



> **Python et XML**

RMLL 2005



Qui je suis ?

- ◆ Julien Anguenot, Ingénieur R&D Nuxeo
- ◆ « Core » développeur CPS Platform
 - ◆ <http://www.cps-project.org>
 - ◆ <http://svn.nuxeo.org/trac/pub/>
- ◆ « Core » développeur Zope3 / ECM
 - ◆ <http://www.z3lab.org>
 - ◆ <http://svn.nuxeo.org/trac/pub/z3lab/>
- ◆ Contributeur CMF et Zope3
- ◆ XML ?
 - ◆ Je ne suis pas un expert ! Consommateur :-)
 - ◆ OOo XML
 - ◆ XPDL (XML Process Definition Language)



Pas de brevets logiciels en Europe ! :-)





Agenda

- ◆ Python et XML ?
- ◆ Bibliothèques standard (SAX, DOM)
- ◆ Exemples d'extensions XML disponibles
 - ◆ PyXML
 - ◆ 4Suites,
 - ◆ libxml2
- ◆ ElementTree / cElementTree
- ◆ lxml
- ◆ Performances
- ◆ Conclusion




Pourquoi utiliser XML ?

- ◆ Indépendant de l'application
 - ◆ fichier texte !
- ◆ Structure hiérarchique
 - ◆ lisible
 - ◆ choix des tags
- ◆ Indépendant de la plateforme
 - ◆ interopérabilité
- ◆ « Multi-linguisme »
 - ◆ ISO / IEC 10646
 - ◆ unicode
- ◆ Standardisé
 - ◆ w3c (transparentes suivants)



XML et standard

- ◆ Avantages des spécifications XML
 - ◆ Orienté technique
 - ◆ Implémentation possible car définition simple
 - ◆ Les « managers » peuvent comprendre XML :-)
- ◆ XML 1.0 recommandations
- ◆ « namespaces » (espace de noms)
- ◆ Outils très simple pour traiter le XML
 - ◆ Plus facile qu'avec SGML
 - ◆ Python par exemple ! :)
 - ◆ Nombreux
- ◆ Littérature et ressources abondantes



Exemple très (très) simple de fichier XML

```
<?xml version="1.0"
encoding='ISO-8859-15'?>
<person sex='male'>
  <nom>Julien Anguenot</nom>
  <address />
</person>
```



XML c'est aussi....

- ◆ DTD / XML Schema / RDF Schema
- ◆ XSL / XSLT
- ◆ CSS
- ◆ XPath, Xforms, etc...
- ◆ RDF
- ◆ Les différentes technologies couvrent
 - ◆ Mise en forme
 - ◆ Affichage
 - ◆ Utilisation des données



Pourquoi Python ?

- ◆ Language de script orienté objet
- ◆ Typage dynamique
- ◆ Facile à utiliser
- ◆ Facile à lire
- ◆ Multi-plateforme
- ◆ Interpréteur permet des phases de debug simple en testant des fragments de code
- ◆ Structure de base du langage très puissante
- ◆ Modèle object « non obscure »
- ◆ Open source et communauté très active ! (croissance)
- ◆ Culture « hacker »



Pourquoi utiliser Python pour du traitement XML ?

- ◆ De nombreuses bibliothèques disponibles (standard et extensions)
- ◆ Même raison qui font de Python est langage attractif sur d'autres domaines :-) (transparentes précédents)
- ◆ Comparons des programmes réalisant des traitements XML avec Python / Perl / Java (pas besoin de comparaison avec PHP hein :-)



Exemple simple en Perl (DOM) (1/2)

http://xmlfr.org/documentations/articles/exemples/ex_dom

```
use strict;
use XML::DOM;
my $dom = new XML::DOM::Parser;           # create
parser
my $doc;                                  # the dom object
if( $ARGV[0]) { $doc= $dom->parsefile( $ARGV[0]); } #
create the dom object
else   { $doc= $dom->parse( \*STDIN);   }
foreach my $tag ('bktlong', 'bktshort')
  { my $nodes= $doc->getElementsByTagName ( $tag);
    my $n = $nodes->getLength;
    for (my $i = 0; $i < $n; $i++)
      { process_title( $doc, $nodes->item ($i)); }
  }
print $doc->toString;
exit;
```



Exemple simple en Perl (DOM) 2/2

http://xmlfr.org/documentations/articles/exemples/ex_dom

```
sub process_title                                # title handler
{ my ( $doc, $title)= @_;
  my $title_pCDATA= $title->getFirstChild;      # get first text element
  my $title_text= $title_pCDATA->getData;      # get it's data
  my $title_no;
  ($title_no, $title_text)=                    # separate num and the
    ( $title_text=~ /A(\d+)\.?\s*(.*)\Z/);    # rest of the title
  $title->setAttribute( num => $title_no);      # create attribute numm
  my $num= $doc->createElement( 'num');        # create element num
  $title->insertBefore( $num, $title_pCDATA);  # insert it in title
  my $num_pCDATA= $doc->createTextNode( $title_no); # create the text
  $num->appendChild( $num_pCDATA);            # insert it in num
  $title_pCDATA->setData( $title_text);       # set title new text
}
```



Exemple simple en Java (DOM) (1/2)

http://www.sogid.com/javalist/fils2001/exemple_xml_parser.html

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.io.File;
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
public class DomEcho01{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;
    public static void main(String argv[])
    {
        if (argv.length != 1) {
            System.err.println("Usage: java DomEcho filename");
            System.exit(1);
        }
    }
}
```



Exemple simple en Java (2/2) (DOM) (2/2)

http://www.sogid.com/javalist/fils2001/exemple_xml_parser.html

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
//factory.setValidating(true);
//factory.setNamespaceAware(true);
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    document = builder.parse( new File(argv[0]) );
} catch (SAXException sxe) {
    // Error generated during parsing)
    Exception x = sxe;
    if (sxe.getException() != null)
        x = sxe.getException();
    x.printStackTrace();
} catch (ParserConfigurationException pce) {
    // Parser with specified options can't be built
    pce.printStackTrace();
} catch (IOException ioe) {
    // I/O error
    ioe.printStackTrace();
}
} // main
}
```



Exemple simple en PHP (DOM) (1/2)

