PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

FACULTAD DE LETRAS

Métodos y técnicas de investigación cuantitativa

César Antonio Aguilar

Facultad de Lenguas y Letras

12/06/2012

Cesar.Aguilar72@gmail.com

Tareas realizables con corpus

En esta clase, vamos a tratar de hacer algunos ejercicios mínimos con algunas herramientas disponibles para análisis de corpus lingüísticos, en concreto las suites de herramientas **Jaguar** (IULA-UPF), y **Natural Language-Tool Kit** (NLTK), desarrollada en lenguaje Python.

http://melot.upf.edu//cgi-bin/jaguar/jaguar.pl

Natural Language Toolkit

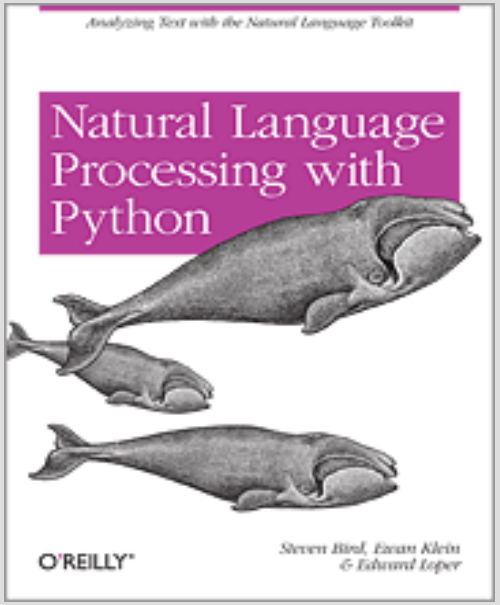
http://nltk.org

Introducción rápida a Python (1)

Python es un lenguaje de programación que nos permite emplear una colección de programas para hacer tareas de procesamiento de textos. A esta colección se le conoce como **NLTK** (***Natural Language Tool-Kit***).

python™

http://www.python.org/



https://sites.google.com/site/naturallanguagetoolkit/book

Introducción rápida a Python (2)

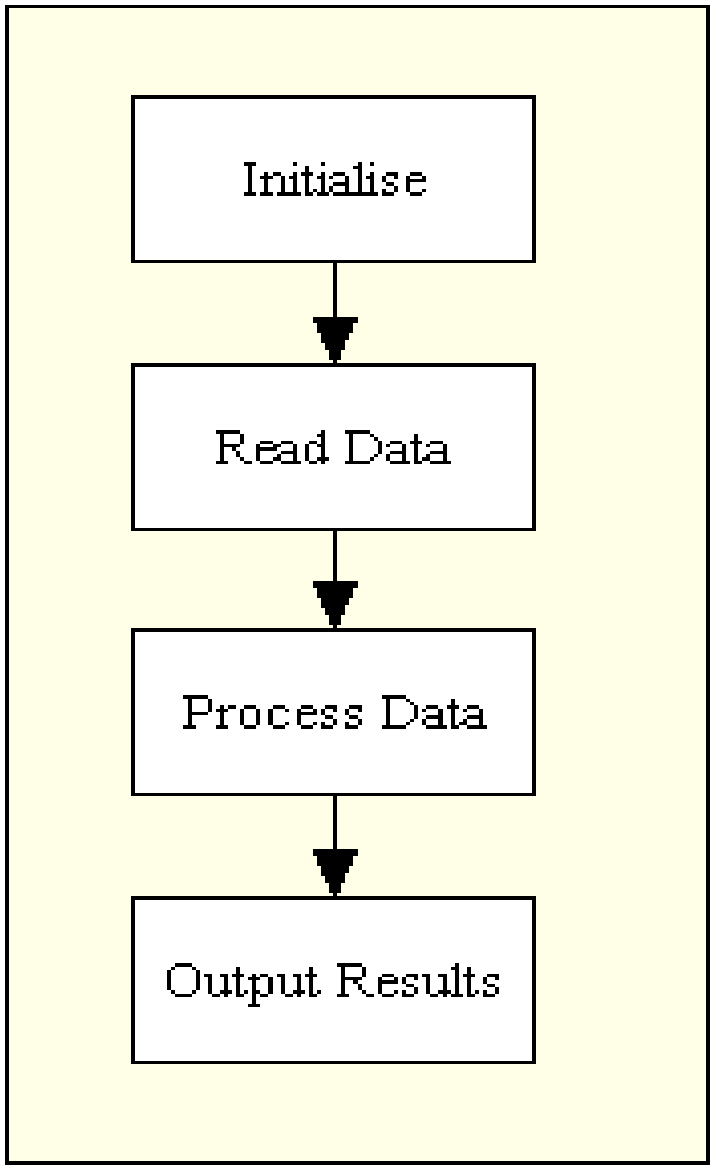
A grandes rasgos, Python es un lenguaje bastante versátil que se ocupa en muchísimas tareas. Digamos que la publicidad de Python pone énfasis en decir que:

1. Es fácil de usar.
2. Cuenta con una sintaxis muy clara.
3. Es gratuito (*open source*).
4. Cuenta con una amplia comunidad de programadores en el mundo.
5. Cuenta con una gran serie de módulos de programación para distintas tareas.
6. Permite hacer diversos análisis simbólicos y estadísticos.

¿Qué es programar? (1)

Ahora bien, para empezar, tratemos de entender algunas cosas básicas, la primera de todas: ¿qué cosa es *programar*? Consideremos que se trata simplemente de un proceso por el cual desarrollamos una secuencia ordenada de algoritmos enfocados en la resolución de un problema.

En general, cualquier programa se estructura de la siguiente forma:



¿Qué es programar? (2)

Ingredientes:

Para 4 Personas

500 gr. de espagueti
2 cucharadas de aceite de oliva
2 dientes de ajo
1 cebolla grande
1 rama de apio
1 zanahoria mediana
1/2 kg. de carne de res molida
2 tazas de caldo de carne
1 y 1/2 tazas de vino tinto
850 gr. de tomate
1 cucharada de azúcar
2 cucharadas de perejil fresco
Sal y pimienta
Queso parmesano

Modo de Preparación:

Freír el apio, cebolla, ajo y zanahoria picados. Añadir la carne removiéndola y desmeduzándola hasta que esté cocinada.

Incoporar el caldo, el vino, los tomates machacados sin semilla y el perejil picado. Al hevir, cocinar a fuego lento 90 minutos removiendo. Añadir sal y pimienta a gusto.

Cocer la pasta en abundante agua hirviendo con aceite y una pizca de sal.
Escurrir y servir la salsa sobre la pasta. Espolvorear con queso parmesano a gusto.

Ahora bien, ¿qué es un algoritmo? Es un término que designa un conjunto de instrucciones que siguen un orden determinado, en aras de resolver justo el problema que nos interesa.

Así, supongamos que tenemos un algoritmo que se llama: “receta para preparar *Spaghetti* la *Bolognesa*”:

¿Qué es programar? (3)

¿Qué es lo que caracteriza a los lenguajes orientados a objetos? En concreto, que conciben todo problema como un **objeto**, esto es, como una entidad que puede ser descrita a partir de dos rasgos:

1. Contar con **atributos**, los cuales son representados a partir de **variables.**

2. Desempeñar alguna **función**, la cual se conoce como **método del objeto.**

Así, digamos que para un lenguaje como Python, lo que existe en el mundo son objetos que tienen ciertas propiedades, además de que pueden ser parte de determinados procesos.

¿Por dónde empezar? (2)

De acuerdo con la definición anterior, podemos definir a todos los objetos que pertenezcan a la clase *coche* a partir de 1 atributo (“Gasolina”), y dos funciones (“arrancar” y “conducir”).

Ahora, de acuerdo con esta lógica, lo que nos importa es para que un objeto sea reconocido como un *coche*, lo primero es que sea capaz de poseer gasolina.

Una vez cumplido este requisito, lo que sigue es que el objeto sea capaz de desempeñarse dentro del marco de las funciones que hemos establecido. Si esto ocurre, decimos que el objeto es *verdadero*, dado que cumple con los requisitos necesarios para pertenecer a la clase.

En términos más computacionales, decimos que nuestro objeto **muestra un comportamiento acorde con la clase *coche*,**

¿Por dónde empezar? (3)

Pregunta: pensando como programadores, ¿qué clase de objetos nos interesa procesar con nuestros códigos?

Object Type	Example Constants/Usage
Numbers	`3.1415, 1234, 999L, 3+4j`
Strings	`'spam', "guido's"`
Lists	`[1, [2, 'three'], 4]`
Dictionaries	`{'food':'spam', 'taste':'ym'}`
Tuples	`(1, 'spam', 4, 'U')`
Files	`text = open ('eggs', 'r'). read()`

Instalado Python (1)

Una vez visto lo anterior, pongamos manos a la obra: hay que instalar primero Python en nuestras computadoras.

