Python crash course

2010-01-15 v0.07

Philippe Lagadec

http://www.decalage.info

Introduction

- “Python is an **easy to learn, powerful programming language**. It has efficient **high-level** data structures and a simple but effective approach to **object-oriented programming**. Python's **elegant syntax** and **dynamic typing**, together with its **interpreted** nature, make it an ideal language for scripting and **rapid application development** in many areas on **most platforms**.”
  - *The Python Tutorial, http://docs.python.org/tutorial/*

Why “Python” ?

- “Guido van Rossum, the creator of the Python language, named the language after the BBC show "**Monty Python's Flying Circus**". He doesn't particularly like snakes that kill animals for food by winding their long bodies around them and crushing them.” ☺

Swaroop C H.

Download a Python interpreter

- Python official website: http://www.python.org
- Download: http://www.python.org/download
- Warning: for now (Jan 2010), Python **2.6** or **2.5** are still recommended.
  - Many third-party modules have issues with Python 3.x, which is very different from v2.x.

Install a Python-aware editor

- On Windows, I recommend PyScripter to get an effective IDE with syntax highlighting, code completion, integrated Python shell and debugger: http://mmm-experts.com/Products.aspx?ProductId=4 or http://code.google.com/p/pyscripter/ for latest versions.
- On other OSes such as Linux or MacOSX, try Eclipse + PyDev.

Python Documentation

- On Windows, the manual provided with the interpreter (Start menu / All Programs / Python / Python Manuals) is usually the most convenient way to access the Python documentation.
  - Use the index tab !
- Online official documentation startpoint: http://www.python.org/doc
- (pretty long) official tutorial: http://docs.python.org/tut/tut.html

Python shell

- On Windows, use Start Menu / Python / Python command line.
- Alternatively you may run "**python**" from a CMD window.
  - but since python.exe is not in the PATH environment variable by default you may want to add it, or type its complete path such as "C:\python25\python.exe".
- Quit with Ctrl+Z or simply close the window when finished.

Python basics

Variables and Constants

- Variables are simply names which point to any value or object:

```
a_string = "hello, world"
an_integer = 12
a_float = 3.14
a_boolean = True
nothing = None
```

- Dynamic typing: the value or type of a variable may be changed at any time.

Print

- To print a constant or variable:

```
print "hello, world"
print a_string
print 12
print (5+3)/2
```

- To print several items, use commas (items will be separated by spaces):

```
print "abc", 12, a_float
```

Long lines

- A long statement may be split using a **backslash**:

```
my_very_long_variable = something + \
    something_else
```

- It is not necessary if there are parentheses or square brackets:

```
average_value = (some_value1 + some_value2 +
    some_value3) / 3
```

Strings

- There are 3 syntaxes for string constants:

```
string_single_quotes = 'abc'

string_double_quotes = "abc"

string_triple_quotes = """this is
a multiline
string."""
```

Strings

- It's useful when we need to include quotes in a string:

```
string1 = 'hello "world"'
string2 = "don't"
```

- otherwise we have to use backslashes:

```
string2 = 'don\'t'
```

- Be careful about backslashes in paths:

```
Win_path = 'C:\\Windows\\System32'
```

String operations and methods

- Strings are objects which support many operations:

```
strings = string1 + " : " + string2
```

- Get the length of a string:

```
len(strings)
```

- Convert to uppercase:

```
strings_uppercase = strings.upper()
```

- Strip spaces at beginning and end of a string:

```
stripped = a_string.strip()
```

- Replace a substring inside a string:

```
newstring = a_string.replace('abc', 'def')
```

- All string methods: http://docs.python.org/lib/string-methods.html
- Note: a string is **immutable**, all operations create a new string in memory.

