Jython

Introduction to Jython programming

Agenda

- Module 1 - Introduction to Jython
- Module 2 - Jython language and semantics
- Module 3 - Data types
- Module 4 - Regular expressions
- Module 5 - Functions, debugging, modules, and packages
- Module 6 - Objects, classes and exceptions
- Module 7 - Java integration
- Module 8 - Testing
- Module 9 - System programming
- Module 11 - Conclusion

Topics

- Python program structure
- Variables
- Reserved words
- Operators & precedence
- Looping
- Conditional testing
- Dynamic code evaluation
- Printing
- Comments
- Quiz
- Q & A

Python program structure

- Python programs are composed of modules
- Modules contain statements
- Statements contain expressions
- Expressions create and process objects

Lines

- Statement separator
- Continuation lines
  - Opening context (parentheses, square bracket, curly brace) makes continuation unnecessary

Block orientation

- Python is a block-oriented language
- Any of the following keywords starts a block

  `class, def, if, elif, else, while, for, try, except, finally`

  - The colon ("*`:`*") character starts the block
  - Indentation is key

Block structure

- Jython block structure
- Java block structure

```
>>> x = 100
>>> if x > 0:
...     print "this is jython"
... else:
...     print "indentation is key"
...
this is jython
>>>
```

":" denotes block

Indented lines

```
class java_structure {
public static void main(String[] args){
System.out.println("\n*********> Java file contents <*********\u001B[31m");
try{
Process p=Runtime.getRuntime().exec("cat java_structure.java");p.waitFor();
BufferedReader reader=new BufferedReader(new InputStreamReader(p.getInputStream()));
```

Block delimiter

Line delimiter

Variables

Names I

- Allowed characters: a-z, A-Z, 0-9, underscore
  - Must begin with letter or underscore
  - Unlimited length
  - Special name classes Single and double underscores.
  - Single leading single underscore Suggests a "private" method or variable name. Not imported by "from module import *".
  - Single trailing underscore Can be used to avoid conflicts with Python keywords.
  - Double leading underscores Used in a class definition to cause name mangling (weak hiding). But, not often used.

Names2

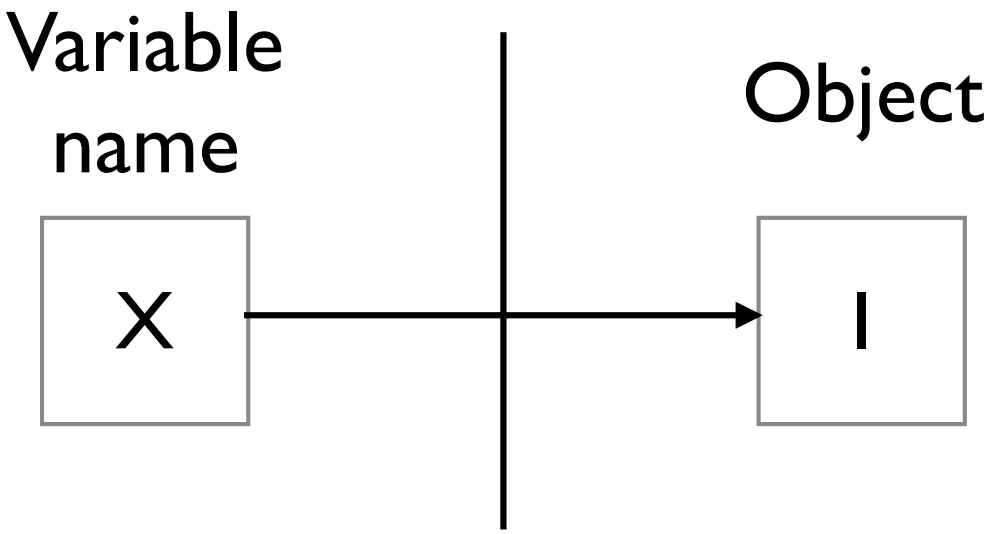
- Naming conventions Not rigid, but:
  - Modules and packages: all lower case.
  - Globals and constants : Upper case.
  - Classes: Bumpy caps with initial upper.
  - Methods and functions: All lower case with words separated by underscores.
  - Local variables: Lower case (with underscore between words) or bumpy caps with initial lower or your choice

Good advice - follow the conventions used in the code on which you are working.

