

Python

Nando Quintana Hernández
fquintana@codesyntax.com



Algunos derechos reservados...

<http://creativecommons.org/licenses/by-sa/2.5/es/>

(Reconocimiento: Diego López de Ipiña)

Contenido

- Sintaxis.
- Instrucciones básicas.
- Tipo de datos.
- Orientación a objetos
- Paquetes esenciales de Python:
XML, bases de datos,
programación web.

Guido van Rossum

- Guido van Rossum
 - ▶ Da este nombre al lenguaje inspirado por el popular grupo cómico británico Monty Python
 - ▶ creó Python durante unas vacaciones de navidad en las que (al parecer) se estaba aburriendo

Hola Mundo en Python

```
#!/usr/local/bin/python  
  
print "Hola Mundo" # "Hola Mundo"  
print "hola", "mundo" # "hola mundo"  
print "Hola" + "Mundo" # "HolaMundo"
```

Características de Python I

- Muy legible y elegante
 - ▶ Imposible escribir código ofuscado
- Simple y poderoso
 - ▶ Minimalista: todo aquello innecesario no hay que escribirlo (;, {, }, '\n')
 - ▶ Muy denso: poco código hace mucho
 - ▶ Soporta objetos y estructuras de datos de alto nivel: strings, listas, diccionarios, etc.
 - ▶ Múltiples niveles de organizar código: funciones, clases, módulos, y paquetes
 - Python standard library: contiene un sinfín de clases de utilidad
 - ▶ Si hay áreas que son lentas se pueden reemplazar por plugins en C o C++, siguiendo la API para extender o empotrar Python en una aplicación, o a través de herramientas como SWIG, sip, Psyco o Pyrex.

Características de Python II

- De scripting
 - ▶ No tienes que declarar constantes y variables antes de utilizarlas
 - ▶ No requiere paso de compilación/linkage
 - La primera vez que se ejecuta un script de Python se compila y genera bytecode que es luego interpretado
 - ▶ Alta velocidad de desarrollo y buen rendimiento
- Código interoperable (como en Java "write once run everywhere")
 - ▶ Se puede utilizar en múltiples plataformas (más aún que Java)
 - ▶ Puedes incluso ejecutar Python dentro de una JVM (Jython)

Características de Python II

- Free Software
 - ▶ Razón por la cual la Python Library sigue creciendo
- De propósito general
 - ▶ Puedes hacer en Python todo lo que puedes hacer con C# o Java, o más

Peculiaridades sintácticas

- Python usa tabulación (o espaciado) para mostrar estructura de bloques
 - ▶ Tabula una vez para indicar comienzo de bloque
 - ▶ Des-tabula para indicar el final del bloque

Código en C/Java	Código en Python
<pre>if (x) { if (y) { f1(); } f2(); }</pre>	<pre>if x: if y: f1() f2()</pre>

Python vs. Perl

- Los dos están basados en un buen entendimiento de las herramientas necesarias para resolver problemas
 - ▶ Perl está basado en awk, sed, y shell scripting y su misión es hacer las tareas de administradores de sistemas más sencillas
 - ▶ Python está basado e inspirando en OOP (Object-oriented programming)
 - Guido van Rossum diseñó un lenguaje simple, poderoso, y elegante orientado a la creación de sistemas a partir de componentes

Python vs. Java

- Java es un lenguaje de programación muy completo que ofrece:
 - ▶ Amplio abanico de tipos de datos
 - ▶ Soporte para threads
 - ▶ Tipado estático y fuerte
 - ▶ Y mucho más ...
- Python es un lenguaje de scripting:
 - ▶ Tipado dinámico y fuerte
 - Cuadratura del círculo:
 - ▶ tipado dinámico no es tipado debil
 - ▶ Todo lo que puedes hacer con Java también lo puedes hacer con Python
 - Incluso puedes acceder a través de Python a las API de Java si usas Jython (<http://www.jython.org>)

Python vs. Jython

- Python
 - ▶ También llamado Cpython
 - ▶ Implementación del lenguaje Python en C
 - ▶ Python C API permite extender Python con librerías realizadas en C
 - ▶ Partes que requieren mayor rendimiento en Python están implementadas en C o C++ y tan sólo contienen una pequeña capa de Python encima
- Jython
 - ▶ Implementación de Python en Java
 - ▶ Permite acceder a todas las APIs de Java
 - P.E. Podemos producir Swing GUIs desde Python

¿Para qué [no] es útil?

- Python no es el lenguaje perfecto, no es bueno para:
 - ▶ Programación de bajo nivel (system-programming), como programación de drivers y kernels
 - Python es de demasiado alto nivel, no hay control directo sobre memoria y otras tareas de bajo nivel
 - ▶ Aplicaciones que requieren alta capacidad de computo
 - No hay nada mejor para este tipo de aplicaciones que el viejo C
- Python es ideal:
 - ▶ Como lenguaje "pegamento" para combinar varios componentes juntos
 - ▶ Para llevar a cabo prototipos de sistema
 - ▶ Para la elaboración de aplicaciones cliente
 - ▶ Para desarrollo web y de sistemas distribuidos
 - ▶ Para el desarrollo de tareas científicas, en los que hay que simular y prototipar rápidamente

Instalar Python

```
$ wget http://www.python.org/ftp/python/2.4.3/Python-2.4.3.tar.bz2
$ tar -jxvf Python-2.4.3.tar.bz2
$ cd Python-2.4.3
$ ./configure --prefix=/opt/Python-2.4.3;
$ make; make install
```

Usando Python desde línea de comando

- Para arrancar el intérprete (Python interactivo) ejecutar:

```
$ python
Python 2.4 (#60, Nov 30 2004, 11:49:19) [MSC v.1310 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

- Un comando simple:

```
>>> print "Hola Mundo"
Hola Mundo
>>>
```

- Para salir del intérprete Ctrl-D (o Ctrl-Z en hasefroch) o:

```
>>> import sys
>>> sys.exit()
$
```

Ejecutando programa ho lamundo.py

- Python desde script:
 - ▶ Guardar las siguientes sentencias en fichero:
ho lamundo.py

```
#!/usr/local/bin/python
```

```
print "Hola mundo!"
```

- Ejecutar el script desde línea de comando:

```
$ ./helloworld.py  
Hola mundo!  
$
```

Sentencias y bloques

- Las sentencias acaban en nueva línea, no en ;
- Los bloques son indicados por *tabulación* que sigue a una sentencia acabada en ':'. E.j. (bloque.py):

```
#!/usr/local/bin/python
```

```
name = "Nando1" # asignación de valor a variable
if name == "Nando":
    print "Aupa Nando"
else:
    print "¿Quién eres?"
    print "¡No eres Nando!"
```

```
$ ./bloque.py
¿Quién eres?
¡No eres Nando!
```

Identificadores

- Los identificadores sirven para nombrar variables, funciones y módulos
 - ▶ Deben empezar con un carácter no numérico y contener letras, números y '_'
 - ▶ Python es *case sensitive* (sensible a la capitalización)
- Palabras reservadas:
 - ▶ `and elif global or assert else if pass break except import print class exec in raise continue finally is return def for lambda try del from not while`
- Variables y funciones delimitadas por `__` corresponden a símbolos implícitamente definidos:
 - ▶ `__name__` nombre de función
 - ▶ `__doc__` documentación sobre una función
 - ▶ `__init__()` constructor de una clase
 - ▶ `__dict__`, diccionario utilizado para guardar los atributos de un objeto

Tipos de datos I

- Numéricos (integer, long integer, floating-point, and complex)

```
>>> x = 4
>>> int (x)
4
>>> long(x)
4L
>>> float(x)
4.0
>>> complex (4, .2)
(4+0.2j)
```

- Booleanos
True y False

Tipos de datos II

- Strings, delimitados por un par de (' , " , "" "")
 - ▶ Dos string juntos sin delimitador se unen

```
>>> print "Hi" "there"
Hithere
```
 - ▶ Los códigos de escape se expresan a través de '\':

```
>>>print '\n'
```
 - ▶ Raw strings

```
>>> print r'\n\' # no se 'escapa' \n
```
 - ▶ Es lo mismo ' que ", p.e. "\\[foo\\]" r'[foo\]'
 - ▶ Algunos de los métodos que se pueden aplicar a un string son:

```
>>> len('La vida es mejor con Python.')
>>> 34
>>> 'La vida es mejor con Python.'.upper()
'LA VIDA ES MEJOR CON PYTHON'
>>> "La vida es mejor con Python".find("Python")
27
>>> "La vida es mejor con Python".find('Perl')
-1
>>> 'La vida es mejor con Python'.replace('Python', 'Jython')
'La vida es mejor con Jython'
```

Tipos de datos III

- El módulo `string` de la Python library define métodos para manipulación de strings:

```
>>> import string
>>> s1 = 'La vida es mejor con Python'
>>> string.find(s1, 'Python')
21
```

- `'%'` es el operador de formateo de cadenas:

```
>>> provincia = 'Araba'
>>> "La capital de %s es %s" % (provincia, "Gasteiz")
'La capital de Araba es Gasteiz'
```

- ▶ Los caracteres de formateo son los mismos que en C, p.e. `d`, `f`, `x`

Tipos de datos IV

- Para poder escribir caracteres con acentos es necesario introducir la siguiente línea al comienzo de un programa Python:

```
# -*- coding: iso-8859-1 -*-
```

- Los strings en formato unicode se declaran precediendo el string de una 'u':

- ▶ `print u'¿Qué tal estás?'`

Tipos de datos V

- Listas []

- ▶ Indexadas por un entero comienzan en 0:

```
>>> meses = ["Enero", "Febrero"]
```

```
>>> print meses[0]
```

```
Enero
```

```
>>> meses.append("Marzo")
```

```
>>> print meses
```

```
['Enero', 'Febrero', 'Marzo']
```

- ▶ Dos puntos (:) es el operador de rodajas, permite trabajar con una porción de la lista, el elemento indicado por el segundo parámetro no se incluye:

```
>>> print meses[1:2]
```

```
['Febrero']
```

- ▶ Más (+) es el operador de concatenación:

```
>>> print meses+meses
```

```
['Enero', 'Febrero', 'Marzo', 'Enero', 'Febrero', 'Marzo']
```

Tipos de datos VI

- ▶ Las listas pueden contener cualquier tipo de objetos Python:

```
>>> meses.append(meses)
>>> print meses
['Enero', 'Febrero', 'Marzo', ['Enero', 'Febrero', 'Marzo']]
>>> meses.append(1)
['Enero', 'Febrero', 'Marzo', ['Enero', 'Febrero', 'Marzo'], 1]
```

- ▶ Para añadir un elemento a una lista:

```
>>> items = [4, 6]
>>> items.insert(0, -1)
>>> items
[-1, 4, 6]
```

- ▶ Para usar una lista como una pila, se pueden usar `append` y `pop`:

```
>>> items.append(555)
>>> items [-1, 4, 6, 555]
>>> items.pop()
555
>>> items # [-1, 4, 6]
```

Tipos de datos VII

- Tuplas (), lo mismo que listas, pero no se pueden modificar

```
>>> mitupla = ('a', 1, "hola")
>>> mitupla[2]
'hola'
>>> dir(mitupla)
['__add__', '__class__', '__contains__', '__delattr__',
 '__doc__', '__eq__', '__ge__', '__getattr__',
 '__getitem__', '__getnewargs__', '__getslice__', '__gt__',
 '__hash__', '__init__', '__iter__', '__le__', '__len__',
 '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__rmul__', '__setattr__',
 '__str__']
```

Tipos de datos VIII

- Diccionarios {} arrays asociativos o mapas, indexados por una clave, la cual puede ser cualquier objeto Python, aunque normalmente es una tupla:

```
>>> mydict = {"altura" : "media", "habilidad" : "intermedia",  
"salario" : 1000 }  
>>> print mydict  
{'altura': 'media', 'habilidad': 'intermedia', 'salario': 1000}  
>>> print mydict["habilidad"]  
intermedia
```

- ▶ Puedes comprobar la existencia de una clave en un diccionario usando `has_key`:

```
if mydict.has_key('altura'):  
    print 'Nodo encontrado'
```

- ▶ Lo mismo se podría hacer:

```
>>> if 'altura' in mydict:  
>>>     print 'Nodo encontrado'
```

Control de flujo: condicionales

- E.j. (condicional.py)

```
q = 4
h = 5
if q < h :
    print "primer test pasado"
elif q == 4:
    print "q tiene valor 4"
else:
    print "segundo test pasado"
```

```
$ ./condicional.py
primer test pasado
```

- Operadores booleanos: "or," "and," "not"
- Operadores relacionales: ==, >, <, !=

Control de flujo: bucles

- for se utiliza para iterar sobre los miembros de una secuencia
 - ▶ Se puede usar sobre cualquier tipo de datos que sea una secuencia (lista, tupla, diccionario)
- Ej. bucle.py

```
for x in range(1,5):  
    print x
```

```
$ ./bucle.py  
1 2 3 4
```
- La función range crea una secuencia descrita por ([start,] end [,step]), donde los campos start y step son opcionales. Start es 0 y step es 1 por defecto.

Control de flujo: bucles

- `while` es otra sentencia de repetición. Ejecuta un bloque de código hasta que una condición es falsa.
- `break` nos sirve para salir de un bucle
- Por ejemplo:

```
reply = 'repite'  
while reply == 'repite':  
    print 'Hola'  
    reply = raw_input('Introduce "repite" para hacerlo de nuevo: ')
```

```
Hola  
Introduce "repite" para hacerlo de nuevo: repite  
Hola  
Introduce "repite" para hacerlo de nuevo: adiós
```

Funciones

- Una función se declara usando la palabra clave def

```
# funcionesimple.py
def myfunc(a,b):
    sum = a + b
    return sum
print myfunc (5,6)
$ ./funcionsimple.py
11
```

- A una función se le pueden asignar parámetros por defecto:

```
# funcionvaloresdefecto.py
def myfunc(a=4,b=6):
    sum = a + b
    return sum
print myfunc()
print myfunc(b=8) # a es 4, sobrescribir b a 8
$ ./funcion.py
10
12
```

Funciones

- Listas de argumentos y argumentos basados en palabras clave:

```
# funcionargumentosvariablesyclave.py
```

```
def testArgLists_1(*args, **kwargs):
```

```
    print 'args:', args
```

```
    print 'kwargs:', kwargs
```

```
testArgLists_1('aaa', 'bbb', arg1='ccc', arg2='ddd')
```

```
print '=' * 40
```

```
def testArgLists_2(arg0, *args, **kwargs):
```

```
    print 'arg0: "%s"' % arg0
```

```
    print 'args:', args
```

```
    print 'kwargs:', kwargs
```

```
testArgLists_2('primero', 'aaa', 'bbb', arg1='ccc', arg2='ddd')
```

- Visualizaría:

```
args: ('aaa', 'bbb')
```

```
kwargs: {'arg1': 'ccc', 'arg2': 'ddd'}
```

```
=====
```

```
arg0: "primero"
```

```
args: ('aaa', 'bbb')
```

```
kwargs: {'arg1': 'ccc', 'arg2': 'ddd'}
```

Clases

- Una clase contiene una colección de métodos. Cada método contiene como primer parámetro (`self`) que hace referencia a un objeto
 - ▶ `self` equivalente a `this` en C++
- Existe un soporte limitado para variables privadas mediante *name mangling*.
 - ▶ Un identificador `_spam` es reemplazado por `__classname__spam`.
 - ▶ El identificador es todavía accesible por `_classname_spam`.
- En Python se soporta la herencia múltiple

Classes

```
# clasepinguinos.py
class PenguinPen:
    def __init__(self):
        self.penguinCount = 0
    def add (self, number = 1):
        """ Add penguins to the pen. The default number is 1 """
        self.penguinCount = self.penguinCount + number
    def remove (self, number = 1):
        """ Remove one or more penguins from the pen """
        self.penguinCount = self.penguinCount - number
    def population (self):
        """ How many penguins in the pen? """
        return self.penguinCount
    def __del__(self):
        pass
```

