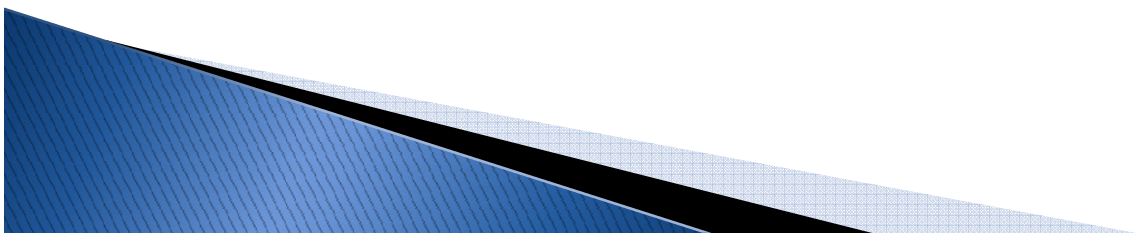
Introduction to Python

Adapted from a Tuturial by Guido van Rossum
Director of PythonLabs at Zope Corporation

Presented at
LinuxWorld - New York City - January 2002

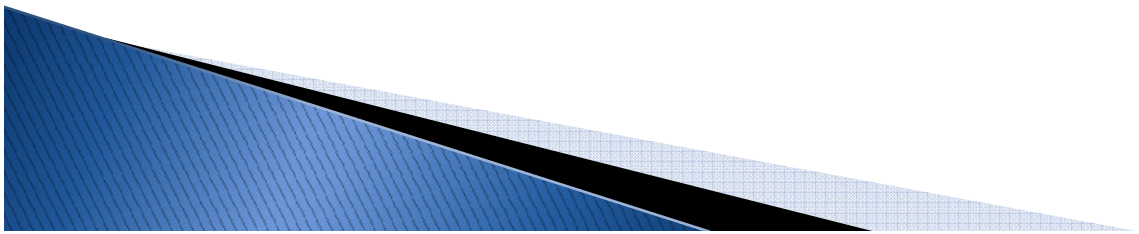
Why Python?

- Interpreted language
- Intuitive syntax
- Dynamic typing
- **Loads** of built-in libraries and available extensions
- Shallow learning curve
- Easy to call C/C++ for efficiency
- Object-oriented
- Simple, but extremely powerful



Tutorial Outline

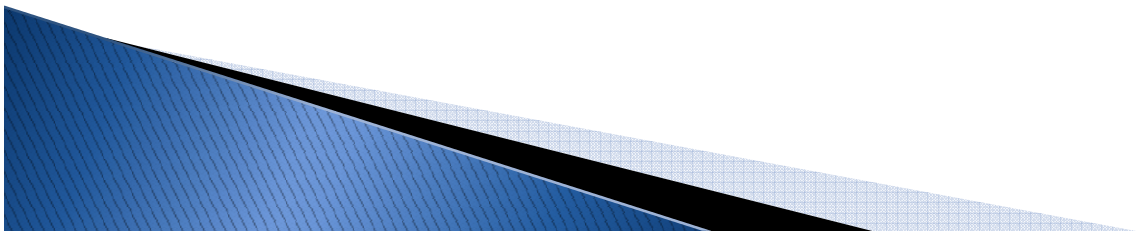
- interactive "shell"
- basic types: numbers, strings
- container types: lists, dictionaries, tuples
- variables
- control structures
- functions & procedures
- classes & instances
- modules
- exceptions
- files & standard library



Interactive "Shell"

- Great for learning the language
- Great for experimenting with the library
- Great for testing your own modules
- Two variations: IDLE (GUI), python (command line)
- Type statements or expressions at prompt:

```
>>> print "Hello, world"
Hello, world
>>> x = 12**2
>>> x/2
72
>>> # this is a comment
```



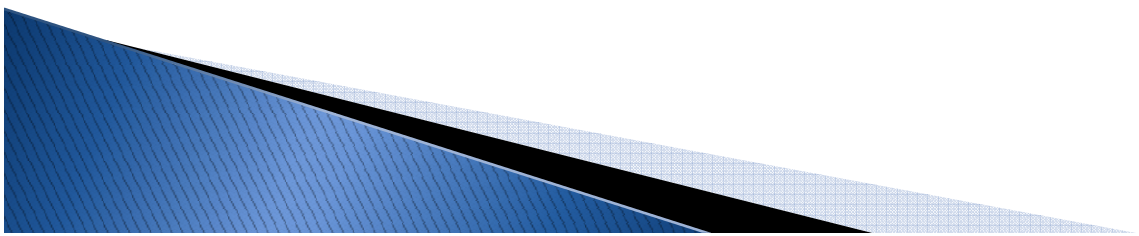
Python Program

- To write a program, put commands in a file

```
#hello.py
print "Hello, world"
x = 12**2
x/2
print x
```

- Execute on the command line

```
~> python hello.py
Hello, world
72
```

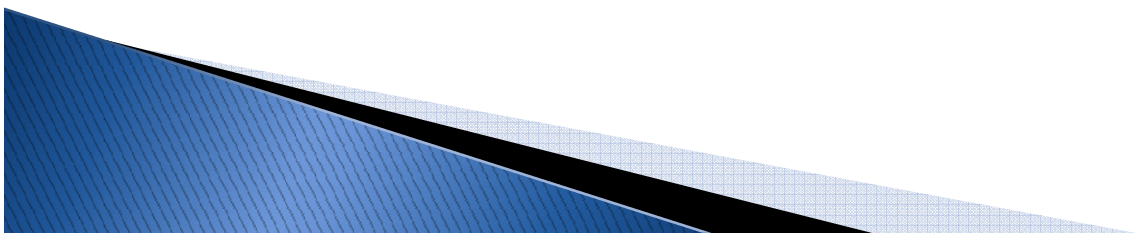


Variables

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Not typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- ***Everything*** is an “object":
  - Even functions, classes, modules