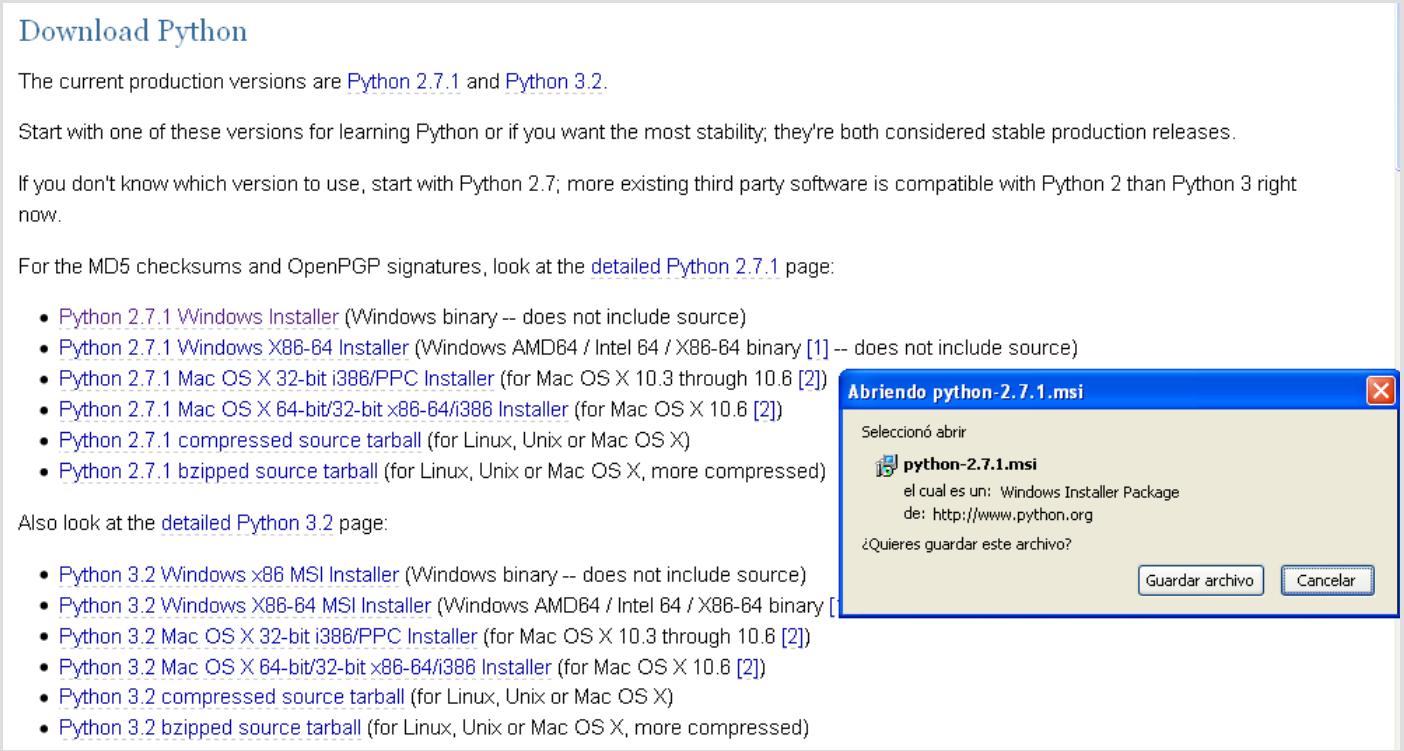
Para esto, hay que acceder al siguiente sitio electrónico:

http://nltk.org/install.html

Hay varias versiones de Python. En nuestro caso, vamos a ocupar la 2.7.3, la cual está disponible tanto para sistemas Windows, Mac y Linux.

Instalado Python (2)

Una vez que estén en esta página, den un click en el botón **Download**, y buscan la opción Python 2.6. Windows Installer. Al darle un click, les debe mostrar una ventana para descargar su programa:



Download Python

The current production versions are Python 2.7.1 and Python 3.2.

Start with one of these versions for learning Python or if you want the most stability; they're both considered stable production releases.

If you don't know which version to use, start with Python 2.7; more existing third party software is compatible with Python 2 than Python 3 right now.

For the MD5 checksums and OpenPGP signatures, look at the detailed Python 2.7.1 page:

- Python 2.7.1 Windows Installer (Windows binary -- does not include source)
- Python 2.7.1 Windows X86-64 Installer (Windows AMD64 / Intel 64 / X86-64 binary [1] -- does not include source)
- Python 2.7.1 Mac OS X 32-bit i386/PPC Installer (for Mac OS X 10.3 through 10.6 [2])
- Python 2.7.1 Mac OS X 64-bit/32-bit x86-64/i386 Installer (for Mac OS X 10.6 [2])
- Python 2.7.1 compressed source tarball (for Linux, Unix or Mac OS X)
- Python 2.7.1 bzipped source tarball (for Linux, Unix or Mac OS X, more compressed)

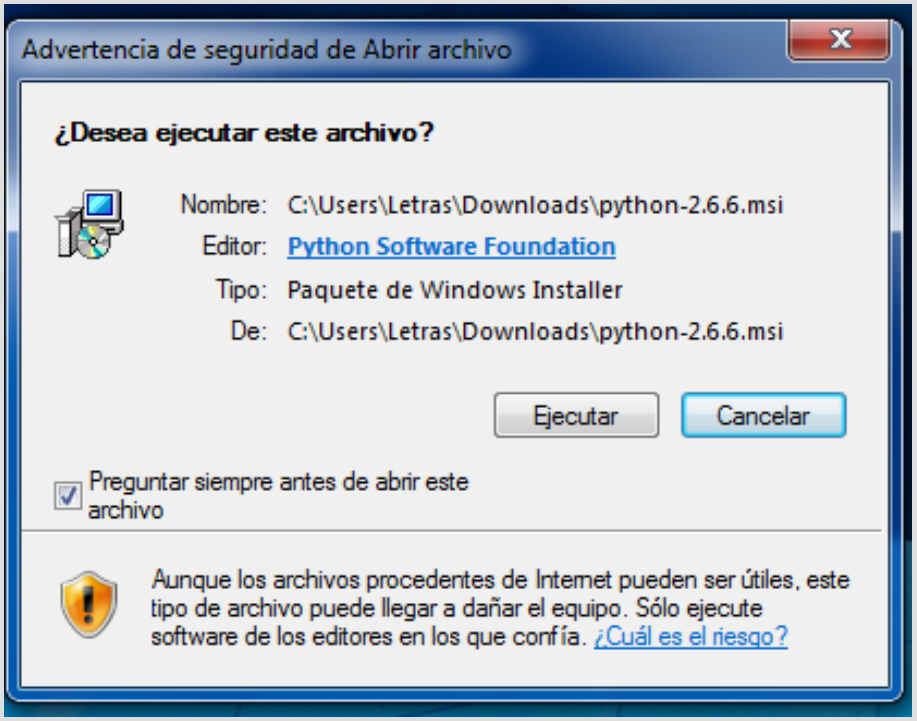
Also look at the detailed Python 3.2 page:

- Python 3.2 Windows x86 MSI Installer (Windows binary -- does not include source)
- Python 3.2 Windows X86-64 MSI Installer (Windows AMD64 / Intel 64 / X86-64 binary [
- Python 3.2 Mac OS X 32-bit i386/PPC Installer (for Mac OS X 10.3 through 10.6 [2])
- Python 3.2 Mac OS X 64-bit/32-bit x86-64/i386 Installer (for Mac OS X 10.6 [2])
- Python 3.2 compressed source tarball (for Linux, Unix or Mac OS X)
- Python 3.2 bzipped source tarball (for Linux, Unix or Mac OS X, more compressed)

Nota: para aquellos que manejen sistemas Mac, Leopard, o cualquiera de la familia Linux (Ubuntu, RedHat, Madrake, Mandriva, Suse, and so on…), podemos revisarlos aparte.

Instalado Python (3)

Luego que de hayan descargado el archivo de instalación de Python, viene una tarea complicada: responder a la pregunta: *¿Desea ejecutar este archivo?:*



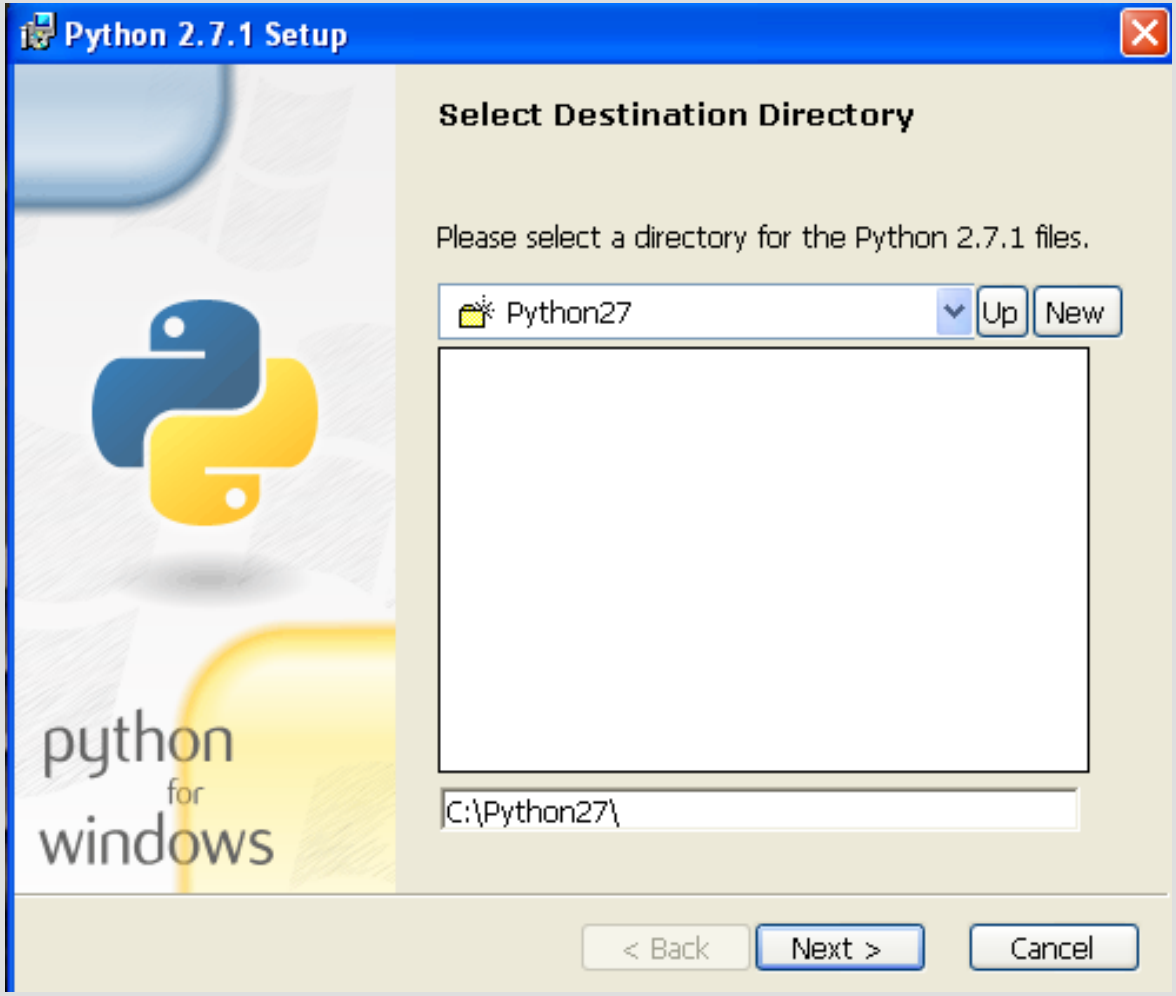
En caso de que respondan favorablemente, den un click en el botón *Ejecutar*, y pasamos a la siguiente fase.

