACIÓN pÚblica correcta para ese envío no debería haber sido más que una transacción.`
	`Texto sin acentos`
	`La informacion publica correcta para ese envio no deberia haber sido mas que una transaccion.`
	`>>>`

También es lógico pensar en ellos cuando queremos guardar información de diferentes entidades para acceder directamente por su clave de guardado. Por ejemplo: información de alumnos de una materia (nombre, notas, condición, etc.) para acceder por legajo; información de libros accesible por isbn; o lista de películas accesibles por categoría.

Acá tenemos un ejemplo de mapeo de información organizada por clave. En este caso se arma un diccionario con datos por cada código de producto de una tienda (descripción, unidad de medida,precio

unitario) y luego se accede directamente al precio unitario de un producto usando su código para calcular una venta:

```
#facturación de Productos
productos={}
cgo=int(input('Ingrese código, 0 para terminar '))
while cgo!=0:
      if cgo not in productos:
            desc=input('Descripción de %d '%cgo)
            unidad=input('Unidad de Medida de %s '%desc)
            precio=float(input('Precio unitario de %s '%desc))
            productos[cgo]=(desc,unidad,precio)
      cgo=int(input('Ingrese código, 0 para terminar '))
for cgo in productos:
      print(cgo,*productos[cgo])
total=0
cgo=int(input('Qué lleva? 0 para salir '))
while cgo not in productos and cgo!=0:
      cgo=int(input('Qué lleva? 0 para salir '))
while cgo!=0:
      cant=float(input('Cantidad de %s '%(productos[cgo][0])))
      total+=cant*productos[cgo][2]
      cgo=int(input('Qué lleva? 0 para salir '))
      while cgo not in productos and cgo!=0:
            cgo=int(input('Qué lleva? 0 para salir '))
print('Debe abonar: $%.2f'%total,sep='')
```

Una salida ejemplo:

```
11 velitas unid 32.5
25 azúcar glass gr 0.65
33 dulce de leche kg 85.7
12 blonda unid 22.75
Qué lleva? 0 para salir 10
Qué lleva? 0 para salir 11
Cantidad de velitas 8
Qué lleva? 0 para salir 12
Cantidad de blonda 3
Qué lleva? 0 para salir 0
Debe abonar: $328.25
>>>
```

Veremos cómo armar menú de opciones a partir de un diccionario. Ya que resulta más práctico que las tuplas para esto.
Tenemos un ejemplo de un menú de opciones de dificultad en un programa empleando una tupla:

```
#traducción de opciones de menú usando tuplas
dificultad=('','Alta','Media','Baja')
platos=[]
nomPlato=input('Ingresá un plato, * para salir ')
while nomPlato!='*':
    opc=0
    while opc==0:
        print('Dificultad')
        for i in range(1,len(dificultad)):
            print('%d-'%i,dificultad[i])
        opc=int(input())
        if opc not in range(1,len(dificultad)):
            opc=0
    platos.append([nomPlato,dificultad[opc]])
    nomPlato=input('Ingresá un plato, * para salir ')
print('\nLista de Platos')
for p in platos:
    print(*p)
```

Salida:

```
Lista de Platos
Lasagna Rellena Alta
Bife a la Plancha Baja
Puré de Papas Media
Puré de Manzanas Media
Pollo a la Crema Alta
>>>
```

Ahora miremos usando diccionarios:

```
#traducción de opciones de menú usando diccionarios
dificultad={1:'Alta',2:'Media',3:'Baja'}
platos=[]
nomPlato=input('Ingresá un plato, * para salir ')
while nomPlato!='*':
    opc=0
    while opc==0:
        print('Dificultad')
        for i in dificultad:
            print('%d-'%i,dificultad[i])
        opc=int(input())
        if opc not in dificultad:
            opc=0
    platos.append([nomPlato,dificultad[opc]])
    nomPlato=input('Ingresá un plato, * para salir ')
print('\nLista de Platos')
for p in platos:
    print(*p)
```

¿Podemos ordenar un Diccionario?

