James Bo de Interness

ExactasPrograma - Datos

Facultad de Ciencias Exactas y Naturales, UBA

Invierno 2020

¿Qué vamos a hacer hoy y el viernes?

- El objetivo es aprender a trabajar procesando textos.
- Vamos a ver cómo conseguir información de Internet, pero a *gran* escala.
- Vamos a aprender un poco más de programación.
- De paso, vamos a aprender una técnica de visualización que es interesante (la nube de palabras).
- Y vamos a ver unos detalles de cómo funciona la web.

Importante

Vamos a usar la consola (terminal) para probar algunas cosas y vamos a volver al entorno de desarrollo local: *Welcome back, spyder! We missed you!*

¿Qué es una página web?

La **web** está tan metida en nuestra vida de todos los días, que ni le prestamos demasiada atención.
Hasta que, cuando no hay acceso, no podemos trabajar.
Hay cuatro *cosas* que intervienen en el proceso de acceder a contenidos en la web:

- **Web page**: es un archivo de texto, pero como queremos que las páginas tengan algún formato *lindo*, se le agregan *marcas*. Por ejemplo, para que una palabra se vea en negrita (*bold*), se la encierra entre **tags** que lo indican: `<b>pelado</b>`.
- **HTML**: viene de *Hyper Text Markup Language*, es la forma de ponerse de acuerdo en qué significa cada marca (*tag*) dentro del archivo de una página. En el ejemplo anterior, es darle el sentido a que la 'b' significa *negrita*.
- **Web browser**: es un programa que corre en nuestras máquinas o dispositivos. El usuario escribe una dirección de una página y el browser se conecta con el servidor para obtener una página para mostrar. La página que se pide se descarga desde el servidor y se muestra en nuestra máquina. El formato (cómo se muestra) está escrito dentro de la misma página usando *tags*.
- **Web server**: corresponde con un programa que responde pedidos (**requests**) de un web browser. La respuesta consiste en un archivo *html* que el browser, luego, muestra al usuario (es decir, a nosotros).

Mirando la matriz

- Es bastante sencillo ver el código html detrás de una página web.
- En cualquier browser razonable, se puede hacer click con el botón derecho del mouse y poner **View page source** (o algo similar).
- Por ejemplo, el comienzo de una página de wikipedia se ve algo así:

```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Michael Jordan - Wikipedia</title>
```

- Los *tags* aparecen encerrados entre un `<` (signo de menor) y un `>` (signo de mayor).
- Para indicar el final, se usa `/` (barra invertida). Por ejemplo:
  `<title>Michael Jordan - Wikipedia</title>`

Estructura mínima de una página

Ya van a ir viendo que la web es un universo de posibilidades y de *modas* en cómo hacer ciertas cosas (a veces forzando los standards del momento).
Vamos a ver un poco de estructura para que sepan qué mirar en una página. Los *tags* que aparecen en cualquier página son:

- `<HTML>`: le indica al browser que comienza la página.
- `<HEAD>`: indica el comienzo del encabezado. En esta sección se incluye información que no se muestra directamente como parte de la página. El ejemplo más claro es el título (`<TITLE>`) que se muestra en la ventana del browser y no como parte del documento.
- `<BODY>`: esta sección del documento incluye todo lo que se muestra en la página: el texto, las tablas, las imágenes, etc.
- Para que el documento tenga bien el formato, todo *tag* debe tener su complemento o cierre: `</HEAD>`, `</BODY>` y `</HTML>`.

Rebobinando

- En lo que nos interesa para esta actividad, una página web es un archivo de texto.
- Tiene ciertas secciones, pero la que tiene la información que me se muestre está en la sección `<BODY>`.
- Para traer una página no hace falta tener usar browser, solo hace falta tener una forma de mandar los **requests** a un web server.
- Inclusive uno podría hacer la tarea que realiza un browser a mano para traernos el html de cualquier página, pero es un poco molesto.
- Para poder traerse **mucha** información desde la web, se desarrollaron distintas plataformas. Una de las más usadas basadas en `Python` es `scrapy` (otra es `selenium`).

Importante

En esta actividad vamos a usar `scrapy` y la consola (o terminal), esperamos que no sufran... tanto.

Scrapy

- Es un *framework* basado en `Python` que permite traer páginas web y extraer la información que nos interesa.
- Se puede automatizar para que siga *links* y siga trayendo información desde otras páginas.
- Se puede limitar para que no visite ciertos dominios (páginas web) que no nos interesan o que sabemos que nos pueden traer problemas.
- Obviamente es de código abierto y se puede ver la documentación en `https://scrapy.org/`.