```
penguinPen = PenguinPen()
penguinPen.add(5) # Tux y su familia
print penguinPen.population()
```

Más clases

```
# clasherencia.py
class Basic:
    def __init__(self, name):
        self.name = name
    def show(self):
        print 'Basic -- name: %s' % self.name
class Special(Basic): # entre paréntesis la clase base
    def __init__(self, name, edible):
        Basic.__init__(self, name) # se usa Basic para referir a
        self.upper = name.upper() # clase base
        self.edible = edible
    def show(self):
        Basic.show(self)
        print 'Special -- upper name: %s.' % self.upper,
        if self.edible:
            print "It's edible."
        else:
            print "It's not edible."
    def edible(self):
        return self.edible
```

Probando clases

```
obj1 = Basic('Manzana')
obj1.show()
print '=' * 30
obj2 = Special('Naranja', True)
obj2.show()
```

- Visualizaría:

```
Basic -- name: Manzana
```

```
=====
```

```
Basic -- name: Naranja
```

```
Special -- upper name: NARANJA. It's edible.
```

Excepciones

- Cada vez que un error ocurre se lanza una excepción, visualizándose un extracto de la pila del sistema. E.j.
excepcion.py:

```
#!/usr/bin/python  
print a  
$ ./excepcion.py  
Traceback (innermost last): File "excepcion.py", line 2,  
in ? print a NameError: a
```
- Para capturar la excepción se usa except:

```
try:  
    fh=open("new.txt", "r")  
except IOError, e:  
    print e  
$ python excepcion.py  
[Errno 2] No such file or directory: 'new.txt'
```
- Puedes lanzar tu propia excepción usando el comando raise:

```
raise MyException  
raise SystemExitModules
```

Excepciones personalizadas

```
# excepcionpersonalizada.py
class E(RuntimeError):
    def __init__(self, msg):
        self.msg = msg
    def getMsg(self):
        return self.msg
try:
    raise E('mi mensaje de error')
except E, obj:
    print 'Msg:', obj.getMsg()
```

- Visualizaría:

Msg: mi mensaje de error

Módulos

- Un módulo es una colección de métodos en un fichero que acaba en `.py`. El nombre del fichero determina el nombre del módulo en la mayoría de los casos.

- E.j. `modulo.py`:

```
def one(a):  
    print "in one"  
def two (c):  
    print "in two"
```

- Uso de un módulo:

```
>>> import modulo  
>>> dir(modulo) # lista contenidos módulo  
['_builtins__', '__doc__', '__file__', '__name__', 'one',  
'two']  
>>> modulo.one(2)  
in one
```

Módulos II

- `import` hace que un módulo y su contenido sean disponibles para su uso.
- Algunas formas de uso son:
 - `import test`
 - ▶ Importa modulo test. Referir a x en test con "test.x".
 - `from test import x`
 - ▶ Importa x de test. Referir a x en test con "x".
 - `from test import *`
 - ▶ Importa todos los objetos de test. Referir a x en test con "x".
 - `import test as theTest`
 - ▶ Importa test; lo hace disponible como theTest. Referir a objeto x como "theTest.x".

Paquetes I

- Un paquete es una manera de organizar un conjunto de módulos como una unidad. Los paquetes pueden a su vez contener otros paquetes.
- Para aprender como crear un paquete consideremos el siguiente contenido de un paquete:

```
package_example/  
package_example/__init__.py  
package_example/module1.py  
package_example/module2.py
```

- Y estos serían los contenidos de los ficheros correspondientes:

```
# __init__.py  
# Exponer definiciones de módulos en este paquete.  
from module1 import class1  
from module2 import class2
```

Paquetes II

```
# module1.py
class class1:
    def __init__(self):
        self.description = 'class #1'
    def show(self):
        print self.description
```

```
# module2.py
class class2:
    def __init__(self):
        self.description = 'class #2'
    def show(self):
        print self.description
```

Paquetes III

```
# testpackage.py
import package_example
c1 = package_example.class1()
c1.show()
c2 = package_example.class2()
c2.show()
```

- Visualizaría:

```
class #1
```

```
class #2
```

- La localización de los paquetes debe especificarse o bien a través de la variable de entorno PYTHONPATH o en código del script mediante `sys.path`

Paquetes IV

- Como en Java el código de un paquete puede recogerse en un .zip:

```
>>> import zipfile
>>> a=zipfile.PyZipFile('mipackage.zip', 'w', zipfile.ZIP_DEFLATED)
>>> a.writepy('package_example')
>>> a.close()
```

- Luego lo puedes importar y usar insertando su path en `sys.path` o alternativamente añadiendo a la variable de entorno `PYTHONPATH` una referencia al nuevo .zip creado:

```
$ mkdir prueba; cp mipackage.zip prueba
$ export PYTHONPATH=/path/to/prueba/mipackage.zip
>>> import sys
>>> sys.path.insert(0, '/path/to/prueba/mipackage.zip')
>>> import package_example
>>> class1 = package_example.module1.class1()
>>> class1.show()
class #1
```

Manejo de ficheros

- Leer un fichero (leerfichero.py)

```
fh = open("holamundo.py") # open crea un objeto de tipo fichero
for line in fh.readlines() : # lee todas las líneas en un fichero
    print line,
fh.close()
$ python leerfichero.py
#!/usr/bin/python
print "Hola mundo"
```

- Escribir un fichero (escribirfichero.py)

```
fh = open("out.txt", "w")
fh.write ("estamos escribiendo ...\n")
fh.close()
$ python escribirfichero.py
$ cat out.txt
estamos escribiendo ...
```

Más sobre print

- **print** (printredirect.py)
 - ▶ stdout en Python es `sys.stdout`, stdin es `sys.stdin`:

```
import sys
class PrintRedirect:
    def __init__(self, filename):
        self.filename = filename
    def write(self, msg):
        f = file(self.filename, 'a')
        f.write(msg)
        f.close()
sys.stdout = PrintRedirect('tmp.log')
print 'Log message #1'
print 'Log message #2'
print 'Log message #3'
```

VARIABLES GLOBALES EN PYTHON

- Usar identificador `global` para referirse a variable global:

```
# variableglobal.py
NAME = "Manzana"
def show_global():
    name = NAME
    print '(show_global) nombre: %s' % name
def set_global():
    global NAME
    NAME = 'Naranja'
    name = NAME
    print '(set_global) nombre: %s' % name
show_global()
set_global()
show_global()
```

- Lo cual visualizaría:
(show_global) nombre: Manzana
(set_global) nombre: Naranja
(show_global) nombre: Naranja

Serialización de objetos

- **Pickle: Python Object Serialization**
 - ▶ El módulo `pickle` implementa la serialización y deserialización de objetos
 - Para serializar una jerarquía de objetos, creas un `Pickler`, y luego llamas al método `dump()`, o simplemente invocas `dump()` del módulo `pickle`
 - Para deserializar crear un `Unpickler` e invocas su método `load()` method, o simplemente invocas `load()` del módulo `pickle`
 - ▶ Se serializa el contenido del objeto `__dict__` de la clase, se puede cambiar este comportamiento cambiando los métodos `__getstate__()` y `__setstate__()`.

Serialización de objetos:

Ejemplo pickle I

```
import pickle # pickleunpickle.py
class Alumno:
    def __init__(self, dni, nombre, apellido1, apellido2):
        self.dni = dni
        self.nombre = nombre
        self.apellido1 = apellido1
        self.apellido2 = apellido2
    def __str__(self):
        return "DNI: " + self.dni + "\n\tNombre: " +
            self.nombre + "\n\tApellido1: " +
            self.apellido1 + "\n\tApellido2: " +
            self.apellido2 + "\n"

    def get_dni(self):
        return self.dni
    def get_nombre(self):
        return self.nombre
    def get_apellido1(self):
        return self.apellido1
    def get_apellido2(self):
        return self.apellido2
```

Serialización de objetos:

Ejemplo pickle II

