



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 10 - Conclusion



Topics



- Functions
- Scope and namespaces
- Debugging
- Modules
- Packages
- Quiz
- Q & A



Functions

Overview

- Performs a specific task
 - Reducing duplication of code
 - Decomposing complex problems into simpler pieces
 - Improving clarity of the code
 - Reuse of code
 - Information hiding
- First class citizen
 - Equal status with other objects (classes, types, etc.)
 - Can be assigned to variables, passed as argument to classes, functions, etc.

Main concepts I

- The “def” keyword declares a function
 - Stand alone function
 - Classes
- “*def*” is an executable statement
 - Can be nested
 - if statements
 - while statements
 - Other def statements
- Creates an object and assigns a name (reference to the object)

```

cerro-bravo:python2.5.3 rereidy$ ./jython
jython2.5.3 - java - 144x70
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_31
Type "help", "copyright", "credits" or "license" for more information.
>>> def x():
...     print "in x"
...
>>> type(x)
<type 'function'>
>>> x()
in x
>>> 
```

Object and name created; object reference assigned to the name

Function execution

```

>>> y = x
>>> type(y)
<type 'function'>
>>> y
<function x at 0x2>
>>> y()
in x
>>> 
```

Assign to variable name

Execute function via reference to assigned variable

Main concepts₂

- “*lambda*” creates an anonymous, in-line function
- “*return*” sends result object back to caller
- “*yield*” sends result back to caller, but remembers where it left off in processing
- “*global*” declares module-level variables that are assigned later in processing



Built-in functions

Built-in functions

- Always available
- Part of the interpreter (module `__builtins__`)

Built-in Functions				
<code>abs()</code>	<code>divmod()</code>	<code>input()</code>	<code>open()</code>	<code>staticmethod()</code>
<code>all()</code>	<code>enumerate()</code>	<code>int()</code>	<code>ord()</code>	<code>str()</code>
<code>any()</code>	<code>eval()</code>	<code>isinstance()</code>	<code>pow()</code>	<code>sum()</code>
<code>basestring()</code>	<code>execfile()</code>	<code>issubclass()</code>	<code>print()</code>	<code>super()</code>
<code>bin()</code>	<code>file()</code>	<code>iter()</code>	<code>property()</code>	<code>tuple()</code>
<code>bool()</code>	<code>filter()</code>	<code>len()</code>	<code>range()</code>	<code>type()</code>
<code>bytearray()</code>	<code>float()</code>	<code>list()</code>	<code>raw_input()</code>	<code>unichr()</code>
<code>callable()</code>	<code>format()</code>	<code>locals()</code>	<code>reduce()</code>	<code>unicode()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>long()</code>	<code>reload()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>map()</code>	<code>repr()</code>	<code>xrange()</code>
<code>cmp()</code>	<code>globals()</code>	<code>max()</code>	<code>reversed()</code>	<code>zip()</code>
<code>compile()</code>	<code>hasattr()</code>	<code>memoryview()</code>	<code>round()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hash()</code>	<code>min()</code>	<code>set()</code>	
<code>delattr()</code>	<code>help()</code>	<code>next()</code>	<code>setattr()</code>	
<code>dict()</code>	<code>hex()</code>	<code>object()</code>	<code>slice()</code>	
<code>dir()</code>	<code>id()</code>	<code>oct()</code>	<code>sorted()</code>	

<https://docs.python.org/2/library/functions.html>



User-defined functions

Creating functions

- Format

```
def <name> (<arg1>, <arg2>, ...<argn>):  
    <statements>
```

<return>

- “*def*” executes at runtime
 - Function object and name created and assigned when file is read

Polymorphism

- Meaning of expression depends on object types

Function arguments

- Positional
- Named
- Default arguments
- Arbitrary arguments
- Dictionary arguments



Argument passing mechanism

- Some languages pass by value
 - Make a copy of object and pass copy
- Python uses pass by reference
 - Mutable objects changed in function are permanently changed

return keyword

- All functions return a value
- Explicit using the “*return*” keyword
 - Multiple values separated by comma “,”
return a, b, c, d ...
- Implicit (falling off the end of the function)
 - *None* type

Anonymous functions

- Functional programming style
- Implemented by keyword “*lambda*”