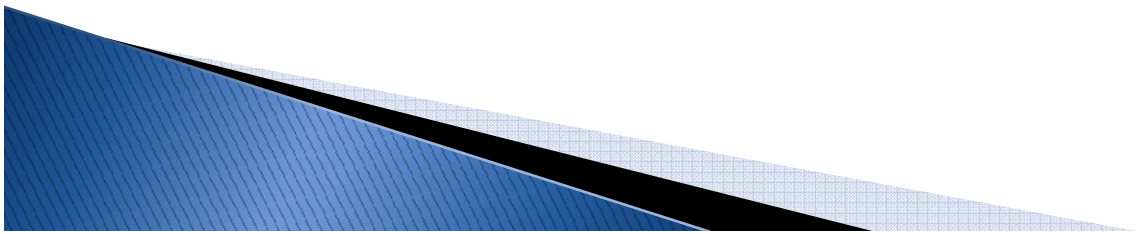
Control Structures

if *condition*:
    statements
[elif *condition*:
    statements] ...
else:
    statements

while *condition*:
    statements

for *var* in *sequence*:
    statements

break
continue



Grouping Indentation

In Python:

```
for i in range(20):
    if i%3 == 0:
        print i
        if i%5 == 0:
            print "Bingo!"
    print "---"
```

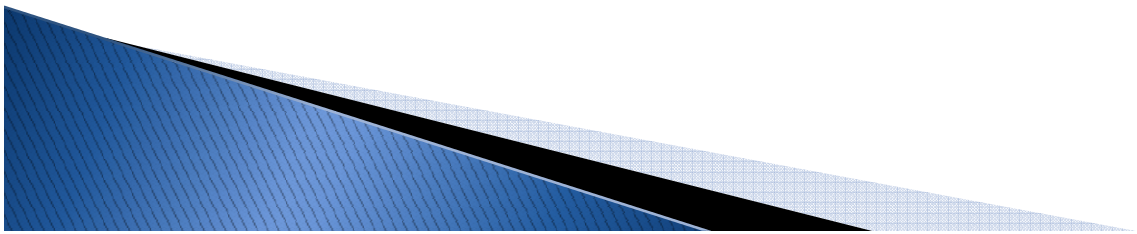
In C:

```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {

   printf("Bingo!\n"); }
      }
      printf("---\n");
}
```

```
0
Bingo!
---
---
---
3
---
---
---
6
---
---
---
9
---
---
---
12
---
---
---
15
Bingo!
---
---
---
18
---
---
```

Numbers

- The usual suspects
  - 12, 3.14, 0xFF, 0377, (-1+2)*3/4**5, abs(x), 0<x<=5
- C-style shifting & masking
  - 1<<16, x&0xff, x|1, ~x, x^y
- Integer division truncates :-(
  - 1/2 -> 0 # 1./2. -> 0.5, float(1)/2 -> 0.5
  - Will be fixed in the future
- Long (arbitrary precision), complex
  - 2L**100 -> 1267650600228229401496703205376L
    - In Python 2.2 and beyond, 2**100 does the same thing
  - 1j**2 -> (-1+0j)

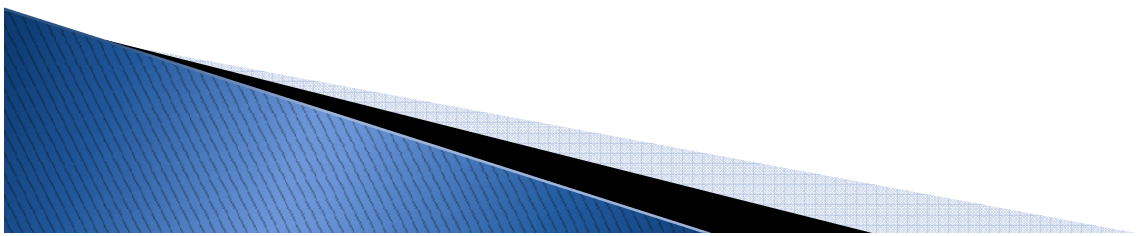


Strings

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello" 1 # search
- "escapes: \n etc, \033 etc, \if etc"
- 'single quotes' """triple quotes""" r"raw strings"

