Python: descubre el poder del lenguaje scripting de moda en la comunidad open source

Dr. Diego Lz. de Ipiña Gz. de Artaza
http://paginaspersonales.deusto.es/dipina

El otro lenguaje de programación que empieza con 'P'

- 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"
```

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)
- Open source
  - Razón por la cual la Python Library sigue creciendo y 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

`if (x) {`
`  if (y) {`
`  f1();`
`  }`
`  f2();`
`}`
	`if x:`
	`  if y:`
	`  f1()`
	`  f2()`

Python vs. Perl

- Los dos están basados en un buen entendimiento de las herramientas necesarias para resolver problemas
  - Perl está basado en awk, sed, and shell scripting y su misión es hacer las tareas de administradores de sistemas más sencillas
  - Python está basado e inspirando por 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
  - Strong typing
  - Y mucho más ...
- Python es un lenguaje de scripting:
  - No ofrece strong typing
    - 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 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

- Bajar versión de Python de http://www.python.org/download/
  - Para Windows ejecutar instalador
  - Para Linux, usar rpms disponibles en: http://www.python.org/2.3.3/rpms.html
    - `rpm -iv python2.3-2.3.3-pydotorg.i386.rpm`

Usando Python desde línea comando

- Para arrancar el intérprete (Python interactivo) ejecutar:

```
$ python
Python 2.3.3 (#1, Dec 30 2003, 08:29:25)
[GCC 3.3.1 (cygwing special)] on cygwin
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 Linux) o:

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

Ejecutando programa holamundo.py

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

```
#!/usr/bin/env
python print "Hello World"
```

- Ejecutar el script desde línea de comando:

```
$ python helloworld.py
Hello World
$
```

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
- 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

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:

```
>>> 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

- 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 IV

- 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 V

- Tuplas (), lo mismo que listas, pero no se pueden modificar, e.j. (1, 2)
- 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"
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`

```
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.

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`

```
# 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
```

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++

```
# 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

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

$ python PenguinPen.py
5
```

Más clases

```
# clasesherencia.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', 1)
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
$ print 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
```

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 objecto `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 sus contenidos:

```
# __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()
>>> ^Z
```

- 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=/home/dipina/examples/prueba/mipackage.zip
>>> import sys # esta y la siguiente no hacen falta si se ha
   inicializado PYTHONPATH
>>> sys.path.insert(0, '/home/dipina/examples/prueba/mipackage.zip')
>>> import package_example
>>> class1 = package_example.module1.class1()
>>> class1.show()
class #1
>>> ^Z
```

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()`
    - Para deserializar creas un `Unpickler` e invocas su método `load()` method.
- 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
```

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 `conection`
- El objeto `connection` tiene el método `cursor()` que sirve para recuperar un cursor de la BD
  - Otros métodos definidos en `connection` son `close()`, `commit()`, `rollback()`, `cursor()`
- 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-0.9.2.win32-py2.3.exe para Windows
    - MySQL-python-0.9.2-1.i386.rpm para Linux
  - Etc.

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/accesodbeventos.py
import MySQLdb, time, string, _mysql, _mysql_exceptions
def executeSQLCommand(cursor, command):
    result = ""
    command = string.strip(command)
    if len(command):
        try:
            cursor.execute(command) # Ejecuta el comando
            if string.lower(command).startswith('select'):
                # si es una select ...
                lines = cursor.fetchall() # recuperar todos los
   resultados
                for line in lines:
                    for column in line:
                        if column == None:
                            result = result + 'null '
                        else:
                            result = result + str(column) + ' '
                    result = result + '\n'
        except _mysql_exceptions.ProgrammingError, e:
            print e
            sys.exit()
```

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))
    results = executeSQLCommand(cursor, "select * from eventos")
    print results
    print results.split() # crear una lista y la visualiza
    del cursor
```

- Visualizando lo siguiente:

```
$ python accesodbeventos.py
0 SEMANA ESIDE ESIDE-DEUSTO 1078901556610 Charla sobre Python

['0', 'SEMANA', 'ESIDE', 'ESIDE-DEUSTO', '1078901556610', 'Charla',
   'sobre', 'Python']
```

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() # lee el contenido de
   una url
# reemplaza todas las ocurrencias de "Artaza" por "artaza"
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