Instalado Python (4)

Posteriormente, el ejecutable de Python les da dos opciones de instalación: una que permite a todos los usuarios de su equipo acceder a este recurso, u otra que restringe su uso únicamente a ustedes. Mi sugerencia es que opten por la primera opción.

Instalado Python (5)

Ahora, viene lo más complicado del proceso: decirle *Sí* a todas las ventanas que aparezcan de forma sucesiva. Empezamos con la primera: seleccionar la carpeta en donde se van a descargar Python:



Instalado Python (6)

Luego, Python es un lenguaje que trabaja con **librerías**, es decir, colecciones de herramientas construidas para realizar tareas específicas. Ustedes pueden descargar todas las librerías, o seleccionar aquellas que crean necesarias. En nuestro caso, optamos por ocupar todas:

Instalando Python (7)

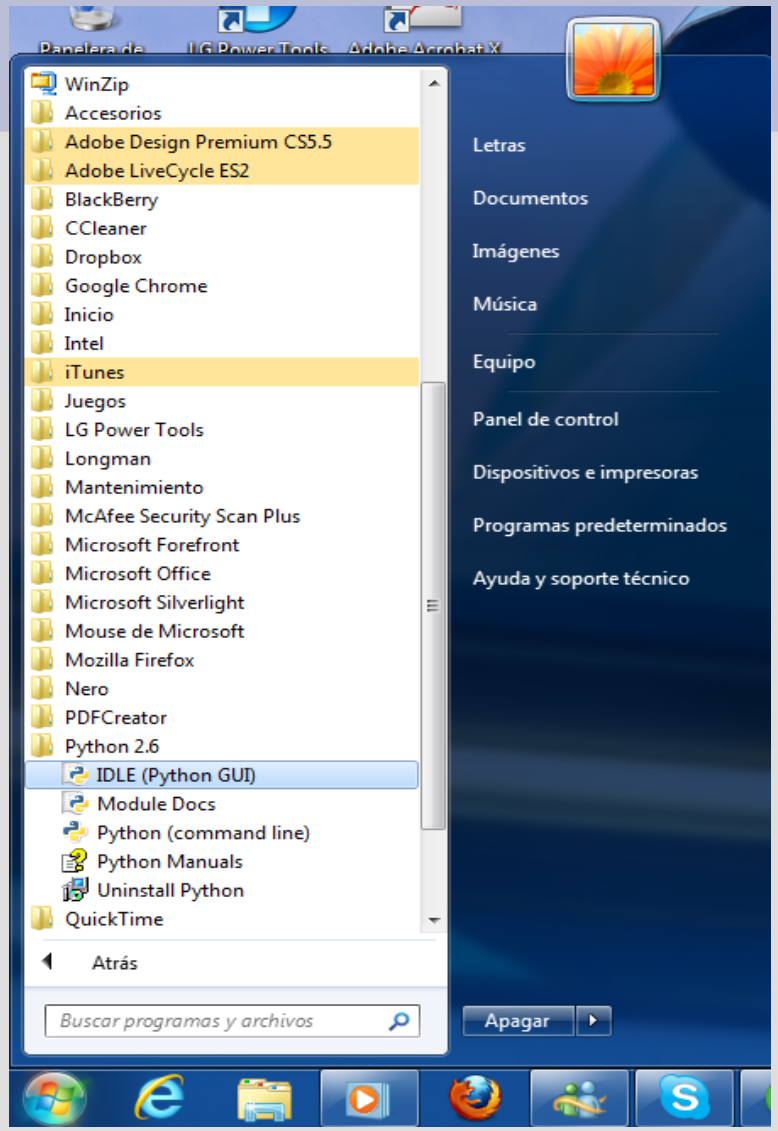
Finalmente, Python les presenta una ventana en donde les notifica que el proceso de instalación ha terminado:

Si dan un click a “Finish”, concluyen satisfactoriamente el proceso de instalaciòn.

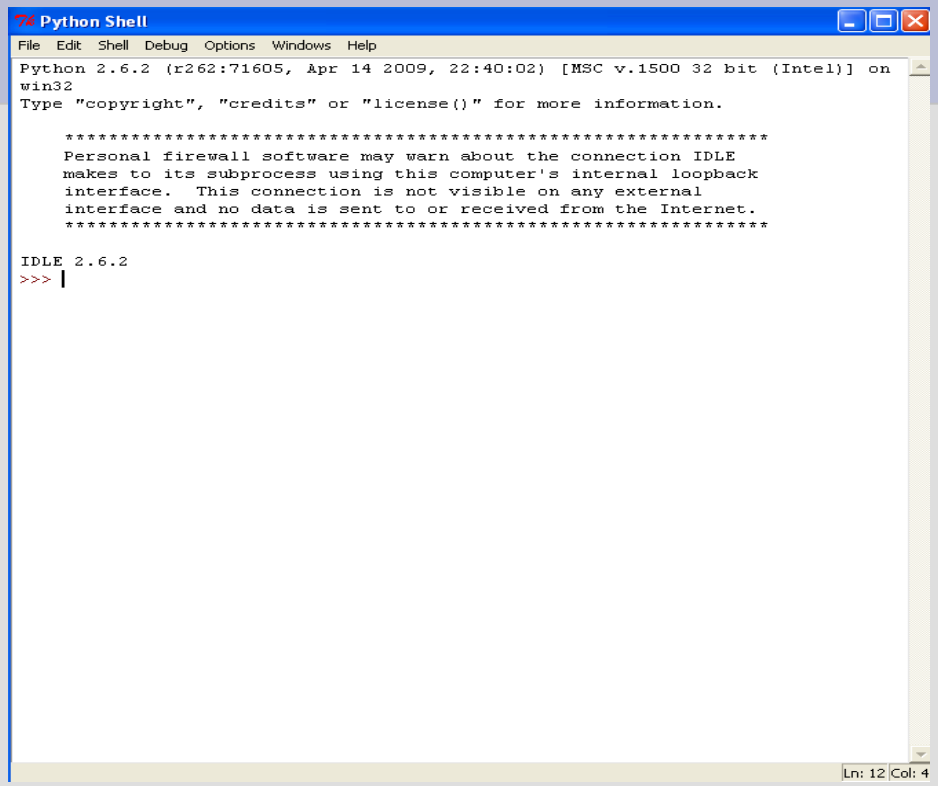
Mi primera sesión en Python (1)

Empecemos a utilizar nuestra sesión en Python, para lo cual primero comprobemos que está funcionando.

Si trabajan con Windows, vayan a su botón de *Inicio*, selecciones la opción *Todos los programas*, escogen la opción *Python 2.7*, y luego dan un click en la opción *IDLE Python*.



Mi primera sesión en Python (2)



Al seleccionar la opción de *IDLE Python*, lo que estamos haciendo es *llamar* al **intérprete** de este lenguaje de programación.

Brevemente, como su nombre lo indica, un *interprete* es una interfaz editora de textos, que nos ayuda a interactuar de una manera "amigable" con Python, evitándonos tener que programar directamente a un nivel de máquina.

Mi primera sesión en Python (3)

Ahora, demos nuestros primeros teclazos con Python. Escriban la siguiente instrucción:

print “¡Hola, Mundo!”

Una vez hecho esto, opriman el botón de *Enter*, y obtenemos:

```
IDLE 2.6.2
>>> print "¡Hola, Mundo!"
¡Hola, Mundo!
>>>
```

Mi primera sesión en Python (4)

Una buena parte de los procesos que vamos a ejecutar en Python siguen la siguiente estructura:

1. **Asignación de variables:** este proceso consiste en crear un objeto que sea identificado con un “nombre”; entre ambos se establece una relación de equivalencia.

2. **Declaración de una operación, función y/o instrucción:** a grandes rasgos, son todo el conjunto de acciones y/o funciones en las cuales pueden operar nuestros objetos, desde una simple suma, hasta un análisis sintáctico (o *parsing*).

3. **Comentarios:** el texto que aparece escrito al lado del signo # se entiende como un **comentario**, esto es, una anotación personal que hacemos para explicar en qué consiste el proceso que vamos a ejecutar.

Algunas aclaraciones (1)

Hay algunas cuestiones que debemos aclarar. La primera tiene que ver con respecto a la selección de un nombre que no sirva de variable para identificar a un objeto: no toda expresión lingüística funciona como una variable, pues hay algunas que aluden a funciones muy específicas, p.e.:

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	not	or	pass
print	raise	return	try	while	with
yield					

Nota: las expresiones que aparecen en esta tabla se les conoce como **nombres reservados**, y sirven para identificar todas las funciones que caracterizan a un lenguaje de programación. En nuestro caso, Python maneja 31 nombres reservados, los cuales sirven hacer funciones concretas.