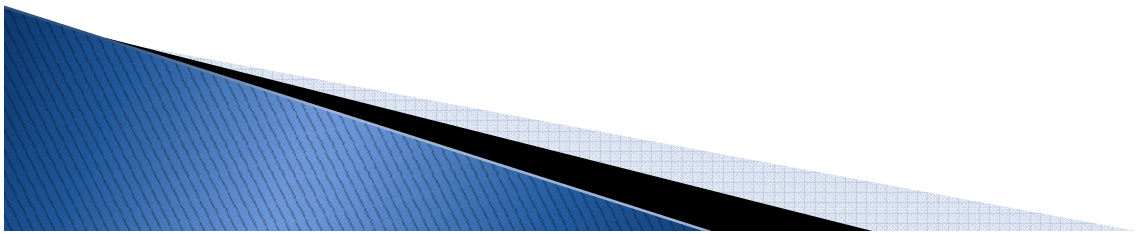
Lists

- Flexible arrays, *not* Lisp-like linked lists
  - a = [99, "bottles of beer", ["on", "the", "wall"]]
- Same operators as for strings
  - a+b, a*3, a[0], a[-1], a[1:], len(a)
- Item and slice assignment
  - a[0] = 98
  - a[1:2] = ["bottles", "of", "beer"]
    -> [98, "bottles", "of", "beer", ["on", "the", "wall"]]
  - del a[-1] # -> [98, "bottles", "of", "beer"]



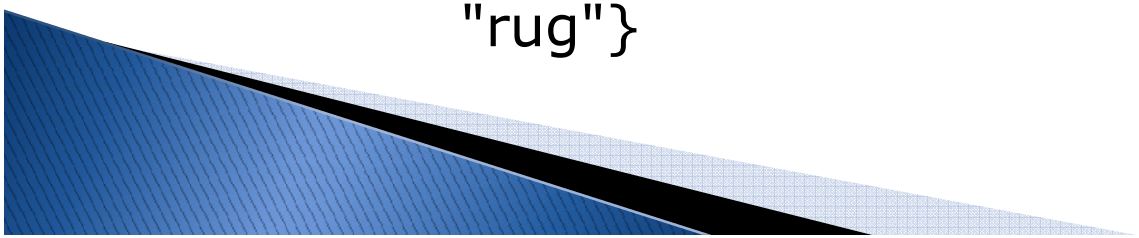
More List Operations

```
>>> a = range(5)            # [0,1,2,3,4]
>>> a.append(5)             # [0,1,2,3,4,5]
>>> a.pop()                 # [0,1,2,3,4]
5
>>> a.insert(0, 42)         # [42,0,1,2,3,4]
>>> a.pop(0)                # [0,1,2,3,4]
5.5
>>> a.reverse()             # [4,3,2,1,0]
>>> a.sort()                # [0,1,2,3,4]
```



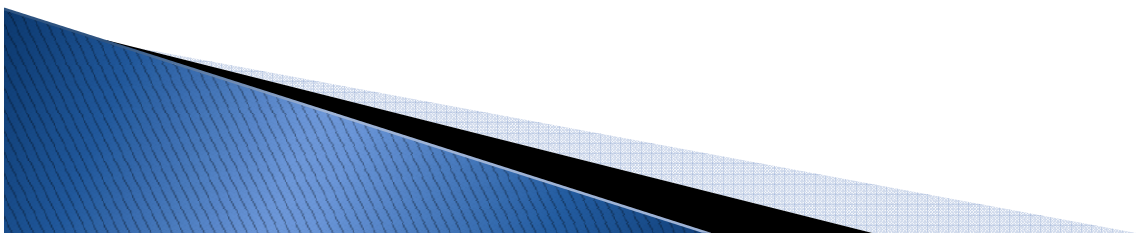
Dictionaries

- Hash tables, "associative arrays"
  - d = {"duck": "eend", "water": "water"}
- Lookup:
  - d["duck"] -> "eend"
  - d["back"] # raises KeyError exception
- Delete, insert, overwrite:
  - del d["water"] # {"duck": "eend", "back": "rug"}
  - d["back"] = "rug" # {"duck": "eend", "back": "rug"}
  - d["duck"] = "duik" # {"duck": "duik", "back": "rug"}



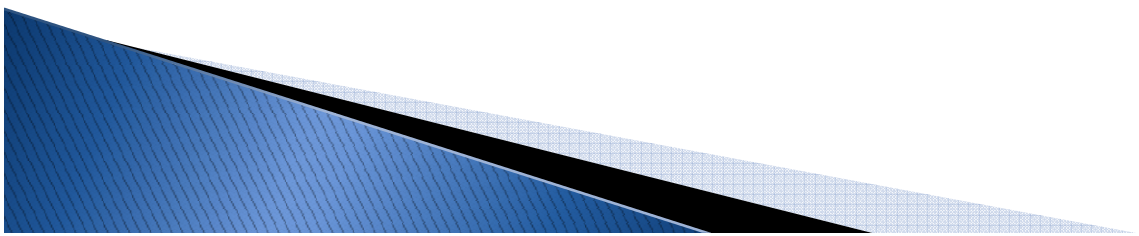
More Dictionary Ops

- Keys, values, items:
  - d.keys() -> ["duck", "back"]
  - d.values() -> ["duik", "rug"]
  - d.items() -> [("duck","duik"), ("back","rug")]
- Presence check:
  - d.has_key("duck") -> 1; d.has_key("spam") -> 0
- Values of any type; keys almost any
  - {"name":"Guido", "age":43, ("hello","world"):1,
    42:"yes", "flag": ["red","white","blue"]}