Conversions

- Convert a string to an integer and vice-versa:

```
i = 12
s = str(i)


s = '17'
i = int(s)
```

Building strings

- Several ways to include a variable into a string:
  - by concatenation:

```
string1 = 'the value is ' + str(an_integer) + '.'
```

  - by "printf-like" formatting:

```
string2 = 'the value is %d.' % an_integer
```

- With several variables, we need to use parentheses:

```
a = 17
b = 3
string3 = '%d + %d = %d' % (a, b, a+b)
```

Building strings

- To include strings into another string:

```
stringa = '17'
stringb = '3'
stringc = 'a = ' + stringa + ', b = ' + stringb
stringd = 'a = %s, b= %s' % (stringa, stringb)
```

- Everything about string formatting: http://docs.python.org/library/stdtypes.html#string-formatting-operations

Lists

- A list is a dynamic array of any objects. It is declared with square brackets:

```
mylist = [1, 2, 3, 'abc', 'def']
```

- Access a specific element by index (index starts at zero):

```
third_item = mylist[2]
```

List operations

- Operations on a list:

```
mylist[2] = 'new value'
mylist.append('another value')
mylist.remove('abc')
length = len(mylist)
longlist = mylist + [4,5,6]
```

- All list operations: http://docs.python.org/lib/typesseq.html

Slicing

- Slicing is extracting a sublist from a list:

```
first_two_items = mylist[0:2]
third_and_fourth = mylist[2:4]
fourth_up_to_end = mylist[3:]
last_two_items = mylist[-2:]
first_two_items2 = mylist[:2]
```

More complex lists

- Lists may contain lists:

```
mylist2 = [mylist, 'abc', mylist,
  [1, 2, 3]]
```

- It works like a two-dimensions array:

```
item1 = mylist2[3][2]
item2 = mylist2[0][4]
```

Tuples

- A tuple is similar to a list but it is a fixed-size, immutable array: once a tuple has been created, its elements may not be changed, removed, appended or inserted.
- It is declared using parentheses and comma-separated values:

```
a_tuple = (1, 2, 3, 'abc', 'def')
```

- But parentheses are optional:

```
a_tuple = 1, 2, 3, 'abc', 'def'
```

- Tuples may be seen as “complex constants”.

Dictionaries

- A dictionary is a mapping between indexes to values.
- Indexes may be almost any type: integers, strings, tuples, objects…
- Example:

```
countries = {'us':'USA', 'fr':'France',
  'uk':'United Kingdom'}
print countries['uk']
countries['de'] = 'Germany'
```

Dictionary operations

- Get a list of all indexes:

```
country_codes = countries.keys()
```

- Get a list of (index, value) tuples:

```
Countries_list = countries.items()
```

- Test if a specific index is there:

```
is_uk_there = 'uk' in countries
```

- More info: http://docs.python.org/lib/typesmapping.html
- And http://docs.python.org/tutorial/datastructures.html#dictionaries

Blocks and Indentation (control flow)

- Blocks of code are delimited using **indentation**, either **spaces or tabs** at the beginning of lines.
  - This is one of the main differences of Python over other languages, and usually the main reason why people love it or hate it. ;-)
- Tip: NEVER mix tabs and spaces in a script, it may result in tricky bugs.
  - From my experience, the safest solution is to **always use 4-spaces indents, never tabs**. (because each editor may convert tabs either to 2, 4 or 8 spaces)

if / else

```
if a == 3:
     print 'The value of a is:'
     print 'a=3'


if a != 'test':
     print 'a is not "test"'
     test_mode = False
else:
     print 'The value of a is:'
     print 'a="test"'
     test_mode = True
```

if / elif / else

```
if choice == 1:
    print “First choice.”
elif choice == 2:
    print “Second choice.”
else:
    print “Wrong choice.”
```

While loop

```
a=1
while a<10:
        print a
        a += 1
```

For loop

```
for a in range(10):
    print a


my_list = [2, 4, 8, 16, 32]
for a in my_list:
    print a
```

Sorting

- Use sorted() to get a sorted version of a list/dict or any other iterable.
- Example:

```
my_list = [32, 4, 16,  8, 2]
for a in sorted(my_list):
        print a
```

- See help for more advanced sorting: http://docs.python.org/library/functions.html?highlight=sorted#sorted

Functions

- See tut1a_functions.py
- And tut1b_default_args.py

Exceptions

- TODO…

- Any error raises an exception:

```
def average (items):
    return sum(items) / len(items)
print average([1, 3, 7])
print average([])
```

Classes and Objects

Object-oriented programming

Classes and objects

- With Python it is possible to use either procedural or object-oriented programming, but most applications mix both.
- You may use objects to model complex data structures in an organized, self-contained way.