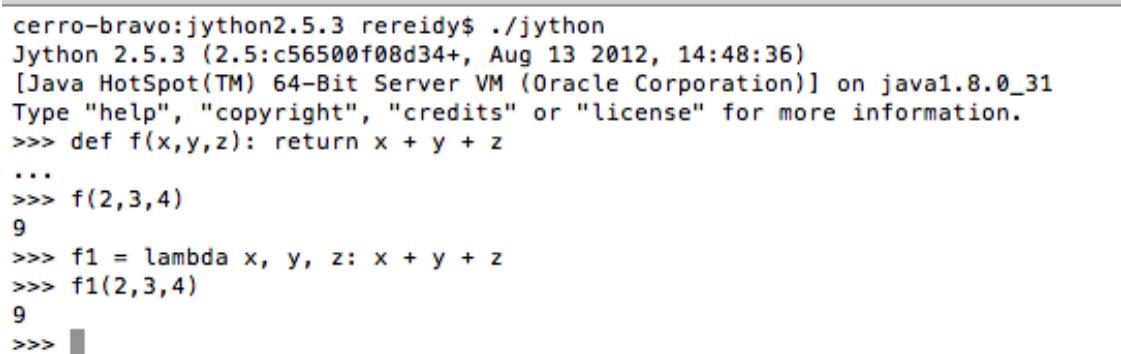
lambda

Why lambdas?

- Function shorthand
- Not mandatory - can use “*def*”ined functions

Anonymous function

- Functions with no name
- Syntax:
lambda argument list: expression
- A *lambda* is an expression, not a statement
 - The body is a single expression, not a block



A screenshot of a terminal window titled "jython2.5.3 — java". The window shows the following Jython session:

```
cerro-bravo:python2.5.3 rereidy$ ./jython
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_31
Type "help", "copyright", "credits" or "license" for more information.
>>> def f(x,y,z): return x + y + z
...
>>> f(2,3,4)
9
>>> f1 = lambda x, y, z: x + y + z
>>> f1(2,3,4)
9
>>>
```



Functional programming

Tools

- Apply functions to sequences and other objects that can be iterated
- `map()`
- `filter()`
- `reduce()`



map



Apply a function to a sequence

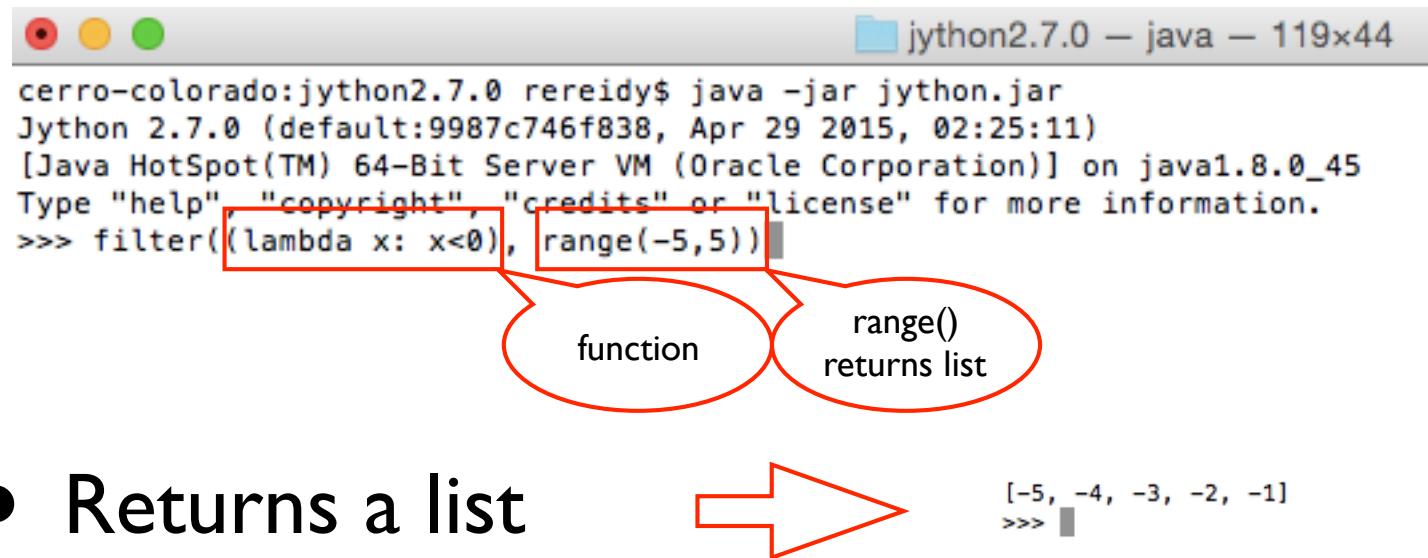
- Apply to all items and collect results
- Easy to do in a *for* loop



filter

Pull items

- Filter items from a list where the called function returns True



A screenshot of a terminal window titled "python2.7.0 – java – 119x44". The window shows the following Jython code:

```
cerro-colorado:jython2.7.0 rereidy$ java -jar jython.jar
Jython 2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_45
Type "help", "copyright", "credits" or "license" for more information.
>>> filter((lambda x: x<0), range(-5,5))
```

Two parts of the code are highlighted with red boxes: the lambda expression `(lambda x: x<0)` and the `range(-5,5)` call. Red arrows point from these boxes to two red ovals below them. The left oval contains the text "function" and the right oval contains the text "range() returns list".

