Python

Tutorial Lecture for CSE 415

Introduction to Artificial Intelligence

Why Python for AI?

- For many years, we used Lisp, because it handled lists and trees really well, had garbage collection, and didn't require type declarations.
- Lisp and its variants finally went out of vogue, and for a while, we allowed any old language, usually Java or C++. This did not work well. The programs were big and more difficult to write.
- A few years ago, the AI faculty started converting to Python. It has the object-oriented capabilities of Java and C++ with the simplicity for working with list and tree structures that Lisp had with a pretty nice, easy-to-use syntax. I learned it with very little work.

Getting Started

- Download and install Python 2.7 from www.python.org on your computer or use it from a lab in the library. They have both 2.7 and 3.2. We will use 2.7, as it is fine for AI search programs.
- Read "Python as a Second Language," a tutorial that Prof. Tanimoto wrote for CSE 415 students (see web page)
- You can also look at the hands-on tutorial provided for majors courses at:

http://courses.cs.washington.edu/courses/cse473/13au/pacman/intro/tutorial.html

Python Data Types

• int	105
• float	3.14159
• str	“Selection:”, ‘a string’
• bool	True, False
• list	[‘apple’, ‘banana’, ‘orange’]
• tuple	(3.2, 4.5, 6.3)
• dict	{‘one’: 1, ‘two’: 2}
• function	lambda x:2*x
• builtin_function_or_method	math.sqrt

Interacting with Python

```
$ python
Python 2.7.5 (default, Nov 12 2013, 16:18:42)
[GCC 4.8.2 20131017 (Red Hat 4.8.2-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 5 + 7
12
>>> x = 5 + 7
>>> x
12
>>> print('x = '+str(x))
x = 12
>>> x = 'apple'
>>> x + x
'appleapple'
>>> print('x is an '+x)
x is an apple
```

Defining Functions

```
>>> def sqr(x):
...     return x*x
...
>>> sqr(5)
25
>>> sqr(75)
5625
>>> sqr(3.14)
9.8596
>>> sqr('notanumber')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in sqr
TypeError: can't multiply sequence by non-int of type 'str'
```

1. You have to indent the lines of the function
2. When typing interactively, CNTL-D escapes

Nice trace for execution errors.

Defining a Recursive Function

```
>>> def factorial(n):
...     if n < 1:
...         return 0
...     if n == 1:
...         return 1
...     return n * factorial(n-1)
...
>>>
>>> factorial(3)
6
>>> factorial(10)
3628800
>>> factorial(-1)
0
```

Bad Version:

```
>>>def fact(n):
        if n==1:
            return 1
        else:
            return n * fact(n-1)
```

```
  File "<stdin>", line 5, in fact
  ...
  File "<stdin>", line 5, in fact
  File "<stdin>", line 5, in fact
RuntimeError: maximum recursion
                    depth exceeded
```

Scopes of Bindings:

In general, declare global variables to save worry, required if you change them.

Global y not needed here and we have two different z's.

```
>>> x = 5
>>> y = 6
>>> z = 7
>>> def fee(x):
...     z = x + y
...     return z
...
>>> r = fee(2)
>>> r
8
```

Global y used here to change y inside the function.

```
>>> def foo(x):
...     global y
...     z = x + y
...     y = y + 1
...   return z
...
>>> q = foo(2)
>>> q
8
>>> y
7
```

Lists

- We use lists heavily in AI.
- Lisp lists had two parts:
  - car (the head or first element of the list)
  - cdr (the tail or remainder of the list)
- Python is MUCH more versatile.
- Lists are like arrays in that you can refer to any element and yet you can also work with the head and tail and much more.

Lists

```
>>> mylist = ['a', 'b', 'c']
>>> mylist[0]                    car (or head)
'a‘
>>> mylist[1]
'b'
>>> mylist[1:]                   cdr (or tail)
['b', 'c']
>>> mylist[2:]
['c']
>>> mylist[-1]
'c‘
>>> mylist.insert(3,'d')         append
>>> mylist
['a', 'b', 'c', 'd']
```

How do you insert at the beginning?

Slices of Lists

```
>>> mylist
['a', 'b', 'c', 'd']
>>> len(mylist)
4
>>> mylist[0:len(mylist)]
['a', 'b', 'c', 'd']
>>> mylist[0:len(mylist):2]
['a', 'c']
>>> mylist[::-1]
['d', 'c', 'b', 'a']
>>> mylist[1:]
?
```

go through mylist by ones

go through mylist my twos

go through mylist in reverse

Iterating through Lists

```
>>> for e in mylist:
...     print('element is '+e)
...
element is a
element is b
element is c
element is d
```

```
>>> count = 0
>>> while count < len(mylist):
...     print(mylist[count])
...     count += 1
...
a
b
c
d
```

Strings

Strings work a lot like lists!

```
>>> mystring = 'abcd'
>>> mystring
'abcd'
>>> mystring[0]
'a'
>>> mystring[0:2]
'ab'
>>> mystring[-1]
'd'
>>> mystring[::-1]
'dcba'
```

Dictionaries

Dictionaries give us look-up table capabilities.

```
>>> translate = {}
>>> translate['I'] = 'Ich'
>>> translate['go'] = 'gehe'
>>> translate['to'] = 'zu'
>>> translate['doctor'] = 'Artz'
>>> translate['the'] = 'der'
>>> print(translate['I'])
Ich
```

How can we print the translation of
I go to the doctor?

Is it correct German?

Functional Programming

- Functions can be values that are assigned to variables or put in lists.
- They can be arguments to or returned by functions.
- They can be created dynamically at run time and applied to arguments.
- They don’t have to have names.
- This is like the lambda capability of Lisp

Example of Function Creation

```
>>> def make_adder(y):
...     return lambda x: x + y
...
>>> f4 = make_adder(4)
>>> f4(5)
9
>>> f7 = make_adder(7)
>>> f7(5)
12
```

What does this mean?

This is actually pretty tame. One can construct strings and make them into functions, too.

Object-Oriented Programming

Unlike Lisp, Python is an object-oriented language, so you can program much as you did in Java.

```
class Coord:
   "2D Point Coordinates"
   def __init__(self, x=0, y=0):
      self.x = x
      self.y = y
#
def describe(self):
    return '('+str(self.x)+','+str(self.y)+')'
#
def euclid(self,p2):
    return ((self.x-p2.x)**2+(self.y-p2.y)**2)**0.5
```

Using the Coord Object

```
>>> p1 = Coord(3,5)
>>> p2 = Coord(2,7)
>>> p1.describe()
'(3,5)'
>>> p2.describe()
'(2,7)'
>>> p1.euclid(p2)
2.23606797749979
>>> p2.euclid(p1)
2.23606797749979
```

Writing Methods

```
class Coord:
  "2D Point Coordinates"
  def __init__(self, x=0, y=0):
     self.x = x
     self.y = y
```

Write a method to add together two points and return a new point p3 = the sum of them

```
def add(self, p2):
```