Clase 2 - Listas y diccionarios

March 25, 2017

1 Introducción

En esta segunda clase resolvemos los ejercicios planteados en la primera y atendemos las posibles dudas al respecto, y entramos en más detalle en listas y diccionarios.

2 Desarrollo de los ejercicios de la clase anterior

2.1 Primer ejercicio

Escriba una función que tome dos listas A y B y devuelva una lista con todos las parejas ordenadas con primera componente en A y segunda en B (es decir, una *lista* $A \times B$).

```
In [1]: def productocartesiano(lista1, lista2):
            producto = []
            for elemento1 in lista1:
                for elemento2 in lista2:
                    producto.append((elemento1, elemento2))

            return producto
```

```
In [2]: productocartesiano([1,2,3], ['a', 'b', 'c'])
```

```
Out[2]: [(1, 'a'),
         (1, 'b'),
         (1, 'c'),
         (2, 'a'),
         (2, 'b'),
         (2, 'c'),
         (3, 'a'),
         (3, 'b'),
         (3, 'c')]
```

2.2 Segundo ejercicio

Supongamos que construimos el objeto *función conjuntista* en python como una lista de parejas ordenadas subconjunto de $A \times B$ con A y B listas, es decir

```
A = [1,2,3]
B = ['a', 'b', 'c']
f = [(1, 'a'), (2, 'b'), (3, 'c')]
```

Escriba una función que, dada una lista f de parejas ordenadas, determine si ella es en efecto una función (es decir, retorne *True* si la lista es función y *False* en caso contrario.)

```
In [3]: def esfuncion(f, A, B):
            lista_de_preimagenes = []

            # Verificamos que f sea subconjunto de A x B
            for (preimagen, imagen) in f:
                lista_de_preimagenes.append(preimagen)
                if preimagen not in A or imagen not in B:
                    # print('f no es subconjunto de AxB')
                    return False

            # Verificamos que todo elemento de A es una preimagen
            if lista_de_preimagenes != A:
                # print('dominio de f no es A')
                return False

            # Verificamos la unicidad de la imagen.
            for (preimagen1, imagen1) in f:
                for (preimagen2, imagen2) in f:
                    if preimagen1 == preimagen2 and imagen1 != imagen2:
                        # print('no hay unicidad en las imágenes')
                        return False

            return True
```

Notamos que en esta implementación preferimos claridad a eficiencia.

```
In [4]: A = [1,2,3]
        f = [(1, 1), (1, 2), (2, 3)]
        print(esfuncion(f, A, A))
        g = [(1, 1), (2, 2), (3, 3)]
        print(esfuncion(g, A, A))
```

```
False
True
```

2.3 Tercer ejercicio

Escriba una función que retorne *True* si un número es par y *False* si un número es impar.

```
In [5]: def espar(n):
            return n%2 == 0
```

```
In [6]: print(espar(2))
        print(espar(3))

True
False
```

3 Almacenamiento y Objetos mutables e inmutables

Hablemos ahora sobre cómo python almacena variables y sobre cómo ciertas variables son mutables y cómo otras son inmutables.

3.1 Almacenamiento de variables

Cuando escribimos en nuestros códigos

```
a = 5
```

ocurren tres cosas:

1. python crea el objeto 5 y lo almacena en la memoria.
2. python crea la variable a.
3. python hace que la variable a apunte al objeto 5.

Vale la pena tener en cuenta que una variable nunca apunta a otra variable: por ejemplo cuando escribimos

```
b = a
```

la variable *b* no apunta a *a* sino al objeto relacionado (i.e. 5).

```
In [7]: a = 5
        b = a
        # print(a is b) (es True)
        a = a + 1
        # print(a is b) (es False, porque ya no son el mismo obj.)
        print(b) # b no se ha modificado.

5
```

3.2 Objetos mutables e inmutables

Un objeto se dice **mutable** si se puede modificar, y se dice **inmutable** en caso contrario. En python, todos los objetos normales son **inmutables** salvo

- las listas
- los conjuntos
- los diccionarios

Es decir: los enteros, los strings, los floats, los bools son todos imutables.

