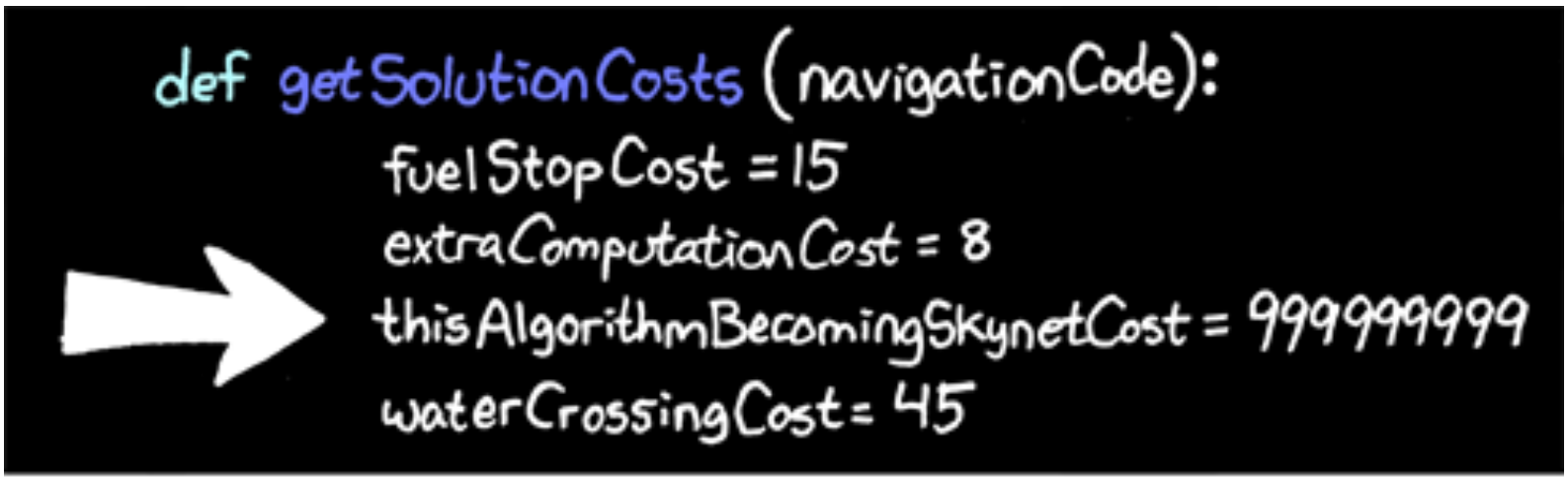
CS3101.3

Lecture 2



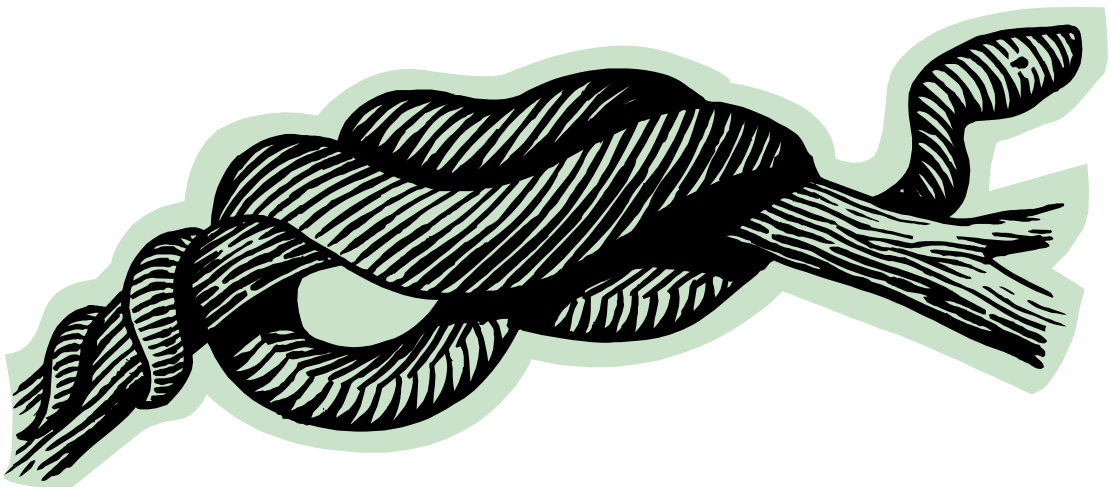
GENETIC ALGORITHMS TIP:
ALWAYS INCLUDE THIS IN YOUR FITNESS FUNCTION

Source: http://xkcd.com/534/

Agenda

- Homework 2
- Jumping into a few practical examples
- Finish dictionaries
- Sorting, Custom Comparators
- Parsing a CSV files to produce a webpage
  - In class exercise (also your hw)
- Functions
- Command line arguments

Questions? Homework, general?