Jython

Variables

- Variables created when first assigned a value
- Variables are references to objects
  - Never have type information or constraints assigned that are traits of a particular data type
  - Type lives with the object
- Variables are not type declared (like in Java or C)
  - Object type is determined by the object reference



1. Create object to represent the literal 1
2. Create the variable name "x"
3. Associate the name "x" with the object 1

Local variables

Characteristics

- The locals() function returns a dictionary of the local variables and their values (within the namespace they are defined - e.g. function or class)
  - Local variables are read only

Global variables

Characteristics

- All variables are local to the namespace they are created
  - A namespace is implemented as a dictionary type
- The globals() function returns a dictionary of the global variables and their values
  - Global variables are read/write (but do not change them yourself)

Reserved words

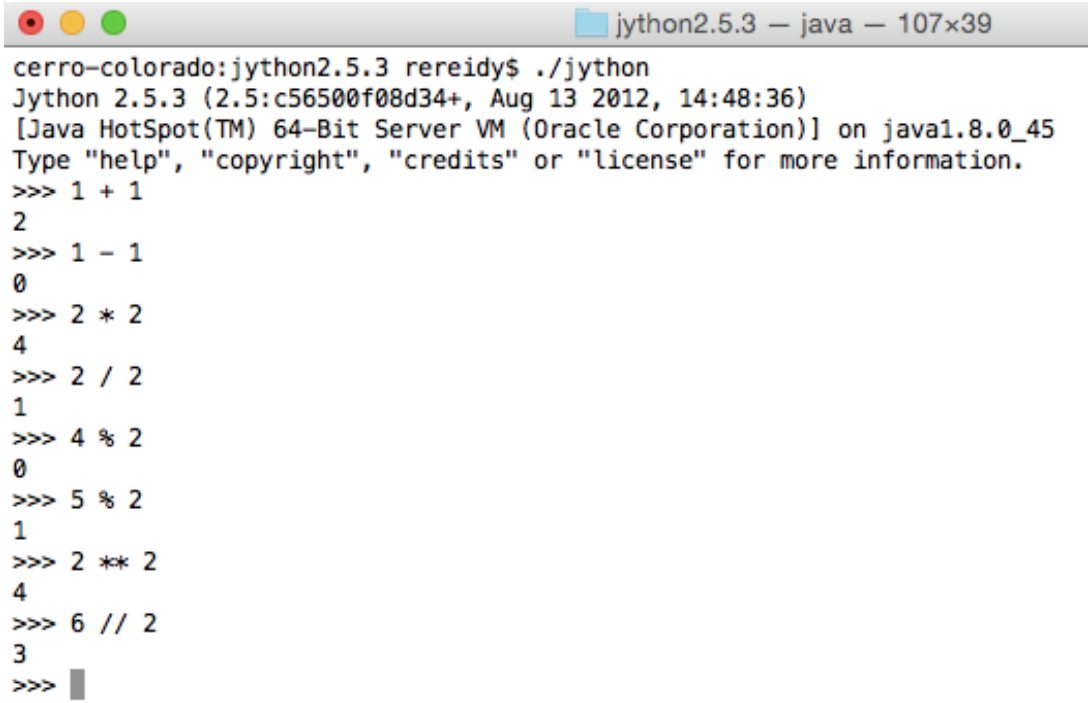
- Do not use as variable names, function names, class names, etc.

```
and, as, assert, break, class, continue,
def, del, elif, else, except, exec, False,
finally, float, for, from, global, if,
import, in, int, is, lambda, local, long,
None, nonlocal, not, or, pass, print, raise,
return, True, try, while, with, yield
```

Operators and precedence

Arithmetic operators

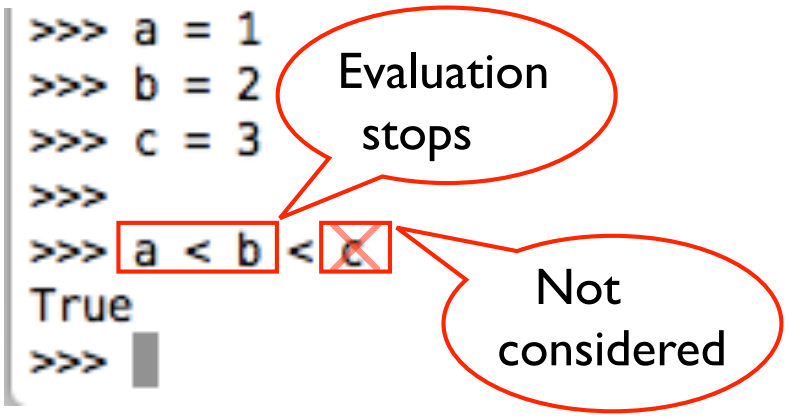
- Default operators
  - `+, -, *, /, %` (modulus)
  - `**` (exponentiation), `//` (floor division)