Algunas aclaraciones (2)

Tener conocimiento de estas limitaciones nos ayuda a evitar problemas como **errores de sintaxis** o **errores de semántica**. Veamos algunos casos:

```
>>> 76trombones = "big parade"
SyntaxError: invalid syntax
>>> more$ = 1000000
SyntaxError: invalid syntax
>>> class = "Computer Science 101"
SyntaxError: invalid syntax
```

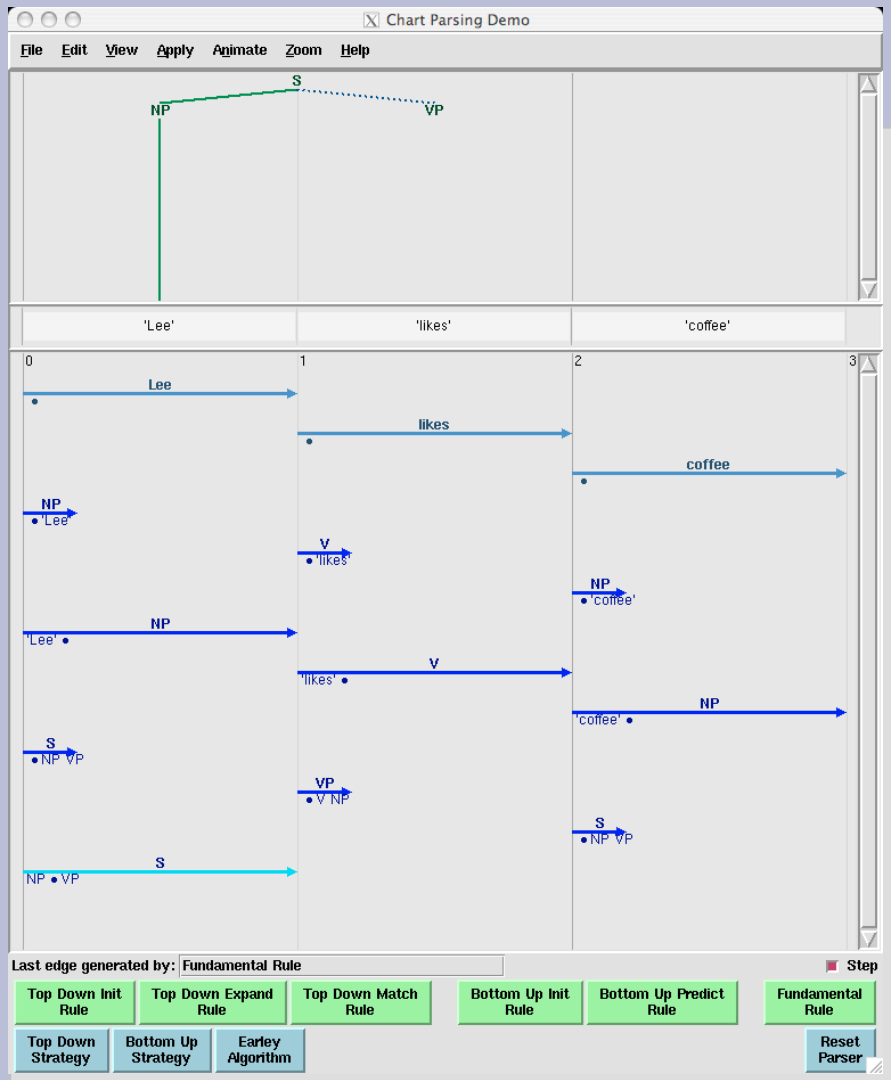
Notas: Estos errores se deben a lo siguiente:

1. Caso uno: **Nunca inicien un nombre de variable con números, siempre van con caracteres.**

2. Caso dos: **el uso del símbolo $ es ilegal a la hora de asignar una variable, ya que este símbolo es una expresión regular.**

3. Caso tres: ***class* es una expresión que sirve para designar funciones que operan al nivel de clases, por lo que se convierte en un nombre reservado dentro de Python.**

Instalando NLTK (1)



Como lo comentamos en una de las láminas, Python es un lenguaje de programación que trabaja con *librerías*, las cuales son útiles para realizar alguna tarea concreta. En Python, tenemos varias librerías útiles para trabajar con expresiones regulares, análisis estadísticos, generación de grafos, etc.

En nuestro curso, la librería con la que vamos a trabajar se llama **NLTK** (Natural Language Tool-Kit), una librería pensada precisamente para hacer tratamientos computacionales con textos.

Instalando NLTK (2)

Al igual que Python, NLTK es un software libre que podemos descargar en nuestra computadora desde el siguiente sitio:

www.nltk.org

Algunos ya lo conocer, otros quizá les resulte nuevo. En todo caso, la idea es que, a partir de estas sesiones, ustedes se hagan una mejor idea sobre lo que es esta librería, y los usos que pueden darle para sus análisis lingüísticos.

Instalando NLTK (3)

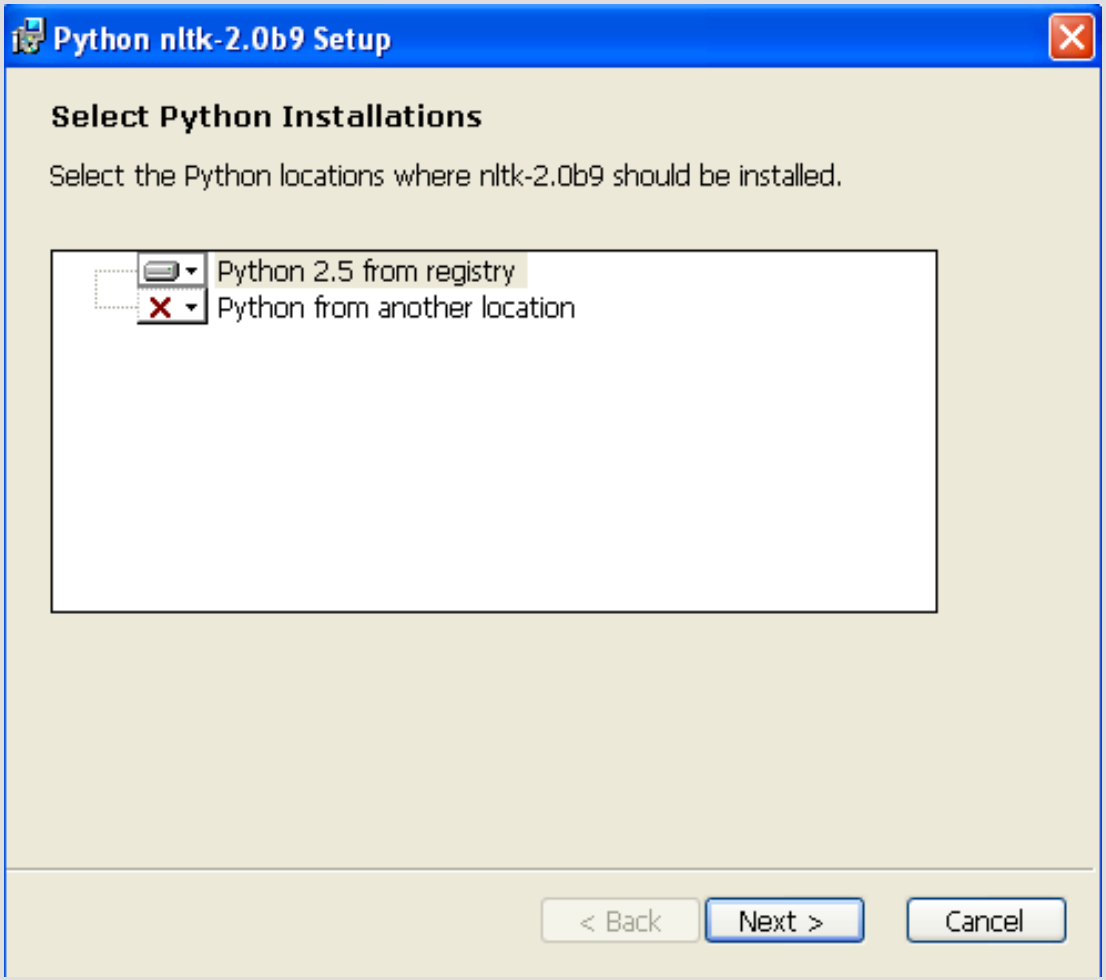
De nuevo, les va a aparecer una ventana con la opción de descarga. Acepten esta opción, y una vez que hayan guardado el ejecutable en su máquina, inicien el proceso de instalación dando un click. A continuación, les saldrá una ventana con la siguiente opción:

Elijan la opción *Install for all users*.

Instalando NLTK (4)

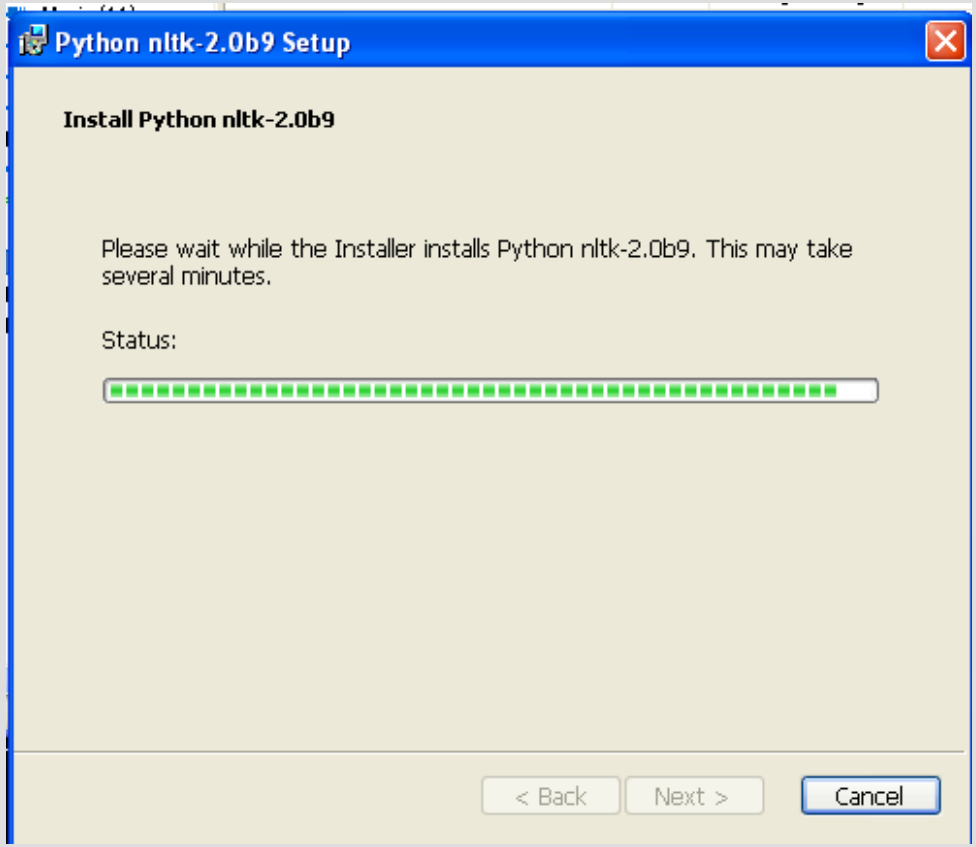
Su versión de NLTK es compatible con Python 2.6, aunque al inicio del proceso les salga esta ventana:

Al igual que como lo hicimos con Python, aceptemos siempre las opciones que nos presenta NLTK.