- Returns a list

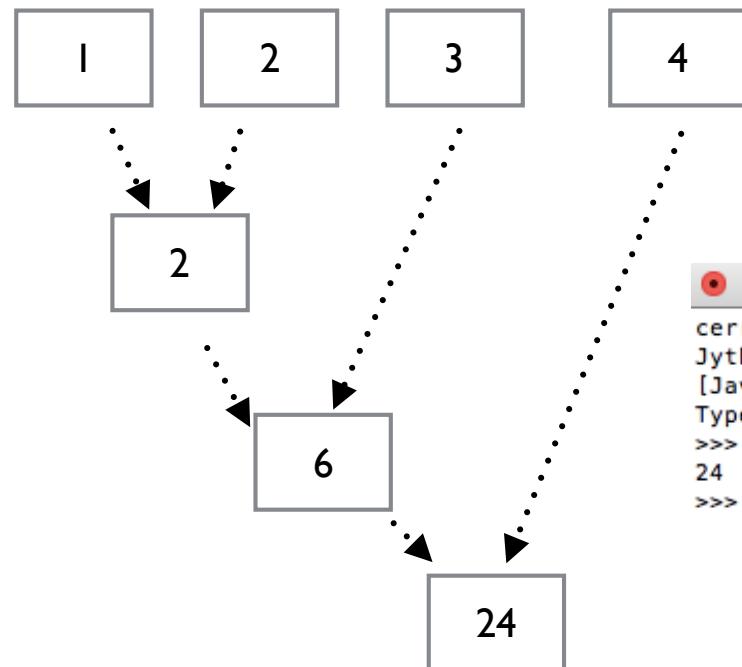
[-5, -4, -3, -2, -1]

reduce

Reduce a sequence

- Continually apply a function to a sequence
 - Reduces the sequence to a single value

*reduce(lambda a, b: a * b, [1,2,3,4])*



```
jython2.5.3 -- java -- 105x43
cerro-colorado:python2.5.3 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.
>>> reduce(lambda a, b: a * b, [1,2,3,4])
24
>>>
```



Scopes

Preliminaries

- When a name is used, Python will
 - Create
 - Change
 - Look up
- This is a namespace - where the names exist
 - Implemented as a dictionary (but that could change)

Scoping rules

Enclosing module

- Exists global scope
 - Variables assigned at the top of a module
 - Attributes of the module
 - Can be used as simple variables within the module file



Scoping rules

Span

- Global scope spans single file only
 - Names at the top of a file are only global to code within that file
- No single, all-encompassing global scope
 - Names are partitioned into modules
 - Modules must be imported explicitly to use the names defined
- In Python: global == module

Scoping rules

Default assignment

- Global scope spans single file only
 - Names at the top of a file are only global to code within that file
- No single, all-encompassing global scope
 - Names are partitioned into modules
 - Modules must be imported explicitly to use the names defined
- In Python: global == module
- All assigned names are local unless declared global
- All other names
 - When not assigned value in function definition
 - Assumed to enclosing scope
 - Local (enclosing def)
 - Global (enclosing module namespace)
 - Built-in (predefined `__builtin__` module)

Name resolution

- Name references search within four scopes
 - Local
 - Enclosing functions
 - Global
 - Built-in
- Name assignments create or change local names by default
- global declarations map assigned names to the enclosing module and function scopes

Built-in

Names preassigned in `__builtin__` module:

Functions (open, range, abs, etc.), exceptions

Global (module/file)

Names assigned at top-level of file or declared “global” in a `def`

Enclosing function locals

Names in local scope of any and all enclosing functions
(`def/lambda`) from inner most out

Local (function)

Names assigned in any way within a function and not
declared global in the function



Minimize global variables

- Because names assigned in functions are local
- Write extra code to change in functions (global)
 - This is a language feature - write more code to do potential harm
 - Default of local is a best software engineering practice
- Value of globals dependent on order of calls to functions

What is the value?

