Introduction to Python as a Functional, Object-Oriented Programming Language

Harold Boley, UNB CS,
NRC Semantic Web Laboratory,
Fredericton, NB
January 13, 2005

Based on “Python” slides by:

K. Naik, M. Raju and S. Bhatkar

Outline

- Introduction
- Installation and Use
- Distinct Features
- Python Basics
- Functional Example
- Comparisons with Other Languages
- Areas of Application
- References

Introduction

- What is Python?

  - Interpreted
  - Interactive
  - Portable
  - Functional
  - Object-Oriented

  programming language

Introduction

- A brief History
  - Invented in 1990 by Guido Van Rossum
  - The name ‘Python’ stems from *"Monty Python's Flying Circus"*
  - Intended to be a scripting language on Amoeba OS
  - Python was influenced by ABC and Modula-3
  - First public release was in 1991

Introduction

- Goals
  - Designed to be simple yet powerful
  - Allow modular programming
  - Great emphasis on readability
  - Rapid application development
  - Easy to embed in and extend with other languages

Introduction

- (Web) Applications
  - Google PageRank algorithm and Interfaces to the Google API
  - Tim Berners-Lee's general-purpose data processor, cwm, for the Semantic Web
  - Scripts for 'gluing' NRC subsystems, e.g. within inDiscover, as pioneered by Daniel Lemire
  - More: Python Package Index

Installation and Use

- Freely available at http://www.python.org/download
- You can download the appropriate installation for your computer
- Can be used in both interactive and batch mode
- IDLE is the editor for writing and running Python programs

Distinct Features

- Extensible (packages)
- Embeddable into applications
- Functional programming
- Object-Oriented programming
- Rapid Prototyping
- Great for readability and presentation
- White space is significant
- Low maintenance costs
- Exception handling
- Free (open source)

Python Basics

Built-in data structures

- Numbers
  - decimal e.g. 631, 3.14
  - octal e.g. O631
  - hexadecimal e.g. oxABC
  - complex e.g. 1 + 3j
  - long e.g. 122233445656455L

  - Normal Arithmetic and Bit operators
  - Integer division truncates e.g. 1/2 = 0

Python Basics

- Strings
  - Concatenation
    - “Hello” + “World” -> “HelloWorld”
  - Repetition
    - “UMBC” * 3 -> “UMBCUMBCUMBC”
  - Indexing
    - “UMBC”[0] -> “U”
  - Slicing
    - “UMBC”[1:3] -> “MB”
  - Size
    - len(“UMBC”) -> 4

Python Basics

- Comparison
  - “UMBC” < “umbc” -> 0
- Search
  - “M” in “UMBC” -> 1

- Can also be enclosed in single quotes
  - e.g. ‘UMBC’

Python Basics

- Lists
  - e.g.

    aList = [631, “Programming languages”,
    [331, “programming languages”]]

  - List items need not have the same type
  - Like indexable arrays, not Lisp-like linked list
  - Extended at right end, not Lisp-like at left end: tail.append(head), not cons(head,tail)
  - Same operators as for strings
  - More operations append(), insert(), pop(), reverse() and sort()

Python Basics

- Tuples
  - E.g.

    aTuple = (631, “Programming Languages”,
    (611, “Computer Architecture”))

  - Nesting is Possible
  - Outer Parentheses are optional
  - Unlike lists and like strings tuples are immutable

Python Basics

- Dictionaries (cf. Lisp property lists, RDF, ...)
  - E.g.

    Map = {“Guido”: “Python”, “Milner”: “ML”}

  - Lookup Map[“Guido”] returns “Python”
    Map[“Milner”] returns “ML”
  - Insert Map[“Ritchie”] = “C”
  - Delete del Map[“Milner”]
  - Iterations keys() values() items()
  - Presence has_key(“Guido”)

  - Values could be anything
  - Keys must be immutable

Python Basics

- Variables
  - No need to declare
  - Not typed

    E.g. F = 2 * 4.5
  - Need to initialize
  - Almost everything can be assigned to a variable (functions, modules, classes)