Instalando NLTK (5)

Una vez que han oprimido el botón *Next* de la ventana anterior, NLTK se instala sin ningún problema en su computadora:



Cuando termine de instalarse, aparecerá el botón *Finish*.

Al oprimirlo, terminarán el proceso, y ya estarán listos para ocupar NLTK.

Instalando NLTK (6)

Junto con NLTK vienen también varias librerías que vamos a requerir para realizar análisis estadísticos, generación de gráficas, árboles sintácticos, etc. En concreto:

* Numpy
* Matplotlib
* Prover9
* MaltParser
* MegaM

Para mayores detalles, vean el sitio WEB:

www.nltk.org/download

Instalando NLTK (7)

Ahora, hagamos una prueba para saber si NLTK se ha cargado correctamente en nuestra computadora. Usemos la siguiente instrucción:

```
>>> import nltk
>>> nltk.download()
```

Esta instrucción nos trae un repositorio electrónico con una serie de recursos disponibles para diferentes tareas: corpus lingüísticos, gramáticas formales, conjuntos de etiquetas, etc. La ventana que tiene que aparecerles es la siguiente:

Instalando NLTK (8)

NLTK Downloader

File View Sort Help

Collections | Corpora | Models | All Packages

Identifier	Name	Size	Status
abc	Australian Broadcasting Commission 2006	1.4 MB	installed
alpino	Alpino Dutch Treebank	2.7 MB	installed
biocreative_ppi	BioCreAtIvE (Critical Assessment of Information Extraction Systems in Biology)	218.3 KB	installed
brown	Brown Corpus	3.2 MB	installed
brown_tei	Brown Corpus (TEI XML Version)	8.3 MB	installed
cess_cat	CESS-CAT Treebank	5.1 MB	installed
cess_esp	CESS-ESP Treebank	2.1 MB	installed
chat80	Chat-80 Data Files	18.8 KB	installed
city_database	City Database	1.7 KB	installed
cmudict	The Carnegie Mellon Pronouncing Dictionary (0.6)	875.1 KB	installed
comtrans	ComTrans Corpus Sample	11.4 MB	installed
conll2000	CONLL 2000 Chunking Corpus	738.9 KB	installed
conll2002	CONLL 2002 Named Entity Recognition Corpus	1.8 MB	installed
conll2007	Dependency Treebanks from CoNLL 2007 (Catalan and Basque Subset)	1.2 MB	installed
dependency_treebank	Dependency Parsed Treebank	446.7 KB	installed
europarl_raw	Sample European Parliament Proceedings Parallel Corpus	12.0 MB	installed
floresta	Portuguese Treebank	1.8 MB	installed
gazetteers	Gazeteer Lists	8.1 KB	installed
genesis	Genesis Corpus	462.1 KB	installed
gutenberg	Project Gutenberg Selections	4.1 MB	installed
ieer	NIST IE-ER DATA SAMPLE	162.3 KB	installed
inaugural	C-Span Inaugural Address Corpus	313.8 KB	installed
indian	Indian Language POS-Tagged Corpus	194.5 KB	installed
jeita	JEITA Public Morphologically Tagged Corpus (in ChaSen format)	15.8 MB	installed
kimmo	PC-KIMMO Data Files	182.6 KB	installed
knbc	KNB Corpus (Annotated blog corpus)	8.4 MB	installed
langid	Language Id Corpus	5.0 MB	installed
mac_morpho	MAC-MORPHO: Brazilian Portuguese news text with part-of-speech tags	2.9 MB	installed
machado	Machado de Assis -- Obra Completa	5.9 MB	installed
movie_reviews	Sentiment Polarity Dataset Version 2.0	3.8 MB	installed
names	Names Corpus, Version 1.3 (1994-03-29)	20.8 KB	installed
nombank.1.0	NomBank Corpus 1.0	6.4 MB	installed

Download Refresh

Server Index: http://nltk.googlecode.com/svn/trunk/nltk_data/index.xml

Download Directory: /home/cesar/nltk_data

NLTK y corpus lingüísticos (1)

El Brown es uno de los corpus electrónicos pioneros que cuenta con una anotación sintáctica. Para saber, vean esta entrada en Wikipedia:

http://en.wikipedia.org/wiki/Brown_Corpus

En la siguiente tabla, podemos ver un ejemplo de su contenido:

ID	File	Genre	Description
A16	ca16	news	Chicago Tribune: *Society Reportage*
B02	cb02	editorial	Christian Science Monitor: *Editorials*
C17	cc17	reviews	Time Magazine: *Reviews*
D12	cd12	religion	Underwood: *Probing the Ethics of Realtors*
E36	ce36	hobbies	Norling: *Renting a Car in Europe*
F25	cf25	lore	Boroff: *Jewish Teenage Culture*
G22	cg22	belles_lettres	Reiner: *Coping with Runaway Technology*
H15	ch15	government	US Office of Civil and Defence Mobilization: *The Family Fallout Shelter*
J17	cj19	learned	Mosteller: *Probability with Statistical Applications*
K04	ck04	fiction	W.E.B. Du Bois: *Worlds of Color*
L13	cl13	mystery	Hitchens: *Footsteps in the Night*
M01	cm01	science_fiction	Heinlein: *Stranger in a Strange Land*
N14	cn15	adventure	Field: *Rattlesnake Ridge*
P12	cp12	romance	Callaghan: *A Passion in Rome*
R06	cr06	humor	Thurber: *The Future, If Any, of Comedy*

NLTK y corpus lingüísticos (2)

A través del intérprete de Python, podemos importar el corpus Brown de la librería de NLTK usando la siguiente instrucción:

```
from nltk.corpus import brown
```

Para saber qué tenemos en el Brown, podemos usar la siguiente instrucción, que nos muestra sus géneros literarios:

```
brown.categories()
```

NLTK y corpus lingüísticos (3)

```
>>> from nltk.corpus import brown
>>> brown.categories()
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies',
'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'reviews', 'romance',
'science_fiction']
```

Ahora, probemos las siguientes instrucciones:

1. brown.words(categories='news')

2. brown.words(fileids=['cg22'])

3. brown.sents(categories=['news', 'editorial', 'reviews'])

Pregunta: ¿Qué es lo que queremos obtener del Brown?

NLTK y corpus lingüísticos (4)

Veamos los resultados:

```
>>> brown.words(categories='news')
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
>>> brown.words(fileids=['cn15'])
['Early', 'in', 'November', 'the', 'clouds', 'lifted', ...]
>>> brown.sents(categories=['news', 'editorial', 'reviews'])
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investiga
tion', 'of', "Atlanta's", 'recent', 'primary', 'election', 'produced', '``', 'no
', 'evidence', "''", 'that', 'any', 'irregularities', 'took', 'place', '.'], ['T
he', 'jury', 'further', 'said', 'in', 'term-end', 'presentments', 'that', 'the',
 'City', 'Executive', 'Committee', ',', 'which', 'had', 'over-all', 'charge', 'o
f', 'the', 'election', ',', '``', 'deserves', 'the', 'praise', 'and', 'thanks',
'of', 'the', 'City', 'of', 'Atlanta', "''", 'for', 'the', 'manner', 'in', 'which
', 'the', 'election', 'was', 'conducted', '.'], ...]
>>> brown.sents(categories=['fiction', 'mystery', 'humor'])
[['Thirty-three'], ['Scotty', 'did', 'not', 'go', 'back', 'to', 'school', '.'],
...]
>>>
```

De acuerdo con esto, podemos consultar el contenido del Brown usando dos clases de objetos: palabras y oraciones, y mostrar en pantalla los resultados de nuestras consultas.

NLTK y corpus lingüísticos (5)

Ahora, tratemos de meternos al interior del corpus Brown. Escriban las siguientes instrucciones:

```
>>> from nltk.corpus import brown
>>> news_text = brown.words(categories='news')
>>> fdist = nltk.FreqDist([w.lower() for w in news_text])
>>> modals = ['can', 'could', 'may', 'might', 'must', 'will']
>>> for m in modals:
        print m + ':', fdist[m],
```

Lo que nuestro programa realiza con este código es un conteo de palabras, en este caso concreto contabiliza cuántos verbos modales hay en la categoría *News*. Veamos el resultado:

```
can: 94 could: 87 may: 93 might: 38 must: 53 will: 389
```

NLTK y corpus lingüísticos (6)

Veamos un código un poco más complicado, con el cual podremos ver la distribución de verbos modales dentro del Brown. Para ello, necesitamos importar de la librería de NLTK la función **Conditional FreqDist** (o distribución de frecuencia condicional). Veamos:

```
cfd_brown01 = nltk.ConditionalFreqDist(
        (genre, word)
        for genre in brown.categories()
        for word in brown.words(categories=genre))
```

Nota: Fíjense en las expresiones *for* y *in*: ¿para qué las estamos ocupando?, ¿qué estamos pidiéndole al intérprete de Python con ellas?

