Clase 3: Archivos (texto, csv, dbf, Excel)

Archivos

En líneas generales los archivos se leen y se escriben en 3 pasos:

Lectura:

1. Abrir (open)
2. Leer (read, readlines, readline)
3. Cerrar (close)

Escritura:

1. Abrir (open)
2. Guardar (write)
3. Cerrar (close)

Leyendo un archivo

La función open crea una referencia a un archivo (usualmente llamado *file handle*) que se usa para leer los datos:

```
open(filename[, mode[, bufsize]])
```

Ejemplo de generación de filehandler asociado al archivo *mi_archivo.txt*.

```
fh = open('mi_archivo.txt','r')
```

La 'r' indica "modo de lectura" y es el modo por defecto (por lo que se podría obviar la 'r').

Sobre el file handle se puede:

- read(n): Lee n bytes, por defecto lee el archivo entero.
- readline(): Devuelve str con una sola linea.
- readlines(): Devuelve una lista con una cadena como elemento por cada línea del archivo.

Distintas maneras de leer el contenido de un archivo:

```
fh = open('archivo.txt')
contenido = fh.read()
print contenido

fh = open('archivo.txt')
contenido = fh.readlines()
print contenido

contenido = ''
fh = open('archivo.txt')
while True:
  line = fh.readline()
  contenido += line
  if line=''
```

```
    break
print contenido

# Para todos los casos:
fh.close()
```

Acceso secuencial al contenido de un archivo:

```
fh = open('archivo.txt')
contenido = ''
for linea in fh:
  contenido += linea
fh.close()
```

Leyendo con 'with' (Python 2.6 en adelante)

Manera genérica:

```
with <Expresion> as <Variable>:
  <Codigo>
```

Ejemplo:

```
with open('archivo.txt') as fh:
  for line in fh:
    print(line)
```

La ventaja de *with* es que al salir del bloque, se ejecuta un método especial que cierra el archivo automaticamente.

En el caso que querramos leer un archivo con un encoding en particular:

```
import codecs
f = codecs.open('arch.txt', encoding='utf-8')
```

Escribiendo archivos

Modos de escritura:

- w: Write, graba un archivo nuevo, si existe, lo borrar.
- a: Append (agregar), agrega información al final de un archivo pre-existente. Si no existe, crea uno nuevo (uso típico: logs).

Ejemplo:

```
fh = open('/home/yo/archivo.txt','w')
fh.write('1\n2\n3\n4\n5\n')
fh.close()
```

Archivos CSV (Comma separated file)

Este tipo de archivos es un "estandar de facto" para muchas aplicaciones, incluyendo planillas de cálculo y bases de datos.

Archivo CSV:

```
Sebastián,Perez,33,23566777
Jose,Martinez,23,42121329
Karina,Gonzalez,32,24159857
Maria Laura,Yañes,19,43852144
```

Archivo CSV con otro separador (;):

```
Sebastián;Perez;33;23566777
Jose;Martinez;23;42121329
Karina;Gonzalez;32;24159857
Maria Laura;Yañes;19;43852144
```

Leyendo CSV sin usar módulo CSV

Con las herramientas vistas hasta este momento podemos procesar dichos archivos:

```
fh = open('archivo.txt')
for line in fh:
  linea = line.split(',')
  # procesar elementos:
  nombre = linea[0]
  apellido = linea[1]
  # etc, etc
fh.close()
```

Aunqe Python tiene un módulo especialmente preparado para esto:

CSV con módulo CSV

Uso del módulo CSV:

```
import csv
f = open('fn.csv')
lns = csv.reader(f)
for line in lns:
  nombre = line[0]
  apellido = line[1]
#Cambiando separador (delimitador):
f = open('fn.csv')
lns = csv.reader(f,delimiter=';')
```

Dialectos

No todos los CSV son hechos iguales, hay diferencias sutiles como las comillas en los campos de texto y otras diferencias menores que agregan algunos programas. Por eso existe un parámetro optativo llamado **dialect**. Estos son los valores que puede tomar:

```
>>> csv.list_dialects()
['excel-tab', 'excel']
```

