Anexo funciones Python - Taller de Web Scrapping

Vicent Blanes Selva

20/02/2017

Índice

1. Estructuras

En esta sección haremos un recordatorio/pequeña introducción de la sintaxis en Python. Si estás familiarizado con otros lenguajes de programación como C, Java, JavaScript o R puedes leerla por encima para no cometer errores de sintácticos. Si no estás familiarizado con la programación te recomendamos que le eches un buen ojo ;)

Enlace directo a la documentación https://docs.python.org/3.6/tutorial/datastructures.html

1.1. Estructuras de datos

1.1.1. Listas

Una lista es una estructura muy sencilla que consiste en una colección de elementos almacenados de forma consecutiva. A diferencia de lo que ocurre en otros lenguajes python puede contener elementos de distintos tipos. Se puede definir una lista vacía o con algunos objetos ya en su interior.

```
# Manera sencilla de definir listas
lista_vacia1 = list()
# o equivalentemente
lista_vacia2 = []
# tambien podemos inicializarla con algunos valores
primos = [2,3,5,7,9,11]
# en python las listas pueden contener elementos de distintos tipos
# una lista puede contener elementos 'complejos' como otras listas
lista_heterogenea = ['Hola que tal', 42, primos]
```

La forma más sencilla de recuperar un elemento de la lista es referirse a él mediante el nombre de la lista y el índice que ocupa entre corchetes. **IMPORTANTE:** en python el primer indice de una lista es el 0.

```
# podemos acceder a los elementos de una lista usando []
print(primos[2])
# obtenemos el TERCER elemento y lo mostramos por pantalla
# tambien funciona con indices negativos
print(primos[-1]) #mostraria el ULTIMO elemento
```

También poseemos dos método para insertar elementos en una lista, **append e insert**.

```
# append anyade un elemento al final de la lista
primos.append(13)
# el estado de la lista 'primos' seria el siguiente
# [2,3,5,7,9,11,13]

# insert requiere dos argumentos, la posicion donde insertar y el
    elemento
primos.insert(0, 1)
# insertamos el 1 en la primera posicion desplazando el resto
# [1,2,3,5,7,9,11]
```

Y entre otros, un par de método para borrar los elementos de una lista. Se recomienda revisar la documentación (enlace al principio de la sección).

```
# dos formas sencillas de borrar
#1) si conocemos el elemento que queremos eliminar
primos.remove(1) #borra la primera aparicion de 1 en la lista
#2) si conocemos su indice
primos.pop(0) #borrar y duelve el valor en el indice 0
# con cualquiera de los [2,3,5,7,9,11]
```

1.1.2. Diccionarios

Los diccionarios son estructuras de datos similares a las listas, sólo que en estos se guardan pares clave-valor. Cada una de las entradas debe poseer una clave única, pudiendo repetirse su valor. También son conocidos como **Maps** o con implementaciones concretas como **tablas hash**. Poseemos métodos para insertar, borrar y consultar las información de los diccionarios. Lo vemos con un ejemplo:

```
# crear un diccionario vacio
dicc_vacio1 = dict()
# otra manera
dicc_vacio2 = {}
# crear un diccionario con algunos elementos ya
dicc_edades = {'Vicent' : 24, 'Maria' : 20, 'Damian' : 47}

# recuperar un valor es similar al metodo usado en listas
# solo que en lugar de un indice escribimos su clave
print(dicc_edades['Vicent'])
# esto nos pintaria por pantalla: 24

# anyadir es muy sencillo
dicc_edades['Encarna'] = 50

# si quisieramos eliminar una entrada del diccionario
del dicc_edades['Damian']
```

Tenemos también método para recuperar en forma de lista las clave y los valores. Debemos destacar que al contrario de las listas, los diccionarios en python no tienen un orden y por tanto cada vez que recuperemos las claves y/o los valores van a tener un orden arbitrario que nada tiene que ver con el orden de inserción.

```
# se puede obtener tanto la lista de claves existentes
# en un diccionario como la lista de valores
claves = dicc_edades.keys()
valores = dicc_edades.values()
```

