NOESIS: Byte The Code(Python)

Outline

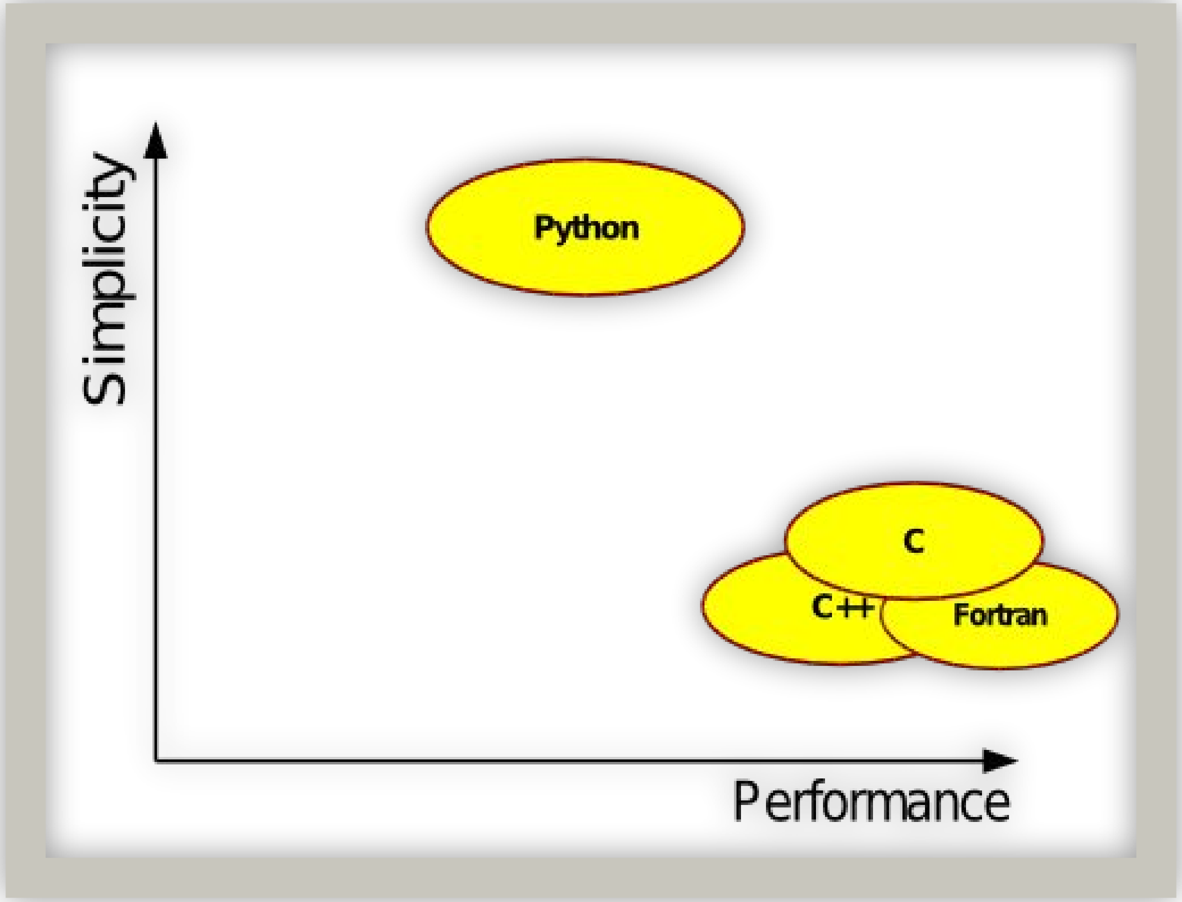
- Overview
- Built-in objects
- Functions and scopes
- Object-oriented programming
- Applications of Python
- Web Scraping

Why use Python?

- Simple syntax: easy to learn *and* remember
- Portable
- Flexible
- Large standard library
- Short development time
- Lots of 3rd party tools/add-ons
- Many good implementations:

  CPython, PyPy, IronPython, Jython
- Active open-source community
- Versions: 2.7.x, 3.4.x

Python vs C/C++



Hello, World!