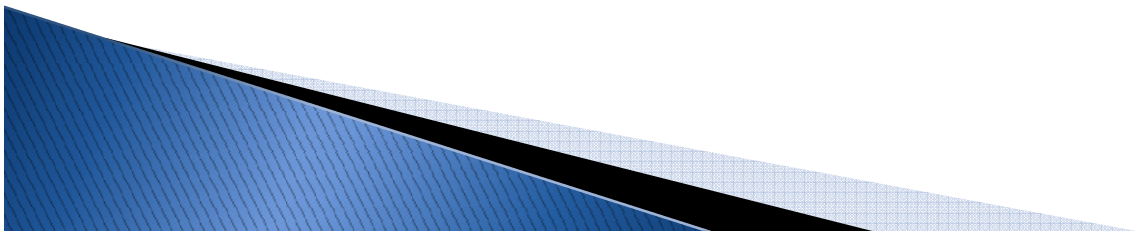
Dictionary Details

- Keys must be **immutable**:
  - numbers, strings, tuples of immutables
    - these cannot be changed after creation
  - reason is *hashing* (fast lookup technique)
  - **not** lists or other dictionaries
    - these types of objects can be changed "in place"
  - no restrictions on values
- Keys will be listed in **arbitrary order**
  - again, because of hashing



Tuples

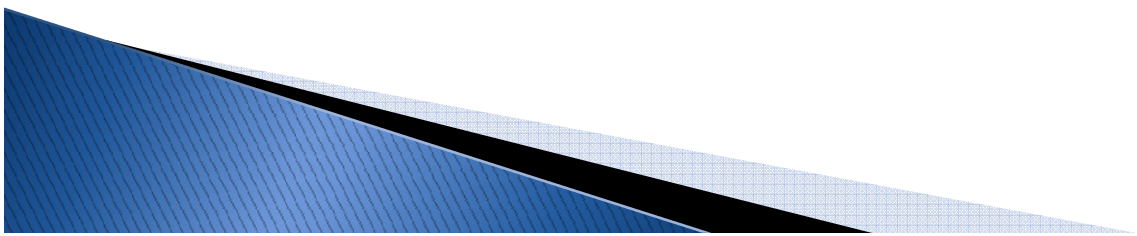
- key = (lastname, firstname)
- point = x, y, z # parentheses optional
- x, y, z = point # unpack
- lastname = key[0]
- singleton = (1,) # trailing comma!!!
- empty = () # parentheses!
- tuples vs. lists; tuples immutable