<http://developpeur.journaldunet.com/tutoriel/php/040927-php-seguy-simplexml-2b.shtml>

```
$dom->loadXML('<books><book><title>blah </title></book></books>');

if (!$dom) {

    echo 'Erreur de traitement XML';

    exit;

}

$liste = $dom->getElementsByTagName('books');

$noeud = $liste->item(0);

$title = $dom->createElement("titre","boo");

$node = $dom->createElement("book");

$node->appendChild($title);
```



Exemple Simple en PHP (DOM) (2/2)

```
$noeud->appendChild($node);
```

```
$s = simplexml_import_dom($dom);
```

```
echo $s->asxml();
```

```
?>
```



Exemple en Python (DOM) (1/1)

<http://www.oreilly.com/catalog/pythonxml/chapter/ch01.html>

```
import pprint

import xml.dom.minidom
from xml.dom.minidom import Node

doc = xml.dom.minidom.parse("books.xml")

mapping = {}

for node in doc.getElementsByTagName("book"):
    isbn = node.getAttribute("isbn")
    L = node.getElementsByTagName("title")
    for node2 in L:
        title = ""
        for node3 in node2.childNodes:
            if node3.nodeType == Node.TEXT_NODE:
                title += node3.data
        mapping[isbn] = title

# mapping now has the same value as in the SAX example:
pprint.pprint(mapping)
```



Conclusion sur les exemples choisis aléatoirement ?

- ◆ Perl
 - ◆ Illisible !
- ◆ Java
 - ◆ lourd
- ◆ PHP
 - ◆ Illisible (moins que Perl mais bon...)
- ◆ Python
 - ◆ La classe :)



Librairies XML standard sur Python

- ◆ SAX
 - ◆ Simple API for XML
- ◆ DOM
 - ◆ Document Object Model

- ◆ Plus ancien parseur « standard »
- ◆ « stream based » parseur
- ◆ Les gestionnaires (handlers)
 - ◆ ContentHandler
 - ◆ ErrorHandler
 - ◆ DTDHandler
 - ◆ EntityResolver
 - ◆ DeclHandler / LexicalHandler
- ◆ Gestionnaires customs avec intelligence

Composants SAX

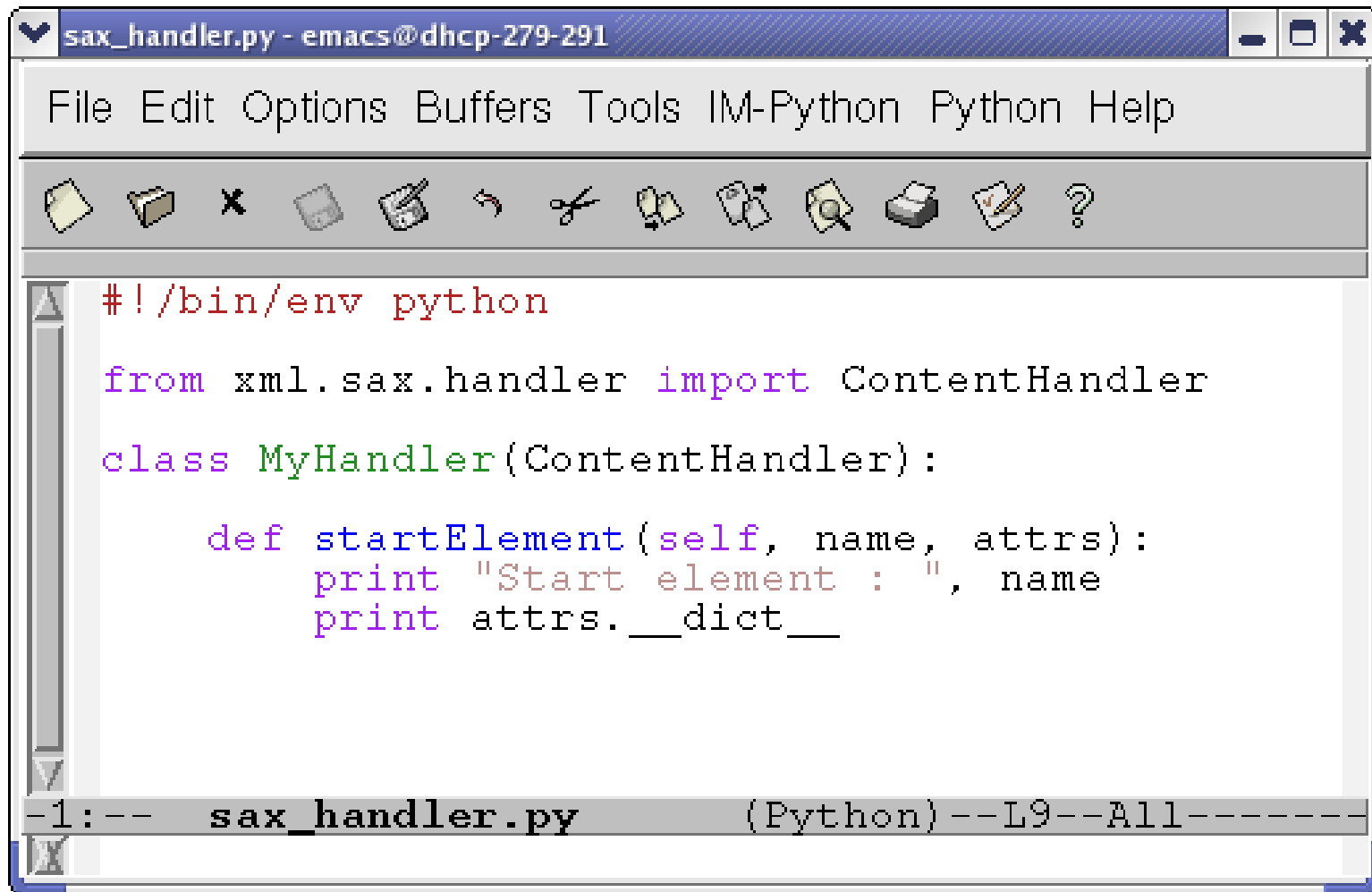


Exemple SAX : fichier XML de tests (1/4)

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
<addressbook>
  <person sex="male">
    <name>Anguenot</name>
    <givenName>Julien</givenName>
    <phone />
  </person>
</addressbook>
```

-0:-- test.xml (nXML Valid)--L3--A11-----

Exemple SAX : définition d'un questionnaire (2/4)



The image shows a screenshot of an Emacs editor window titled "sax_handler.py - emacs@dhcp-279-291". The window contains a Python script for a SAX handler. The script defines a class "MyHandler" that inherits from "ContentHandler" and implements the "startElement" method. The method prints the element name and its attributes. The status bar at the bottom of the window shows the current line and column: "-1:-- sax_handler.py (Python) --L9--A11-----".