- Por poder se puede incluso llevar a cabo la programación de sistemas en Python: programación de API de Windows (http://www.python.org/windows/index.html) y 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
- Siguiente transparencia muestra como 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.html'),
         ('http://paginaspersonales.deusto.es/dipina', 'index.html') ]
# Recuperar el contenido de las urls en diferentes threads
for url, file in urls:
   t = FetchUrlThread(url, file)
   t.start()
```

Programación de CGIs

- Pasos para desarrollar CGIs en Python:
  - Instalar Apache 2.0, disponible en: http://httpd.apache.org/download.cgi
  - Instalar mod_python 3.1.2b: http://httpd.apache.org/modules/python-download.cgi
  - Configurar Apache añadiendo a `httpd.conf` las siguientes líneas, para dar soporte a CGIs en Python y PSPs (Python Server Pages):

```
<Directory "<dir-donde-guardar-python-scripts>">
  AddHandler mod_python .py
  PythonHandler mod_python.publisher
  PythonDebug On
</Directory>

<Directory "<dir-donde-guardar-paginas-psp>">
  AddHandler mod_python .psp
  PythonHandler mod_python.psp
  PythonDebug On
</Directory>
```

- Usar el módulo `cgi` de la Python Library para programar y seguir documentación de `mod_python` (http://www.modpython.org/live/current/doc-html/)

Ejemplo CGI I

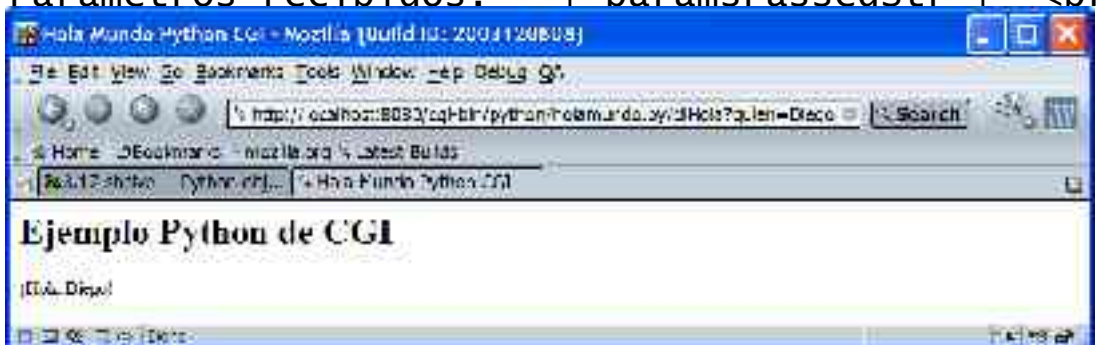
```
# cgi-bin/python/holamundo.py
# metodos de ayuda del CGI
def _formatAsHTML(req, content):
    req.content_type = "text/html"
    return "<html><head><title>Hola Mundo Python
   CGI</title></head><body><h1>Ejemplo Python de CGI</h1><p>" +
   content + "</p></body></html>"

def _usage():
    return "Uso: Debes especificar un parametro de nombre 'quien',
   para saber a quien saludar, e.j: http://localhost:8080/cgi-
   bin/python/holamundo.py/diHola?quien=Diego"
```

Ejemplo CGI II

```
# único método público que se puede invocar al que hay que pasar
   obligatoriamente un parametro
def diHola(req, quien=""):
    if not quien:
        return _formatAsHTML(req, _usage())
    return _formatAsHTML(req, "¡Hola " + quien + "!")