- What is Y

- Timing dependent
 - Which function is called last

```
Y = 100

def f1():
    global Y
    Y = -1

def f2():
    global Y
    Y = 101
```



Debugging

Debugging strategies

- Using “*print*” statements
- Using a debugger



The Python debugger

- Module that allows interactive source code debugging
 - Conditional breakpoints
 - Single stepping
 - Inspect stack frame
 - Evaluation of arbitrary Python code
 - Source code listing
- Invoke from the command line
`./jython -m pdb file.py`

<https://docs.python.org/2/library/pdb.html>



Modules

Modules

- All Python files are modules
- Implement a set of functions, variables, classes
- Modules process with
 - *import* - fetch a whole module
 - *from* - fetch particular names from a module

Using modules

- Use the “*import*” statement

```
cerro-colorado:python2.5.3 rereidy$ ./jython ~/Documents/workspace/Jython/class/demos/Mod4/args.py 1 "now is the time"  
import sys  
  
print "arguments:", sys.argv  
  
arguments: ['/Users/rereidy/Documents/workspace/Jython/class/demos/Mod4/args.py', '1', 'now is the time']  
cerro-colorado:python2.5.3 rereidy$
```

- Can also use “*from mod import ...*”

```
cerro-colorado:python2.5.3 rereidy$ ./jython ~/Documents/workspace/Jython/class/demos/Mod4/args1.py 1 "now is the time"  
from sys import argv  
  
print "arguments:", argv  
  
arguments: ['/Users/rereidy/Documents/workspace/Jython/class/demos/Mod4/args1.py', '1', 'now is the time']  
cerro-colorado:python2.5.3 rereidy$
```

Aliasing imports

- The imported module can be aliased

```
cerro-colorado:jython2.5.3 rereidy$ ./jython
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.
>>> import sys as s
>>> len(s.argv)
1
>>> len(sys.argv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sys' is not defined
>>> 
```

```
cerro-colorado:jython2.5.3 rereidy$ ./jython
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.
>>> from sys import argv as args
>>> len(args)
1
>>> len(sys.argv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sys' is not defined
>>> 
```

Using modules

- When using the “*from mod import ...*” form
 - Do not use “*” (e.g. *from sys import **)

Import mechanics

- Import is not a textual insertion or substitution (like C)
- Runtime operations
 - Find the module file
 - Compile to bytecode (if needed)
 - Run the module code to build objects

Find the module

- The import statement looks for the file with a .py extension implied (do not include it in the import statement)
- Does not include the path to the file (more later)
 - Uses the standard module search path mechanisms

Compile to bytecode

- When modules are first imported
 - Creates a `.pyc` file
 - Jython - `module$py.class` file
- Not when a module file is executed as a program
- Can be done manually



Module import

Compiled module

Run as a program

Jython source

Compile via
py_compile

Compiled
module

Compile via
compilall

Compiled
module

```
cerro-colorado:jython2.7.0 rereidy$ java -jar jython.jar
Jython 2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_45
Type "help", "copyright", "credits" or "license" for more information.
>>> import xyz
>>> cerro-colorado:jython2.7.0 rereidy$ ls -l xyz*
-rw-r--r-- 1 rereidy staff 2606 May 21 17:32 xyz$py.class
-rw-r--r-- 1 rereidy staff 38 May 21 17:28 xyz.py
cerro-colorado:jython2.7.0 rereidy$
```

```
cerro-colorado:jython2.7.0 rereidy$ java -jar jython.jar xyz.py
cerro-colorado:jython2.7.0 rereidy$ ls -l xyz*
-rw-r--r-- 1 rereidy staff 38 May 21 17:28 xyz.py
cerro-colorado:jython2.7.0 rereidy$
```

```
cerro-colorado:jython2.7.0 rereidy$ java -jar jython.jar
imJython 2.7.0 (default:9987c746f838, Apr 29 2015, 02:25:11)
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporation)] on java1.8.0_45
Type "help", "copyright", "credits" or "license" for more information.
>>> import py_compile
>>> py_compile.compile('xyz.py')
>>> cerro-colorado:jython2.7.0 rereidy$ ls -l xyz*
-rw-r--r-- 1 rereidy staff 2640 May 21 17:43 xyz$py.class
-rw-r--r-- 1 rereidy staff 38 May 21 17:28 xyz.py
cerro-colorado:jython2.7.0 rereidy$
```