Por lo tanto, para leer un archivo csv generado por Excel:

```
import csv
f = open('from_excel.csv')
lines = csv.reader(f,dialect="excel")
```

Escribiendo archivos CSV

```
>>> import csv
>>> f = open('eggs.csv', 'w')
>>> obj = csv.writer(f, delimiter=' ', quotechar='|', quoting=csv.QUOTE_MINIMAL)
>>> obj.writerow(['algo', 'algo mas', 'nada'])
>>> f.close()
```

Archivos dbf

Este formato de archivo corresponde a un tipo de base de datos compatible de dBASE diseñado hace mas de 30 años que no tiene las funcionalidades de las bases de datos actuales (basadas en SQL), por lo que no se recomienda su uso. Se lo presenta aca con la idea de poder leer datos en este tipo de bases para transformarlo en un formato actual.

Para acceder a estos archivos se usa el módulo **dbfpy**. Este módulo no es parte de Python, hay que bajarlo e instalarlo:

```
# easy_install dbfpy
```

Apertura de la base de datos:

```
from dbfpy import dbf

f1 = '/home/sbassi/BENEFI.DBF'
dbf1 = dbf.Dbf(f1)
```

Leer el nombre de los campos:

```
>>> print dbf1.fieldNames
['CODRUB', 'DESCRI']
```

Ver el tamaño:

```
>>> len(dbf1)
43
>>> dbf1.header.recordCount
43
```

Acceder a los elementos:

Se accede de la misma manera que a cualquier iterable ordenado.

```
>>> dbf2[20]
CODRUB: 24 (<type 'int'>)
DESCRI: Equipos (<type 'str'>)
>>> dbf2[20][0]
24
>>> dbf2[20][1]
```

```
'Equipos'
```

Lectura y escritura de Excel: xlrd y xlwt

Estos módulos no vienen con Python, pero pueden ser instalados facilmente con *Easy_install* (http://pypi.python.org/pypi/setuptools).

Leyendo Excel

El siguiente programa usa **xlrd** para leer la primera hoja de una planilla llamada sampledata.xls que contiene 2 columnas. Cada par de datos que lee es guardado como un item en un diccionario llamado *iedb*. El módulo **xlrd** tiene que estar instalado para que esto funcione.

```
import xlrd
iedb = {} # diccionario vacio
book = xlrd.open_workbook('datos.xls')
sh = book.sheet_by_index(0)
for i in range(1,sh.nrows):
  iedb[sh.cell_value(rowx=i, colx=1)] = \
       sh.cell_value(rowx=i, colx=2)
```

Creando planillas Excel

Este programa crea una planilla usando **xlwt**:

```
import xlwt
list1 = [1,2,3,4,5]
list2 = [234,267,281,301,331]
wb = xlwt.Workbook()
ws = wb.add_sheet('Primera hoja')
ws.write(0,0,'Columna A')
ws.write(0,1,'Columna B')
i = 1
for x,y in zip(list1,list2):
  # con zip recorro las 2
  # listas a la vez
  ws.write(i,0,x)
  ws.write(i,1,y)
  i += 1
wb.save('mynewfile.xls')
```

Pickle: Persistencia simple de datos

Cualquier dato (variable) en Python puede grabarse a disco usando **pickle** o **cPickle** (en Python 2.x). La única diferencia entre pickle y cPickle es que este último está escrito en C y por lo tanto es mas rápido. En el caso de Python 3, existe solamente **pickle** y tiene componentes en C y en Python y se activa la versión en C cada vez que es posible y de manera transparente al usuario.

Grabar

```
>>> d = {'blue':'azul','red':'rojo'}
>>> d
{'blue': 'azul', 'red': 'rojo'}
>>> import cPickle
>>> fh = open('spd.data','w')
>>> cPickle.dump(d, fh)
>>> fh.close()
```

Leer

Con *load* "levantamos" el objeto que había sido almacenado con *dump*:

```
>>> spd = cPickle.load(open('spd.data'))
>>> spd
{'blue': 'azul', 'red': 'rojo'}
```

Mas información

- Archivos: http://www.devshed.com/c/a/Python/File-Management-in-Python
- Unicode: http://docs.python.org/howto/unicode.html
- Pickle: http://docs.python.org/library/pickle.html