Hay que tener especial cuidado con la asignación con objetos mutables. Por ejemplo:

```
In [8]: L1 = [1,2,3]
        L2 = L1
        L1[0] = 10
        print(L2) # Como L2 estaba apuntando al mismo objeto que L1,
        # aparecerá [10, 2, 3]
```

```
[10, 2, 3]
```

Es decir, cuando hacemos `L1[0] = 10` estamos modificando el objeto mutable `[1,2,3]` a `[10,2,3]`. Como `L2` estaba apuntando a este objeto, queda `L2 = [10, 2, 3]`.

Sin embargo, si volvemos al ejemplo anterior:

```
In [9]: a = 5
        b = a
        a = a + 1
        print(b)
```

```
5
```

vemos que `b` no se ha modificado porque el objeto al que apuntaba `a` (i.e. `5`) es inmutable, entonces cuando decimos `a = a+1` estamos en verdad creando un objeto nuevo (en este caso `6`) y haciendo que `a` apunte a él.

4 Listas

Como habíamos hablado, existen los objetos de tipo `list` en python. Habíamos hablando de cómo acceder a sus elementos, de cómo saber su longitud usando `len` y de cómo pegarle más elementos usando `append`. Ahora vamos a hablar un poco más sobre listas.

4.1 Entrando a elementos en listas

Estamos acostumbrados a entrar a los elementos en las listas usando índices positivos, pero también podemos usar índices negativos (en donde -1 es el último elemento, -2 el penúltimo…). Por ejemplo:

```
In [10]: L = [1,2,3,4]
         print(L[0])
         print(L[3])
         print(L[-1])
```

```
1
4
4
```

4.2 Definiendo listas por extensión y por comprensión

Solemos definir listas así:

```
In [11]: L = [1,2,3,4,5,6]
```

pero, ¿qué pasa si necesitamos una lista con los números del 1 al 100? Podríamos intentar la siguiente aproximación (que no es para nada *pythonica*)

```
In [12]: L = []
         for k in range(1,101):
             L.append(k)
         print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
```

Sin embargo existe una forma mucho más elegante y corta de definir una lista al describir sus elementos:

```
In [13]: L = [k for k in range(1, 101)]
         print(L)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
```

¡Miren eso!, con solo una línea hacemos lo que antes nos demoraba 3. Podemos mezclar condicionales también:

```
In [14]: K = [m for m in L if m >= 50]
         print(K)
```

```
[50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70
```

4.3 Un par de métodos más

Recordemos que un método es una función específica a una clase. Revisemos otros métodos asociados a las listas:

```
In [15]: L = [1,2,3,2,2]
         print(L.count(2)) # para contar todas las incidencias del objeto 2 en la l
         M = L.copy() # para copiar la lista creando un objeto nuevo con el mismo c
         L.insert(3, 'hola') # para insertar un objeto antes del índice indicado.
         print(L)
         L.extend(M) # extiende L pegándole los objetos de M. M puede ser cualquier
         print(L)
         M.sort()
         print(M)
         print(L.index(2))
```

```
3
[1, 2, 3, 'hola', 2, 2]
[1, 2, 3, 'hola', 2, 2, 1, 2, 3, 2, 2]
[1, 2, 2, 2, 3]
1
```

Notamos una diferencia fundamental entre algunos de estos métodos: el método `list.copy()` crea un objeto nuevo, mientras que métodos como `list.sort()` o `list.insert(index, object)` **modifican la lista original**, a éste último tipo de métodos los solemos llamar métodos *in-place*.