NLTK y corpus lingüísticos (7)

Una vez que hemos especificado que queremos calcular frecuencias de palabras dentro de los textos del Brown (considerando su género literario), ahora lo que debemos haces es especificar los objetos que vamos a buscar y contar, esto es:

```
genres = ['news', 'religion', 'hobbies', 'science-fiction',
'romance', 'humor']
modals = ['can', 'could', 'may', 'might', 'must', 'will', 'shall']
```

NLTK y corpus lingüísticos (8)

Y el resultado es:

```
>>> cfd_brown01.tabulate(conditions=genres, samples=modals)
                 can could  may might must will shall
           news   93   86   66   38   50  389     5
       religion   82   59   78   12   54   71    21
        hobbies  268   58  131   22   83  264     5
science_fiction   16   49    4   12    8   16     3
        romance   74  193   11   51   45   43     3
          humor   16   30    8    8    9   13     2
>>>
```

Obviamente, estamos trabajando con un ejercicio *de juguete*. Lo importante es que estamos obteniendo datos de un corpus, y que podemos modificar nuestras consultas tal cual lo necesitemos.

NLTK y corpus lingüísticos (9)

Otra tarea que podemos realizar es mostrar estos datos con un gráfico. Para esto, vamos a usar un corpus llamado **Inaugural Address Corpus**, el cual es una recopilación de los discursos que han dado los presidentes de Estados Unidos al asumir su cargo. Para esto, necesitamos de la instrucción:

```
from nltk.corpus import inaugural
```

Veamos el contenido de este corpus con los siguientes comandos:

1. inaugural.fileids()

2. [fileid[:4] for fileid in inaugural.fileids()]

NLTK y corpus lingüísticos (10)

Primer resultado:

```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()
['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', '1801-Jefferson
.txt', '1805-Jefferson.txt', '1809-Madison.txt', '1813-Madison.txt', '1817-Monro
e.txt', '1821-Monroe.txt', '1825-Adams.txt', '1829-Jackson.txt', '1833-Jackson.t
xt', '1837-VanBuren.txt', '1841-Harrison.txt', '1845-Polk.txt', '1849-Taylor.txt
', '1853-Pierce.txt', '1857-Buchanan.txt', '1861-Lincoln.txt', '1865-Lincoln.txt
', '1869-Grant.txt', '1873-Grant.txt', '1877-Hayes.txt', '1881-Garfield.txt', '1
885-Cleveland.txt', '1889-Harrison.txt', '1893-Cleveland.txt', '1897-McKinley.tx
t', '1901-McKinley.txt', '1905-Roosevelt.txt', '1909-Taft.txt', '1913-Wilson.txt
', '1917-Wilson.txt', '1921-Harding.txt', '1925-Coolidge.txt', '1929-Hoover.txt'
, '1933-Roosevelt.txt', '1937-Roosevelt.txt', '1941-Roosevelt.txt', '1945-Roosev
elt.txt', '1949-Truman.txt', '1953-Eisenhower.txt', '1957-Eisenhower.txt', '1961
-Kennedy.txt', '1965-Johnson.txt', '1969-Nixon.txt', '1973-Nixon.txt', '1977-Car
ter.txt', '1981-Reagan.txt', '1985-Reagan.txt', '1989-Bush.txt', '1993-Clinton.t
xt', '1997-Clinton.txt', '2001-Bush.txt', '2005-Bush.txt', '2009-Obama.txt']
>>>
```

```
>>> [fileid[:4] for fileid in inaugural.fileids()]
['1789', '1793', '1797', '1801', '1805', '1809', '1813', '1817', '1821', '1825',
 '1829', '1833', '1837', '1841', '1845', '1849', '1853', '1857', '1861', '1865',
 '1869', '1873', '1877', '1881', '1885', '1889', '1893', '1897', '1901', '1905',
 '1909', '1913', '1917', '1921', '1925', '1929', '1933', '1937', '1941', '1945',
 '1949', '1953', '1957', '1961', '1965', '1969', '1973', '1977', '1981', '1985',
 '1989', '1993', '1997', '2001', '2005', '2009']
>>>
```

Segundo resultado:

NLTK y corpus lingüísticos (11)

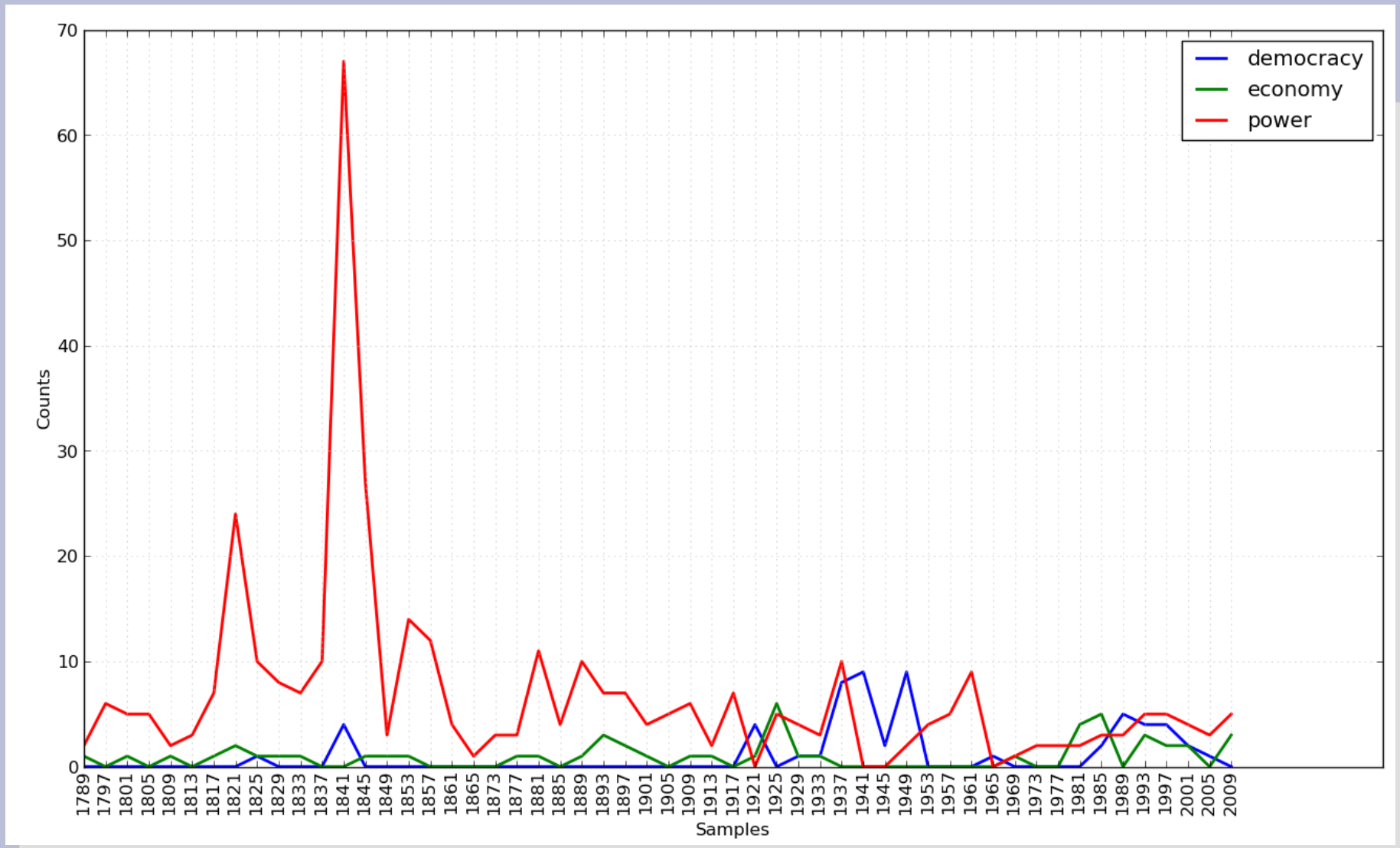
Pasemos entonces a hacer nuestro gráfico, para lo que necesitamos de nuevo la función **Conditional FreqDist**. Del mismo modo, vamos a visualizar la frecuencia del uso de 3 palabras: *democracy*, *economy* y *power*.

```
>>> cfd_inaugual01 = nltk.ConditionalFreqDist(
                        (target, fileid[:4])
                        for fileid in inaugural.fileids()
                        for w in inaugural.words(fileid)
                        for target in ['democracy', 'economy', 'power']
                        if w.lower().startswith(target))

>>> cfd_inaugural01.plot()
```

Veamos el resultado en la siguiente lámina:

NLTK y corpus lingüísticos (12)



NLTK y etiquetado (1)

Para hacer nuestras pruebas, requerimos un nuevo módulo:

```
import os
```

A continuación, probemos esta instrucción:

```
nltk.corpus.brown.tagged_words()
```

El resultado, como se darán cuenta, es la oración inicial del sub-corpus *News* del Brown:

```
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ...]
```

NLTK y etiquetado (2)

Otro corpus etiquetado muy recurrido parta estas tareas es el **Treebank**, el cual ha establecido una serie de estándares útiles para diseñar anotaciones morfosintácticas. Podemos acceder a él del siguiente modo:

```
nltk.corpus.treebank.tagged_words()
```

Al igual que el Brown, la muestra es un fragmento de una oración:

```
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ...]
```

NLTK y etiquetado (3)

Vamos a implementar un proceso de etiquetado que considere:

1. La inserción de un archivo, el cual vamos a transformar en una cadena de caracteres.

2. Posteriormente, lo dividimos en palabras.

3. Una vez convertido en lista de palabras, insertamos etiquetas en tal documento.

4. Presentar los resultados en un nuevo texto plano, en donde las palabras van asociadas a una etiqueta correspondiente.

NLTK y etiquetado (4)

1. Importando archivo:

```
Cadena_Turing01 = open('Escritorio/turing01.txt', 'rU').read()
```

2. Comprobamos:

```
type(Cadena_Turing01)

len(Cadena_Turing01)
```

NLTK y etiquetado (5)

3. Seccionemos nuestra cadena en palabras, usando la función **word_tokenize**:

```
Tokens_Turing01 = nltk.word_tokenize(Cadena_Turing01)
```

4. Comprobamos:

```
type(Tokens_Turing01)

len(Tokens_Turing01)
```

NLTK y etiquetado (6)