C

```
#include<stdio.h>

int main(){
      printf(“Hello, World”);
      return 0;
}
```

Python

```
print("Hello, World”)
```

```
print "Hello, World"
```

Compiler and Interpreter

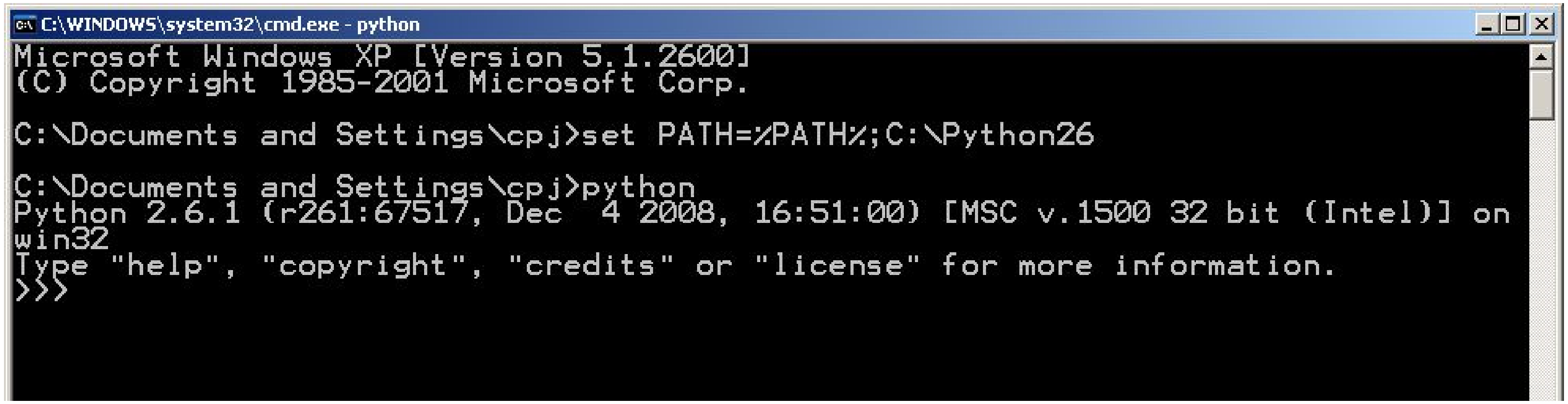
Shell and Editor

Getting Started

- Download from: **`http://python.org/`**

  Add python to PATH to run scripts from command line

- Python is available for most platforms, even mobile.
- Most Linux distributions have Python as package(s)



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\cpj>set PATH=%PATH%;C:\Python26

C:\Documents and Settings\cpj>python
Python 2.6.1 (r261:67517, Dec  4 2008, 16:51:00) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Variables

name *x* means 23

now it means 'foo'

x is undefined

```
>>> x = 23
>>> print(x)
23
>>> x = 'foo'
>>> print(x)
foo
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>>
```

Numeric Types

- Integers
  - Generally signed, 32-bit
- Long Integers
  - Unlimited size
  - Format: <*number*>L
  - Example: `4294967296L`
- Float
  - Platform dependant “double” precision
- Complex
  - Format: <real>+<imag>j
  - Example: `6+3j`

Strings

A sequence of characters enclosed in quotes

3 ways to quote strings:

```
'Single Quotes'
"Double Quotes"
"""Triple Quotes""" or '''triple quotes'''
```

- Triple quotes can span multiple lines

Examples

```
>>> print('This string may contain a "')
This string may contain a "
>>> print("A ' is allowed")
A ' is allowed
>>> print("""Either " or ' are OK""")
Either " or ' are OK
```

Type Conversions

int(), float(), str(), chr(), ord()

- int(2.3)
- int('5')
- int('666')
- float(2)
- float('234')
- float('123.123')
- str(1)
- str(2.8)

- chr(65)
- ord(‘A’)

Type Conversions

Functions to convert between types:

`str()`   `int()`   `float()`   `complex()`   `bool()`

```
>>> str(0.5)
'0.5'
>>> float('-1.32e-3')
-0.00132
>>> int('0243')
243
>>> int(123456000*789101112000)
97420512307200000
>>> bool('hi')
True
>>> bool('False')     # any non-zero, non-null is true
True
```

type determines type of Object

Determine the type of an object

Syntax: `type(object)`

Examples

```
>>> type(2.45)
<type 'float'>
>>> type('x')
<type 'str'>
>>> type(2**34)
<type 'long'>
>>> type(3+2j)
<type 'complex'>
```