```
#!/bin/env python

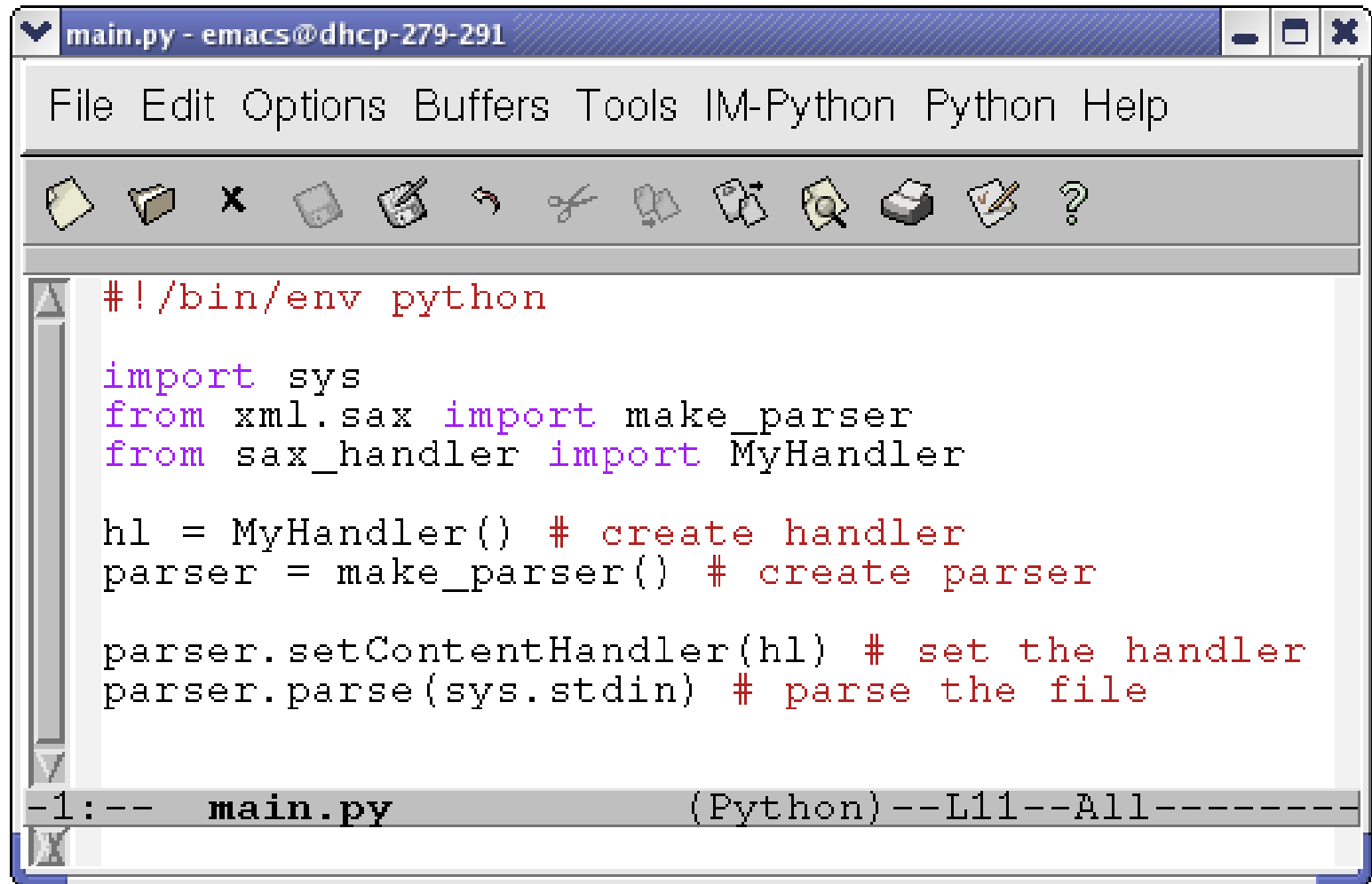
from xml.sax.handler import ContentHandler

class MyHandler(ContentHandler):

    def startElement(self, name, attrs):
        print "Start element : ", name
        print attrs.__dict__
```

-1:-- sax_handler.py (Python) --L9--A11-----

Exemple SAX : programme (3/4)



The image shows a screenshot of an Emacs editor window. The title bar reads "main.py - emacs@dhcp-279-291". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "IM-Python", "Python", and "Help". The toolbar contains various icons for file operations. The main text area contains the following Python code:

```
#!/bin/env python

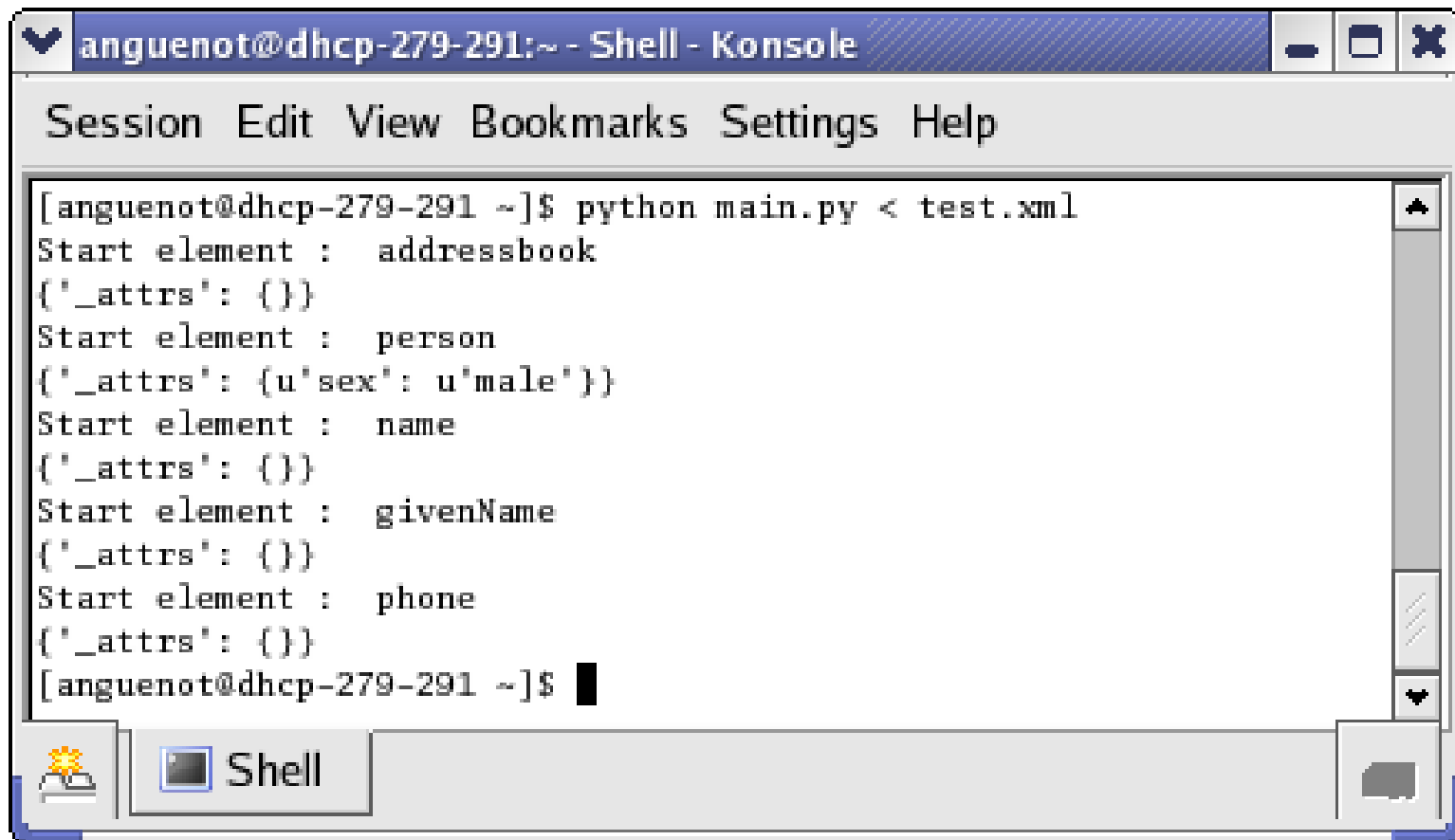
import sys
from xml.sax import make_parser
from sax_handler import MyHandler

h1 = MyHandler() # create handler
parser = make_parser() # create parser

parser.setContentHandler(h1) # set the handler
parser.parse(sys.stdin) # parse the file
```

The status bar at the bottom shows "-1:-- main.py (Python) --L11--A11-----".

Exemple SAX : tester le programme (4/4)



The screenshot shows a terminal window titled "anguenot@dhcp-279-291:~ - Shell - Konsole". The window has a menu bar with "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". The terminal content shows the execution of a Python script:

```
[anguenot@dhcp-279-291 ~]$ python main.py < test.xml
Start element :  addressbook
({'_attrs': {}})
Start element :  person
({'_attrs': {u'sex': u'male'}})
Start element :  name
({'_attrs': {}})
Start element :  givenName
({'_attrs': {}})
Start element :  phone
({'_attrs': {}})
[anguenot@dhcp-279-291 ~]$
```

The terminal window also features a taskbar at the bottom with a "Shell" icon and a "Shell" label.



SAX avantages / inconvénients

- ◆ Avantages

- ◆ Très bon parser « stream based »
- ◆ Rapide
- ◆ Peu coûteux en mémoire
- ◆ Peut gérer de GROS document XML

- ◆ Désavantages

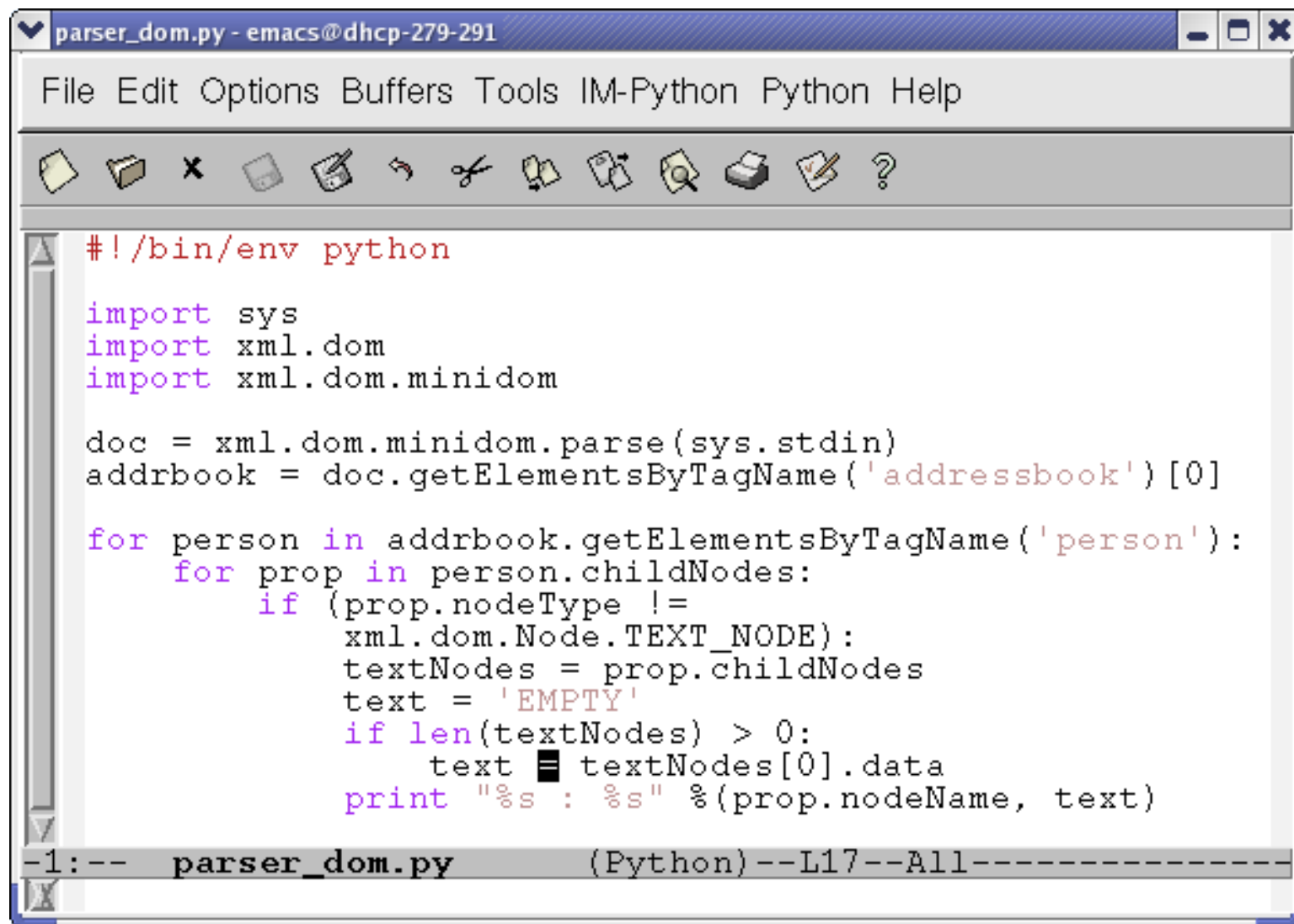
- ◆ Difficilement lisible
- ◆ Les gestionnaires ont tendances à devenir compliqués
- ◆ Pas « pythonique » du tout



DOM : Document Object Model

- ◆ Expose la structure XML de façon programmatique au développeur
- ◆ Documents en mémoire
- ◆ Représentation en arbre
- ◆ Exemple classique
 - ◆ Navigateur / javascript
- ◆ Spécifications DOM
 - ◆ Niveau de spécification
- ◆ Python en standard : minidom et pulldom

Exemple minidom