Comparison operators

- Default operators

  `==, >, <, >-, <=, !=`
- Can be used on many types of data (numbers, collections, etc.)
- Chained comparisons (`a < b < c`)
  - Evaluation guaranteed to stop when the outcome is clear

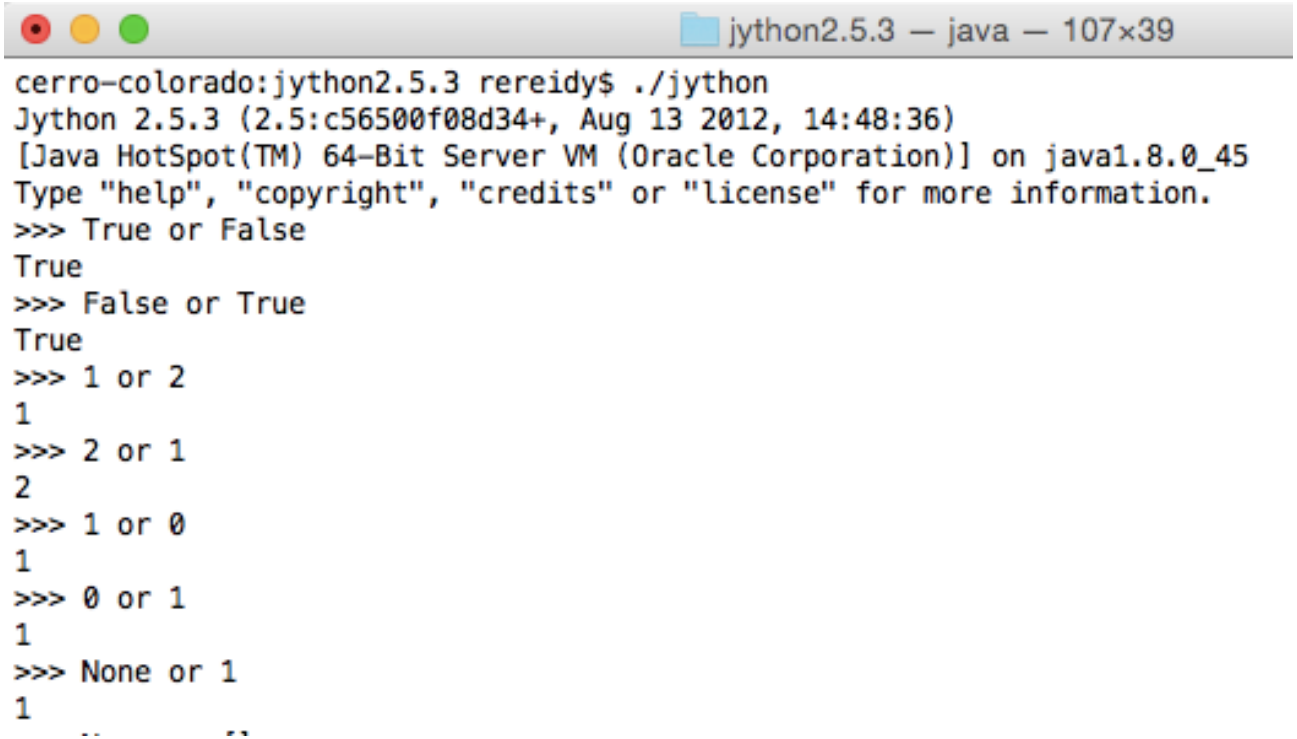


Logical operators

- Default operators
  - `or`
  - `not`
  - `and`

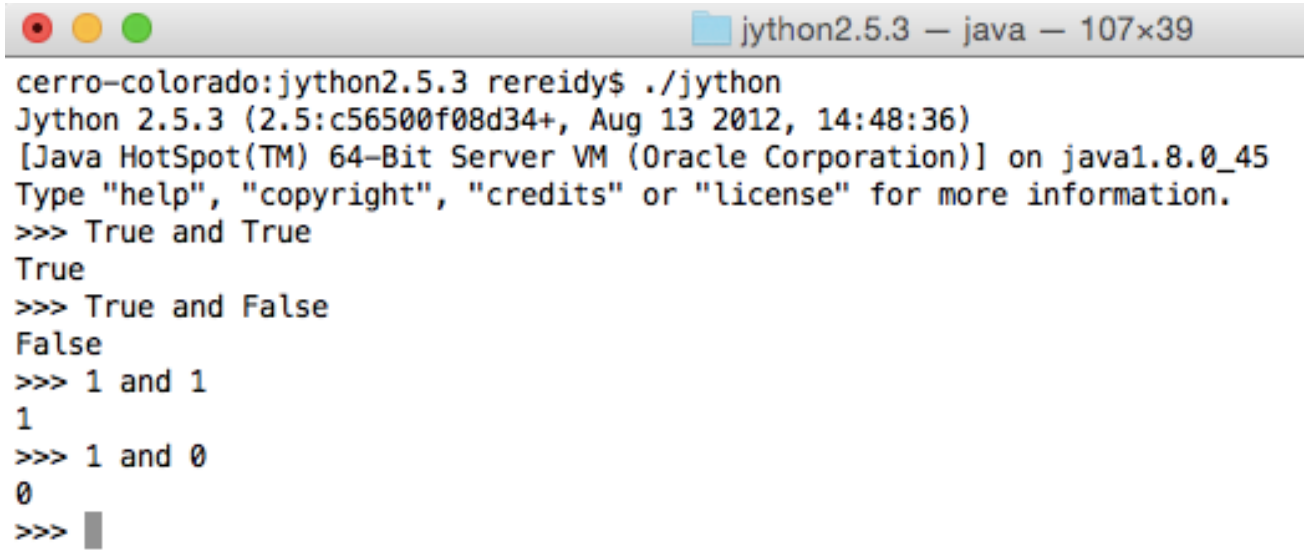
Logical or

- Return first True value found