Basic I/O

input([prompt])

```
>>> x = input('What is your age?')
What is your age? 20
>>> print(x)
20
>>> x = input()
2.3
>>> x = input()
"Hello"
>>> x,y,z = input()
1,3.9,"hello"
```

Basic I/O

`raw_input( [`*`prompt`*`] )`

- Print *prompt* and return user's input as a string
- a built-in function

Example

```
>>> reply = raw_input('Are you awake? ')
Are you awake? not sure
>>> print( reply )
not sure
```

Basic I/O

raw_input([prompt])

```
>>> x = raw_input('What is your age?')
What is your age? 20
>>> x
'20'
>>> x = int(raw_input())
20
>>> print(x)
20
>>> x = float(raw_input())
2.3
```

Basic I/O

print

```
>>> print “hello”
hello
>>> x = 20
>>> print “age is”,x
age is 20
>>> print “age is”+str(x)
age is 20
>>> y = 3.4
>>> print (x,y)
20 3.4
```

Arithmetic Operations

operators: `+   -   *   /   //   **   %   abs`

Examples:

```
>>> 5 + 3           # Addition
8
>>> 2 ** 8          # Exponentiation
256
>>> 13 / 4          # Integer (Truncating) Division*
3
>>> float(13) / 4 # Float Division
3.25
>>> 13 % 4          # Remainder
1
>>> 13.5 % 4        # Remainder in floating point
1.5
>>> 5.0//2          # floor
2.0
>>> abs(-3.5)       # Absolute Value
3.5
```

Boolean comparisons

Comparison: `<   <=   >   >=   ==   !=   <>`

Example

```
>>> 4 > 1.5
True
>>> 'this' != 'that'
True
>>> 4+3j == 4-2j
False
>>> '5' == 5
False
>>> 0 < 10 < 20
True
```

Boolean Operations

Operators: **`and`** **`or`** **`not`**

Standard Boolean Algebra

i_1	i_2	i_1 **and** i_2	i_1 **or** i_2
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

i_1	**not** i_1
1	0
0	1

Boolean values

`True`: any non-zero, non-null value.

`False`: `None` (null)

empty string

0

Boolean Expressions

```
>>> 1 == 1 and 2 >= 3
False
>>> 1 == 1 or 2 >= 3
True
>>> not 5.3 != 2.2    # same as: not (5.3 != 2.2)
False
>>> 2 and '23' > '11' or 0
True
```

String Formatting

C-Style formatting (extended `printf`):

```
"format string" % (arg1, arg2, ...)
```

```
>>> "%i %s in the basket" % (2, "eggs")
'2 eggs in the basket'
>>> x = 2.0/9.0
>>> "%f to 2 dec places is %.2f" % (x, x)
'0.222222 to 2 decimal places is 0.22'


>>> length = 5
>>> obj = "fence"
>>> "Length of %(obj)s is %(length)i" % vars()
'Length of the fence is 5'
```

String Format Codes

Format codes begin with "%":

```
x = 10
y = 2.3
"x is %f" % x
"pi is %.8f" % y
"pi is %12.6f" % y
eps = 1.0E-17
```

Building strings

Concatenation (+): `string1 + string2`

Example:

```
>>> 'MANIT' + 'Noesis'
'MANITNoesis'
```

Repetition (*): `string * number`

Example:

```
>>> 'dog' * 5
'dogdogdogdogdog'
```

Flow Control

```
if condition :
        body
elif condition :
        body
else:
        body
```

```
if x%2 == 0:
     y = y + x
else:
     y = y - x
```

```
while condition:
        body
else:
        body
```

```
while count < 10:
  count = 2*count
```

```
for iterator in sequence:
        body
else:
```

```
for x in [1,2,3]:
     sum = sum + x
```

Indentation

Python vs C

C

Python

```
if(i>5){
printf(“i is greater than 5”);
}
else{
printf(“i is less than 5”);
}
```

`range`: create a sequence

`range([start,] stop[, step])`

Generate a list of numbers from `start` to `stop` stepping every `step`

`start` defaults to 0, `step` defaults to 1

Example

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(1, 9)
[1, 2, 3, 4, 5, 6, 7, 8]
>>> range(2, 20, 5)
[2, 7, 12, 17]
```

for loop using range( )

Use `range` to generate values to use in for loop