Reference Semantics

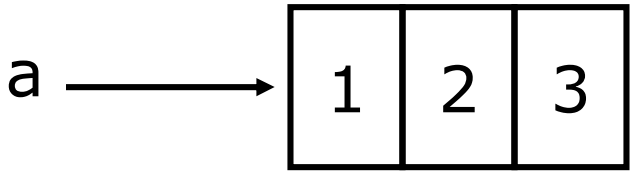
- Assignment manipulates references
  - x = y **does not make a copy** of y
  - x = y makes x **reference** the object y references
- Very useful; but beware!
- Example:

```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

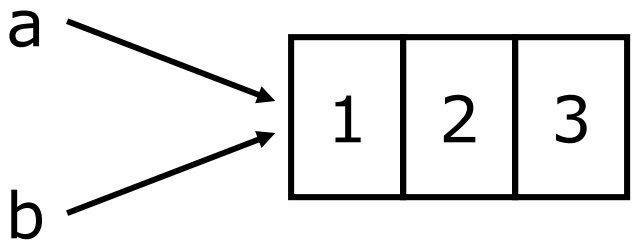


Changing a Shared List

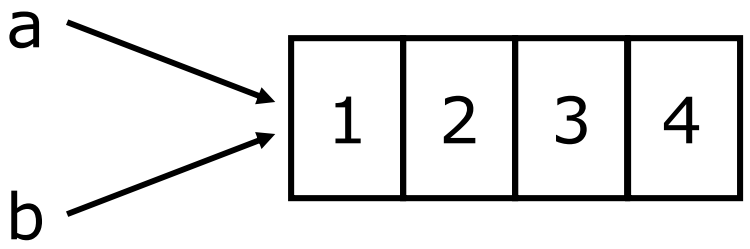
a = [1, 2, 3]



b = a



a.append(4)

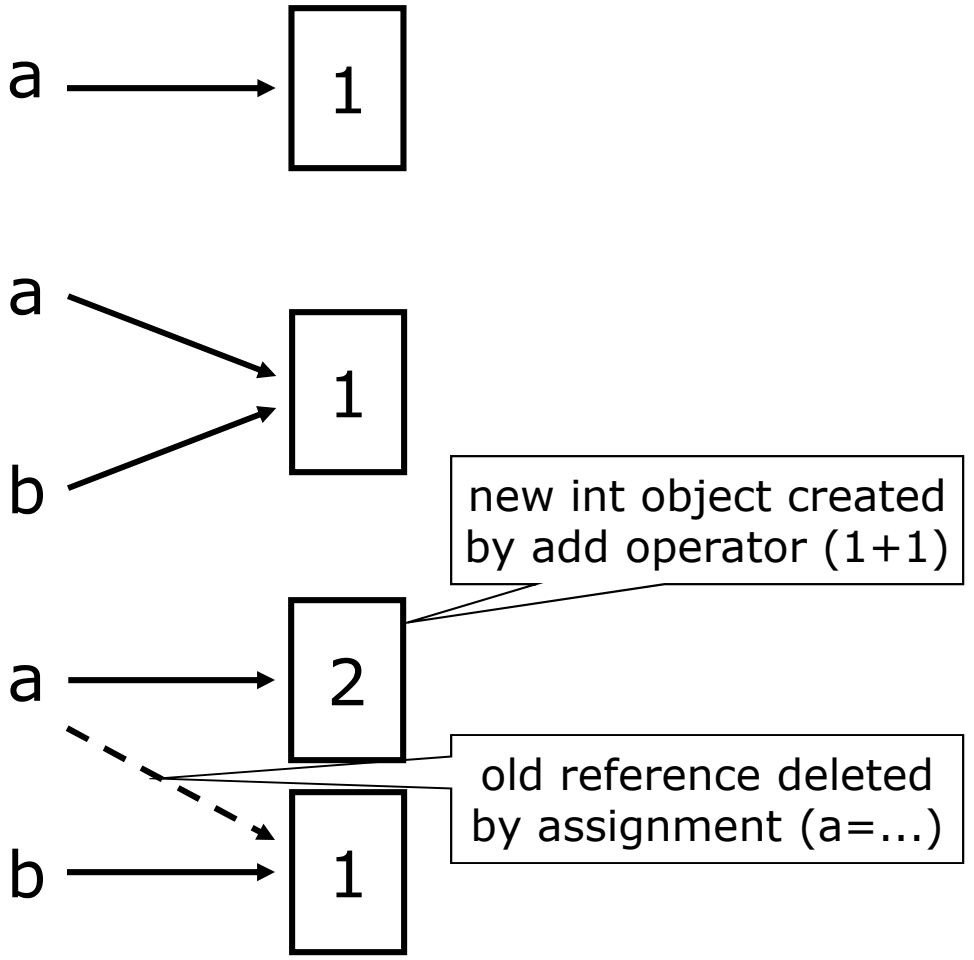


Changing an Integer

a = 1

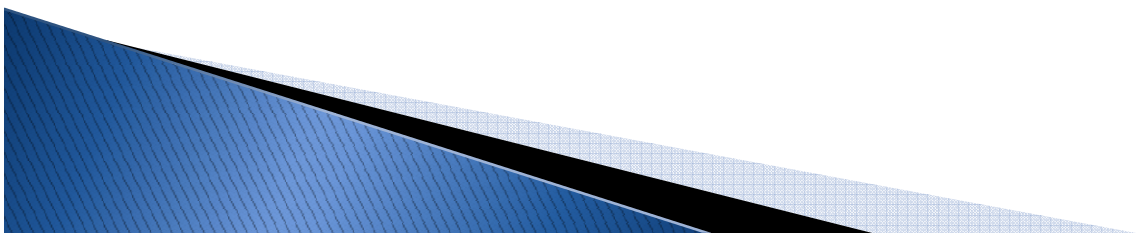
b = a

a = a+1



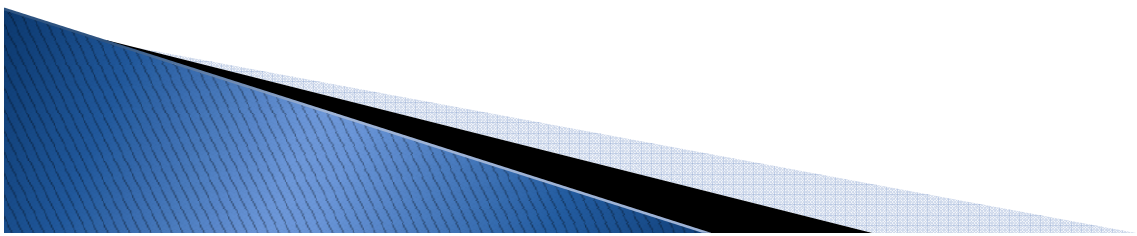
Functions, Procedures

```
def name(arg1, arg2, ...):
    """documentation"""     # optional doc
  string
    statements

return                      # from procedure
return expression                 # from function
```



Example Function

```
def gcd(a, b):
    "greatest common divisor"
    while a != 0:
        a, b = b%a, a     # parallel assignment
    return b

>>> gcd.__doc__
'greatest common divisor'
>>> gcd(12, 20)
4
```



Classes

class *name*:

    "*documentation*"

    statements

-or-

class *name*(*base1*, *base2*, ...):

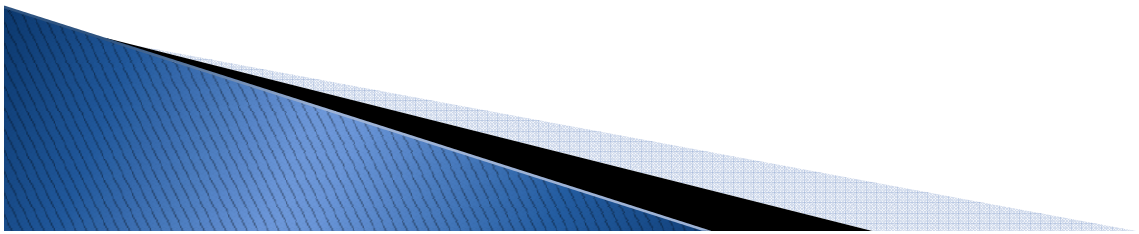
    ...

Most, *statements* are method definitions:

    def *name*(self, *arg1*, *arg2*, ...):

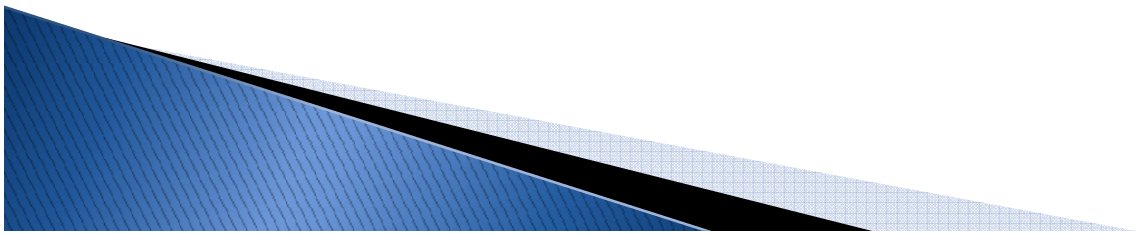
        ...

May also be *class variable* assignments



Example Class