```
alum = Alumno("00001111A", "Nando", "Quintana",  
             "Hernández")  
print "Alumno a serializar:\n", alum  
f = open("Alumno.db", 'w')  
pickle.dump(alum, f)  
f.close()  
  
f = open("Alumno.db", "r")  
alum2 = pickle.load(f)  
f.close()  
print alum2.get_dni()  
print "Alumno leído:\n", alum2
```

Serialización de objetos: Otro ejemplo más sofisticado

- Revisar ejemplos:
 - ▶ `picklingexample.py`
 - ▶ `unpicklingexample.py`
- Utilizan los métodos especiales `__setstate__()` y `__getstate__()`

Serialización de objetos

- El módulo `shelve` define diccionarios persistentes, las claves tienen que ser strings mientras que los valores pueden ser cualquier objeto que se puede serializar con `pickle`

```
import shelve
d = shelve.open(filename) # abre un fichero
d[key] = data # guarda un valor bajo key
data = d[key] # lo recupera
del d[key] # lo borra
d.close()
```

Programación de BD en Python

- Lo que es JDBC en Java es DB API en Python
 - ▶ Información detallada en: <http://www.python.org/topics/database/>
- Para conectarnos a una base de datos usamos el método `connect` del módulo de base de datos utilizado que devuelve un objeto de tipo `connection`
- El objeto `connection` define el método `cursor()` que sirve para recuperar un cursor de la BD
 - ▶ Otros métodos definidos en `connection` son `close()`, `commit()`, `rollback()`
- El objeto `cursor` define entre otros los siguientes métodos:
 - ▶ `execute()` nos permite enviar una sentencia SQL a la BD
 - ▶ `fetchone()` recuperar una fila
 - ▶ `fetchall()` recuperar todas las filas
- Hay varios módulos que implementan el estándar DB-API:
 - ▶ DCOracle (<http://www.zope.org/Products/DCOracle/>) creado por Zope
 - ▶ MySQLdb (<http://sourceforge.net/projects/mysql-python>)
 - MySQL-python-1.2.0.tar.gz para Linux
 - `apt-get install python2.4-mysqldb`
 - ▶ Etc.



-
- La base de datos open source más popular
 - ▶ Desarrollada por MySQL AB, compañía sueca cuyo negocio se basa en labores de consultoría sobre MySQL
 - <http://www.mysql.com>
 - Diseñada para:
 - ▶ Desarrollo de aplicaciones críticas
 - ▶ Sistemas con altos requerimientos de carga
 - ▶ Ser embebida en software
 - Existen otras buenas alternativas open source como PostgreSQL (<http://www.postgresql.org/>)
 - MySQL 5.0 (development release) soporta procedimientos almacenados
 - ▶ Desde MySQL 4.1 (production release) se soportan subqueries

Instalación MySQL

- Para instalar la última versión de producción de MySQL (4.1) tanto en Linux como Windows:
 - ▶ `dev.mysql.com/downloads/mysql/4.0.html`
- En las distribuciones que soportan `apt-get`, instalar con el comando:
`apt-get install mysql-server php4-mysql`

Ejemplo programación BD en Python con MySQL I

- Creamos una base de datos de nombre codesyntax a la que podemos hacer login con usuario nando y password nando, a través del siguiente SQL:

```
CREATE DATABASE codesyntax;
```

```
GRANT ALTER, SELECT, INSERT, UPDATE, DELETE, CREATE, DROP  
ON codesyntax.*  
TO nando@'%'  
IDENTIFIED BY 'nando';
```

```
GRANT ALTER, SELECT, INSERT, UPDATE, DELETE, CREATE, DROP  
ON codesyntax.*  
TO nando@localhost  
IDENTIFIED BY 'nando';
```

```
use codesyntax;
```

```
CREATE TABLE EVENTOS(ID int(11) NOT NULL PRIMARY KEY,  
NOMBRE VARCHAR(250), LOCALIZACION VARCHAR(250), FECHA bigint(20),  
DESCRIPCION VARCHAR(250));
```

```
INSERT INTO EVENTOS VALUES (0, 'curso python', 'tknika', 0, 'Cursillo54  
sobre Python');
```

Ejemplo programación BD en Python con MySQL II

```
# db/accesodbeventos.py
import MySQLdb, time, string, _mysql, _mysql_exceptions
def executeSQLCommand(cursor, command):
    rowSet = []
    command = string.strip(command)
    if len(command):
        try:
            cursor.execute(command) # Ejecuta el comando
            if string.lower(command).startswith('select'): # si es select
                lines = cursor.fetchall() # recuperar todos los resultados
                for line in lines:
                    row = []
                    for column in line:
                        row.append(column)
                    rowSet.append(row)
        except _mysql_exceptions.ProgrammingError, e:
            print e
            sys.exit()
    return rowSet
```

Ejemplo programación BD en Python con MySQL III

```
if __name__ == '__main__':
    db=MySQLdb.connect(host="localhost",user="nando",
                       passwd="nando", db="codesyntax")
    cursor = db.cursor()

    executeSQLCommand(cursor, "update eventos set fecha=" +
                        str(time.time()*1000))
    rowSet = executeSQLCommand(cursor, "select * from eventos")
    for row in rowSet:
        print row
    del cursor
```

- Visualizando lo siguiente:

```
$ python accesodbeventos.py
[0L, 'curso python', 'tknika', 1110133067870L, 'Cursillo sobre Python']
[1L, 'curso zope', 'tknika', 1110133067870L, 'Curso sobre Zope']
```

Programación de expresiones regulares I

- A través del módulo `re`, Python permite el uso de expresiones regulares similares a como se hace en Perl (una razón más para moverse de Perl a Python)

```
# regex/procesaUrlConRe.py
import re, urllib, sys
if len(sys.argv) <= 4:
    print "Usage: procesaUrl <url-a-procesar> <palabra-a-
    reemplazar> <nueva-palabra> <fichero-html-a-crear>"
    sys.exit(0)
print sys.argv[1]
s = (urllib.urlopen(sys.argv[1])).read()
t = re.sub(sys.argv[2], sys.argv[3], s)
backupFile = open(sys.argv[4], "w")
backupFile.write(t)
backupFile.close()
print 'Fichero ' + sys.argv[4] +
    ' escrito con contenido de url: ' + sys.argv[1] +
    ' al reemplazar palabra ' + sys.argv[2] +
    ' con palabra ' + sys.argv[3]
```

Programación de expresiones regulares II

```
# conseguir el titulo del documento HTML
tmatch = re.search(r'<title>(.*?)</title>', s, re.IGNORECASE)
if tmatch:
    title = tmatch.group(1)
    print 'Titulo de pagina ' + sys.argv[1] + ' es: ' + title

# extraer lista de enlaces url:
pat = re.compile(r'(http://[\w-]*[\.\w-]+)')
addrs = re.findall(pat, s)

print 'La lista de enlaces encontrados en esta pagina es: '
for enlace in addrs:
    print enlace
```

Programación de sistemas

- Python permite la programación de sistema tanto accediendo a la API de Windows (<http://www.python.org/windows/index.html>) como a las llamadas al sistema de UNIX (módulo `os`)
- El módulo `os` nos da acceso a:
 - ▶ El entorno del proceso: `getcwd()`, `getgid()`, `getpid()`
 - ▶ Creación de ficheros y descriptores: `close()`, `dup()`, `dup2()`, `fstat()`, `open()`, `pipe()`, `stat()`, `socket()`
 - ▶ Gestión de procesos: `execle()`, `execv()`, `kill()`, `fork()`, `system()`
 - ▶ Gestión de memoria `mmap()`
- El módulo `threading` permite la creación de threads en Python
- Revisar ejemplo del servidor web en Python
- Siguiente transparencia muestra cómo usar módulo `threading` para recuperar el contenido de varias urls

Ejemplo threads