```
>>> for i in range(1,4):
        print i
1
2
3
```

loop iteration using **`continue`**

skip to next iteration of a loop

```
for x in range(10):
    if x%2 == 0:
        continue
    print x
```

```
1
3
5
7
```

break

break out of the inner-most loop

```
for number in range(10):
    if number == 4:
      print 'Breaking'
      break
    else:
        print number
```

```
0
1
2
3
Breaking
```

Getting Help

The `help` function gives help for a module or function.

```
>>> help(str)
Help on class str in module __builtin__:

class str(basestring)
 |  str(object) -> string
 |
 |  Return a nice string representation of the object.
 |
 |  Method resolution order:
 |      str
 |      basestring
 |      object
 |
 |  Methods defined here:
 |  ...
```

Functions

Defining Functions

Syntax: **`def func(arg1, …):`**

`body`

- Body of function must be indented
- If no value is returned explicitly, function will return `None`

```
def average(num1, num2, num3):
    sum = num1 + num2 + num3
    avg = sum / 3.0
    return avg
```

Function Parameters

- Parameters can be any type
- A function can take any number of parameters or none at all

```
def usage(programName, version):
     print("%s Version %i" % (programName, version))
     print("Usage: %s arg1 arg2" % programName)

>>> usage('Test', 1.0)
Test Version 1.0
Usage: Test arg1 arg2
```

Function Default Parameter values

- Parameters can be given a default values
- The function can be called with fewer arguments than there are parameters
- Parameters with default values must come last

```
>>> def printName(last, first, mi=""):
      print("%s, %s %s" % (last, first, mi))

>>> printName("Smith", "John")
Smith, John
>>> printName("Smith", "John", "Q")
Smith, John Q
```

Functions as Values

- Functions can be used just like any other data type
- Functions can be assigned to variables

```
def sub(a, b):
   return a-b

>>> op = sub
>>> print op(3, 5)
-2
>>> type(op)
<type 'function'>
```

Functions as Parameters

Functions can be passed to other functions

```
def convert(data, convertFunc):
      for i in range(len(data)):
         data[i] = convertFunc(data[i])
      return data

>>> convert(['1', '5', '10', '53'], int)
[1, 5, 10, 53]
>>> convert(['1', 'nerd', '10', 'hi!'], len)
[1, 4, 2, 3]
>>> convert(['1', '5', '10', '53'], complex)
[(1+0j), (5+0j), (10+0j), (53+0j)]
```

Functions can return multiple values

Return a tuple of values.

Example: **`string.split( )`** returns a tuple of substrings.

```
def separate(text, size=3):
    '''divide a string into two parts'''
    head = text[:size]
    tail = text[size:]
    return (head,tail)
# ok to omit parens: start,last = separate(...)
>>> (start,last) = separate('GOODBYE', 4)
>>> start
GOOD
>>> last
BYE
```

OOPs

- Encapsulation
- Polymorphism
- Inheritance
- Abstraction

Built-in Data Structures

- List l = [2, 3, 5, 8]
- Tuple (read-only list) t = (2, 3, 5, 8)
- Set s = { 2, 5, 3, 8 }
- Dictionary (key-value map) d = {"two":2, "three": 3, ...}

Lists

Syntax: **`[elem1, elem2, ...]`**

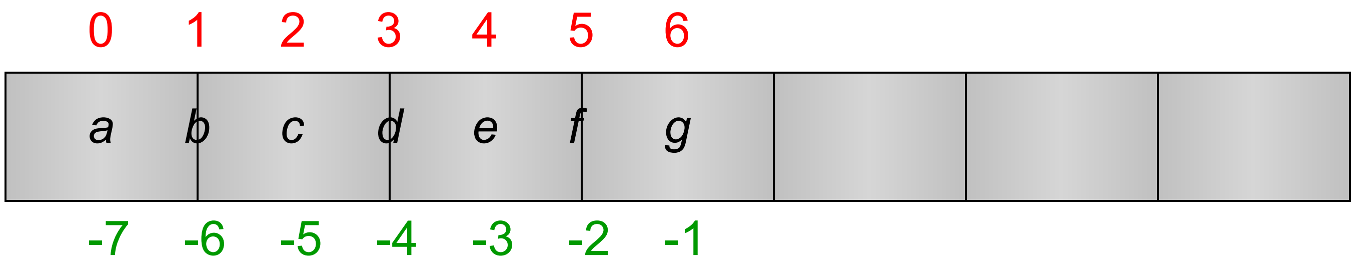
- Ordered sequence of any type (mixed types ok)
- Mutable