```
class Stack:
    "A well-known data structure…"
    def __init__(self):             # constructor
        self.items = []
    def push(self, x):
        self.items.append(x)        # the sky is the limit
    def pop(self):
        x = self.items[-1]                  # what happens if it's empty?
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items) == 0         # Boolean result
```



Using Classes

- To create an instance, simply call the class object:

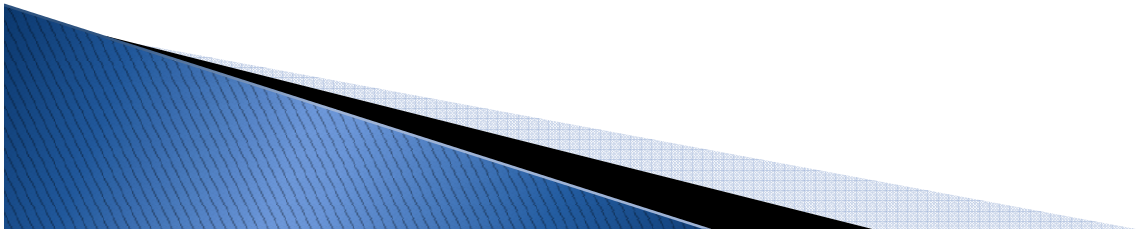
```
x = Stack()# no 'new' operator!
```

- To use methods of the instance, call using dot notation:

```
x.empty()   # -> 1
x.push(1)                       # [1]
x.empty()   # -> 0
x.push("hello")                             # [1, "hello"]
x.pop()                 # -> "hello"        # [1]
```

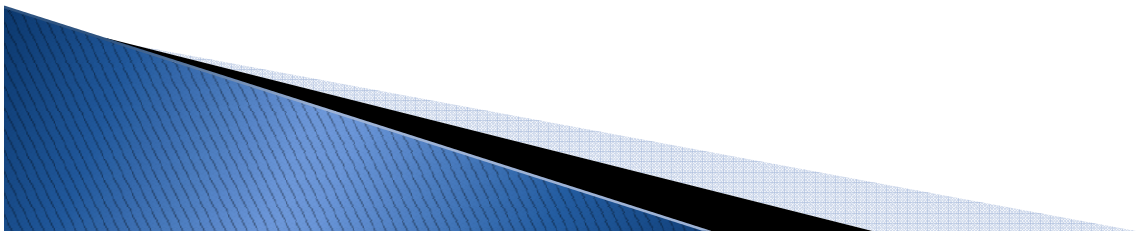
- To inspect instance variables, use dot notation:

```
x.items                 # -> [1]
```



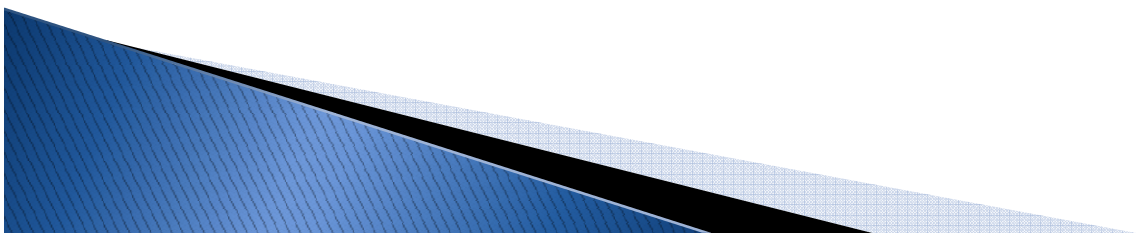
Class / Instance Variables