```
#!/usr/bin/env python
import threading # threading/ejemplothreading.py
import urllib
class FetchUrlThread(threading.Thread):
    def __init__(self, url, filename):
        threading.Thread.__init__(self)
        self.url = url
        self.filename = filename
    def run(self):
        print self.getName(), "Fetching ", self.url
        f = open(self.getName()+self.filename, "w")
        content = urllib.urlopen(self.url).read()
        f.write(content)
        f.close()
        print self.getName(), "Saved in ", (self.getName()+self.filename)
urls = [ ('http://www.python.org', 'index_p.html'),
         ('http://www.google.es', 'index_g.html') ]
# Recuperar el contenido de las urls en diferentes threads
for url, file in urls:
    t = FetchUrlThread(url, file)
    t.start()
```

¿Por qué usar XML?

- Un documento XML puede ser fácilmente procesado y sus datos manipulados
- Existen APIs para procesar esos documentos en Java, C, C++, Perl.. (y por supuesto Python)
- XML define datos portables al igual que Java define código portable

Componentes documento XML

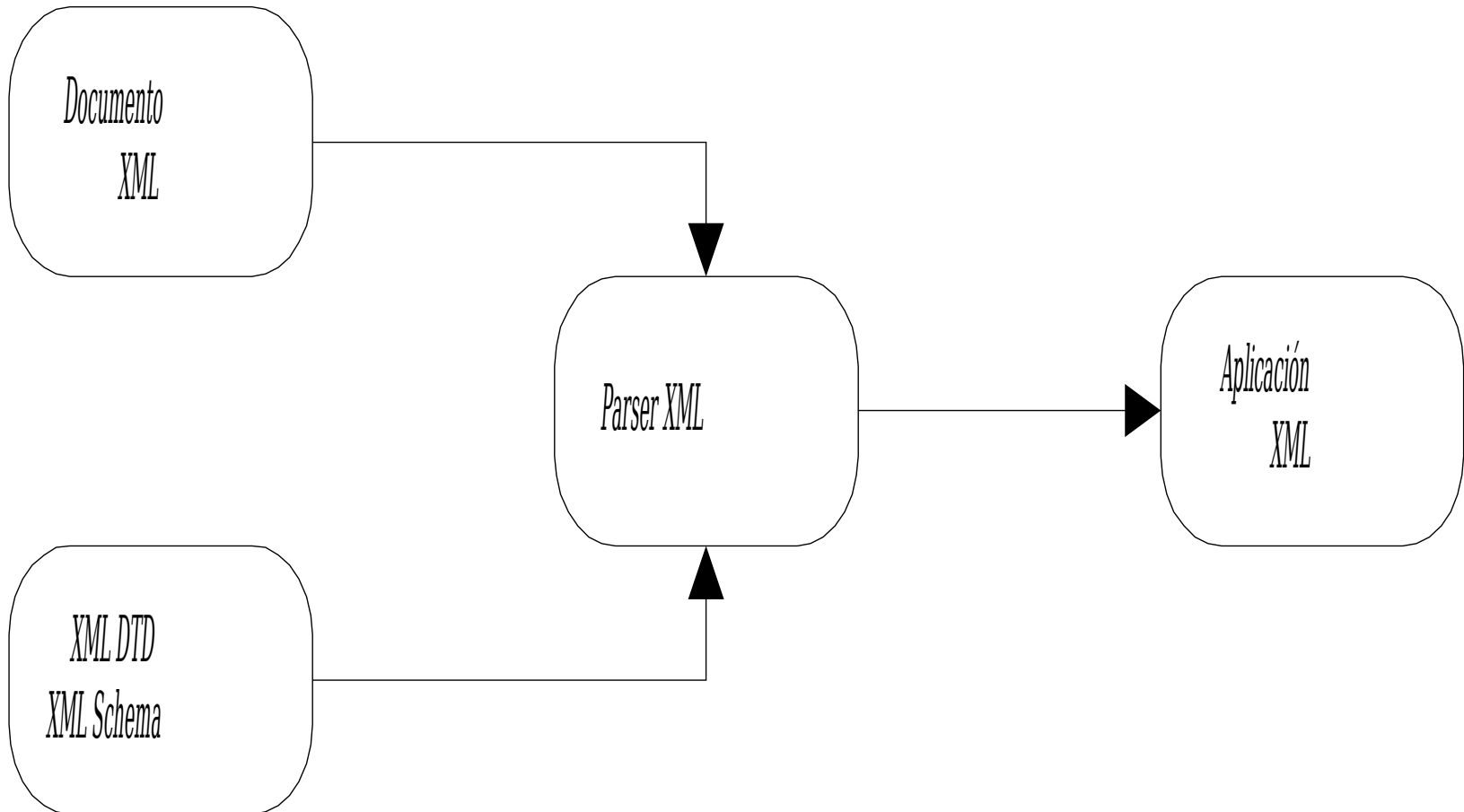
- Los documentos XML constan de:
 - ▶ Instrucciones de procesamiento (*processing instructions – PI*)
 - ▶ Declaraciones de tipo de documento
 - ▶ Comentarios
 - ▶ Elementos
 - ▶ Referencias a entidades
 - ▶ Secciones CDATA

Ejemplo Documento XML

```
<?xml version="1.0"?>
<!DOCTYPE mensaje SYSTEM "labgroups.dtd">

<lab_group>
  <student_name dni="44670523">
    Josu Artaza
  </student_name>
  <student_name dni="44543211">
    Nuria Buruaga
  </student_name>
  <student_name dni="23554521" tutor="33456211">
    Inga Dorsman
  </student_name>
</lab_group>
```

XML Parsing



XML Parsing (cont)

- SAX
 - ▶ Define interfaz dirigido por eventos (*event-driven*) para el procesamiento de un documento XML
 - ▶ Definido por David Megginson y lista correo XML-DEV : <http://www.megginson.com/SAX>
- DOM
 - ▶ Provee una representación de un documento XML en forma de un árbol
 - ▶ Carga todo el documento XML en memoria
 - ▶ <http://www.w3.org/DOM>

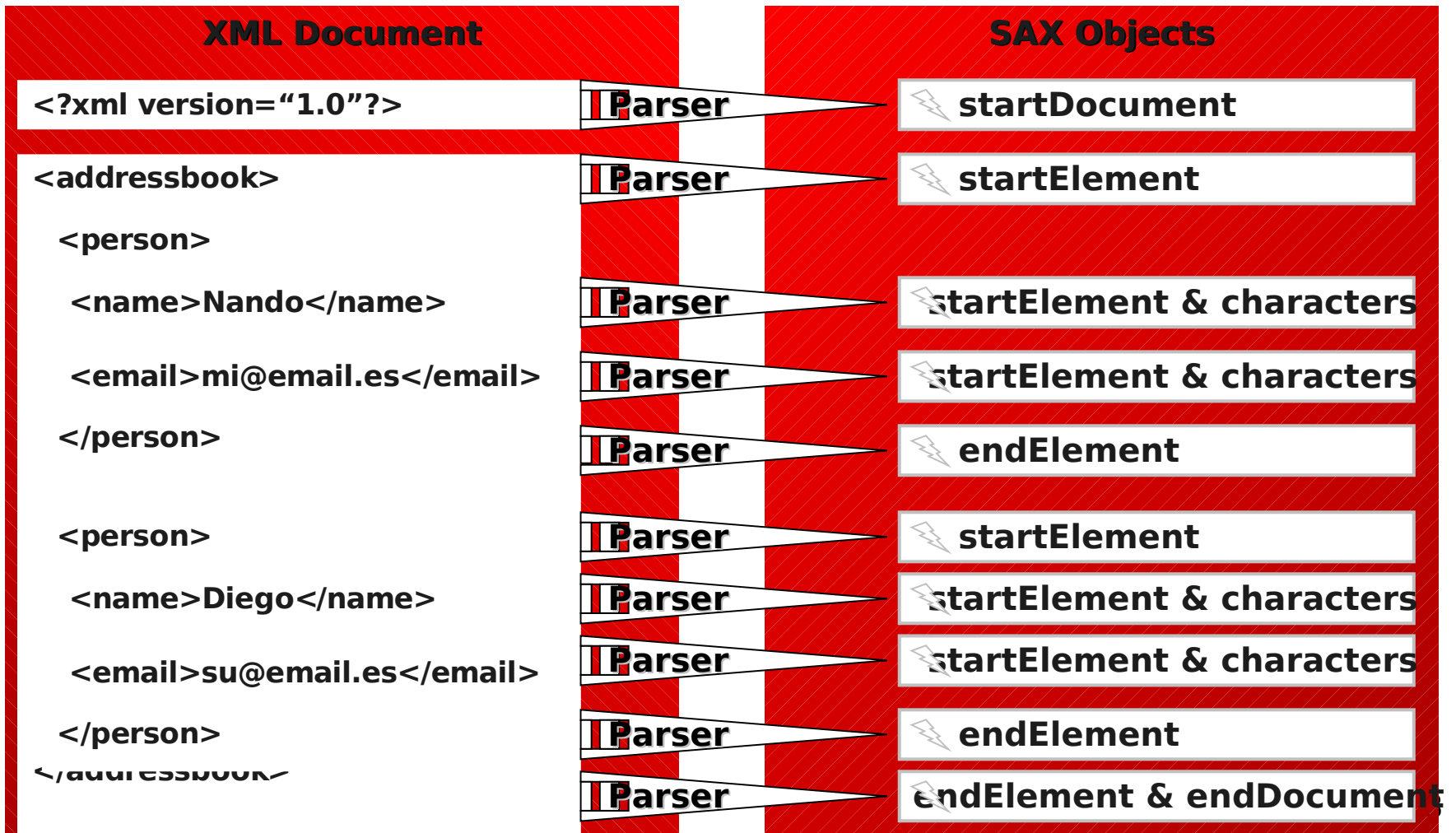
Simple API for XML: SAX

- Define un interfaz común implementado por muchos XML Parsers
- Es el estándar de-facto para procesamiento de XML basado en eventos
- SAX no es un parseador de XML
- SAX2 añade soporte para XML Namespaces
- La especificación de SAX 2.0/Java en:
 - ▶ <http://www.saxproject.org/apidoc/overview-summary.html>

Características de SAX

- Analizador o parser SAX:
 - ▶ Detecta cuándo empieza y termina un elemento o el documento, o un conjunto de caracteres, etc. (genera eventos)
 - ▶ Gestiona los espacios de nombres
 - ▶ Comprueba que el documento está bien formado
- Las aplicaciones necesitan implementar manejadores de los eventos notificados
- SAX lee secuencialmente de principio a fin, sin cargar todo el documento en memoria
- **Ventaja:** *eficiencia* en cuanto al tiempo y la memoria empleados en el análisis
- **Desventaja:** *no disponemos de la estructura en árbol* de los documentos

¿Cómo funciona SAX?



Programación en XML con SAX

- Soporte para SAX en Python es ofrecido por el módulo `xml.sax` de la Python Library
- Define 2 métodos:
 - ▶ `make_parser([parser_list])`
 - Crea y devuelve un objeto SAX XMLReader
 - ▶ `parse(filename_or_stream, handler[, error_handler])`
 - Crea un parser SAX y lo usa para procesar el documento a través de un handler
- El módulo `xml.sax.xmlreader` define readers para SAX
- El módulo `xml.sax.handler` define manejadores de eventos para SAX: `startDocument`, `endDocument`, `startElement`, `endElement`

Ejemplo procesamiento SAX I