```
>>> list1 = [1, 'hello', 4+2j, 123.12]
>>> list1
[1, 'hello', (4+2j), 123.12]
>>> list1[0] = 'a'
>>> list1
['a', 'hello', (4+2j), 123.12]
```

Indexing

Syntax: **`list1[n]`**

- Positive indices count from the left: `list1[0]`
- Negative indices count from the right: `list1[-1]`



```
list1[0] == a     list1[-1] == g
list1[2] == c     list1[-2] == f
list1[6] == g     list1[-7] == a
```

Joining and Repeating Lists

Concatenation: `list1 + list2`

```
>>> [1, 'a', 'b'] + [3, 4, 5]
[1, 'a', 'b', 3, 4, 5]
```

Repetition: `list * count`

```
>>> [23, 'x'] * 4
[23, 'x', 23, 'x', 23, 'x', 23, 'x']
```

Adding Elements to a List

```
>>> list1 = [ "apple", "banana" ]
```

Append item to end

```
>>> list1.append( "orange" )
```

Append another list

```
>>> list1.extend( list2 )
```

- *Same as* `list1 + list2`

Insert item anywhere

```
>>> list1.insert( 0, "radish" )
>>> list1.insert( 2, "carrot" )
```

Removing Elements from a List

```
>>> list1 = [ "a", "b", "c", "b" ]
```

- **Remove a matching element (w/o returning it)**

```
>>> list1.remove( "b" )
```

Throws exception if argument is not in the list

- **Remove last element and return it**

```
>>> list1.pop( )

'b'
```

- **Remove by position**

```
>>> list1.pop( 1 )   # 'b' removed already
'c'

>>>del list1[ index ]
```

Delete element by index.

Not so object-oriented `len( )`

len( ) returns length of a list

```
>>> list1 = [ "a", "b", "c" ]
>>> len( list1 )
3
```

List Slicing: get a sublist

list1[m:n] return elements **m** (inclusive) up to **n** (exclusive)

```
>>> x = [0, 1, 2, 3, 4, 5, 6, 7]
>>> x[1:4]
[1, 2, 3]
>>> x[2:-1]
[2, 3, 4, 5, 6]
# Missing Index means start or end of list
>>> x[:2]
[0, 1]
>>> x[0:6:2]
[0,2,4]
```

Sorting a list

`List.sort( [`*`comparator`*`] )`

Sort `List` *in place*. Result is applied to the list!

Example:

```
>>> list3 = [4, 12, 3, 9]
>>> list3.sort()
>>> list3
[3, 4, 9, 12]
```

Reverse order of elements

`list.reverse( )`

Reverse elements of `list` *in place.*

Example:

```
>>> list3 = [4, 12, 3, 9]
>>> list3.reverse()
>>> list3
[9, 3, 12, 4]
```

Count or find elements in a list

`list.count( element )`

count number of occurences of element.

`n = list.index( element )`

return index of first occurence of `element`.

Throws **`ValueError`** if element is not in list.

Tuples

Immutable list

Syntax: **(elem1, elem2, …)**

A tuple cannot be changed.

Example:

```
>>> tuple1 = (1, 5, 10)
>>> tuple1[2] = 2
Traceback (most recent call last):
  File "<pyshell#136>", line 1, in ?
  tuple1[2] = 2
TypeError: object doesn't support item
assignment
```

Converting between list and tuple

```
>>> list1 = ['a', 'b', 'c']
>>> tuple1 = tuple( list1 )
>>> type( tuple1 )
<class 'tuple'>


>>> tuple2 = ('cat', 'dog')
>>> list2 = list(tuple2)
>>>type( list2 )
<class ‘list'>
```

Multiple assignment using tuples

```
(a,b,c) = (10, 20, 50)

>>> b

20
```

This can be used in **for** loops.