Un poco de terminología

Si bien en la guía están los detalles, viene bien tener un panorama de cómo funciona scrapy:

- **Proyecto**: el programa que vamos a hacer para bajar las páginas tiene una estructura especial con distintos archivos y directorios. Hay un comando en `scrapy` para que lo arme por nosotros y que no lo pifiemos.
- **Spider**: es el *laburante* del universo de `scrapy`. Es un programa en `Python` que es el encargado de traer las páginas y que tiene funciones que podemos escribir para guardar solo la parte de la información que nos interesa.
- **fetch**: es el procedimiento que consiste en contactar a un servidor web y traer una página (incluye un **request** al servidor y una **response** del mismo).
- **parse**: es una función que uno debe escribir dentro de un `spider`, recibe el *html* con la página que acaba de traerse el `spider` y debe quedarse solo con lo que nos interesa. Esto suele involucrar *jugar* con cadenas de caracteres.
- **response**: además de representar una respuesta genérica de un servidor, en el universo `scrapy` suele ser variable que tiene la página que nos acabamos de bajar.

Breve ejemplo

`scrapy` tiene una modalidad de uso que permite aprender cómo funciona:

- Desde una terminal deben ejecutar el siguiente comando: `scrapy shell`.
- Esto me abre una programa en un entorno `Python` (podemos hacer `import` y las cosas habituales). Pero además podemos usar algunas funciones de `scrapy` directamente sin tener que escribir un programa (es decir, un `spider`).
- Dentro del *shell* de scrapy, podemos ejecutar el siguiente comando: `fetch("https://en.wikipedia.org/wiki/Michael_Jordan")`
- El mensaje no es muy descriptivo, pero lo vamos a aprender a apreciar:

```
2020-08-24 20:40:27 [scrapy.core.engine] INFO: Spider opened
2020-08-24 20:40:28 [scrapy.core.engine] DEBUG: Crawled (200)
<GET https://en.wikipedia.org/wiki/Michael_Jordan> (referer: None)
```

- Creanme que dice que pudo bajar la página, pero para verificarlo, podemos usar este comando: `view(response)`, que nos abre la página que tenemos bajada en el browser de nuestro sistema.

Joya

¡Así podemos bajar todas las páginas que nos interesen!

Diccionarios

- Hay una estructura muy útil que vamos a usar bastante en el contexto de esta guía y que conviene repasarla (¡o aprenderla!).
- Un diccionario es una **colección** desordenada en la que cada ítem tiene la forma *clave-valor*.
- Se utilizan las llaves para declararlo:

```
# diccionario vac'io
my_dict = {}
```

- Para la clave solo se pueden usar **tipos inmutables** (`string`, `number` o `tupla`) y no se admiten repetidos:

```
my_dict = {'Esteban': 26, 'Jose': 45, 'Matias': 133}
```

Esto declara un diccionario con tres entradas identificadas por nombres y cada una asociada a un número.

Diccionarios: cómo se usa

- Para recuperar el valor asociado a una clave, usamos los corchetes con la clave:

```
my_dict['Esteban'] #devuelve el n'umero 26
```

- Si la clave buscada no está, `Python` no ahorra en insultos:

```
my_dict['Mariela'] # exception KeyError: 'Mariela'
```

- Si quiero agregar una nueva clave con su valor, hacemos:

```
my_dict['Mariela']=34 # agrego Mariela->34
```

- Si quiero cambiar una clave existente, también es bastante directo:

```
my_dict['Mariela']=89 # agrego Mariela->89
```

- Para ver todo lo que hay en el diccionario, simplemente lo imprimo:

```
print(my_dict)
{'Esteban': 26, 'Jose': 45, 'Matias': 133, 'Mariela': 89}
```

- Si quiero sacar un ítem:

```
my_dict.pop('Mariela')  #Devuelve el valor asociado a la clave
```

Terminando

- El diccionario es muy útil porque buscar por la clave es eficiente.
- Podemos asociar más o menos lo que queramos como valor. Pueden ser listas, cadenas de caracteres u otros objetos.
- En la actividad de hoy, lo vamos a usar bastante, así que vamos a practicar a full.

¡A trabajar!

La guía está en el campus. Intentamos hacerla bastante autocontenida y paso a paso. Van a tener que usar `Spyder` (o el entorno que les guste) para escribir la aplicación y una consola para correr `scrapy`. ¡Bye bye colab!