4.4 Slicing

Slicing consiste en sacar ciertas partes de una lista. La sintaxis es la siguiente: `lista[inicio:fin:paso]`. Como siempre, nunca se toma el elemento final, entonces si queremos sacar por ejemplo el segmento inicial hasta la posición 4, debemos escribir `lista[0:5]` en vez de `lista[0:4]`. Por último, el paso siempre está predeterminado a ser 1 y podemos evitar escribir las posiciones iniciales y finales si éstas son el comienzo y el fin de la lista respectivamente.

```
In [16]: L = ['a','b','c','d']
         M = L[0:3]
         print(M)
         K = L[:3]
         print(K)
         G = L[-2:]
         print(G)
         L_invertida = L[::-1]
         print(L_invertida)

['a', 'b', 'c']
['a', 'b', 'c']
['c', 'd']
['d', 'c', 'b', 'a']
```

Todo objeto secuenciable puede ser sujeto a slicing, por ejemplo:

```
In [17]: nombre = 'Miguel'
         diminutivo = nombre[:-1]
         print(diminutivo)
         sucesion = (1,2,3,4,5,6,7,8)
         print(sucesion[3:-3])

Migue
(4, 5)
```

4.5 La función `enumerate`

Usualmente es útil recuperar el índice en el que está cierto elemento a la hora de hacer un ciclo for. Para hacerlo, usamos el método enumerate.

```
In [18]: print(enumerate(['a', 'b', 'c']))
         print(list(enumerate(['a', 'b', 'c'])))

<enumerate object at 0x7fefe85c8af8>
[(0, 'a'), (1, 'b'), (2, 'c')]
```

```
In [19]: lista_de_invitados = ['Iván', 'Daniel', 'Jorge']
         for indice, invitado in enumerate(lista_de_invitados):
             print(invitado + ' es el invitado número ' + str(indice))

Iván es el invitado número 0
Daniel es el invitado número 1
Jorge es el invitado número 2
```

5 Diccionarios

5.1 ¿Qué es un diccionario?

Un diccionario es un objeto en python que se compone de llaves (en inglés *keys*) y valores (*values*).

```
edades = {'Miguel': 22, 'Daniel': 25, 'Bruma': 9}
```

podemos pensar en `edades` como una *función conjuntista* que a la llave (o preimagen) `'Miguel'` le asocia el valor (o imagen) 22. Las llaves deben siempre ser objetos inmutables.

Diferente a las listas, los diccionarios no tienen ningún sentido del orden.

```
In [20]: edades = {'Miguel': 22, 'Daniel': 25, 'Bruma': 9}
         edades['Bruma']

Out[20]: 9
```

Podemos ingresar nuevas *parejas* en el diccionario de la siguiente forma:

```
In [21]: edades = {'Miguel': 22, 'Daniel': 25, 'Bruma': 9}
         edades['Diana'] = 49
         print(edades)

{'Daniel': 25, 'Miguel': 22, 'Bruma': 9, 'Diana': 49}
```

Podemos también modificar los valores de cierta llave:

```
In [22]: edades['Miguel'] += 1
         print(edades)
```

```
{'Daniel': 25, 'Miguel': 23, 'Bruma': 9, 'Diana': 49}
```

```
In [23]: print(edades.keys()) # es un iterable con las preimágenes
         print(edades.items()) # es un iterable con las parejas ordenadas
         print(edades.values()) # es un iterable con valores.
```

```
dict_keys(['Daniel', 'Miguel', 'Bruma', 'Diana'])
dict_items([('Daniel', 25), ('Miguel', 23), ('Bruma', 9), ('Diana', 49)])
dict_values([25, 23, 9, 49])
```

5.2 Diccionarios por extensión y por comprensión

Como con las listas, podemos definir diccionarios por comprensión.

```
In [24]: invitados = ['Miguel', 'Agatha', 'Bruma']
         diccionario_de_invitados = {invitado: indice for (indice, invitado) in enu
         print(diccionario_de_invitados)
```

```
{'Agatha': 1, 'Miguel': 0, 'Bruma': 2}
```