1.2. Estructuras de control

1.2.1. Condicional: if

La clausula if, seguida de una condición que se evalúa a verdadero o falso y dos puntos, dan paso a un bloque de código que sólo se ejecutará si la mencionada condición es verdadera. La mejor forma de verlo es con un ejemplo:

```python
dicc_edades = {'Vicent' : 24, 'Maria' : 20, 'Damian' : 47}
# queremos saber si Vicent esta en nuestro diccionario
# y si lo esta mostrar su edad
if 'Vicent' in dicc_edades.keys():
    print(dicc_edades['Vicent'])

# es posible anyadir un bloque de codigo que se ejecute
# si la condicion no se cumple

if 'Laura' not in dicc_edades.keys():
    print(dicc_edades['Laura'])
else:
    dicc_edades['Laura'] = 27
# si Laura esta en el diccionario mostramos su edad
# si no esta la anyadimos
# En este ejemplo el programa ignoraria el bloque de codigo
# interior al if y ejecutaria el del else
```

Nota: los bloques de código en python están marcados por la identación. ***El estándar son 4 espacios*** *así que se recomienda configurar tu editor favorito para que sustituya la inserción de tabuladores por la de 4 espacios.*

1.2.2. Bucle: for

Al igual que los condicionales, los bucles son un parte muy importante de la programación ya que permiten realizar un conjunto de acciones varias veces sin tener que repetir el código. Los usaremos principalmente para recorrer elementos de una lista y poder ejecutar acciones sobre ellos. Los bucles for en python son algo distintos a los bucles de otros lenguajes más clásicos. De hecho serían el equivalente a un bucle **for each** tradicional.

```python
primos = [2,3,5,7,9,11]
# en cada una de las iteraciones la variable
# num va tomando los valores de la lista primos
# en la primera num vale 2, en la segunda vale 3
# hasta que num llega a valer 11
for num in primos:
    print(num)
# esto mostraria por pantalla todos los elementos
# que estan en la lista 'primos'
```

2. Funciones python

En este apartado veremos algunas funciones útiles que posee python en una distribución sin librerías externas y que nos ayudarán a desarrollar nuestro código.

- **print**: Función que imprime por la salida estándar.
- **str(obj)/int(obj)/float(obj)**: Devuelve, si es posible, el objeto transformado en cadena de texto, entero o número de coma flotante dependiendo de la función utilizada. Puede producir error.

- **len(obj)** Devuelve la longuitud de una colección, como el tamaño de una lista o el número de clave de un diccionario. Produce error si se aplica sobre variables simples como un número o un literal.
- **range(numero)**: Devuelve una secuencia de números enteros desde 0 hasta el número especificado sin incluirlo (Intervalo abierto por la derecha). Se pueden añadir más argumentos, como el número de inicio o el tamaño del paso.

  - range(4) devuelve [0,1,2,3]
  - range(2,4) devuelve [2,3]
  - range(1,4,2) devuelve [1,3] #de uno hasta 4 yendo de 2 en 2

3. Funciones de urllib

La librería urllib es un conjunto de paquetes diseñados para hacer más fácil las interacciones a partir de peticiones web. Nos vamos a centrar en **urllib.request** pero es una librería muy útil y recomendamos revisar su documentación (https://docs.python.org/3/library/urllib.html). Podemos ver que se ofrecen montones de funciones para realizar peticiones web de modo muy avanzado, su exploración concreta queda fuera del taller pues nosotros únicamente centraremos en aquellas que nos permitan hacer un *scrapeado* eficiente sin entrar en temas de *proxies* o identificación.

- **urlopen(url)** se trata de una función para establecer conexión con un sitio web, se pueden especificar algunos argumentos opcionales más pero en nuestro caso sólo la usaremos con la página web a la que deseemos conectar.
- Sobre el objeto retornado por la función anterior se puede aplicar la función **read()** que nos devuelve el contenido web de la página a la que hayamos realizado la petición como una única cadena de texto.

4. Funciones de beautifulsoup4

BeautifulSoup es una librería que permite tratar documentos html de cualquier tamaño para poder hacerlo más legible o realizar consultas como por ejemplo, obtener el primer texto que esté entre las etiquetas $< p >$. Aquí puedes encontrar su documentación con numerosos ejemplos: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

- **BeautifulSoup(cadenahtml, 'lxml')** esta es la función principal que transforma una cadena de texto con información html en un objeto del tipo BeautifulSoup que representa información anidada y posee varios métodos para tratar la información.

5. Para más información...

Este pequeño anexo no es más que una ayuda para entender algunos de los muchos elementos del lenguaje que vamos a usar en este taller, si se quiere aprender más en profundidad recomendamos acudir a fuentes bibliográficas. Nos gustaría recomendar el libro **Python para todos**, que a pesar de estar enfocado a Python 2.7 es libre y muy didáctico. Se puede descargar en este enlance: https://launchpadlibrarian.net/18980633/Python%20para%20todos.pdf