A simple class - Attributes

- Attributes are defined in the constructor method called __init__().
- “self” is a variable pointing to the object itself.
- Each attribute is stored in self.attribute.

```
class Person:

    def __init__(self, lastname, firstname, age):
        self.lname = lastname
        self.fname = firstname
        self.age = age

john = Person('Doe', 'John', 45)
print john.fname, “is”, john.age
john.age = 46
```

Methods

- Like a function defined inside the class.
- “self” must always be the first argument of a method. It represents the object itself.

```
class Person:

    def __init__(self, lastname, firstname, age):
        self.lname = lastname
        self.fname = firstname
        self.age = age

    def get_fullname(self):
        return '%s %s' % (self.fname, self.lname)

    def add_years(self, years):
        self.age += years

john = Person('Doe', 'John', 45)
print john.get_fullname()
john.add_years(2)
```

Common error: missing self

- When adding a new method, it's a common mistake to forget "self".

```
class Person:
    […]
    def display():
        print self.get_fullname()

john = Person('Doe', 'John')
john.display()
```

- Here is the cryptic error message you get in this case:
  - `TypeError: display() takes no arguments (1 given)`
- Self is a hidden argument, hence "1 given".

Attributes are flexible

- Contrary to other languages like Java or C++, attributes are not protected or private.
- Attributes may be modified from anywhere outside of the class.
- Attributes may even be created or deleted by any method, or from outside.
- Simple naming convention: an attribute starting with underscore is expected to be private. But it’s only a convention.

Initializing list and dict attributes

- Common pitfall: never use an empty list or dict as default value for an attribute.
- All objects would then a get a pointer to the same list/dict in memory.
- Tip: use “None” instead:

```
class Person:

    def __init__(self, lastname, firstname, age, children=None):
        self.lname = lastname
        self.fname = firstname
        self.age = age
        self.children = children
        if children == None:
            self.children = []
```

The special __str__() method

- Used to return a custom string representation of an object.
- Will be called when Python needs to convert an object to a string, such as:
  - `print my_object`
  - `s = str(my_object)`
  - `s = "the object is %s." % my_object`
- See tut3a_class.py

Inheritance

- See tut3b_class_inherit.py

Class attributes

- TODO

Class methods

- TODO

Standard Library

Overview: a few useful modules

Tip

- The CHM Python manual installed with the Python interpreter is very handy:
  - Start Menu / All Programs / Python / Python Manuals
- Use the index tab to quickly find anything

Script arguments

- **sys.argv** is a list containing all arguments that were specified when launching a script.

```
import sys
print “all arguments:”, sys.argv
print “number of args:”, len(sys.argv)
print “first arg:”, sys.argv[1]
```

Launching a process/command

- Simplest way:

```
import os
os.system(“net accounts”)
```

- More effective and flexible:
  - See subprocess module
    - Capture output
    - Control keyboard input
    - Execution in parallel

Exercises

- 1) Write a script which computes the SHA1 hash of your name.
- 2) Compute the SHA1 hash of a file given as argument.

Solution

- TODO…

Network protocols

- Example 1: download a web page (HTTP client)
  - See urllib2
- Example 2: send an e-mail (SMTP client)
  - See smtplib