Python Basics

- References
  - a = b does not make copy of b
  - b = a, a and b refer to the same object

E.g.

```
>>> a = [1,2,3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

Python Basics

- Flow of Control
  - if condition : statements
    (elif condition : statements)*
    [else : statements]
  - while condition : statements
  - for var in sequence : statements
  - break
  - continue

Python Basics

- An Example

(Fibonacci series)

```
>>> a = 0
>>> b = 1
>>> while b < 1000
...         print b
...         a, b = b, a + b
```

Python Basics

- Procedures and Functions
  - General Form

```
def(arg1, arg2, …)
    Statements
    return                    # from procedure        OR
    return expression         # from function
```

Procedures can omit any 'return'
E.g.

```
>>> def fibproc(n):     # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while b < n:
...         print b,
...         a, b = b, a+b
...
>>> fibproc(1000) # calling the procedure
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Python Basics

- Pure Functions
  - It is simple to write a function that returns a list of the numbers of the Fibonacci series, instead of a procedure printing them:

```
>>> def fibfun(n): # return Fibonacci series up to n
... """Return a list containing the Fibonacci series up to n."""
... result = []
... a, b = 0, 1
... while b < n:
...         result.append(b)
...         a, b = b, a+b
... return result
...
>>> fibfun (1000) # calling the function
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
```

Python Basics

- Modules
  - A module is a file containing Python definitions and statements
  - File should have suffix .py
  - Within a module, the module's name is available as through global variable _name_.
  - Use "import module-name" to import the functions in this module
  - It is not required to place all import statements at the beginning of a module
  - Some modules are built-in e.g. sys

Python Basics

- Packages
  - Structure Python's module namespace using dotted module names
  - E.g. A.B.C refers to the submodule C of module B in package A
  - To import module C ->
    - "import A.B.C" and use the fully qualified name OR
    - "from A.B import C" and use only the module name
  - Subpackages need to use fully qualified names to refer to each other

Python Basics

- Classes

  E.g.

  ```
  class ClassName:
          statements
  ```

  OR

  ```
  class ClassName(BaseClass1, BaseClass2…)
          statements
  ```

- Objects

  x = ClassName() creates a new instance of class ClassName and assigns it to the variable x

Python Basics

- An Example

```
class stack:
    "A well known data structure."
    def __init__(self) :            #constructor
        self.items = []
    def push(self, x) :
        self.items.append(x)
    def pop(self) :
        x = self.items[-1]
        del self.items[-1]
        return x
    def empty(self)
        return len(self.items) == 0
```

Functional Example

- Pure function returning index of item in data

```
def binarySearch(data, item):
    min = 0; max = len(data) - 1
    while 1:
        if max < min:
            return “not-found”
        m = (min + max) / 2
        if item > data[m]:
                min = m + 1
        elif item < data[m]:
                max = m - 1
        else:
                return m
```

Comparisons

- Vs perl
  - Easier to learn
  - More readable
  - Fewer side effects
  - Less Unix bias
- Vs Tcl
  - Much faster
  - Less need for C extensions
  - Better java integration

Comparisons

- Vs java
  - More concise code
  - Dynamic typing
  - Runs slower but development is fast
  - No native-code compilation
  - Can be integrated with java using **JPython**

Comparisons

- Vs lisp
  - Indexed arrays instead of linked lists
  - Whitespace instead of parentheses ☺
  - Compiler still (much) slower: bytecode only
  - More: http://www.norvig.com/python-lisp.html
- Vs relfun
  - Mutable objects instead of logical relations
  - More: http://www.cs.unb.ca/~boley/FLP

Comparisons – Example

- python
- **def fac(n):**
- ** if n==0: return 1**
- ** else: return fac(n-1) * n**

- relfun
- **fac(0) :& 1.**
- **fac(N) :& *(fac(-(N,1)), N).**

Areas of Application

- Glue language
- Graphical applications
- Database applications
- Multimedia applications
- Internet protocol applications
- Web (scripting) applications: see above

References

- Python Homepage
  - http://www.python.org/
- Python Tutorial
  - http://www.python.org/tut
- Python documentation
  - http://www.python.org/doc