5.3 Ejercicio: diccionario inverso

1. Escriba una función que tome un diccionario y retorne True si éste es una función biyectiva y False en caso contrario.
2. Escriba una función que tome un diccionario biyectivo y devuelva el diccionario inverso.

```
In [25]: def esinvertible(diccionario):
             if len(set(diccionario.keys())) != len(diccionario):
                 return False

             if len(set(diccionario.values())) != len(diccionario):
                 return False

             return True
```

```
In [26]: print(esinvertible({1: 1, 2: 2, 3: 2}))
         print(esinvertible({1: 1, 2: 2, 3: 3}))
```

```
False
True
```

```
In [27]: def diccionarioinverso(diccionario):
             if esinvertible(diccionario):
                 return {value: key for (key, value) in diccionario.items()}
             else:
                 raise ValueError('diccionario no es invertible')
```

```
In [28]: diccionario_invertible = {1: 'a', 2: 'b', 3: 'c'}
         inverso = diccionarioinverso(diccionario_invertible)
         print(inverso)
```

```
{'b': 2, 'a': 1, 'c': 3}
```

5.4 Proyecto: encriptador básico usando python.

Construyamos un encriptador para mensajes usando diccionarios en python.

```
In [29]: alfabeto = 'abcdefghijklmnopqrstuvwxyz'
         valor_de_cambio = 2
         encriptador = {alfabeto[i]: alfabeto[(i+valor_de_cambio) % len(alfabeto)]
         print(encriptador)
         desencriptador = diccionarioinverso(encriptador)
         print(desencriptador)
```

```
{'j': 'l', 'y': 'a', 'z': 'b', 'u': 'w', 'a': 'c', 'q': 's', 'i': 'k', 'b': 'd', 'w
{'j': 'h', 'z': 'x', 'u': 's', 'a': 'y', 'q': 'o', 'i': 'g', 'b': 'z', 'w': 'u', 'h
```

```
In [30]: def encriptar(mensaje, alfabeto, offset):
             encriptador = {alfabeto[i]: alfabeto[(i+offset) % len(alfabeto)] for i
             desencriptador = diccionarioinverso(encriptador)
             mensaje_encriptado = ''
             for letra in mensaje:
                 mensaje_encriptado = mensaje_encriptado + encriptador[letra]

             return mensaje_encriptado

         def desencriptar(mensaje, alfabeto, offset):
             encriptador = {alfabeto[i]: alfabeto[(i+offset) % len(alfabeto)] for i
             desencriptador = diccionarioinverso(encriptador)
             mensaje_original = ''
             for letra in mensaje:
                 mensaje_original = mensaje_original + desencriptador[letra]

             return mensaje_original
```

```
In [31]: alfabeto = 'abcdefghijklmnopqrstuvwxyz'
         print(encriptar('hola', alfabeto, 5))
         print(desencriptar('mtqf', alfabeto, 5))
```

```
mtqf
hola
```

6 Ejercicios

6.1 Primer ejercicio

Escriba por compresión las siguientes listas:

1. Una lista con el cuadrado de todos los números en range(0,100).
2. Una lista con todas las letras de su nombre.

6.2 Segundo ejercicio

Cree una función que tome un string y devuelva un diccionario cuyas llaves son las letras del string y sus respectivos valores son la cantidad de veces que ellas aparecen en el string. Por ejemplo,

```
contador('esternocleidomastoideo')
{'e': 4, 's': 2, 't': 2, ...}
```

6.3 Tercer ejercicio

Cree una función que tome un string y que devuelva un diccionario que cuente las palabras y sus repeticiones bajo las siguientes especificaciones: 1. Todas las llaves deben estar en minúsuculas (es decir, 'Hola' y 'hola' deberían sumar al mismo contador). 2. Se deben ignorar los signos de puntuación (es decir, " '(),.;:?¡¿[]{}-' ").

Por ejemplo

```
contadorpalabras('Hola, mi nombre es Miguel y el nombre de mi gata es Agatha')
{'hola': 1, 'mi': 2, 'nombre': 2, 'es': 2, ...}
```

Pista: para este ejercicio vendría bien revisar los métodos `split` y `replace` en strings.