```
parser_dom.py - emacs@dhcp-279-291
File Edit Options Buffers Tools IM-Python Python Help
[Icons: File, Folder, Close, Save, Print, Undo, Cut, Copy, Paste, Find, Zoom, Help]
#!/bin/env python

import sys
import xml.dom
import xml.dom.minidom

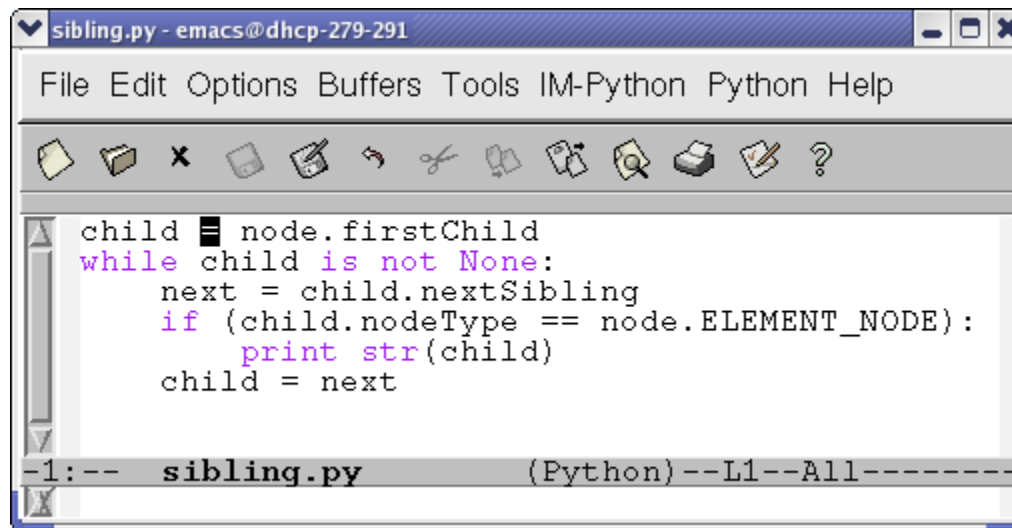
doc = xml.dom.minidom.parse(sys.stdin)
addrbook = doc.getElementsByTagName('addressbook')[0]

for person in addrbook.getElementsByTagName('person'):
    for prop in person.childNodes:
        if (prop.nodeType !=
            xml.dom.Node.TEXT_NODE):
            textNodes = prop.childNodes
            text = 'EMPTY'
            if len(textNodes) > 0:
                text = textNodes[0].data
            print "%s : %s" % (prop.nodeName, text)

-1:-- parser_dom.py (Python)--L17--A11-----
```

Introspection de l'arbre

- ◆ firstChild / lastChild / childNodes
- ◆ getElementByTagName()
- ◆ Siblings (frères suivants)
- ◆ ELEMENT_NODE / TEXT_NODE



The screenshot shows an Emacs editor window titled 'sibling.py - emacs@dhcp-279-291'. The menu bar includes 'File Edit Options Buffers Tools IM-Python Python Help'. The toolbar contains icons for file operations and editing. The main text area contains the following Python code:

```
child = node.firstChild
while child is not None:
    next = child.nextSibling
    if (child.nodeType == node.ELEMENT_NODE):
        print str(child)
    child = next
```

The status bar at the bottom indicates the current line is 1, column 1, and the file is sibling.py in Python mode.



Modification de l'arbre : API

- ◆ createElement()
- ◆ createTextNode()
- ◆ appendChild(newChild)
- ◆ InsertBefore(newChild, refChild)
- ◆ replaceChild(newChild, oldChild)
- ◆ removeChild(oldChild)



DOM avantages / inconvénients

- ◆ Avantages
 - ◆ Représentation en arbre
 - ◆ API de assez haut niveau
- ◆ Inconvénients
 - ◆ Gourmand en mémoire
 - ◆ Lent sur des gros documents



Quelques extensions XML pour Python

- ◆ PyXML
 - ◆ <http://pyxml.sf.net>
- ◆ 4Suites
 - ◆ <http://4suite.org/index.xhtml>
- ◆ Libxml2
 - ◆ <http://xmlsoft.org/python.html>

- ◆ Collection de bibliothèques Python pour XML
- ◆ xmlproc
 - ◆ Parseur validateur XML
- ◆ expat
 - ◆ Parseur non validateur XML (rapide)
- ◆ PySAX
 - ◆ SAX1 et SAX2
- ◆ 4DOM
 - ◆ Implémentation DOM niveau 2
- ◆ + d'autres

- ◆ Ft.Xml.Domlette
 - ◆ DOM orienté XPath (10x plus rapide que 4DOM)
 - ◆ Extensions C
- ◆ Ft.Xml.XPath
 - ◆ XPath 1.0 pour Domlette
- ◆ Ft.Xml.Xslt
 - ◆ Processeur XSLT 1.0
- ◆ Ft.Rdf
- ◆ Ft.Server
 - ◆ Repository et documents RDF / Web
- ◆ Ft.lib



Exemple Domlette