Simple XML parsing

using ElementTree

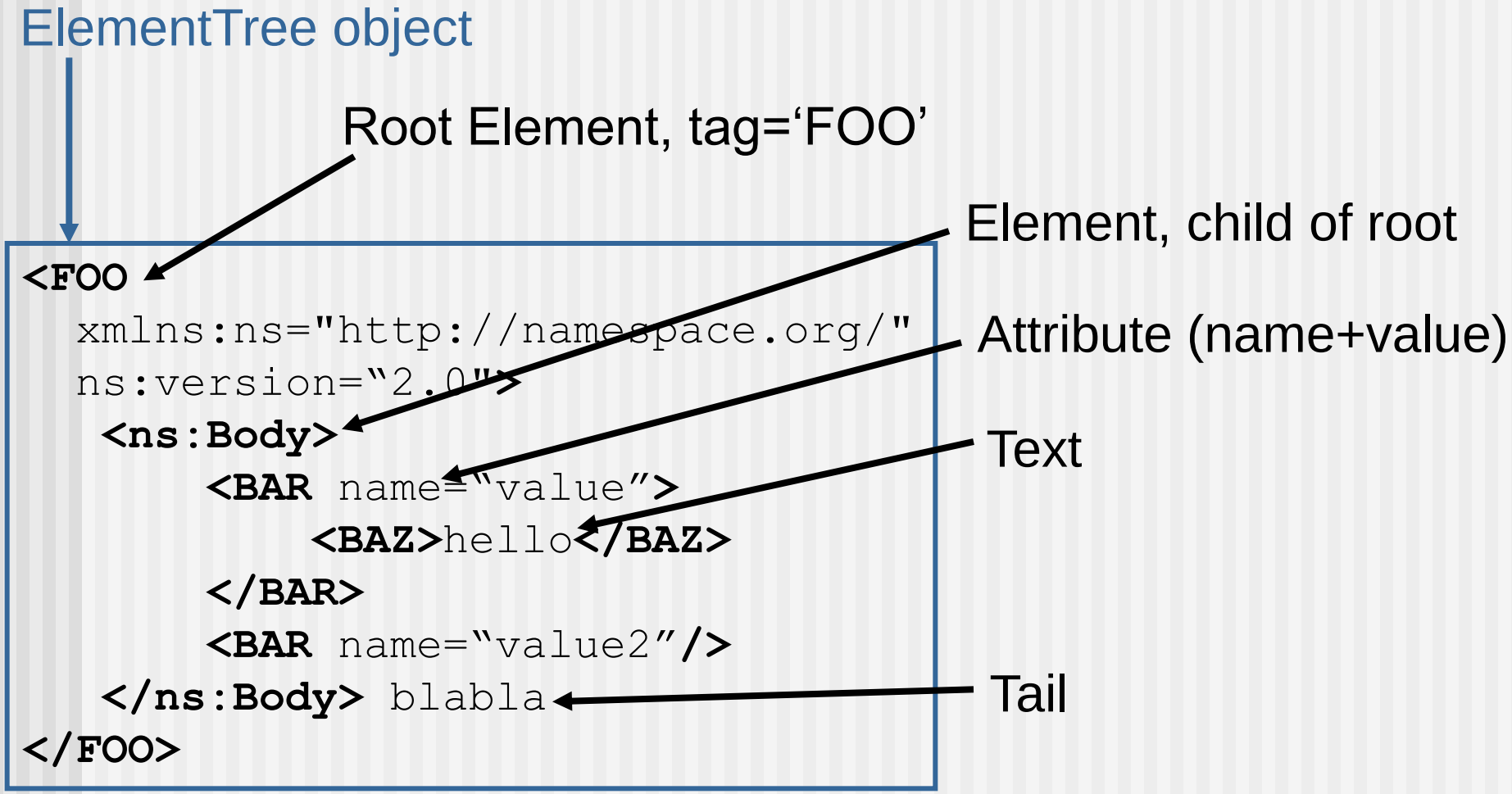
XML parsers

- There are several XML parsers for Python, usually with DOM or SAX API.
  - Quite complex
- ElementTree provides a simpler, more pythonic API to handle XML
  - Included in Python standard library since v2.5

ElementTree: base concepts

- An **ElementTree** object is a representation in memory of a complete XML file (tree with a root)
- An **Element** object is a single XML node. It has:
  - A **tag name**, with optional **namespace**
  - **Attributes** (optional) with values
  - **Text** (optional)
  - **Children** (optional sub-elements)
  - **Tail** (optional)