# si no se especifica un metodo en la url se invoca index por defecto,
# es decir http://localhost:8080/cgi-bin/python/holamundo.py
def index(req, **params):
    paramsPassedStr = ""
    if params:
        for param in params:
            paramsPassedStr += (param + "\n")
    return _formatAsHTML(req, "Unico metodo publico en CGI es
   diHola<br>Parametros recibidos: " + paramsPassedStr + "<br>" + _usage
   ())
```



LCE Sentient Library

LCE Sentient Library Catalog Functions:

- Browse LCE Library Catalog
- Search for book
- Create new book category
- Enter new book details
- Edit book details and/or Print TRIPtag
- Enter new shelf details
- Edit shelf details and/or Print TRIPtag

Created by Diego López de Ipiña (dl231@eng.cam.ac.uk)

Book ('1115377') belonging to category root.LCE-LIBRARY details:

Title:	The C++ Programming Language
Author:	Bjarne Stroustrup
Year:	1997
Publisher:	Addison Wesley
ISBN:	0-201-88954-4
Borrowed by:	Diego
Owner:	Diego

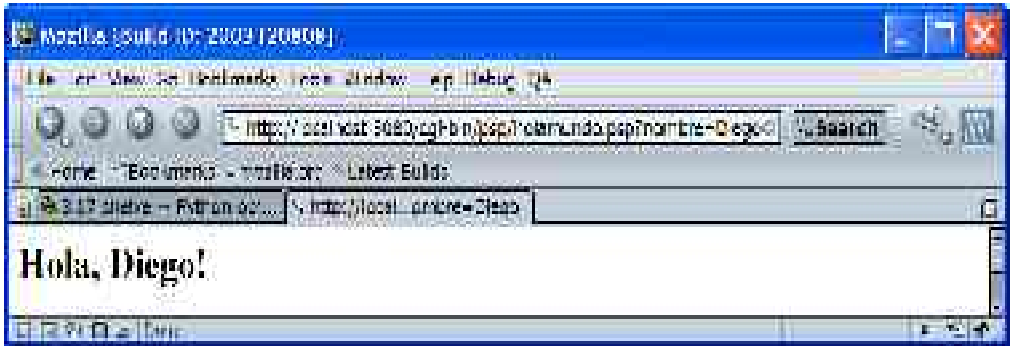
Book ('1115377') LOCATION details:

The book was last seen at shelf inside Diego's computer in room D beside books: | Programming Mobile Objects with Java., Design Patterns Elements of Reusable Object Oriented Software | on Wed Nov 7 11:17:39 2001

Ejemplo PSP

- Python Server Pages es la versión Python de JSPs o ASPs
  - Permite la inserción de código Python en un documento HTML
  - Usa los mismos códigos de escape que los JSPs
  - Se permite su uso a través de mod_python de Apache
    - http://www.onlamp.com/pub/a/python/2004/02/26/python_server_pages.htm

```
<!- cgi-bin/psp/holamundo.psp -->
<html>
<%
if form.has_key('nombre'):
   saludo = 'Hola, %s!' % form['nombre'].capitalize()
else:
   saludo = 'Hola mundo!'
# end
%>
  <h1><%= saludo %></h1>
  <p>Usa parametro 'nombre' para recibir un saludo, e.g.
    holamundo.psp?nombre=Diego</p>
</html>
```



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` object
  - `parse(filename_or_stream, handler[, error_handler])`
    - Crea un parser SAX parser 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])
```

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/
  - La distribución 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("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
```

Programación distribuida: CORBA con omniORBpy

- Desde Python se puede usar tanto CORBA (omniORBpy) como servicios web (SOAPpy disponible en http://pywebsvcs.sourceforge.net/)
- En este curso nos concentramos sólo en CORBA:
  - Download omniORBpy de: http://omniorb.sourceforge.net/
    - Desarrollada por Duncan Grisby en AT&T Labs Cambridge
    - Basada en la ORB para C++: omniORB
  - Descomprimir y compilar en Linux o simplemente descomprimir en Windows
  - Las siguientes variables de entorno son necesarias:
    - `PYTHONPATH=<omniORBpy-install-dir>/lib/python;<omniORBpy-install-dir>\lib\x86_win32`
    - `PATH=$PATH:=<omniORBpy-install-dir>/bin/x86_win32`
    - `LD_LIBRARY_PATH=<omniORBpy-install-dir>\lib\<platform>`
  - Para compilar IDL usar: `omniidl -bpython <fichero-idl-a-compilar>`

Ejemplo CORBA: IDL

```
// corba/example_echo.idl
module Example {
  interface Echo {
    string echoString(in string mesg);
  };
};
```

Ejemplo CORBA: servidor