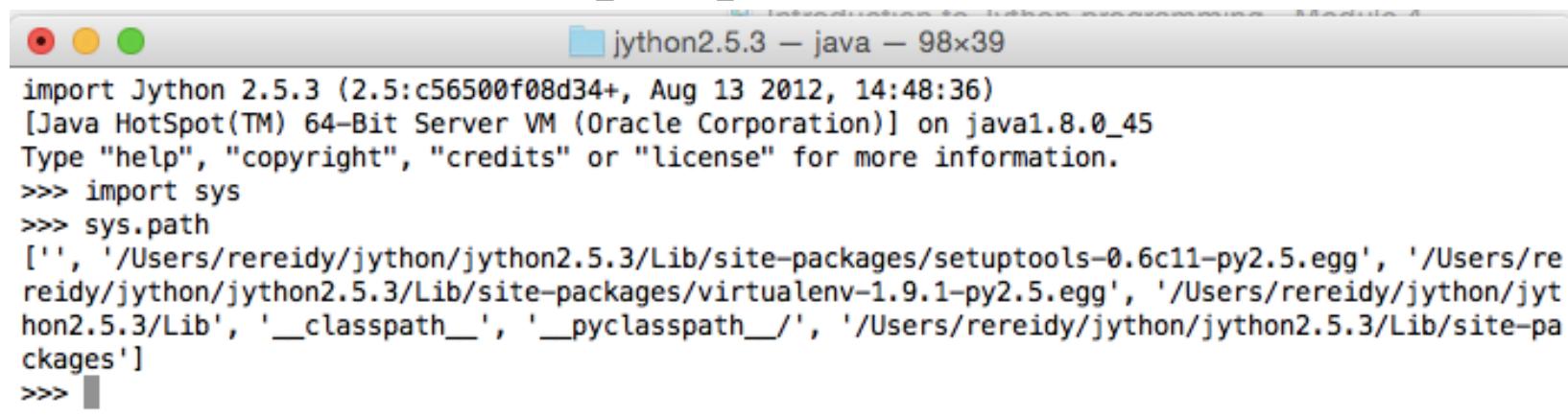
```
cerro-colorado:jython2.7.0 rereidy$ java -jar jython.jar -m compileall xyz.py
Compiling xyz.py ...
cerro-colorado:jython2.7.0 rereidy$ ls -l xyz*
-rw-r--r-- 1 rereidy staff 2640 May 21 17:48 xyz$py.class
-rw-r--r-- 1 rereidy staff 38 May 21 17:28 xyz.py
cerro-colorado:jython2.7.0 rereidy$
```

Run the module

- All statements are executed from the top of the file to the bottom
- Assignments made to names generate attributes of objects

Finding modules

- In the current directory
- In the directory where your script exists
- In \$PYTHONHOME (where Python is installed)
- In \$PYTHONPATH (list of directories separated by ":")
 - Present in *sys.path*



The screenshot shows a terminal window with the title bar 'jython2.5.3 – java – 98x39'. The window contains the following text:

```
import 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.
>>> import sys
>>> sys.path
['', '/Users/rereidy/jython/jython2.5.3/Lib/site-packages/setuptools-0.6c11-py2.5.egg', '/Users/re
reidy/jython/jython2.5.3/Lib/site-packages/virtualenv-1.9.1-py2.5.egg', '/Users/rereidy/jython/jyt
hon2.5.3/Lib', '__classpath__', '__pyclasspath__', '/Users/rereidy/jython/jython2.5.3/Lib/site-pa
ckages']
>>>
```

Module file types

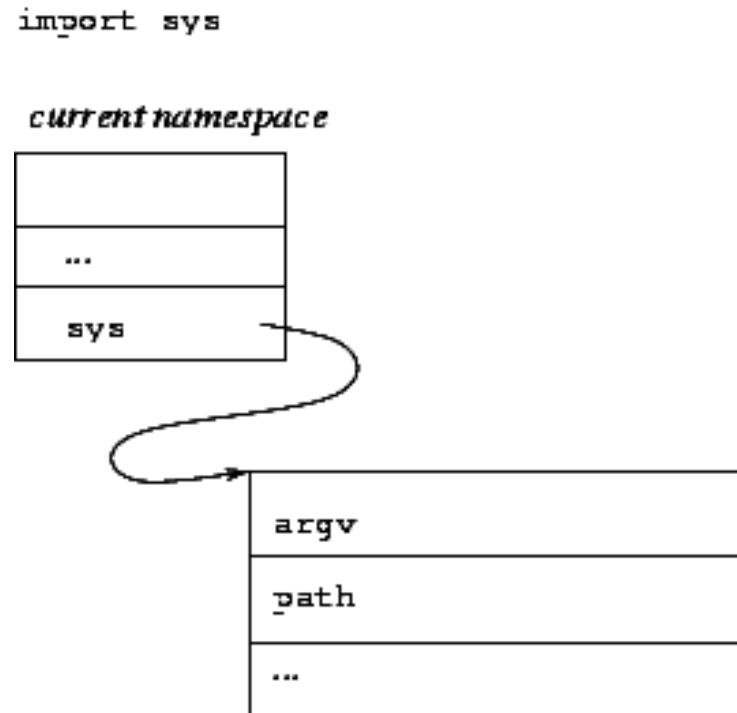
- Python
 - Python files (suffix of .py or compiled Python .pyc)
 - Defined C extension files
 - Modules built in to the interpreter
- Jython
 - Python files (suffix of .py or compiled Python .pyc)
 - Jython Java class files are *file\$py.class*
 - Java .class files
 - Modules built in to the interpreter