```
# xml/ElementCounterSAX.py
# Ejecutar: ./ElementCounterSAX.py Cartelera.xml
import sys
from xml.sax import make_parser, handler
class ElementCounter(handler.ContentHandler):

    def __init__(self):
        self._elems = 0
        self._attrs = 0
        self._elem_types = {}
        self._attr_types = {}

    def startElement(self, name, attrs):
        self._elems = self._elems + 1
        self._attrs = self._attrs + len(attrs)
        self._elem_types[name] = self._elem_types.get(name, 0) + 1
        for name in attrs.keys():
            self._attr_types[name] = self._attr_types.get(name, 0) + 1
```

Ejemplo procesamiento SAX II

```
def endDocument(self):
    print "There were", self._elems, "elements."
    print "There were", self._attrs, "attributes."

    print "---ELEMENT TYPES"
    for pair in self._elem_types.items():
        print "%20s %d" % pair

    print "---ATTRIBUTE TYPES"
    for pair in self._attr_types.items():
        print "%20s %d" % pair
```

```
parser = make_parser()
parser.setContentHandler(ElementCounter())
parser.parse(sys.argv[1])
```

W3C Document Object Model (DOM)

- Documentos XML son tratados como un árbol de nodos
- Cada elemento es un “nodo”
- Los elementos hijos y el texto contenido dentro de un elemento son subnodos
- W3C DOM Site:
<http://www.w3.org/DOM/>

Características DOM

- Documento se carga totalmente en memoria en una estructura de árbol
- **Ventaja:** fácil acceder a datos en función de la jerarquía de elementos, así como modificar el contenido de los documentos e incluso crearlos desde cero.
- **Desventaja:** *coste* en tiempo y memoria que conlleva construir el árbol

W3C XML DOM Objects

- **Element** – un elemento XML
- **Attribute** – un atributo
- **Text** – texto contenido en un elemento o atributo
- **CDATAsection** – sección CDATA
- **EntityReference** – Referencia a una entidad
- **Entity** – Indicación de una entidad XML
- **ProcessingInstruction** – Una instrucción de procesamiento
- **Comment** – Contenido de un comentario de XML
- **Document** – El objeto documento
- **DocumentType** – Referencia al elemento DOCTYPE
- **DocumentFragment** – Referencia a fragmento de documento
- **Notation** – Contenedor de una anotación

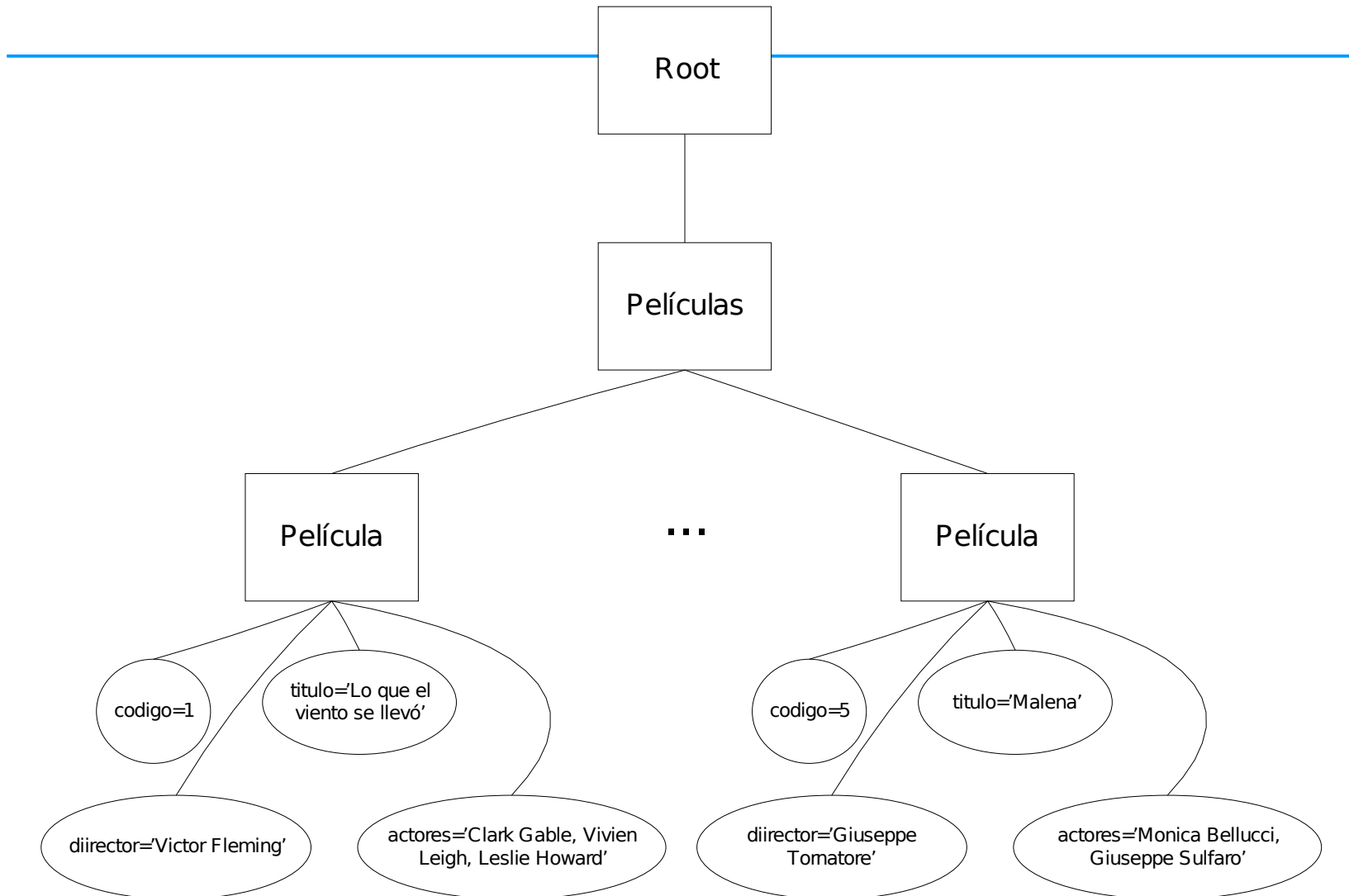
Objetos relacionados con Nodos

- **Node** – un nodo en el árbol de un documento
- **NodeList** – una lista de objetos nodos
- **NamedNodeMap** – permite interacción y acceso por nombre a una colección de atributos

Documento XML como Árbol de Nodos

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Películas>
  <Película código='1'
    título='Lo que el viento se llevó'
    director='Victor Fleming'
    actores='Clark Gable, Vivien Leigh,
            Leslie Howard' />
  <Película código='2' título='Los Otros'
    director='Alejandro Amenabar'
    actores='Nicole Kidman' />
  <Película código="5" título="Malena"
    director="Giuseppe Tornatore"
    actores="Monica Bellucci, Giuseppe Sulfaro" />
</Películas>
```

