Artificial Intelligence Programming

Intro to Python

Chris Brooks

Department of Computer Science
University of San Francisco

What is Python?

- Python is:
  - High-level
  - Object-oriented
  - Free, open-source
  - Dynamically typed
  - Has a large collection of utility libraries
  - garbage-collected
  - Mixable - works nicely as a “glue” language
  - Easy to learn/use

Some uses of Python

- Things that Python is good at:
  - System utilities
    - Python has hooks into standard OS-level services, such as files, processes, pipes, etc.
  - GUIs
    - Python has interfaces to Tk, Qt, and WxPython
  - Embedded integration
    - Python has hooks that allow it to call/be called by C and C++ code, and also COM interfaces.
  - Rapid Interactive Development
    - Like Lisp, Python makes it easy to quickly put together a working program.
  - Scripting
    - Python has utilities for CGI, Database access, HTTP, Sockets, FTP, POP/SMTP, XML parsing, etc.

Invoking Python Programs

- There are four ways to run a Python program:
  - Interactively
  - From the command line
  - As a script
  - From another program

Python Implementations

- /usr/bin/python on all unix machines and on OS X
- IDLE and PythonWin on Windows
- MacPython on Macintosh
- Also for Amiga, PalmOS, BeOS, etc.
  - BoaConstructor is an open-source Python IDE
  - Eclipse has a Python plugin (PyDev)

Python Program Structure

- Programs consist of modules
  - A module corresponds to a source file.
- Modules contain blocks of statements
- Statements create and process objects.

Basic Python

- Python has a nice set of built-in data types
  - Numbers
  - Strings
  - Lists
  - Sets
  - Dictionaries
  - Files
- Using built-in types makes your code easier to write/maintain, more portable, and more efficient.

Numbers

- Numbers work like we'd expect.
- There are integers, equivalent to longs in C. (1, -31311, 4000)
- There are *long integers*, which are of unlimited size (31111L, 12345l)
- There are floats, equivalent to doubles in C or Java. 1.23, 3.1e+5
- There are Octal and Hexadecimal representations as in C. (0155, 0x3af5)
- There are complex numbers, as in Lisp, Matlab, etc. (3.0+4j)

Mathematical Operators

- Python has all the usual mathematical operators
- +,=,*,/, %
- «, »
- ** or pow for exponentiation
- abs, rand, |, &
- This makes Python a very handy desk calculator.
- Operations are coerced to the most specific type.
  - 3 + (4.0 / 2) will produce a float.
  - Common error: since variables do not have declared types, be careful of rounding: 3 /2 == 1 !!

Strings

- One of Python's strong suits is its ability to work with strings.
- Strings are denoted with double quotes, as in C, or single quotes
- s1 + s2 - concatenation
- s1 * 3 - repetition
- s1[i] - indexing, s1[i:j] - slicing
- s1[-1] - last character
- "a % parrot" % "dead" - formatting
- for char in s1 - iteration

Strings

- Strings are immutable sequences - to change them, we need to make a copy.
  - Can't do: s1[3] = 'c'
  - Must do: s2 = s1[0:2] + 'c' + s1[3:]
- As in Java, making lots of copies can be very inefficient. If you need to do lots of concatenation, use join instead.
- We'll return to efficiency issues throughout the semester.

Lists

- Python has a flexible and powerful list structure.
- Lists are mutable sequences - can be changed in place.
- Denoted with square brackets. l1 = [1,2,3,4]
- Can create nested sublists. l2 = [1,2, [3,4, [5], 6], 7]
- l1 + l2 - concatenation.
- l1 * 4 - repetition
- l1[3:5], l1[:3], l1[5:] - slices
- append, extend, sort, reverse built in.
- Range - create a list of integers

Dictionaries

- A Dictionary is a Python hash table (or associative list)
- Unordered collections of arbitrary objects.
- d1 = {} - new hashtable d2 = {'spam' : 2, 'eggs', 3}
- Can index by key: d2['spam']
- Keys can be any immutable object.
- Can have nested Hashtables
  - d3 = {'spam' : 1, 'other' :{'eggs' :2, 'spam' : 3}}
  - d3['other']['spam']
- has_key, keys(), values(), for k in keys()
- Typically, you'll insert/delete with:
  - d3['spam'] = 'delicious!'
  - del d3['spam']

Tuples

- Tuples are like immutable lists.
- Nice for dealing with enumerated types.
- Can be nested and indexed.
- t1 = (1,2,3), t2 = (1,2,(3,4,5))
- Can index, slice, length, just like lists.
  - t1[3], t1[1:2], t1[-2]
- Tuples are mostly useful when you want to have a list of a predetermined size/length.
- Also, constant-time access to elements. (fixed memory locations)
- Tuples are also very useful as keys for dictionaries.