Built in and class files

```
cerro-colorado:python2.5.3 rereidy$ ./jython
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.
>>> import sys, os
>>>
>>> type(sys)
<type '<reflected field public org.python.core.PyObject org.python.core.PySystemState.__name__ at 0x1>'>
>>>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: '<reflected field public org.python.core.PyObject o' object has no attribute '__file__'
>>>
>>>
>>> type(os)
<type 'module'>
>>>
>>> os.__file__
'/Users/rereidy/jython/jython2.5.3/Lib/os$py.class'
>>>
```

How it works

- When importing
 - Interpreter creates a new namespace
 - Code is executed in this namespace



Adding modules

- In Jython, modules can only be
 - Pure Python (no C extensions)
 - Install from PyPi (Python Package Index)
 - Use setuptools (pip)
 - Java class or JAR files
 - Add location to CLASSPATH

Standard library

- Built-in modules (written in C) or Python
 - Standard solutions for everyday programming problems
- Jython implements the standard library in .class and .py files

<https://docs.python.org/2/library/index.html>



Modules of interest

sys module

- Parameters and variables that have strong ties to the interpreter
- Python command line arguments (`sys.argv`)
 - Not interpreter command line arguments

OS

- Miscellaneous operating system functions
 - For path interfaces, import os.path
- In Jython, the os module is different than in CPython

__future__ module

- Future statement definitions

```
all_feature_names = [
    "nested_scopes",
    "generators",
    "division",
    "absolute_import",
    "with_statement",
]
__all__ = ["all_feature_names"] + all_feature_names
```