Element / ElementTree example



First, import the “ET” module

- Tip to support all Python versions:

```
try:
        # ElementTree in Python 2.5
        import xml.etree.ElementTree as ET
except:
    try:
        # external ElemenTree for Python <=2.4
        import elementtree.ElementTree as ET
    except:
        raise ImportError, \
              "the ElementTree module is not installed: "+\
              "see http://effbot.org/zone/element-index.htm"
```

Parsing an XML file

- **ET.parse** returns an ElementTree object:
  - `tree = ET.parse('myfile.xml')`
- To obtain the root Element, use **getroot**:
  - `elem = tree.getroot()`

Parsing a string containing XML

- Use **ET.fromstring**:

```
xmlstring = “<FOO>…</FOO>”
root = ET.fromstring(xmlstring)
```

Element data

- Tag name:
  - `print elem.tag`
- Text and tail:
  - `print elem.text`
  - `print elem.tail`
- Attributes and values (dictionary):
  - `print elem.attrib`
- List of children (sub-elements):
  - `print list(elem)`

Element Attributes

- Iterating over the attributes (dictionary):

```
for name in elem.attrib:
    value = elem.attrib[name]
    print '%s="%s"' % (name, value)
```

- Or simply:

```
for name, value in elem.attrib.items():
    print '%s="%s"' % (name, value)
```

Finding a tag

- Find the first child with a given tag:

  ```
  elem.find("FOO")
  ```

- Obtain a list of children with that tag:

  ```
  elem.findall("FOO")
  ```

- Look for a tag in the whole sub-tree:

  ```
  elem.getiterator("FOO")
  ```

- More info: http://effbot.org/zone/element.htm#searching-for-subelements

Finding a tag by its path

- Knowing all tags on the path:

  ```
  elem.find("FOO/BAR/BAZ")
  ```

- To look for that path anywhere:

  ```
  elem.find(".//FOO/BAR/BAZ")
  ```

- Paths may contain "*"

  ```
  elem.find("FOO/*/BAZ")
  ```

- More info: http://effbot.org/zone/element-xpath.htm

Namespaces

- Modern XML formats use more and more namespaces.
- When XML data contains namespaces, each tag name MUST contain its full namespace URL in ElementTree as "**{**namespace URL**}**tag".
- Sample XML data:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">…
```

- Corresponding ElementTree code:

```
envelope = tree.find(
    "{http://schemas.xmlsoap.org/soap/envelope/}Envelope")
```

Namespaces

- Tip: to ease development and clarify code, prepare all useful tags in global variables before using them.

```
NS_SOAP="http://schemas.xmlsoap.org/soap/envelope/"
ENVELOPE = "{%s}Envelope" % NS_SOAP
BODY = "{%s}Body" % NS_SOAP

root = ET.fromstring(xml_soap_message)
elem_env  = root.find(ENVELOPE)
elem_body = elem_env.find(BODY)
```

More about ElementTree

- More complete tutorials about parsing, generating and editing XML with ElementTree:
  - http://effbot.org/zone/element-index.htm
  - http://effbot.org/zone/element.htm
  - http://www.decalage.info/en/python/etree

Simple web applications

Using CherryPy

Simplest web application

```
import cherrypy

class HelloWorld:
    def index(self):
        return "Hello world!“
    index.exposed = True

cherrypy.quickstart(HelloWorld())
```

Two pages

```
import cherrypy

class TwoPages:
    def index(self):
        return '<a href="page2">Go to page 2</a>'
    index.exposed = True

    def page2(self):
        return "This is page 2!"
    page2.exposed = True

cherrypy.quickstart(TwoPages())
```

CherryPy tutorial

- Tutorial:
  - http://www.cherrypy.org/wiki/CherryPyTutorial
- Sample scripts
  - See also local samples in your Python directory, such as:
    - C:\Python25\Lib\site-packages\cherrypy\tutorial\
- Documentation:
  - http://www.cherrypy.org/wiki/TableOfContents