```
#!/usr/bin/env python
import sysfrom omniORB import CORBA, PortableServer
# Import the stubs and skeletons for the Example module
import Example, Example__POA
# Define an implementation of the Echo interface
class Echo_i (Example__POA.Echo):
   def echoString(self, mesg):
      print "echoString() called with message:", mesg
      return mesg
# Initialise the ORB
orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
# Find the root POA
poa = orb.resolve_initial_references("RootPOA")
# Create an instance of Echo_ie
i = Echo_i()# Create an object reference, and implicitly activate the objecte
o = ei._this()
# Print out the IOR
print orb.object_to_string(eo)
# Activate the POA
poaManager = poa._get_the_POAManager()
poaManager.activate()
# Everything is running now, but if this thread drops out of the end
# of the file, the process will exit. orb.run() just blocks until the
# ORB is shut down
orb.run()
```

Ejemplo CORBA: servidor

```
#!/usr/bin/env pythonimport sys
# Import the CORBA module
from omniORB import CORBA
# Import the stubs for the Example module
import Example
# Initialise the ORB
orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
# Get the IOR of an Echo object from the command line (without
# checking that the arguments are sensible!)
ior = sys.argv[1]
# Convert the IOR to an object reference
obj = orb.string_to_object(ior)
# Narrow reference to an Example::Echo objecte
o = obj._narrow(Example.Echo)
if eo is None:
   print "Object reference is not an Example::Echo"
   sys.exit(1)
# Invoke the echoString operation
message = "Hello from Python"
result  = eo.echoString(message)
print "I said '%s'. The object said '%s'." % (message,result)
```

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 Tcl/tk, no tiene apariencia nativa
  - Es lenta pero su uso es muy sencillo
  - Pmw (Python meta widgets) (http://pmw.sourceforge.net/)
    - Componentes más elaborados encima de Tkinter
- Existen otras toolkits para generación de GUIs:
  - wxPython (http://www.wxpython.org/)
    - Apariencia nativa, basado en wxWindows (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/)

Ejemplo Tkinter I

```
# gui/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

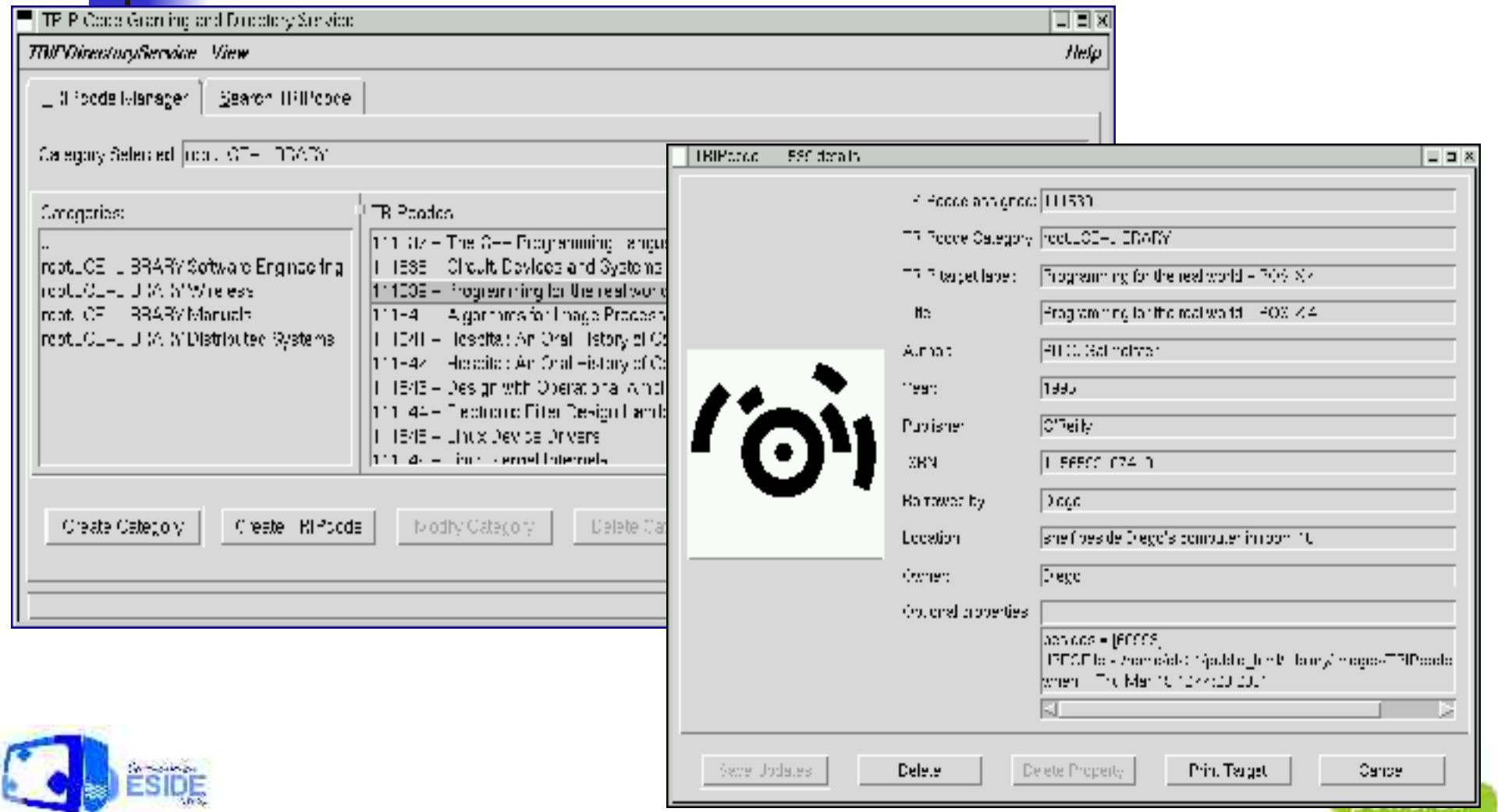
```
    def Start(self):
        """ Start the stopwatch, ignore if running. """
        if not self._running:
            self._start = time.time() - self._elapsedtime
            self._update()
            self._running = 1

    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()
```