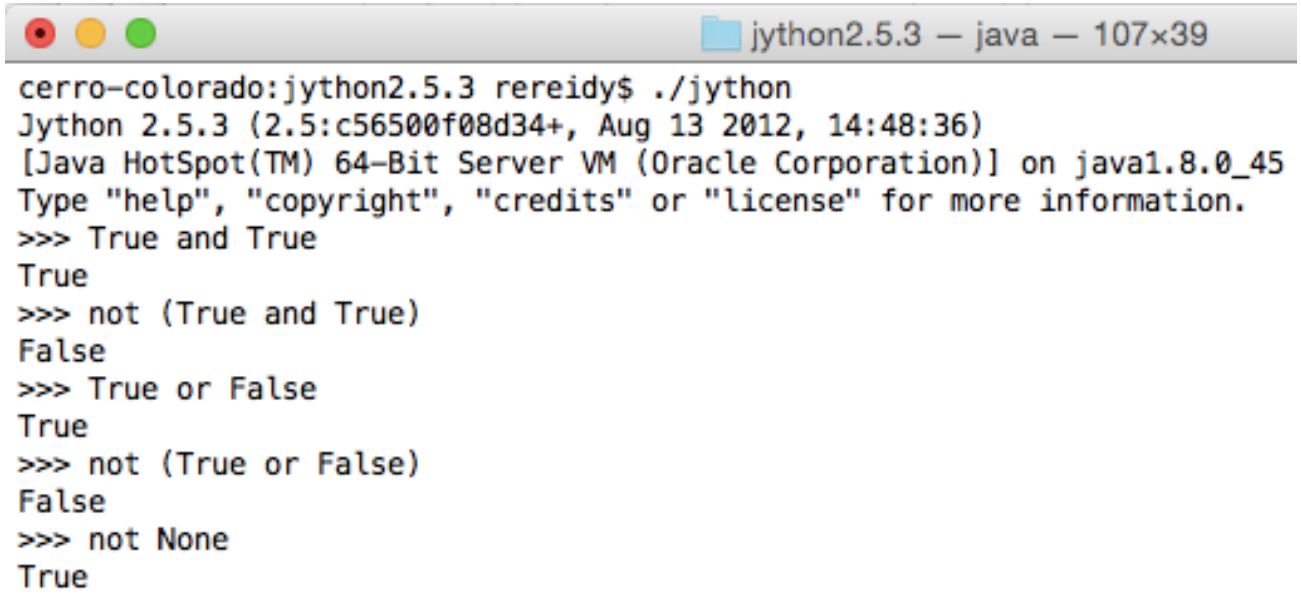
Logical and

- Return True if all values are True; otherwise False



Logical negation

- Negate logical operation



Truth testing

- False
  - Empty/values of zero are normally treated as False

None, False, zero of any numeric type: 0, 0L, 0.0, 0j
Any empty sequence - *"", (), []*
Any empty mapping - *{}*
User defined classes that implement
__nonzero__() or *__len__()* that returns integer 0 or boolean False