Files

- Since it's a scripting language, Python has a lot of support for file I/O
- Operators are not too different from C.
- Outfile = file('fname', 'w') or infile = file('fname', 'r')
  - 'r' is default and can be left out
- S = infile.read() - read the entire file into the string S.
- S = infile.read(N) - read N lines into the string S.
- S = input.readline() - read one line
- S = input.readlines() - read the whole file into a list of strings.
  - Unless the file is *really* huge, it's fastest to read it all in at once with read() or readlines()

Files

- outfile.write(S) - write the string S into the file.
- outfile.writelines(L) - write the list of strings L into the file.
- outfile.close() (this is also done by the garbage collector)

Basic Python statements

- Python uses dynamic typing.
  - No need to pre-define variables.
- Variables are instantiated by assigning values to them
  - Referencing a variable before assignment is an error
- You can assign multiple variables simultaneously

```
spam = 4
eggs = 5
spam, eggs = eggs, spam
spam, eggs = 4,5
```


Python variables

- Variables must:
  - begin with a letter or underscore
  - contain any number of letters, numbers, or underscores.
- No $, @, #, etc.
- Case matters.
- Can't use reserved words as variables.

Printing

- We've already seen the basic print.
  - print "hello world"
- To use a formatting string, do:
  - print "hello %s" % "bob"
  - print "%s %s" % ("hello" , "world")
- To suppress the linefeed, include a ,

Conditionals

- The general format for an if statement is:

```
if <test1> :
    <statement1>
    <statement2>
elif: <test2> :
    <statement3>
else:
    <statementn>
```

- Notice the colons after the conditionals.
- Compound statements consist of the colon, followed by an indented block.

Conditionals

- Logical tests return 1 for true and 0 for false.
- "True" and "False" are shorthand
- and, or, not are available for compound tests

Syntax

- Indentation is used to delimit blocks
  - (If you are going to be editing your code on multiple machines with different editors, you may want to configure your editor to use spaces instead of tabs.)
  - Statements can cross multiple lines if:
    - They're within delimiters
    - You use a backslash

Iteration

- Python has the familiar while loop.
- One wrinkle: it has an optional else clause to execute if the test is false.
- Additional loop control
  - break
    - Exit the innermost loop without doing an else
  - continue
    - Return to the beginning of the loop
  - pass
    - Do nothing

For

- For is a generic iterator in Python.
- Lets you loop over any indexable data structure.
  - for item in [1,2,3,4,5]
  - for char in "This is a string"
  - for key in dictionary
- This provides a uniform (polymorphic) interface to a set of different data structures.
- Note: it's much faster to use the polymorphic operator than to access manually.
  - i.e. for item in list is faster than for i in len(list)
  - for item in dictionary is faster than for item in dictionary.keys()

Functions

- def is used to define a function.
- return returns a value
- Functions maintain local scope
- Names are resolved locally, then globally, then built-in

Functions

- Multiple values can be returned in a tuple.
- We can also provide default arguments.
- Functions can be called with keywords.
  - myfunc(spam=1, eggs=2)
- *args can be used to catch arbitrary argument lists and store them in a tuple.
- **args can be used to catch arbitrary argument lists and store them in a dictionary.

Basic Python advice

- Let the language do the work for you
  - e.g.: if x in [1,2,3,4] rather than an an explicit loop
  - this is both faster and more readable
- Use a Python-aware editor/IDE
- Take advantage of the built-in modules whenever possible.
- Use the interpreter to help test your code
- All built-in modules are readable
  - Looking at these can give insights into how they work, as well as how to write good Python code.