```
class Connection:
    verbose = 0                         # class variable
    def __init__(self, host):
        self.host = host                # instance variable
    def debug(self, v):
        self.verbose = v                # make instance variable!
    def connect(self):
        if self.verbose:                # class or instance variable?
            print "connecting to", self.host
```



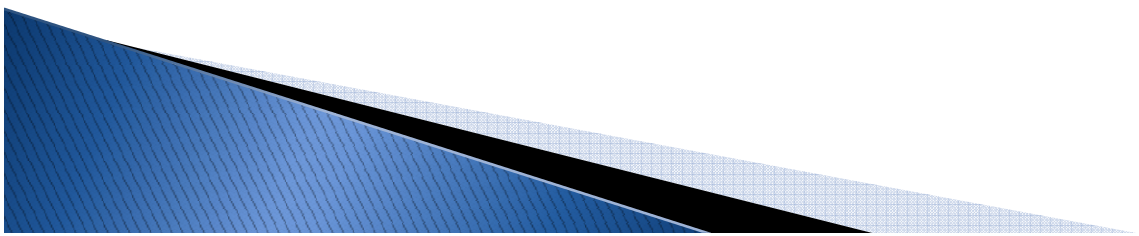
Instance Variable Rules

- On use via instance (self.x), search order:
  - (1) instance, (2) class, (3) base classes
  - this also works for method lookup
- On assignment via instance (self.x = ...):
  - always makes an instance variable
- Class variables "default" for instance variables
- But...!
  - mutable *class* variable: one copy *shared* by all
  - mutable *instance* variable: each instance its own



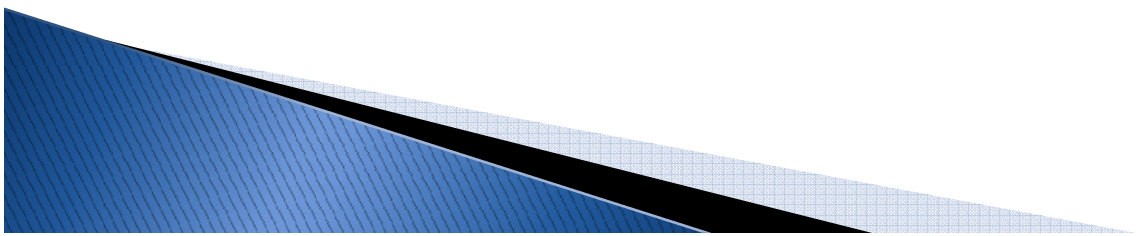
Modules

- Collection of stuff in *foo*.py file
  - functions, classes, variables
- Importing modules:
  - import re; print re.match("[a-z]+", s)
  - from re import match; print match("[a-z]+", s)
- Import with rename:
  - import re as regex
  - from re import match as m



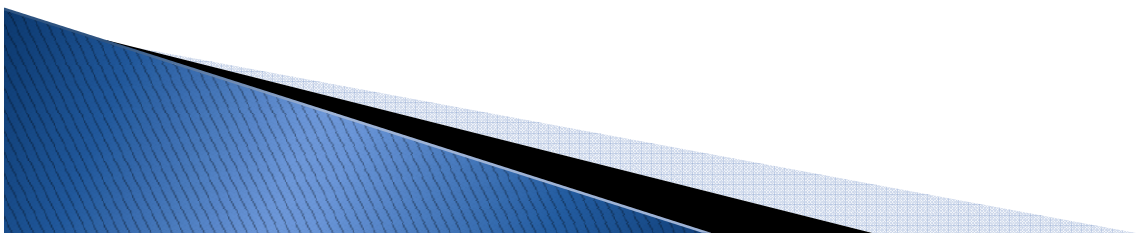
Catching Exceptions

```
def foo(x):
    return 1/x

def bar(x):
    try:
        print foo(x)
    except ZeroDivisionError, message:
        print "Can’t divide by zero:", message