- True
  - Non-empty/values != 0 are normally treated as True

Precedence

Operator	Description
lambda	Lambda expression
or	Boolean or
and	Boolean and
not	Boolean not
in, not in	Membership tests
is, is not	Identity testing
<, <=, >, >=, !=, ==	Camparison
|	Bitwise or
^	Bitwise XOR
&	Bitwise &
<<, >>	Shifting
+, -	Addition/subtraction
***, /, %**	Multiplication, division, modulus
+x, -x	Positive/negative
~	Bitwise not
******	Exponentiation
.	Attribute reference
[]	Subscripting
[:]	Slicing
f()	Function call
(expr,…)	Binding or tuple display
[expr,..]	List display
{key:data,…}	Dictionary display
`expr,…`	String conversion

Looping

while statement

- Executes a block of code until a condition is met

for statement

- Iterates over items in a sequence in the order they appear in the sequence for n times

map statement

- Apply a function to every element of an sequence

Loop subversion

break statement

- Exit a for or while loop

continue statement

- Move control to the beginning of a fr or while loop

Conditional testing

if statement

- Simple testing

```
>>> x = 1
>>> if x == 1:
...     print "x == 1"
...
x == 1
>>> s = "Now is the time for all good men to come to the aid of their country"
>>> if s.startswith("Now"):
...     print s
...
Now is the time for all good men to come to the aid of their country
>>>
```

Conditional expressions

- More complex

  a if x else b

```
cerro-colorado:jython2.5.3 rereidy$ java -jar jython.jar
Jython 2.5.3 (2.5:c56500f08d34+, Aug 13 2012, 14:48:36)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_45
Type "help", "copyright", "credits" or "license" for more information.
>>> "yes" if 1 == 1 else "no"
'yes'
>>> "yes" if 1 == 2 else "no"
'no'
>>>
```

switch statement

- Python does not have a switch statement like C

```
#include <stdio.h>

int main(void)
{
    char c = 'a';
    switch (c) {
        case 'z':
            printf("z\n");
        case 'y':
            printf("y\n");
        default:
            printf("not tested\n");
    }
}
cerro-colorado:demos rereidy$ ./switch
not tested
cerro-colorado:demos rereidy$
```

- Use if … elif … else

```
>>> x = 2
>>> if x == 0:
...     print "x = 0"
... elif x < 0:
...     print "x < 0"
... elif x == 1:
...     print "x = 1"
... else:
...     print "x is something else"
...
x is something else
>>>
```

Dynamic code evaluation

- Allows interaction with the interpreter
- Three methods
- Be wary of code injection vulnerabilities

eval()

- Built-in function
- Evaluates an expression (string) and returns the result
- Good for "what if" testing

exec()

- Statement in Jython 2; function in Jython 3
- Compiles and execute statement(s) in a string
- Syntax
  - exec(statements)
    - Statements is one or more Python statements
    - Return is ignored

Jython

compile()

- Low level version of exec() and eval()
- Compiles source code to byte code
- Compiles abstract syntax trees
  - Can modify source code on the fly
- Does not execute or evaluate statements or expressions
  - Returns a code object that can be executed

Caution

- Beware
  - Code injection vulnerabilities
  - Memory and CPU exhaustion

Printing

Basic printing

- The print statement
  - In Python 2.5, print is a statement ("`print …`")
  - In Python 3, print is a function ("`print()`")
  - In Python 2.6 and above
    - Print can be a function

```
from __future__ import print_function
```

Comments

One line comments

- Anything after the "#" is considered a comment

```
#print "hello world"
print "hello world" # print greeting
```

Multi-line comments

- Just like a "here" document in UNIX

```
"""
This is a
multi-line
comment
"""
```

Coding standards

- Follow PEP 8 - code readability
  - Indentation (spaces **NOT** tabs)
  - Maximum line length
  - Blank lines
  - Source file encoding
  - Imports
- Much more

https://www.python.org/dev/peps/pep-0008/

Quiz

1. How are variables stored internally in Python?

A. Variables are stored within a dictionary.

2. Give two examples of values that will result in a test returning False.

A. None
empty collection (`[], {}, ""`)
numeric zero (`0, 0L, 0.0, 0j`)

3. Are the following assignments valid (why or why not)?

```
1. float = 1.0
2. int x = 7
3. f = 1.0
```

A. Answers

1. `float = 1.0` # No - "float" is a reserved word
2. `int x = 7` # No - Python variables are not type declared
3. `f = 1.0` # Yes

Q & A

Exercises

1. Write a program to loop through a list of integers (e.g. "*ints = range(10)*")
   a. Print the sum of all numbers that are in the even number members of the list (use the "`%`" operator).
2. Write a program to count down from 10 to 0.
   a. Print each number ("`print num`").
   b. Bonus - sleep before printing each number

      import time

      time.sleep(1)

      Print ellipsis (…) of digits with the number