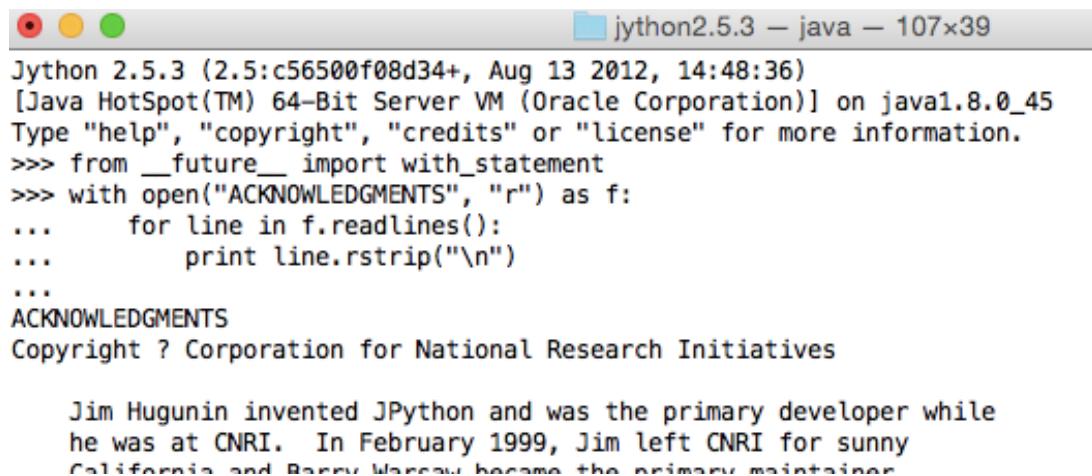
from __future__ import with_statement

- Imports *from __future__* must always be first lines in Python file

https://docs.python.org/2/library/__future__.html

Reading a file

- When you use the “*with*” statement
 - No need to close file at end of I/O operations



```
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.
>>> from __future__ import with_statement
>>> with open("ACKNOWLEDGMENTS", "r") as f:
...     for line in f.readlines():
...         print line.rstrip("\n")
...
ACKNOWLEDGMENTS
Copyright ? Corporation for National Research Initiatives

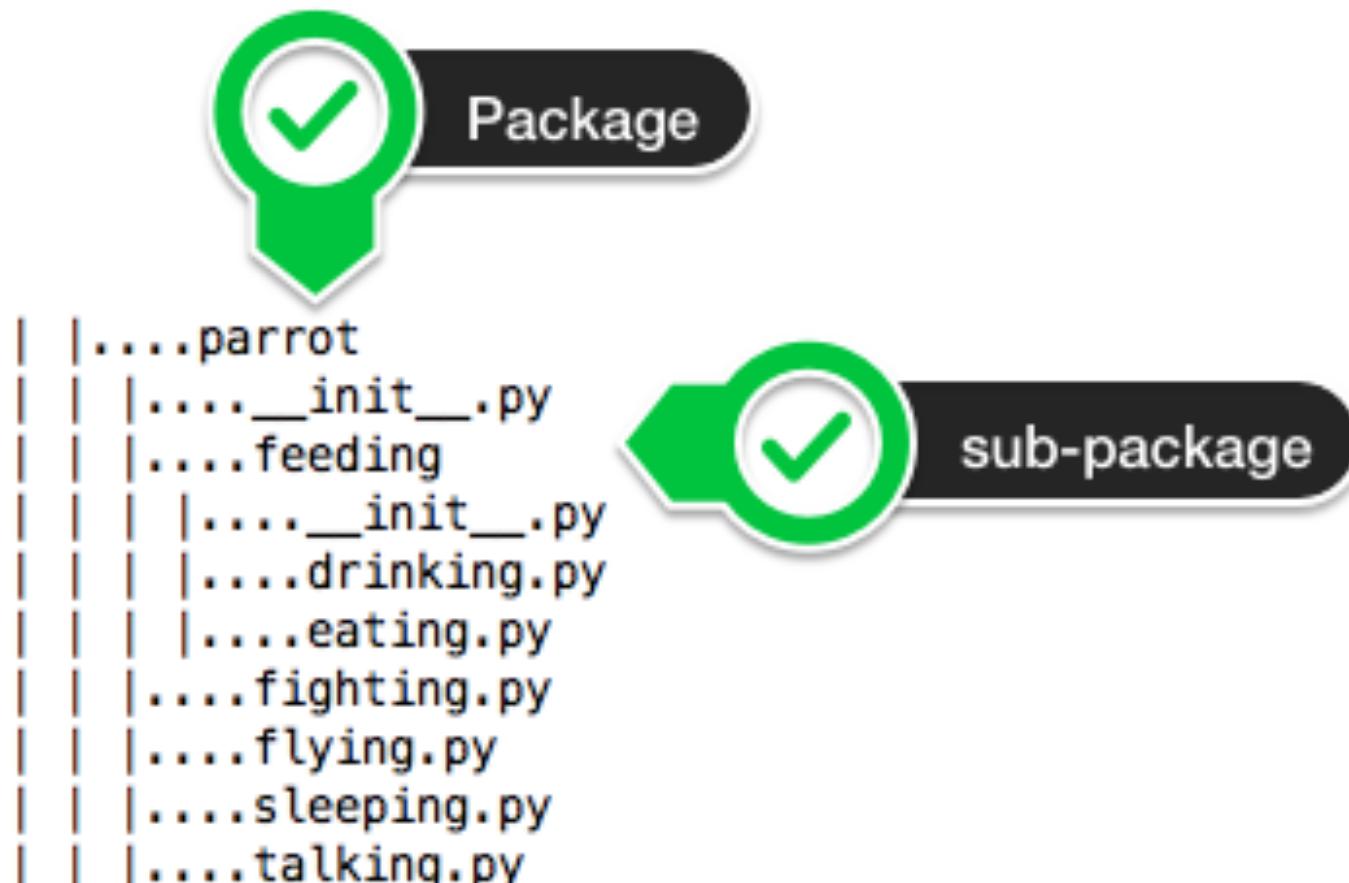
Jim Hugunin invented JPython and was the primary developer while
he was at CNRI. In February 1999, Jim left CNRI for sunny
California and Barry Warsaw became the primary maintainer
```

Packages

Packages

- A package is a directory with potentially several Python files
- Also contains a `__init__.py` file
 - Used to define classes, functions, variables that are directly visible
 - **Not** sub-packages or sub-modules

Package example



Package use

- Many ways to use the package

```
import parrot                                # loads parrot/__init__.py
import parrot.talking                         # loads parrot/talking.py
from parrot import sleeping                   # load parrot/sleeping.py
import parrot.feeding                         # load parrot/feeding.py
from parrot.feeding.eating import eat_cracker # object within eating.py
```