bar(0)
```



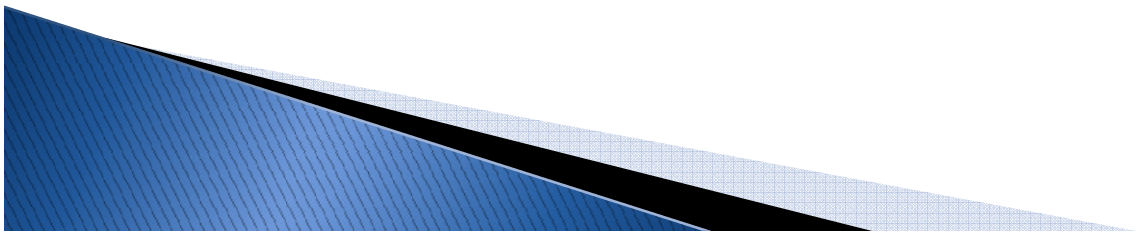
Try-finally: Cleanup

```
f = open(file)
try:
    process_file(f)
finally:
    f.close()       # always executed
print "OK"          # executed on success only
```



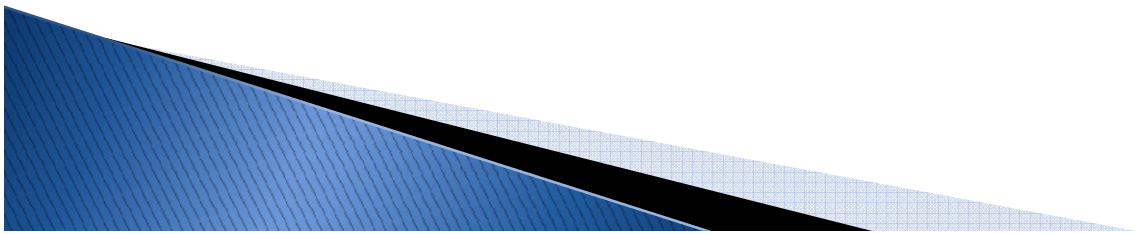
Raising Exceptions

- raise IndexError
- raise IndexError("k out of range")
- raise IndexError, "k out of range"
- try:
  something
  except: # catch everything
  print "Oops"
  raise # reraise



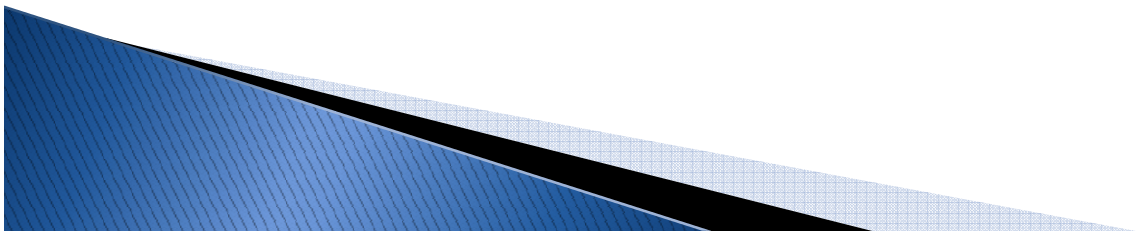
More on Exceptions

- User-defined exceptions
  - subclass Exception or any other standard exception
- Old Python: exceptions can be strings
  - WATCH OUT: compared by object identity, not ==
- Last caught exception info:
  - sys.exc_info() == (exc_type, exc_value, exc_traceback)
- Last uncaught exception (traceback printed):
  - sys.last_type, sys.last_value, sys.last_traceback
- Printing exceptions: traceback module



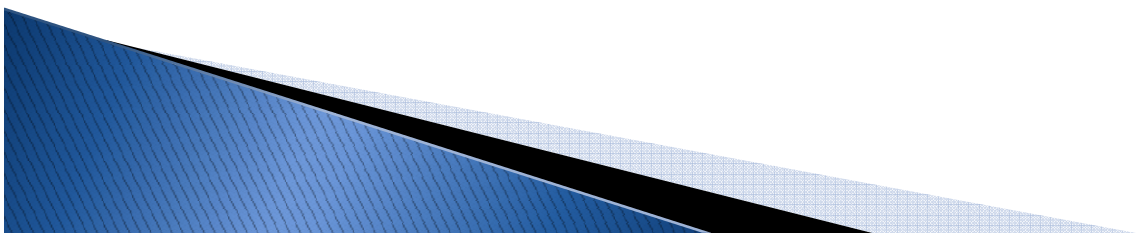
File Objects

- f = open(*filename*[, *mode*[, *buffersize*])
  - mode can be "r", "w", "a" (like C stdio); default "r"
  - append "b" for text translation mode
  - append "+" for read/write open
  - buffersize: 0=unbuffered; 1=line-buffered; buffered
- methods:
  - read([*nbytes*]), readline(), readlines()
  - write(*string*), writelines(*list*)
  - seek(*pos*[, *how*]), tell()
  - flush(), close()
  - fileno()



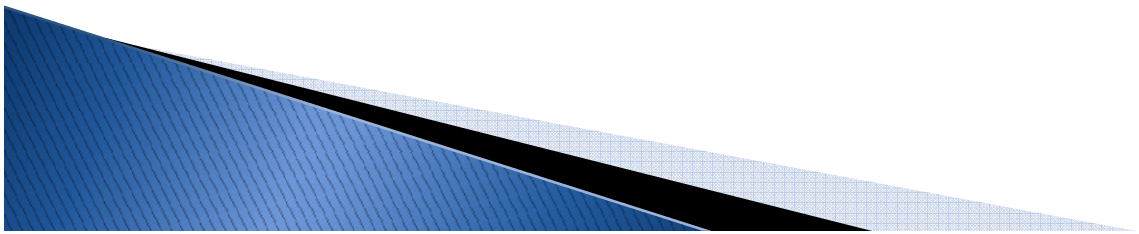
Standard Library

- Core:
  - os, sys, string, getopt, StringIO, struct, pickle, ...
- Regular expressions:
  - re module; Perl-5 style patterns and matching rules
- Internet:
  - socket, rfc822, httplib, htmllib, ftplib, smtplib, ...
- Miscellaneous:
  - pdb (debugger), profile+pstats
  - Tkinter (Tcl/Tk interface), audio, *dbm, ...



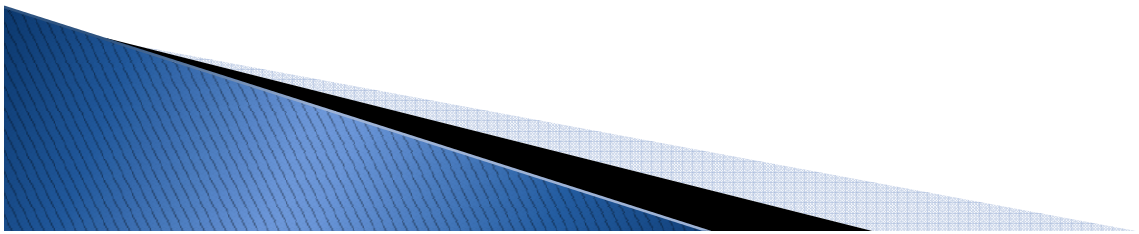
URLs

- http://www.python.org
  - official site
- http://starship.python.net
  - Community
- http://www.python.org/psa/bookstore/
  - (alias for http://www.amk.ca/bookstore/)
  - Python Bookstore



Further Reading

- Learning Python: Lutz, Ascher (O'Reilly '98)
- Python Essential Reference: Beazley (New Riders '99)
- Programming Python, 2nd Ed.: Lutz (O'Reilly '01)
- Core Python Programming: Chun (Prentice-Hall '00)
- The Quick Python Book: Harms, McDonald (Manning '99)
- The Standard Python Library: Lundh (O'Reilly '01)
- Python and Tkinter Programming: Grayson (Manning '00)
- Python Programming on Win32: Hammond, Robinson (O'Reilly '00)
- Learn to Program Using Python: Gauld (Addison-W. '00)
- And many more titles...



QUESTIONS