```
points = [ (1,0), (0.2,0.9), (1,2) ]

for (x,y) in points:

   r = math.hypot(x,y)

   print("distance of (%f,%f) from the Origin is%f" % 
(x,y,r) )
```

`split` a String

Syntax: **`string.split([seperator])`**

Returns a list of substrings

```
>>> text = "1 2 4 5 1"
>>> text.split()
['1', '2', '4', '5', '1']
>>> test = "a, b, c, d, e"
>>> test.split(',')
['a', ' b', ' c', ' d', ' e']
# notice the space before b c d e
```

String functions

`s = '''Now is the time for all good men'''`	Multi-line strings (triple quote)
`list = s.splitlines()`	return list of lines in string
`s.lower()`	to lowercase
`s.upper()`	to uppercase
`s.title()`	title case
`s.index('me')`	index of first occurrence, throw exception if substring not found
`s.count('me')`	count occurrences
`s[1:10]`	slice, just like list slice
`s.replace("men","people")`	replace substring.

Slicing in String

```
>>> "Hello nerd"[3:]
    'lo Nerd'
    >>> "Hello nerd"[1:10:2]
```

- 'el ed'

`strip` leading/trailing whitespace

`string.strip()`

Remove leading and trailing white space (tab, new line, etc)

```
>>> padded = "   stuff   "
>>> unpadded = padded.strip()
>>> unpadded
'stuff'
>>> padded
'   stuff   '
# strings are immutable
```

String format functions

`>>> "Hello".`**`ljust`**`(8)`
`"Hello   "`
	Left justify to given length.

`>>> "Hello".`**`rjust`**`(8)`
`"   Hello"`
	Right justify.

`>>> "Hello".`**`center`**`(8)`
`" Hello  "`
	Center, of course.

`>>> u = "Bird"`
`>>> "Hello {0}".`**`format`**`(u)`
`'Hello Bird'`
	Format using template.

Set

An unordered collection, without duplicates (like Java).

Syntax is like dictionary, but no ":" between key-value.

```
>>> aset = { 'a', 'b', 'c' }
>>> aset
{'a', 'c', 'b'}
>>> aset.add('c")      # no effect, 'c' already in set
>>> aset
{'a', 'c', 'b'}
```

Set Methods

`set.discard('cat')`	remove cat. No error if not in set.
`set.remove('cat')`	remove cat. Error if not in set.
`set3 = set1.union(set2)`	doesn't change set1.
`set4 = set1.intersection(set2)`	doesn't change set1.
`set2.issubset( set1 )`	
`set2.issuperset( set1 )`	
`set1.difference( set2 )`	`element in set1 not set2`
`set1.symmetric_difference(set2)`	xor
`set1.clear( )`	remove everything

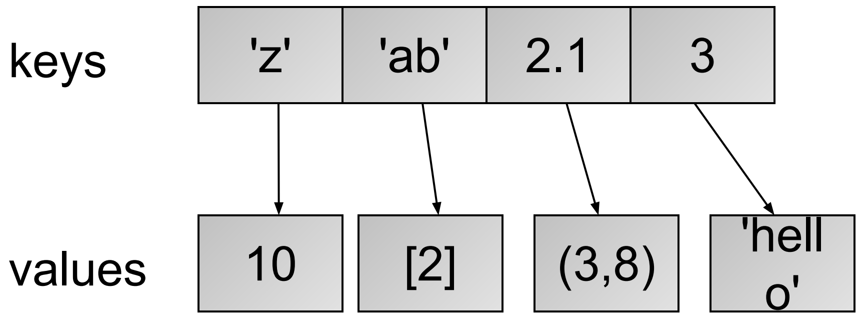
Test for element in Set

item* in *set

```
>>> aset = { 'a', 'b', 'c' }
>>> 'a' in aset
True
>>> 'A' in aset
False
```

Dictionary: mapping key to value

- A mapping of keys to values
  - Associate a key with a value
  - Each key must be unique



Using Dictionaries

Syntax: dict = `{key1: value1, key2: value2, ...}`