5. Ahora, vamos a añadir etiquetas morfosintácticas a esta lista de tokens, usando la función **pos_tag**, la cual importa tales etiquetas desde el corpus Brown:

```
Tagged_Turing01 = nltk.pos_tag(Tokens_Turing01)
```

4. Comprobamos:

```
type(Tagged_Turing01)

len(Tagged_Turing01)

Tagged_Turing01[0:101]
```

NLTK y etiquetado *again* (8)

6. Para concluir, vamos a escribir un código que nos permita imprimir nuestros resultados en un archivo de texto plano. Así, lo que hacemos es:

```
Final_Turing01 = open('Escritorio/Resultados_Turing01.txt', 'w')
for elem in Tagged_Turing01:
        Final_Turing01.write(str(elem)+" ")

Final_Turing01.close()
```

Vamos a explicar este código. Lo que estamos pidiendo es: *en un nuevo archivo llamado **Final_Ariel01**, el cual se imprimirá con el nombre **Resultados_Ariel01.txt**, escribe los elementos que aparecen en el archivo **Tagger_Ariel01**, transformándolos de una cadena a una lista de objetos. Al final, cierra el archivo **Final_Ariel01**.*

NLTK y etiquetado (7)

Resultado:

Resultados_Turing01: Bloc de notas

Archivo Edición Formato Ver Ayuda

('The', 'DT') ('Turing', 'NNP') ('Test', 'NNP') ('First', 'NNP') ('published', 'VBD') ('Wed', 'NNP') ('Apr', 'NNP') ('9', 'CD') (',', ',') ('2003', 'CD') (';', ':') ('substantive', 'JJ') ('revision', 'NN') ('Wed', 'NNP') ('Jan', 'NNP') ('26', 'CD') (',', ',') ('2011', 'CD') ('The', 'DT') ('phrase', 'NN') ('\xe2', ':') ('\x80', ':') ('\x9c', ':') ('The', 'DT') ('Turing', 'NNP') ('Test', 'NNP') ('\xe2', 'NNP') ('\x80', 'NNP') ('\x9d', 'NNP') ('is', 'VBZ') ('most', 'RBS') ('properly', 'RB') ('used', 'JJ') ('to', 'TO') ('refer', 'VB') ('to', 'TO') ('a', 'DT') ('proposal', 'NN') ('made', 'VBN') ('by', 'IN') ('Turing', 'NNP') ('(', 'NNP') ('1950', 'CD') (')', 'CD') ('as', 'IN') ('a', 'DT') ('way', 'NN') ('of', 'IN') ('dealing', 'VBG') ('with', 'IN') ('the', 'DT') ('question', 'NN') ('whether', 'IN') ('machines', 'NNS') ('can', 'MD') ('think.', 'NNP') ('According', 'NNP') ('to', 'TO') ('Turing', 'NNP') (',', ',') ('the', 'DT') ('question', 'NN') ('whether', 'IN') ('machines', 'NNS') ('can', 'MD') ('think', 'VB') ('is', 'VBZ') ('itself', 'PRP') ('\xe2', ':') ('\x80', ':') ('\x9c', ':') ('too', 'RB') ('meaningless', 'JJ') ('\xe2', 'NN') ('\x80', ':') ('\x9d', ':') ('to', 'TO') ('deserve', 'VB') ('discussion', 'NN') ('(', ':') ('442', 'CD') (')', 'CD') ('.', '.') ('However', 'RB') (',', ',') ('if', 'IN') ('we', 'PRP') ('consider', 'VBP') ('the', 'DT') ('more', 'RBR') ('precise', 'JJ') ('\xe2', 'NN') ('\x80', ':') ('\x94', ':') ('and', 'CC') ('somehow', 'NN') ('related', 'VBD') ('\xe2', 'CD') ('\x80', 'CD') ('\x94', 'CD') ('question', 'NN') ('whether', 'IN') ('a', 'DT') ('digital', 'JJ') ('computer', 'NN') ('can', 'MD') ('do', 'VB') ('well', 'RB') ('in', 'IN') ('a', 'DT') ('certain', 'JJ') ('kind', 'NN') ('of', 'IN') ('game', 'NN') ('that', 'IN') ('Turing', 'NNP') ('describes', 'VBZ') ('(', ':') ('\xe2', ':') ('\x80', ':') ('\x9c', ':') ('The', 'DT') ('Imitation', 'NNP') ('Game', 'NNP') ('\xe2', 'NNP') ('\x80', 'NNP') ('\x9d', 'NNP') (')', 'NNP') (',', ',') ('then', 'RB') ('\xe2', ':') ('\x80', ':') ('\x94', ':') ('at', 'IN') ('least', 'JJS') ('in', 'IN') ('Turing', 'NNP') ("'s", 'POS') ('eyes', 'NNS') ('\xe2', ':') ('\x80', ':') ('\x94', ':') ('we', 'PRP') ('do', 'VBP') ('have', 'VBP') ('a', 'DT') ('question', 'NN') ('that', 'WDT') ('admits', 'VBZ') ('of', 'IN') ('precise', 'NN') ('discussion.', 'NNP') ('Moreover', 'NNP') (',', ',') ('as', 'IN') ('we', 'PRP') ('shall', 'MD') ('see', 'VB') (',', ',') ('Turing', 'NNP') ('himself', 'PRP') ('thought', 'VBD') ('that', 'IN') ('it', 'PRP') ('would', 'MD') ('not', 'RB') ('be', 'VB') ('too', 'RB') ('long', 'RB') ('before', 'IN') ('we', 'PRP') ('did', 'VBD') ('have', 'VB') ('digital', 'JJ') ('computers', 'NNS') ('that', 'WDT') ('could', 'MD') ('\xe2', 'VB') ('\x80', ':') ('\x9c', ':') ('do', 'VBP') ('well', 'RB') ('\xe2', ':') ('\x80', ':') ('\x9d', ':') ('in', 'IN') ('the', 'DT') ('Imitation', 'NNP') ('Game', 'NNP') ('.', '.') ('The', 'DT') ('phrase', 'NN') ('\xe2', ':') ('\x80', ':') ('\x9c', ':') ('The', 'DT') ('Turing', 'NNP') ('Test', 'NNP') ('\xe2', 'NNP') ('\x80', 'NNP') ('\x9d', 'NNP') ('is', 'VBZ') ('sometimes', 'RB') ('used', 'VBN') ('more', 'RBR') ('generally', 'RB') ('to', 'TO') ('refer', 'VB') ('to', 'TO') ('some', 'DT') ('kinds', 'NNS') ('of', 'IN') ('behavioural', 'JJ') ('tests', 'NNS') ('for', 'IN') ('the', 'DT') ('presence', 'NN') ('of', 'IN') ('mind', 'NN') (',', ',') ('or', 'CC') ('thought', 'VBD') (',', ',') ('or', 'CC') ('intelligence', 'NN') ('in', 'IN') ('putatively', 'RB') ('minded', 'VBN') ('entities.', 'NNP') ('So', 'NNP') (',', ',') ('for', 'IN') ('example', 'NN') (',', ',') ('it', 'PRP') ('is', 'VBZ') ('sometimes', 'RB') ('suggested', 'VBN') ('that', 'IN') ('The', 'DT') ('Turing', 'NNP') ('Test', 'NNP') ('is', 'VBZ') ('prefigured', 'VBN') ('in', 'IN') ('Descartes', 'NNP') ("'", 'POS') ('Discourse', 'NNP') ('on', 'IN') ('the', 'DT') ('Method.', 'NNP') ('(', 'NNP') ('Copeland', 'NNP') ('(', 'NNP') ('2000', 'CD') (':', ':') ('527', 'CD') (')', 'CD') ('finds', 'NNS') ('an', 'DT') ('anticipation', 'NN') ('of', 'IN') ('the', 'DT') ('test', 'NN') ('in', 'IN') ('the', 'DT') ('1668', 'CD') ('writings', 'NNS') ('of', 'IN') ('the', 'DT') ('Cartesian', 'NNP') ('de', 'IN') ('Cordemoy.', 'NNP') ('Gunderson', 'NNP') ('(', 'NNP') ('1964', 'CD') (')', 'CD') ('provides', 'NNS') ('an', 'DT') ('early', 'JJ') ('instance', 'NN') ('of', 'IN') ('those', 'DT') ('who', 'WP') ('find', 'VBP') ('that', 'IN') ('Turing', 'NNP') ("'s", 'POS') ('work', 'NN') ('is', 'VBZ') ('foreshadowed', 'VBN') ('in', 'IN') ('the', 'DT') ('work', 'NN') ('of',

ES 16:56 09-05-2012

NLTK y otros etiquetadores (1)

Así visto, parece que la cuestión de anotar de manera automática un documento o un corpus es muy sencilla, y sí, digamos que de algún modo el uso de estas herramientas debiera facilitarnos ese trabajo.

El etiquetador de NLTK no es el único que hay: por lo menos podemos mencionar otros dos: un viejo conocido, el **TreeTagger**, y uno más reciente, el **Freeling**.

TreeTagger ·

www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html

FreeLing 2.2

AN OPEN SOURCE SUITE OF LANGUAGE ANALYZERS

http://nlp.lsi.upc.edu/freeling/

NLTK y otros etiquetadores (2)

Consideremos los resultados que nos arroja el TreeTagger: como saben, este etiquetador nos muestra un tripleta: palabra/etiqueta/lema:

word	pos	lemma
The	DT	the
TreeTagger	NP	TreeTagger
is	VBZ	be
easy	JJ	easy
to	TO	to
use	VB	use
.	SENT	.

¿Qué tan bueno o malo es esta clase de etiquetado en comparación con el de NLTK? Y más importante: ¿cuál de los dos etiquetadores comete menos errores?

Una buena tarea sería precisamente elaborar un método estadístico y lingüístico, que permita evaluar cuáles son las ventajas y desventajas de cada uno de estos dos etiquetadores.