- Common (but not mandatory) is for
parrot/__init__.py to import all modules

```
import parrot                                # loads parrot/__init__.py
parrot.feeding.eating.eat_cracker()
```

- No need to import sub package(s)

`__init__.py`

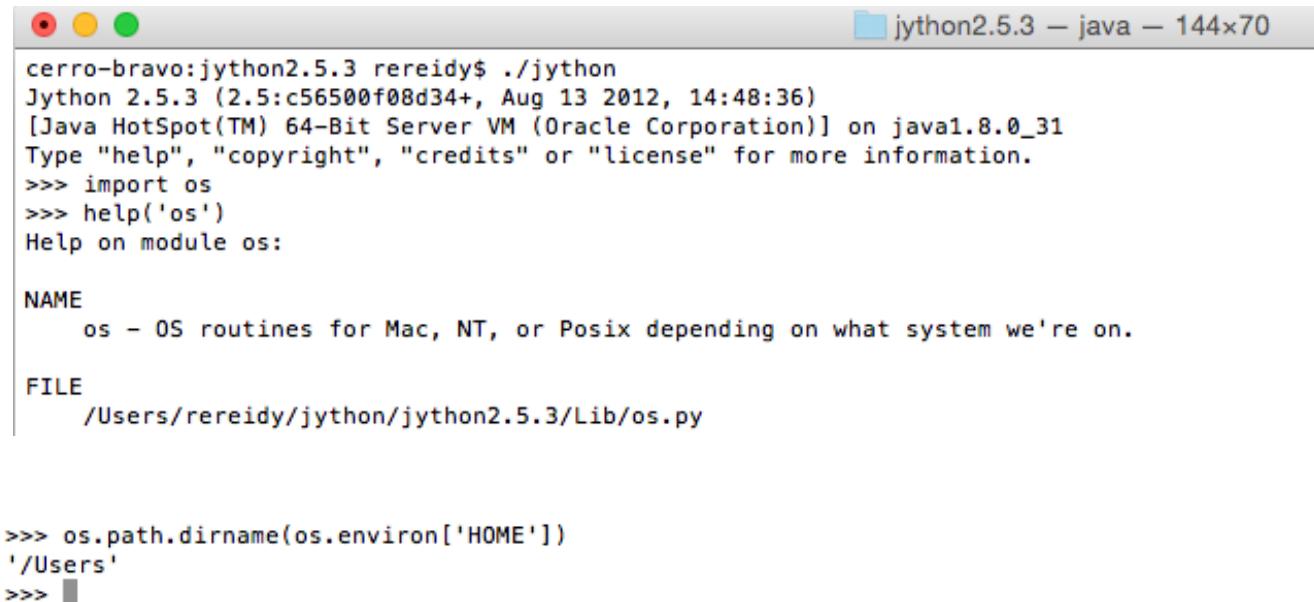
- To import objects in the package
 - Set list object (`__all__`) to the values

```
__all__ = ['feeding', 'fighting', 'flying', 'sleeping', 'talking']
```

- List of public objects in the package

os module

- Behaves similarly
 - import os
 - Can access objects in os.path package



The screenshot shows a terminal window titled "jython2.5.3 – java – 144x70". It displays the help documentation for the "os" module in Jython. The output includes the module's NAME, FILE, and a command example:

```
cerro-bravo:jython2.5.3 rereidy$ ./jython
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_31
Type "help", "copyright", "credits" or "license" for more information.

>>> import os
>>> help('os')
Help on module os:

NAME
    os - OS routines for Mac, NT, or Posix depending on what system we're on.

FILE
    /Users/rereidy/jython/jython2.5.3/Lib/os.py

>>> os.path.dirname(os.environ['HOME'])
'/Users'
>>> 
```



Quiz

- I. What is returned from a function when no “return” statement is issued?
 - A. None type

2. What does the lambda statement do?

A. Creates an anonymous function

3. What does the following code print?

```
>>> s = "Test"
>>> def f():
...     global s
...     s = "no test"
...
>>> f()
>>> print s
```

A. The text “no test” is printed.

4. Why?

A. Because the name “s” was identified as global in the function “f” and then modified.

5. What is the result of the following code?

```
>>> from os import getenv as ge  
>>> print getenv('HOME') █
```

- A. Throws the `NameError` exception because “`getenv`” is not defined

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



Q & A



Exercises



- I. Write a program to read a file using the “with” statement - e.g.

```
from __future__ import with_statement
```



2. Modify the program in #1 to write the contents of the file read into a new file.

3. Create a function that solves the fibonacci sequence.