```
>>> dict = {'a': 1, 'b': 2}
>>> dict
{'a': 1, 'b': 2}
>>> dict['a']
1
>>> dict['b']
2
>>> dict[3] = 'three'
>>> dict
{'a': 1, 'b': 2, 3: 'three'}
```

Dictionary Methods

`dict = {'a': 1, 'b':2, 'c':30}`	Example

`dict.keys()`
`['a', 'b', 'c']`
	list of keys

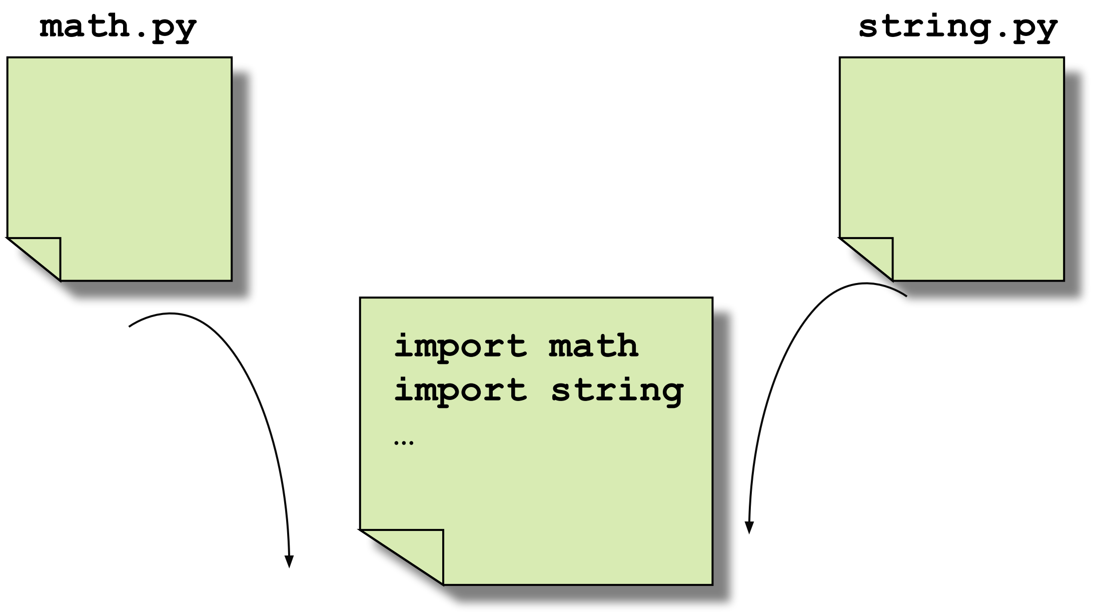
`dict.values( )`
`[1, 2, 30]`
	list of values

`dict.has_key('d')`
`False`
	Test for key in dictionary

Modules

A file containing Python definitions and statements

- Modules can be “imported”
- Module file name must end in .py
- Used to divide code between files



```
import math
import string
…
```

`import` Statement

`import <module name>`

- **`module name`** is the file name without **`.py`** extension
- You must use the module name to call functions

```
>>> import math
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'e', 'exp', 'fabs',
'floor', 'fmod', 'frexp', ...]
>>> math.e
2.71828182846
>>> math.sqrt(2.3)
1.51657508881
```

`import` specific names

`from <module> import <name>`

- Import a specific name from a module into global namespace
- Module name is not required to access imported name(s)

```
>>> from math import sqrt
>>> sqrt(16)
4
>>> dir(math)
  Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: name 'math' is not defined
```

`import` all names from module

`from <module> import *`

- Import everything into global namespace

```
>>> dir()
['__builtins__', '__doc__', '__name__']
>>> from time import *
>>> dir()
['__builtins__', '__doc__', '__name__',
'accept2dyear', 'altzone', 'asctime', 'clock',
'ctime', 'daylight', 'gmtime', 'localtime', 'mktime',
'sleep', 'strftime', 'time', ... ]
>>> time()
1054004638.75
```

Python Standard Libraries

`sys`	System-specific parameters and functions
`time`	Time access and conversions
`thread`	Multiple threads of control
`re`	Regular expression operations
`email`	Email and MIME handling
`httplib`	HTTP protocol client
`tkinter`	GUI package based on TCL/Tk (in Python 2.x this is named Tkinter)
`Urllib`	Open an Arbitrary URL.

See http://docs.python.org/library/index.html

Applications Of Python