NLTK en español: lo que hay (1)

Siendo demasiado francos, si bien NLTK cuenta con un corpus en español, las colecciones de árboles sintácticos llamadas CESS Treebanks (que son derivados del proyecto Freeling), en español y catalán, estos recursos **no permiten todavía desarrollar un proceso de etiquetado de nuevos documentos, como pasa con el corpus Brown.** Las razones son varias:

1. Los corpus CESS están diseñados para mostrar resultados respecto a las consultas que hagamos en ellos, **pero no permiten implementar algún proceso de aprendizaje para un etiquetador en español.**

2. Todavía no se ha desarrollado algún método en NLTK para entrenar un etiquetador en español.

3. No hay un corpus etiquetado de referencia en español, el cual sea de libre uso, y pueda formar parte de las librerías de NLTK.

4. No hay criterios unificados sobre el proceso de etiquetado de palabras en español.

NLTK en español: lo que hay (2)

Si probamos la *demo* que ofrece Freeling en línea, podemos ver que se cuentan con algunos recursos para el etiquetado de oraciones, p.e.:

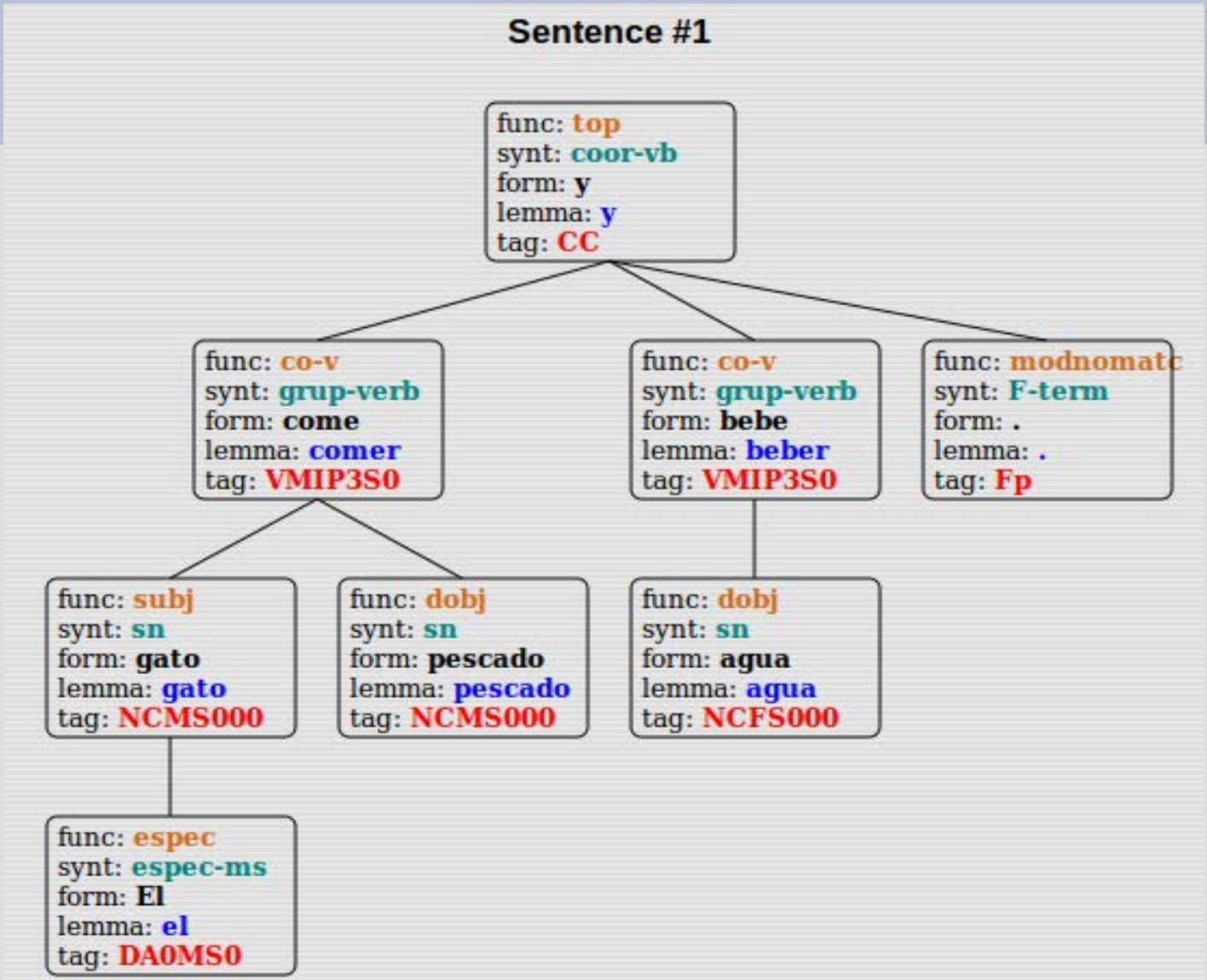
Analysis Results

Sentence #1

El	gato	come	pescado	y	bebe	agua	.
el	*gato*	*comer*	*pescado*	*y*	*beber*	*agua*	*.*
DA0MS0	NCMS000	VMIP3S0	NCMS000	CC	VMIP3S0	NCFS000	Fp

Incluso podemos graficar árboles sintácticos con cierto grado de complejidad, digamos un *parseo* basado en gramática de dependencias. Veamos la siguiente lámina:

NLTK en español: lo que hay (3)



NLTK en español: lo que hay (4)

Esto último sí podemos hacerlo desde NLTK: lanzar consultas al corpus CESS en español, mostrar sus resultados, e incluso generar árboles sintácticos con las oraciones que hay en tal corpus. Así, si ocupan la instrucción:

```
from  nltk.corpus import cess_esp

nltk.corpus.cess_esp.words()
```

Nos muestra el ejemplo de una oración:

```
['El', 'grupo', 'estatal', 'Electricit\xe9_de_France', ...]
```

NLTK en español: lo que hay (5)

Las etiquetas que ocupa este corpus son una variante del sistema de anotación EAGLES,el cual ustedes ya conocen. Si escriben:

```
nltk.corpus.cess_esp.tagged_words()
```

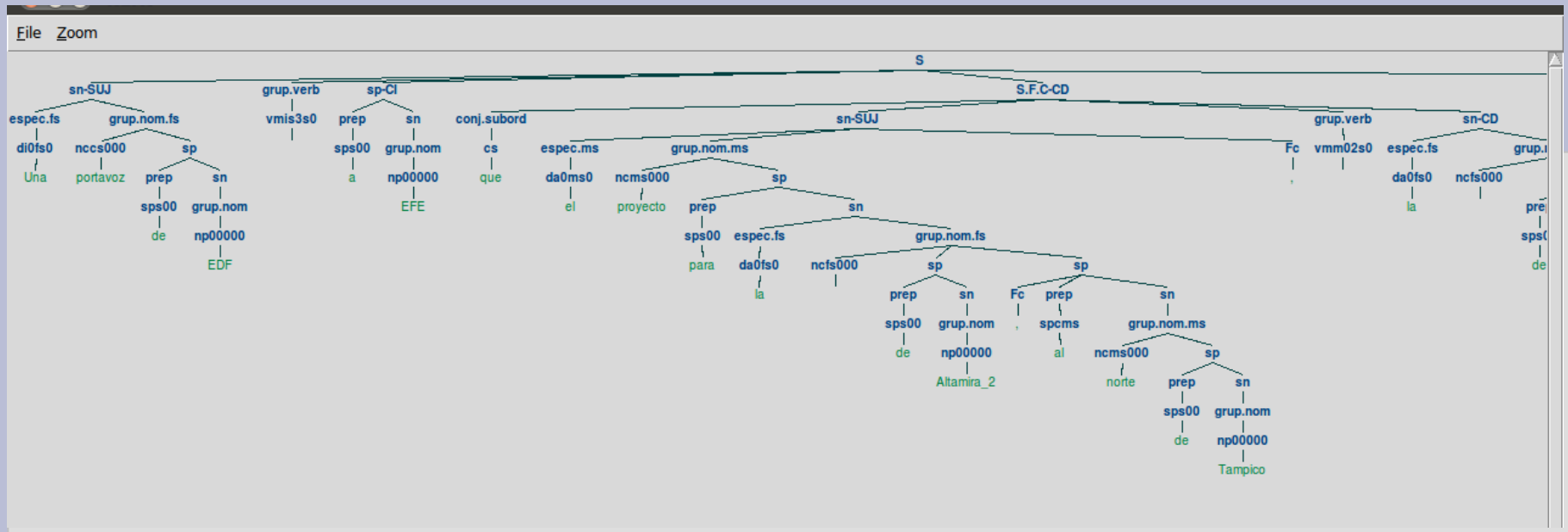
Nos arroja como resultado:

```
[('El', 'da0ms0'), ('grupo', 'ncms000'), ...]
```

Finalmente, podemos generar un esquema arbóreo, seleccionado alguna de las oraciones que cuenta este corpus. Usemos esta instrucción, y veamos el resultado en la siguiente lámina:

```
Tree01 = nltk.corpus.cess_esp.parsed_sents()[1].draw()
```

NLTK en español: lo que hay (6)



A manera de comentario final: hay muchas cosas por hacer todavía respecto al uso de NLTK para el procesamiento de corpus, pero lo importante es que hay soluciones. ¿Qué posibles vías creen que puedan ser útiles para aprovechar estos recursos para el análisis de textos en español?

Esta cuestión, claro, es para discutirla con una taza de café en la mano.

Algunas referencias

Para aquellos que quieran aprender más sobre cómo programar en Python, existen un sinnúmero de manuales libres disponibles en línea, los cuales pueden consultar desde el sitio de Python:

http://wiki.python.org/moin/BeginnersGuide/NonProgrammers

En particular, uno muy bueno es:

How to Think Like a Computer Scientist:
http://openbookproject.net/thinkcs/python/english2e/index.html

Gracias por su atención

Blog del curso:
http://cesaraguilar.weebly.com/meacutetodos-y-teacutecnicas-de-investigacioacuten-cuantitativa.html