from python import lecture

A gentle introduction to the python programming language

Muli Ben-Yehuda

mulix@mulix.org

IBM Haifa Research Labs

what is python?

- Python is
  - procedural
  - object-oriented
  - functional
  - interactive
  - dynamic
- Indentation is meaningful.
- (Almost) everything is an object.
- Object attributes can be added/removed at run time.
- There is one way to do it.
- Interpreted, not compiled (to native machine code).

history

- 1990: Python invented by Guido van Rossum.
- 1991: Python announced on usenet, first public release.
- Python was really named after Monty Python.
- Free (as in speech) from the beginning, nowdays released under a GPL compatible license.
- Latest version: Python 2.3b2

variables

```
numbers: 1, 2, 3.0, (3+9j), complex(7,9)

strings: "a", 'a', """aaa"""
a = "a", b = "b", c = a + b + a*3 # a,b,c are strings
d = c[3] # d == 'a'
e = c[0:1] # e == 'a', the slice of c from 0 to 1
# strings are immutable (cannot be changed)
e[0] = 'g' # error
len(e) # 1
# len is a built-in - don't name your variables len
>>> print len
<built-in function len>
>>> len = 1
>>> len(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'int' object is not callable
```

lists

```
lists: [1,3], ['a', 7, 9L, 7j] # L is 'long', j is for complex

# lists are mutable (can be changed)
lst = [1,2]
lst[1] = 3 # lst == [1,3]

#lists can be nested
lst = [1, ['a', 'b'], []]

# more on lists:
append, extend, insert, remove, pop, index, count, sort, reverse

# list comprehensions
[x for x in range(4) if x < 2]

lst = [x*2 for x in range(1, 100) if x%2]
>>> len(lst), lst[-1]
(50, 198)
```

math fun

```
>>> # Fibonacci series:
... # the sum of two elements defines the next
... a, b = 0, 1
>>> while b < 10:
...       print b,
...       (a, b) = (b, a+b)
1 1 2 3 5 8

>>> # number of combinations of k items out of n
>>> # first, we need a factorial function
>>> fac = lambda n:[1,0][n>0] or fac(n-1)*n
>>> def comb(n, k):
...   return (fac(n) / (fac(n-k) * (fac(k))))
>>> # sanity check
>>> comb(4, 2)
6
>>> comb(3, 0)
1
```

control flow

```
a = raw_input("please enter a number: ");
if (a < 3): print "too low, joe"
elif (a >7): print "too big, nick"
else: print "you win, jim"

lst = ['aaaa', 'bv', 'ccc']
for s in lst: print s, len(s)

# range(a,b) generates a list of numbers from a (inclusive)
# to b (exclusive). the first parameter defaults to 0.
range(2) == [0, 1]

# break and continue work like in c
# pass when waiting for an external event or to do nothing
def foo():
    pass
```

functions

```
def hello_func(n = "beautiful"):
    """ this is a doc string """
    print "hello %s world" % (n)

# functions can be renamed and assigned to
f = hello_func;  f = None
# functions can be called with keyword arguments
f(a = 1, b = 2)

def cheesehop(kind, *arguments, **keyword):...
# kind is a regular parameter
# *arguments is a tuple of parameters
# **keyword is a dictionary of keyword:value

>>> def foo(**args):
...  print args
>>> foo(a=1)
{'a': 1}
```

functional programming

```
# lambda forms
add = lambda x: x + x
add(2) # returns 4

def make_pow() : return lambda x: x * x
make_pow()(3) # returns 9

# functional programming: filter, map, reduce
>>> def f(a): return 'b' <= a <= 'c'
>>> filter(f, string.split("a b c d")); # return ['b', 'c']
['b', 'c']
>>> filter(lambda x: x % 3 == 0, range(1, 10))
[3, 6, 9]

>>> car = lambda x: x[0]
>>>cdr = lambda x: x[1:]
>>> car(cdr(string.split("a b c d"))
'b'
```

functional programming cont’

```
def split(s): return s[0:len(s)/2]
map(split, ["aaaaa", "BBBb", "C"]) # returns [’aa’, ’BB’, ’’]

def add(x,y): return x + y
reduce(add, range(1,10)) # returns 45
# again, using a lambda
reduce(lambda x: x + x, range(1,10)) # returns 45

compilers_file = open(’tests/compilers.dat’, "r")
# get a list of compilers (strings), stripped of whitespaces
compilers = [string.strip(z) for z in compilers_file.readlines()]
```