Árbol de Nodos



Procesando XML con DOM

- Python provee el módulo `xml.dom.minidom` que es una implementación sencilla de DOM
- El método `parse` a partir de un fichero crea un objeto DOM, el cual tiene todos los métodos y atributos estándar de DOM: `hasChildNodes()`, `childNodes`, `getElementsByTagName()`
- Para más información sobre procesamiento XML en Python ir a: <http://pyxml.sourceforge.net/topics/>
 - ▶ El módulo PyXML, que no viene en la distribución por defecto de Python, permite procesamiento un poco más sofisticado
 - <http://pyxml.sourceforge.net/topics/>

Ejemplo DOM I

```
# xml/ejemploDOM.py
# Ejecutar: python ejemploDOM.py Cartelera.xml

#!/usr/bin/env python
import xml.dom.minidom, sys
class Pelicula:
    def __init__(self, codigo, titulo, director, actores):
        self.codigo = codigo
        self.titulo = titulo
        self.director = director
        self.actores = actores

    def __repr__(self):
        return "Codigo: " + str(self.codigo) + " - titulo: " +
            self.titulo + " - director: " + self.director +
            " - actores: " + self.actores

class PeliculaDOMParser:
    def __init__(self, filename):
        self.dom = xml.dom.minidom.parse(filename)
        self.peliculas = []
```

Ejemplo DOM II

```
def getPeliculas(self):
    if not self.peliculas:
        peliculaNodes = self.dom.getElementsByTagName("Película")
        numPelis = len(peliculaNodes)
        for i in range(numPelis):
            pelicula = peliculaNodes.item(i)
            # Recuperar los atributos de cada nodo Película
            peliAttribs = pelicula.attributes
            codigo = peliAttribs.getNamedItem("codigo").nodeValue
            titulo = peliAttribs.getNamedItem("titulo").nodeValue
            director = peliAttribs.getNamedItem("director").nodeValue
            actores = peliAttribs.getNamedItem("actores").nodeValue

        self.peliculas.append(Película(codigo,titulo,director,actores))
        return self.peliculas

if __name__ == '__main__':
    domParser = PelículaDOMParser(sys.argv[1])
    for peli in domParser.getPeliculas():
        print peli
```

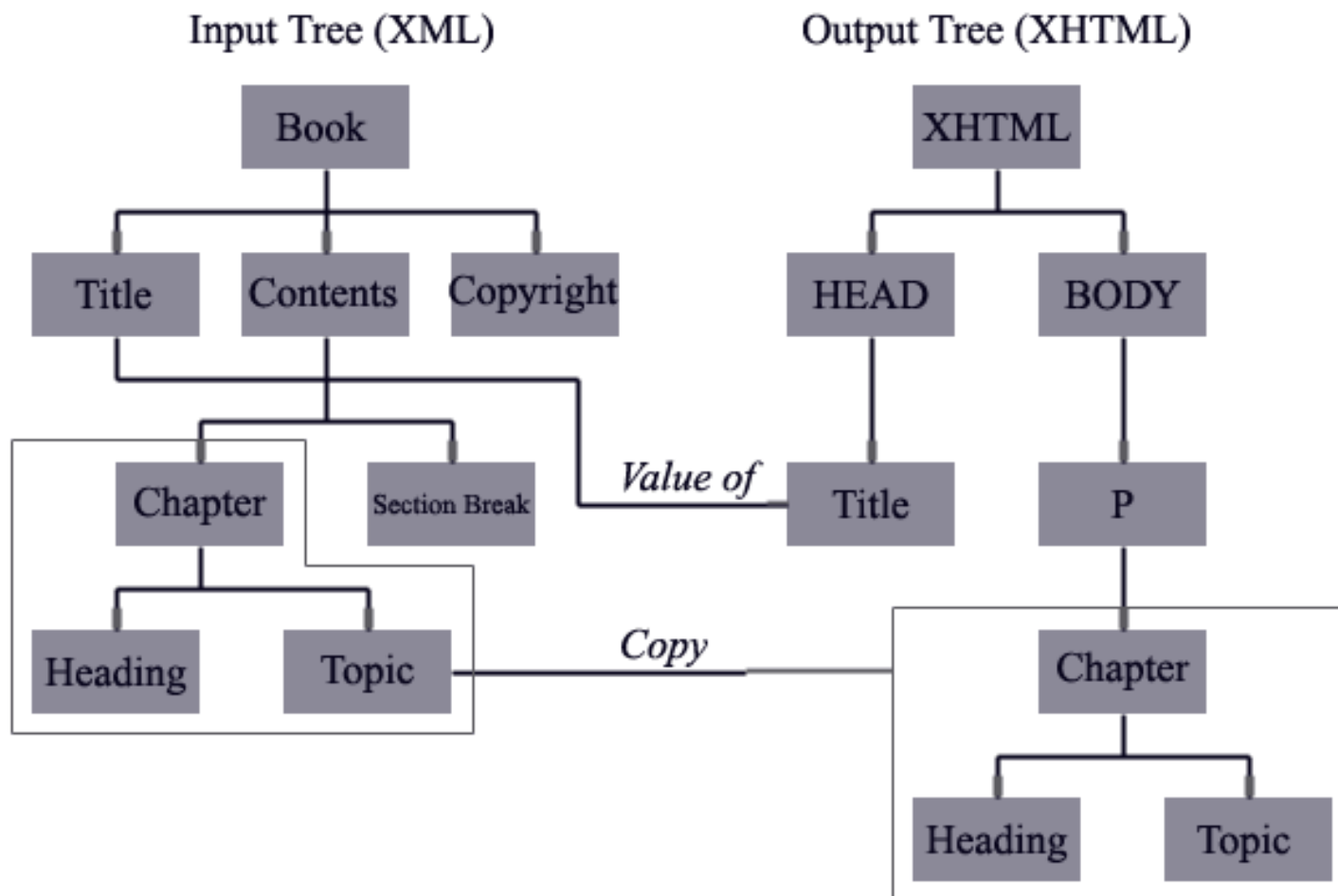
Extensible Style Language Transformations (XSLT) I

- Con la diversidad de lenguajes de presentación que hay (WML, HTML, cHTML) existen dos alternativas para desarrollar las aplicaciones:
 - ▶ Desarrollar versiones de los procesos de generación de presentación (JSP, ASP, CGI,..) para cada lenguaje.
 - ▶ Desarrollar solo una versión que genere XML y conversores de XML a los lenguajes de presentación.