```
OOCalcHelperDomlette.py - emacs@localhost.localdomain
File Edit Options Buffers Tools IM-Python Python Help
[Icons: File, Folder, Close, Save, Undo, Cut, Copy, Paste, Find, Print, Help]
    phome_path = phome.__path__[0]
    dtd_dir = os.path.join(phome_path, 'dtd')
    return os.path.join(dtd_dir, 'office.dtd')
def _getParsedXMLFromString(self, xml_str, uri=''):
    """Returns a parsed XML doc

    Instance of a Domlette parser
    """
    _factory = InputSource.DefaultFactory
    _doc = _factory.fromString(xml_str, uri=uri)
    return Domlette.NonvalidatingReader.parse(_doc)
def _getZipFileFromFile(self, file):
    """Returns a ZipFile instance given a file

--:-- OOCalcHelperDomlette.py (Python)--L109-- 6%-----
```



Conclusion cDomlette

- ◆ 10x plus rapide que 4DOM
 - ◆ Mais pas suffisant...
- ◆ Implémentation très « light » de DOM
 - ◆ Pas complètement compatible avec 4DOM
- ◆ We want more !



libxml2

- ◆ C implementation
- ◆ Bindings sur plusieurs langages
 - ◆ libxml2-python
 - ◆ lxml
 - ◆ Pour Pascal également ! :-)
- ◆ « full featured » => XPATH / XSLT ...
- ◆ XML experts !
- ◆ FAST FAST FAST !!!!!



Exemple libxml2-python

```
libxml2.py - emacs@localhost.localdomain
File Edit Options Buffers Tools IM-Python Python Help
[Icons: File, Folder, Close, Save, Undo, Cut, Copy, Paste, Find, Print, Help]
import libxml2, sys
doc = libxml2.parseFile("tst.xml")
if doc.name != "tst.xml":
    print "doc.name failed"
    sys.exit(1)
root = doc.children
if root.name != "doc":
    print "root.name failed"
    sys.exit(1)
child = root.children
if child.name != "foo":
    print "child.name failed"
    sys.exit(1)
doc.freeDoc()
-1:** libxml2.py (Python) --L1--All-----
```



Défauts de libxml2-python

- ◆ Trop bas niveau et « C styled »
- ◆ Pas assez de doc
- ◆ UTF-8 sur l'API versus Python unicode strings
- ◆ Génère des « segfaults » depuis Python



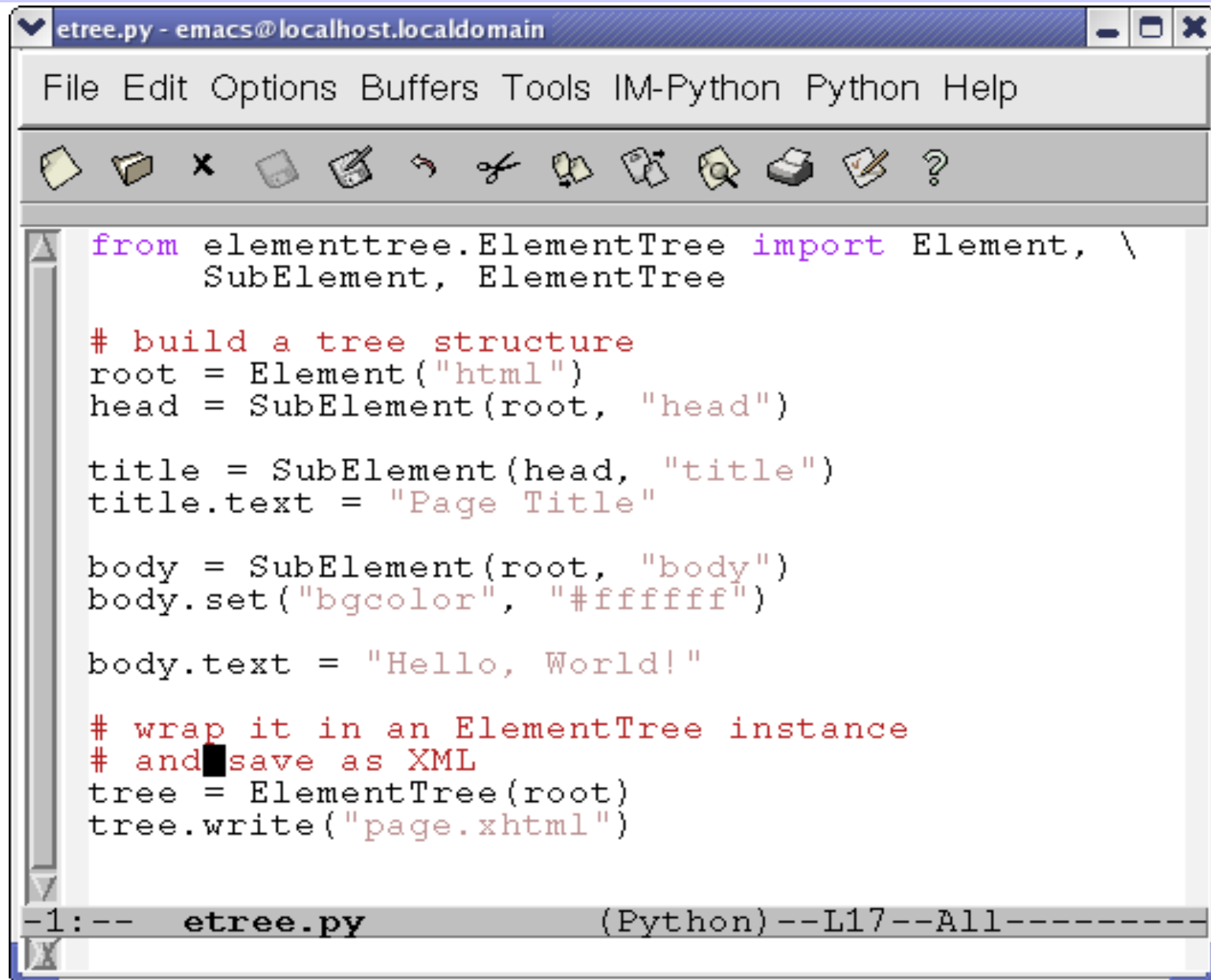
- ◆ "XML with elementtree is what makes me never have [to] think about XML again."
 - ◆ Istvan Albert



ElementTree / cElementTree

- ◆ Nouvelle API
- ◆ Plus « Pythonique »
- ◆ Notion d'élément manipulant des structures Python
 - ◆ dictionnaires / listes
 - ◆ plus flexible
- ◆ Version C extrêmement rapide
- ◆ Prenons un exemple

Exemple ElementTree (1/2)



