Python: A great programming toolkit

Asokan Pichai
Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

10, October 2009

Outline

1. Python
   - Dictionary
   - Set
   - Functions Reloaded!
   - Functional programming
   - Debugging

Dictionary

- aka associative arrays, key-value pairs, hashmaps, hashtables ...
- `d = { "Hitchhiker's guide" : 42, "Terminator" : "I'll be back"}`
- lists and tuples index: 0 ...n
- dictionaries index using strings
- aka key-value pairs
- what can be keys?

Dictionary . . .

- Unordered

Standard usage

for key in dict:
<use> dict[key] # => value

- `d.keys()` returns a list
- can we have duplicate keys?

5 m

Problem Set 2.1

2.1.1 You are given date strings of the form “29, Jul 2009”, or “4 January 2008”. In other words a number a string and another number, with a comma sometimes separating the items.Write a function that takes such a string and returns a tuple (yyyy, mm, dd) where all three elements are ints.

2.1.2 Count word frequencies in a file.

2.1.3 Find the most used Python keywords in your Python code (import keyword).

15 m

Outline

1. Python
   - Dictionary
   - Set
   - Functions Reloaded!
   - Functional programming
   - Debugging

Set

- Simplest container, mutable
- No ordering, no duplicates
- usual suspects: union, intersection, subset . . .
- >, >=, <, <=, in, . . .

```
>>> f10 = set([1,2,3,5,8])
>>> p10 = set([2,3,5,7])
>>> f10|p10
set([1, 2, 3, 5, 7, 8])
>>> f10&p10
set([2, 3, 5])
>>> f10-p10
set([8, 1])
```

Set

```
>>> p10-f10, f10^p10
set([7]), set([1, 7, 8])
>>> set([2,3]) < p10
True
>>> set([2,3]) <= p10
True
>>> 2 in p10
True
>>> 4 in p10
False
>>> len(f10)
5
```

Problem set 2.2

2.2.1 Given a dictionary of the names of students and their marks, identify how many duplicate marks are there? and what are these?

2.2.2 Given a string of the form “4-7, 9, 12, 15” find the numbers missing in this list for a given range.

30 m

Outline

1. Python
   - Dictionary
   - Set
   - Functions Reloaded!
   - Functional programming
   - Debugging

Advanced functions

- default args
- var args
- keyword args
- scope
- `global`

Functions: default arguments

```
def ask_ok(prompt, retries=4,
           complaint='Yes or no!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop',
                  'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError, 'bad user'
        print complaint
```

Functions: keyword arguments

```
def parrot(voltage, state='a stiff',
           action='voom', type='Royal Blue'):
    print "-- This parrot wouldn't", action,
    print "if you supply", voltage, "Volts."
    print "-- Lovely plumage, the", type
    print "-- It's", state, "!"

parrot(1000)
parrot(action = 'VOOOOOM', voltage = 1000000)
parrot('a thousand',
       state = 'pushing up the daisies')
parrot('a million', 'bereft of life', 'jump')
```

Functions: arbitrary argument lists

- Arbitrary number of arguments using `*args` or `*whatever`
- Keyword arguments using `**kw`
- Given a tuple/dict how do you call a function?
  - Using argument unpacking
  - For positional arguments: `foo(*[5, 10])`
  - For keyword args:
    `foo(**{'a':5, 'b':10})`

```
def foo(a=10, b=100):
    print a, b
def func(*args, **keyword):
    print args, keyword
# Unpacking:
args = [5, 10]
foo(*args)
kw = {'a':5, 'b':10}
foo(**kw)
```

45 m

Outline

1. Python
   - Dictionary
   - Set
   - Functions Reloaded!
   - Functional programming
   - Debugging

Functional programming

What is the basic idea?

Why is it interesting?

`map, reduce, filter`

list comprehension

generators 60 m

Outline

1. Python
   - Dictionary
   - Set
   - Functions Reloaded!
   - Functional programming
   - Debugging

Errors

```
>>> while True print 'Hello world'
  File "<stdin>", line 1, in ?
    while True print 'Hello world'
                   ^
SyntaxError: invalid syntax
```

Exceptions

```
>>> print spam
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined

>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division
or modulo by zero
```

Debugging effectively

- `print` based strategy
- Process: Hypothesis, test, refine, rinse-repeat
- Using `%debug` and `%pdb` in IPython

75 m

Debugging: example

```
>>> import pdb
>>> import mymodule
>>> pdb.run('mymodule.test()')
> <string>(1)<module>()
(Pdb) continue
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python2.6/pdb.py", line 1207,
    Pdb().run(statement, globals, locals)
  File "/usr/lib/python2.6/bdb.py", line 368, i
    exec cmd in globals, locals
  File "<string>", line 1, in <module>
  File "mymodule.py", line 2, in test
    print spam
NameError: global name 'spam' is not defined
```

Debugging in IPython

```
In [1]: %pdb
Automatic pdb calling has been turned ON
In [2]: import mymodule
In [3]: mymodule.test()
---------------------------------------------------------------------------
NameError                 Traceback (most recent call last)
/media/python/iitb/workshops/day1/<ipython cons
/media/python/iitb/workshops/day1/mymodule.pyc
      1 def test():
----> 2     print spam
NameError: global name 'spam' is not defined
> /media/python/iitb/workshops/day1/mymodule.py
      0     print spam
ipdb>
```

Debugging: Exercise