Extensible Style Language Transformations (XSLT) II

- Dos partes:
 - ▶ Transformation Language (XSLT)
 - ▶ Formatting Language (XSL Formatting Objects)
- XSLT transforma un documento XML en otro documento XML
- XSLFO formatea y estiliza documentos en varios modos
- XSLT W3C Recommendation - <http://www.w3.org/TR/xslt>

Operaciones entre árboles en XSL



Ventajas y desventajas de XSLT

- Ventajas:
 - ▶ No asume un único formato de salida de documentos
 - ▶ Permite *manipular* de muy diversas maneras un documento XML: reordenar elementos, filtrar, añadir, borrar, etc.
 - ▶ Permite *acceder a todo* el documento XML
 - ▶ XSLT es un *lenguaje XML*
- Desventajas:
 - ▶ Su *utilización* es más compleja que un lenguaje de programación convencional
 - ▶ *Consume* cierta memoria y capacidad de proceso → DOM detrás

Usando hojas de estilo XSLT

- Para crear una transformación XSL necesitamos:
 - ▶ El documento XML a transformar (`students.xml`)
 - ▶ La hoja de estilo que especifica la transformación (`students.xsl`)

Documento XML (students.xml)

```
<?xml version="1.0"?>
<course>
  <name id="csci_2962">Programming XML in Java</name>
  <teacher id="di">Diego Lopez</teacher>
  <student id="ua">
    <name>Usue Artaza</name>
    <hw1>30</hw1>
    <hw2>70</hw2>
    <project>80</project>
    <final>85</final>
  </student>
  <student id="iu">
    <name>Iñigo Urrutia</name>
    <hw1>80</hw1>
    <hw2>90</hw2>
    <project>100</project>
    <final>40</final>
  </student>
</course>
```

Hoja de estilo XSLT

(students.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="course">
    <HTML>
      <HEAD><TITLE>Name of students</TITLE></HEAD>
      <BODY>
        <xsl:apply-templates select="student"/>
      </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="student">
    <P><xsl:value-of select="name"/></P>
  </xsl:template>
</xsl:stylesheet>
```

Resultado de transformación

- (students.html)

```
<HTML>  
  <HEAD> <TITLE>Name of students</TITLE>  
  </HEAD>  
  <BODY>  
    <P>Usue Artaza</P>  
    <P>Iñigo Urrutia</P>  
  </BODY>  
</HTML>
```

XSLT en Python

- Herramientas para procesamiento XSLT tools en Python:
 - ▶ <http://uche.ogbuji.net/tech/akara/nodes/2003-01-01/python-xslt>
- En la siguiente url podemos encontrar adaptaciones Python de las librerías de la toolkit Gnome en C Libxml y Libxslt:
 - ▶ <http://xmlsoft.org/python.html> (Linux)
 - ▶ <http://users.skynet.be/sbi/libxml-python/> (Windows)
 - ▶ El ejemplo en la siguiente página ilustra el uso de esta librería

Ejemplo XSLT

```
# Instalar fichero libxml2-python-2.6.16.win32-py2.4.exe
# Ejecutar: python xsltexample.py Cartelera.xml Cartelera.xml
transform.html
import libxml2
import libxslt
import sys

if len(sys.argv) != 4:
    print 'Usage: python xsltexample <xml-file> <xslt-file>
    <output-file>'
    sys.exit(0)
else:
    styledoc = libxml2.parseFile(sys.argv[2])
    style = libxslt.parseStylesheetDoc(styledoc)
    doc = libxml2.parseFile(sys.argv[1])
    result = style.applyStylesheet(doc, None)
    style.saveResultToFile(sys.argv[3], result, 0)
    style.freeStylesheet()
    doc.freeDoc()
    result.freeDoc()
```

Ejemplo XML (Cartelera.xml)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Cartelera>
  <Cine codigo='1' nombre='Coliseo Java' direccion='Avda. Abaro'
    poblacion='Portugalete'>
    <Pelicula codigo='1' titulo='Lo que el viento se llevo'
      director='Santiago Segura'
      actores='Bo Derek, Al Pacino, Robert Reford'>
      <Sesion>16:00</Sesion>
      <Sesion>19:30</Sesion>
      <Sesion>22:00</Sesion>
    </Pelicula>
    <Pelicula codigo='2' titulo='Los Otros'
      director='Alejandro Amenabar'
      actores='Nicole Kidman'>
      <Sesion>16:30</Sesion>
      <Sesion>19:45</Sesion>
      <Sesion>22:30</Sesion>
    </Pelicula>
  </Cine>
  ...
</Cartelera>
```

Ejemplo XSL (Cartelera.xsl)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform
  version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <style type="text/css">
          table {font-family: arial, 'sans serif';
margin-left: 15pt;}
          th,td {font-size: 80%;}
          th {background-color:#FAEBD7}
        </style>
      </head>
      <body>
        <table border="1">
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
```

Ejemplo XSL (Cartelera.xsl)

```
<xsl:template match="Cartelera">
  <xsl:for-each select="Cine">
    <tr>
      <th><xsl:text>Cine</xsl:text></th>
      <th><xsl:text>Dirección</xsl:text></th>
      <th><xsl:text>Población</xsl:text></th>
      <th></th>
    </tr>
    <tr>
      <td><xsl:value-of select="./@nombre"/></td>
      <td><xsl:value-of select="./@direccion"/></td>
      <td><xsl:value-of select="./@poblacion"/></td>
      <td><xsl:text></xsl:text></td>
    </tr>
```

Ejemplo XSL (Cartelera.xsl)

```
<xsl:for-each select="Película">
  <tr>
    <th></th>
    <th><xsl:text>Película</xsl:text></th>
    <th><xsl:text>Director</xsl:text></th>
    <th><xsl:text>Actores</xsl:text></th>
  </tr>
  <tr>
    <td><xsl:text></xsl:text></td>
    <td><xsl:value-of select="./@titulo"/></td>
    <td><xsl:value-of select="./@director"/></td>
    <td><xsl:value-of select="./@actores"/></td>
  </tr>
```

Ejemplo XSL (Cartelera.xsl)

```
<tr>
  <th></th>
  <th></th>
  <th><xsl:text>Sesión</xsl:text></th>
  <th><xsl:text>Hora</xsl:text></th>
</tr>
<xsl:for-each select="Sesion">
  <tr>
    <td><xsl:text></xsl:text></td>
    <td><xsl:text></xsl:text></td>
    <td><xsl:value-of select="position()"/></td>
    <td><xsl:value-of select="."/></td>
  </tr>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Resultado XSL parsing

Cine	Dirección	Población	Actores
Cineo Java	Avenida Alvaro	Portugalete	
	Película	Director	
	Lo que el viento se llevó	Victor Fleming	Clark Gable, Vivien Leigh, Leslie Howard
		Sesión	Hora
		1	16:00
		2	19:30
		3	22:00
	Película	Director	Actores
	Los Otros	Alejandro Amenabar	Nicole Kidman
		Sesión	Hora
	1	16:30	
	2	19:45	
	3	22:30	
Cine	Dirección <th>Población</th> <th>Actores</th>	Población	Actores
Serantes	Calle Labrador	Sanurtzi	
	Película	Director	
	Los Otros	Alejandro Amenabar	Nicole Kidman
		Sesión	Hora
		1	16:00
		2	19:00
		3	22:00
	Película	Director	Actores
	Matrix - Reloaded	Peter Jackson	Keanu Reeves
		Sesión	Hora
	1	00:30	
	2	16:30	
	3	19:45	
	4	22:30	
Película	Director	Actores	
X-Men II	George Lucas	Sandra Bullock, Wesley Snipes	
	Sesión	Hora	
	1	16:45	
	2	19:45	
	3	22:15	
Película	Director	Actores	
Malena	Giuseppe Tornatore	Monica Bellucci, Giuseppe Sulfaro	
	Sesión	Hora	
	1	16:30	

References

- <http://www.python.org/doc/faq/es/general/>
- <http://diveintopython.org>
- <http://pyspanishdoc.sourceforge.net/tut/>
- <http://docs.python.org/lib/lib.html>
- <http://www.python.org/~guido/>
- <http://paginaspersonales.deusto.es/dipina/>
- <http://www.artima.com/weblogs/viewpost.jsp?thread:>
- http://www.ferg.org/projects/python_java_side-by-side