tuples

```
# deleting something from a list, or deleting a variable

del var # 'print var' is now an error!
a = [1,2,3]
del a[-1] # a == [1,2]

# tuples (collection of values) are immutable
(1,2)
(1,)
t = ('a', 1, 0L)
(x,y,z) = t

# len(t) == 3
# t[0] == 'a'
# t * 2 == ('a', 1, 0L, 'a', 1, 0L)
# t = (t, t); t == (('1', 1, 1.0), ('1', 1, 1.0))
```

dictionaries

```
# dictionaries (aka associative arrays)
d = {1: 9000, 'aa': 9001}
d[1] = 7
del d['aa']
d.keys()
d.has_key(7)

# more conditionals
lst = ['a', 'c', 'd']
# ('a' in lst) == 1, ('b' not in lst) == 1

lst2 = lst[0:len(lst)]
# (lst is not lst2) == 1

>>> [[x] for x in lst]
[[1], [2], [4]]
```

modules

- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix .py appended.
- Within a module, the module's name (as a string) is available as the value of the global variable __name__.

```
>>> import string # refer to string.foo as string.foo
>>> from string import foo # foo is string.foo
>>> from string import * # bring everything in from string.*

>>> import sys
>>> sys.path # this is the module search path
['', '/usr/lib/python2.2', '/usr/lib/python2.2/plat-linux2', ...]
```

the beauty of 'dir'

```
>>> dir() # all currenty defined symbol names
['__builtins__', '__doc__', '__name__', 'car', 'cdr', 'f', 'foo', ...]
>>> dir(sys) # symbol names in the 'sys' module
['__displayhook__', '__doc__', '__excepthook__', '__name__', ...]
>>> dir(len) # symbol names in the 'len' function object
['__call__', '__class__', '__cmp__', '__delattr__', '__doc__', ... ]

# when you don't have any documentation, python gives you built in
# documentation, via __doc__ strings!
>>> len.__doc__ # builtin objects have built in doc strings
'len(object) -> integer\n\nReturn the number of items of ...'
>>> sys.__doc__
"This module provides access to some objects used or maintained by ...''

# try this!
>>> dir(dir)
>>> dir.__doc__
```

fancier I/O

```
>>> s = 'Hello, World.'
>>> print str(s), repr(s)
Hello, World. 'Hello, World.'
>>> str(s), repr(s)
('Hello, World.', "'Hello, World.'")
>>> s = 0.1
>>> str(s), repr(s)
('0.1', '0.10000000000000001')

>>> # The argument to repr() may be any Python object:
... repr((x, y, ('spam', 'eggs')))
"(32.5, 40000, ('spam', 'eggs'))"
```

fancier I/O cont'

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print '%-10s ==> %10d' % (name, phone)
...
Jack       ==>       4098
Dcab       ==>       7678
Sjoerd     ==>       4127

>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print 'Jack: %(Jack)d; Sjoerd: %(Sjoerd)d; Dcab: %(Dcab)d' % table
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

file I/O

```
>>> f=open('/tmp/workfile', 'w')
>>> print f
<open file '/tmp/workfile', mode 'w' at 80a0960>

>>> f.read()
'This is the entire file.\n'
>>> f.read()
''


>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'

>>> f.readlines()
['First line\n', 'Second line\n']
# likewise f.write(), f.writelines()
```

exceptions

```
>>> while 1:
...     try:
...         x = int(raw_input("Please enter a number: "))
...         break
...     except ValueError:
...         print "Oops! That was no valid number.  Try again..."
...

>>> try:
...     spam()
... except NameError, x:
...     print 'name', x, 'undefined'
...
name spam undefined

>>> raise NameError, 'HiThere'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: HiThere
```

exceptions cont’

```
>>> raise NameError, ’HiThere’
Traceback (most recent call last): ...
NameError: HiThere

>>> try:
...     raise NameError, ’HiThere’
... except NameError:
...     print ’An exception flew by!’
...     raise
...
An exception flew by!
Traceback (most recent call last): ...
NameError: HiThere

>>> try:
...     raise KeyboardInterrupt
... finally:
...     print ’Goodbye, world!’
```

classes

```
class MyClass:
   "A simple example class"
   i = 12345
   def f(self):
       return 'hello world'

x = MyClass()

>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

classes cont’

```
x.counter = 1
while x.counter < 10:
    x.counter = x.counter * 2
print x.counter
del x.counter

x.f()

xf = x.f
while 1:
    print xf()

class Employee:
   pass

john = Employee() # Create an empty employee record
john.name = ’John Doe’  # Fill the fields of the record
john.dept = ’computer lab’
```