```
etree.py - emacs@localhost.localdomain
File Edit Options Buffers Tools IM-Python Python Help
[Icons: New, Open, Save, Undo, Redo, Cut, Copy, Paste, Find, Print, Help]

from elementtree.ElementTree import Element, \
    SubElement, ElementTree

# build a tree structure
root = Element("html")
head = SubElement(root, "head")

title = SubElement(head, "title")
title.text = "Page Title"

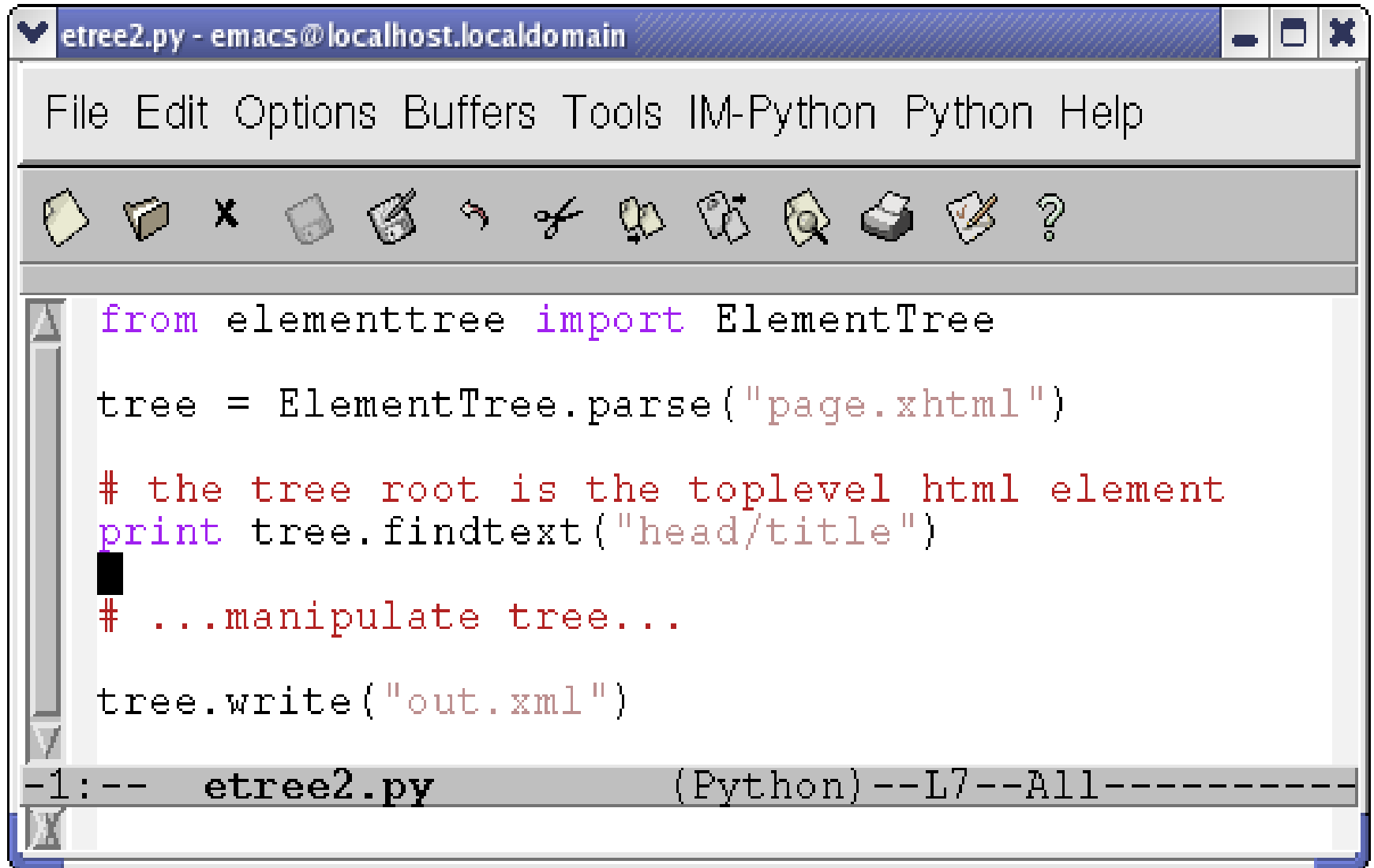
body = SubElement(root, "body")
body.set("bgcolor", "#ffffff")

body.text = "Hello, World!"

# wrap it in an ElementTree instance
# and save as XML
tree = ElementTree(root)
tree.write("page.xhtml")

-1:-- etree.py (Python)--L17--All-----
```

Exemple2 ElementTree (2/2)



The screenshot shows an Emacs editor window titled "etree2.py - emacs@localhost.localdomain". The window has a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "IM-Python", "Python", and "Help". Below the menu bar is a toolbar with various icons for file operations and editing. The main text area contains the following Python code:

```
from elementtree import ElementTree
tree = ElementTree.parse("page.xhtml")
# the tree root is the toplevel html element
print tree.findtext("head/title")
# ...manipulate tree...
tree.write("out.xml")
```

At the bottom of the window, a status bar shows "-1:-- etree2.py (Python) --L7--All-----".



Conclusion ElementTree

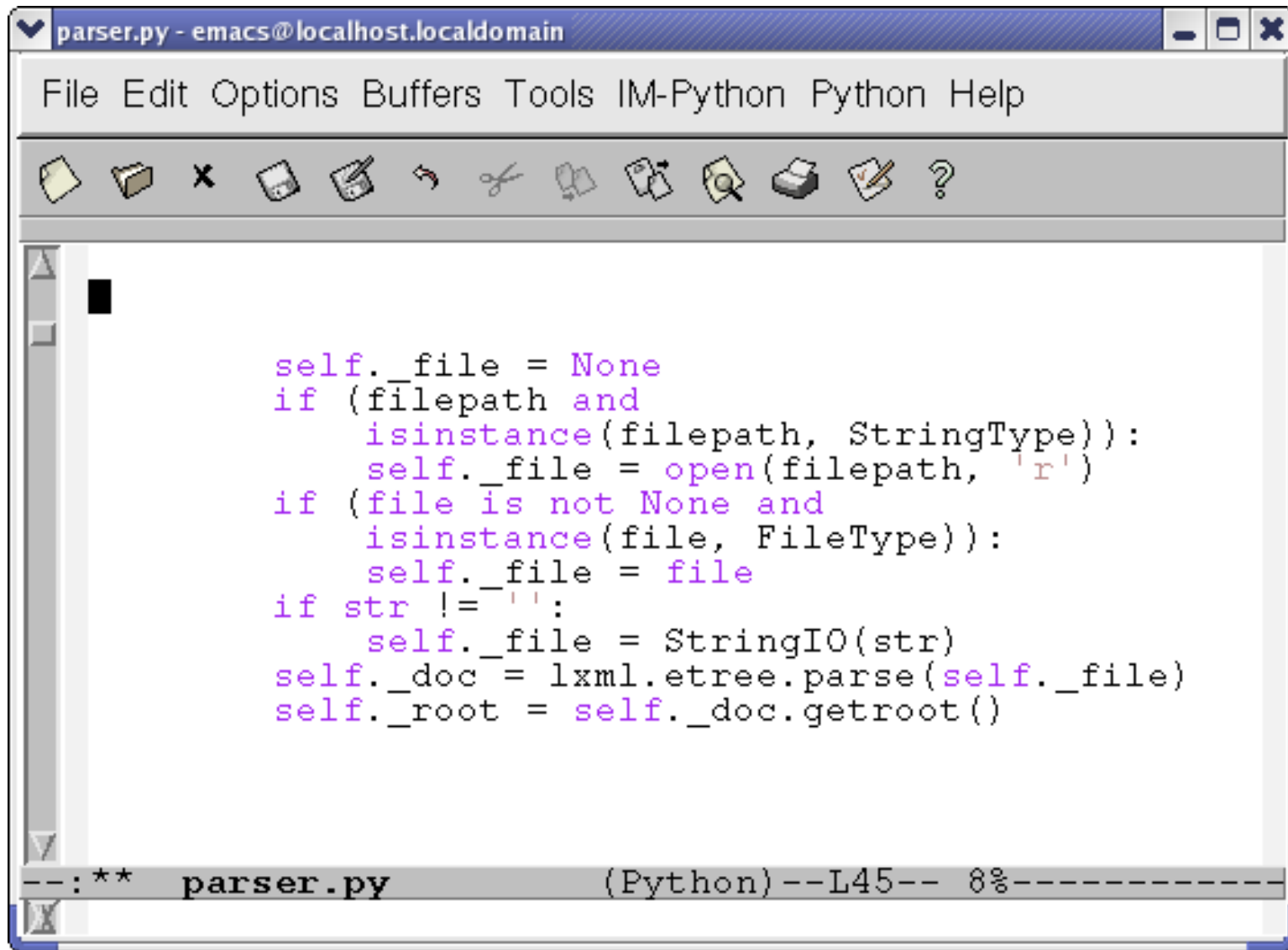
- ◆ Avantages
 - ◆ API « pythonique »
 - ◆ Facile à utiliser
 - ◆ Très rapide (version C)
- ◆ Inconvénients
 - ◆ Support XPath non natif
 - ◆ Pas de XSLT
 - ◆ Sérialisation utilise des aliases sur les namespaces qui n'est pas possible d'éviter...



lxml (<http://codespeak.net/lxml>)

- ◆ Binding Python sur libxml2 / libxslt
- ◆ Implémente l'API de ElementTree
 - ◆ Compatible
- ◆ Extensions
 - ◆ XPath / Relax NG / XML Schema / XSLT
- ◆ Pas de « segfault » depuis Python
- ◆ Bien documentée
- ◆ Optimisation de code avec PyRex

Exemple simple lxml parser (1/2)



The screenshot shows an Emacs editor window titled "parser.py - emacs@localhost.localdomain". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "IM-Python", "Python", and "Help". The toolbar contains icons for file operations like opening, saving, and printing. The main text area contains the following Python code:

```
self._file = None
if (filepath and
    isinstance(filepath, StringType)):
    self._file = open(filepath, 'r')
if (file is not None and
    isinstance(file, FileType)):
    self._file = file
if str != '':
    self._file = StringIO(str)
self._doc = lxml.etree.parse(self._file)
self._root = self._doc.getroot()
```

The status bar at the bottom of the window displays "--: ** parser.py (Python) --L45-- 8%-----".



Exemple simple lxml / xpath (2/2)

The screenshot shows an Emacs editor window titled "parser.py - emacs@localhost.localdomain". The menu bar includes "File Edit Options Buffers Tools IM-Python Python Help". The toolbar contains icons for file operations like opening, saving, and printing. The main text area displays the following Python code:

```
def _elts(self, elt, tag_name, ns=xpdlns_1_0):  
    """Get all elements given a tag name using a given namespace  
    """  
    return elt.xpath('ns:%s'%tag_name, {'ns' : ns})
```

The status bar at the bottom shows "--:** parser.py (Python) --L65--11%--" with a dashed line indicating the current position in the file.

- ◆ Issue du monde Zope
 - ◆ Martijn Faasen
- ◆ Activement maintenu
- ◆ Sera intégré dans le « core » Zope3 très prochainement
- ◆ Grande classe :)
- ◆ « More in the pipe »



Performances des différents parseurs

- ◆ Frederik Lundh benches (devel de ElementTree)
- ◆ Pas subjectif du tout ;)

library	memory	time

minidom (python 2.1)	80000k	6.5s
minidom (python 2.4)	53000k	1.4s
ElementTree 1.3	14500k	1.1s
cDomlette	20500k	0.540s
libxml2	16000k	0.098s
lxml	??	??
cElementTree 0.8	5700k	0.058s

readlines (read as text)	5050k	0.032s



Conclusion Python / XML

- ◆ Python est un langage de choix pour réaliser des traitements XML.
- ◆ XML possède un grand nombre d'avantages
- ◆ Utiliser XML en Python c'est nécessaire pour des questions d'interopérabilité
- ◆ Mais c'est aussi un boulet que l'on se traîne au pied Python ou pas Python...
- ◆ XML n'est pas toujours la solution en Python :
 - ◆ Utiliser XML avec Perl donne l'impression que XML est de la programmation « agile »
 - ◆ Ce n'est pas le cas en Python :)
- ◆ XML n'est pas la solution a tous les problèmes