Ejemplo de GUI con Pmw



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 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()
```

- A través del programa `wxPython\demo\demo.py` se pueden ver demos de todas las capacidades de wxPython y lo que es más importante vienen acompañadas de código fuente

Un poco de Jython

- Download Jython de:
  - http://www.jython.org/download.html
- Para instalar simplemente ejecutar: `java jython-21`
  - Usa Lift-Off Java-Installer: http://liftoff.sourceforge.net/
- Algunos ejemplos de Jython en:
  - http://www.jython.org/applets/index.html
- Documentación básica sobre Jython disponible en:
  - http://www.jython.org/docs/usejava.html

Ejemplo Jython: ButtonDemo

```
# http://www.jython.org/applets/button.html
from java import awt, applet
class ButtonDemo(applet.Applet):

  def init(self):
    self.b1 = awt.Button('Disable middle button',
   actionPerformed=self.disable)
    self.b2 = awt.Button('Middle button')
    self.b3 = awt.Button('Enable middle button', enabled=0,
   actionPerformed=self.enable)
    self.add(self.b1)
    self.add(self.b2)
    self.add(self.b3)

  def enable(self, event):
    self.b1.enabled = self.b2.enabled = 1
    self.b3.enabled = 0

  def disable(self, event):
    self.b1.enabled = self.b2.enabled = 0
    self.b3.enabled = 1
```

Casos de éxito de Python

- BitTorrent (http://bitconjurer.org/BitTorrent/), sistema P2P que ofrece mayor rendimiento que eMule
- PyGlobus, permite la programación de Grid Computing (http://www-itg.lbl.gov/gtg/projects/pyGlobus/)
- ZOPE (www.zope.org) es un 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
- Google usa Python internamente, lo mismo que Yahoo para su sitio para grupos
- Red Hat Linux utiliza Python para la instalación, configuración, y gestión de paquetes.
- Más historias de éxito de Python en: http://pbf.strakt.com/success

Recursos utilizados

- Compilador de Python 2.3.3 y documentación:
  - http://www.python.org/2.3.3/
- Extensiones para Windows:
  - https://sourceforge.net/projects/pywin32/
- Módulo de bases de datos MySQLdb 0.9.2:
  - http://sourceforge.net/projects/mysql-python
- Base de datos MySQL 4.0:
  - http://www.mysql.com/downloads/mysql-4.0.html
- omniORBpy
  - http://omniorb.sourceforge.net/
- Servidor Apache 2.0 (http://httpd.apache.org/)
  - Módulo mod_python para Apache: http://www.modpython.org
- wxPython 2.4:
  - http://www.wxpython.org/download.php#binaries
- Jython 2.1
  - http://www.jython.org/download.html