Ya sabemos que un Diccionario organiza su información de acuerdo a la dirección obtenida al aplicar la función **Hash** a sus claves. O sea que no podemos decirle en qué posición irá cada una de ellas.

Normalmente cuando cargamos un diccionario y luego lo recorremos con un bucle, las claves aparecen en el orden en que fueron guardadas:

```
dicci={}
print('Datos de Clientes, * para terminar')
dni=input('Dni: ')
while dni!='*':
  nom=input('Nombre: ')
  ape=input('Apellido: ')
  edad=int(input('Edad: '))
  while edad not in range(18,130):
    edad=int(input('Edad: '))
  dicci[dni]=[nom,ape,edad]
  dni=input('Dni: ')
for pers in dicci:
  print(pers,dicci[pers][0],dicci[pers][1],dicci[pers][2])
```

Salida Ejemplo:

```
Datos de Clientes, * para terminar
Dni: 18023569
Nombre: Andrea
Apellido: Gómez
Edad: 55
Dni: 17895822
Nombre: Manuel
Apellido: Sánchez Ibarra
Edad: 55
Dni: 21589658
Nombre: Juan
Apellido: Gil
Edad: 48
Dni: *
18023569 Andrea Gómez 55
17895822 Manuel Sánchez Ibarra 55
21589658 Juan Gil 48
>>>
```

Pero ¿Y si en realidad queremos ver toda la información, pero ordenada por el valor de la clave?

No hay mucho estrafalario acá. En realidad, lo que podemos hacer es armar una lista de claves, ordenarla, y luego emplear esa lista para acceder a los datos del diccionario. Más viejo que FORTRAN pero igual de efectivo que él! Los viejos programadores llamamos a la lista ordenada: **Índice**. Y en el caso de Python, nos hace el servicio directo con **sorted**().

Ya que aplicando la función a un diccionario devuelve una lista con las claves ordenadas. Bingo!

El ejemplo anterior sería:

```
dicci={}
print('Datos de Clientes, * para terminar')
dni=input('Dni: ')
while dni!='*':
  nom=input('Nombre: ')
  ape=input('Apellido: ')
  edad=int(input('Edad: '))
  while edad not in range(18,130):
    edad=int(input('Edad: '))
  dicci[dni]=[nom,ape,edad]
  dni=input('Dni: ')
dicciOrden=sorted(dicci)
for pers in dicciOrden:
  print(pers,dicci[pers][0],dicci[pers][1],dicci[pers][2])
```

En este caso la salida será:

```
17895822 Manuel Sánchez Ibarra 55
18023569 Andrea Gómez 55
```
21589658 Juan Gil 48

Cuidado con los Espejitos de Colores!

Estás obteniendo una lista con todas las claves del diccionario (que claramente podrías haber obtenido por tu cuenta). Así que duplicas la información.

A tener en Cuenta!
Los diccionarios, como las listas, se comparten con las funciones de una manera diferente a las otras estructuras de datos que conocemos. Cuando enviamos una lista o diccionario a una función, no se copia en la región de memoria privada de la función (**ámbito** de la función), sino que se le cede a esta última la llave de acceso a la estructura original.
Consecuencias directas de este modus operandi (técnicamente se llama **Pasaje de Parámetros por Referencia**):

- ✔ No necesitamos devolver una lista o diccionario ya que las modificaciones que realizamos en la función permanecen afuera
- ✔ Estamos confiando en que la función modifique la estructura de manera segura
- ✔ Si necesitamos cargar una lista o un diccionario dentro de una función se lo podemos pasar vacío y se irá cargando afuera. Por el contrario, si no lo pasamos y se crea adentro es local a la función y necesitaremos si o sí devolverlo con un return para poder tener acceso una vez que la función termine
- ✔ Si no queremos compartir la estructura original estamos obligados a hacer la copia manualmente. Puede ser antes de pasar la estructura o dentro de la función, una vez recibida. Si este último fuera el caso, nuevamente aplica lo anterior. La estructura copiada es local a la función y debemos devolverla