Exercise 1: Producing a webpage from a CSV file

- In this exercise your goal is to create an html page by opening and parsing the Simpsons CSV file:
  - http://www1.cs.columbia.edu/~joshua/cs3101spring09/code/simpsons_diet.csv
- Step 0. See the lecture slides for help!
- Step 1. You may either download this file manually and read it from disk, or open it directly using the URLLIB package.
- Step 2. Parse simpsons_diet.csv either manually by splitting or using the CSV package.
- Step 3. Open a new file for writing "simpsons.html"
- Step 4. In any way you choose, parse the simpsons_diet.csv file and format it with html. This doesn't have to be fancy, just show me you understand how to access the fields of the csv file.

```
Homer Simpson,Dinner,Broccoli,2,So long cruel world!
Lisa Simpson,Lunch,Kale,2,Einstein's favorite
Lisa Simpson,Dinner,Tofu,4,Animals are cute!
Klang,Snack,Humans,40, *how*to*cook*for*forty?*humans*
Montgomery Burns,Snack, Life Extension Elixer,1,Excellent...
```

```
<html><body>
Montgomery Burns had a
 <b> Life Extension Elixer</b>
for a <b>Snack</b>
...
```

This weeks extra credit

For a **very large** triangle in the form:

40
73 11
52 10 40
26 53 06 34
10 51 87 86 90

Compute the maximum path value beginning at the top, and proceeding by single vertical or adjacent steps to the bottom (i.e., the path in blue).

http://www1.cs.columbia.edu/~joshua/cs3101spring09/code/triangle.txt

Exercise 2

- Briefly implement two strategies to tackle the extra credit problem
- Write a script which takes a command line parameter indicating the chosen strategy
- Output the maximum path value computed with that approach
- For this exercise you will be graded only on the correctness of your program, not whether your strategies actually find the optimal answer

Sequence basics

- Sequence concatenation
  - sequences of the same type can be concatenated with the (+) operator
    - note type references to the sequence itself (i.e., a list or a tuple) – NOT the contents
  - likewise (less commonly) you can assemble multiple copies using (*)
- Membership testing
  - Use “x in sequence” – returns True / False
  - More general in the case of strings, checking whether “x” is any substring of sequence

Subsequences and Slicing

- Given a sequence, a subsequence may be extracted via slicing
- Form: s[x:y] where s is a sequence, x is the lower bound (inclusive) and y is the upper bound (exclusive)
- if x is less than y the extract will be empty, if y is greater than len(s) it will be the upper bound
- x can be omitted if it's 0, i.e. s[:5]
- y can be omitted, in which case it will be the upper bound

More slicing syntax

- Slicing can also use the syntax s[x:y:n], where n is the stride (i.e., step) of the slice
- the same as saying n is the positional difference between successive elements
- think s[x:y] == s[x:y:1]
- think s[x:y:2] give me every second element in the range
- consider s[::3] - every third element in the sequence
- indices may also be negative - s[::-1] reverses a list

Sorting sequences

- A sort method on a list causes elements to be sorted in place
- General syntax
  - L.sort(cmp=cmp, key=None, reverse=False)
    - cmp is a pairwise comparison function, we'll see how to write those. cmp is the default
    - these return -1,0,1
    - key is None by default
  - Search the web for Python TIMSORT if for the technical details
- In general, passing arguments to sort slow down the process, later lectures will cover the decorate, sort, undecorate pattern
-

Quick word on sets

- Useful containers - word on program consistency
- Methods
  - Non-mutating
    - (set).intersection(another set)
    - (set).issubset..
    - (set).issuperset...
  - Mutating
    - (set).add
    - (set).clear
- Operators are overloaded for the non-mutating methods: e.g., Set1&Set2 for instance is intersection_update
- Careful...

Dictionaries

- Dictionaries are containers, so built in functions like len() will work (returning the number of key value pairs)
- Dictionaries are iterable (but only the keys by default, an in arbitrary order)
  - min(myDictionary)
  - max(myDictionary)
  - returns the smallest and largest keys
- Membership:
  - if key in dictionary
  - can also use has_key but the above is better
- Accessing: d['x'] returns a value or KeyError
- Assignment d['x'] = 5 creates or reassigns

Major dictionary methods

- If d is a dictionary:
- d.keys() – returns a list of the keys in arbitrary order
- d.values() – returns a list of the values in arbitrary order
- d.items() – returns a list of key / value pair tuples
- d.iter(iterms / keys / values)
- d.clear()
- d.popitem() – removes and returns an arbitrary item
- d.pop(key)
- d.setdefault(k[,x]) – returns d[k] if the key exists, otherwise sets d[k] equal to x and returns x

List comprehensions

- Common use of the for loop:
  - inspect each item in an iterable
  - build a new list by appending the result of an expression
  - general syntax [expression for target in iterable clauses]
  - result = []
    - for item in sequence:
      - result.append(item + 5)
  - result = [item + 5 for item in sequence]
  - result = [item + 5 for item in sequence if item > 3]
  - result = [x+y for x in seq1 for y in seq2]
- Notes
  - list comprehensions are expressions rather than a block of statements - so you use it almost anywhere
  - performance is high due to the underlying implementation
  - elegance - but NEVER sacrifice readability
  - general rule: sort is ok, long and complex skip it

Functions

- In Python a function is a group of statements that execute upon calling
- Functions are objects that are handled like anything else, i.e.,
  - they may be passed as arguments to other functions
  - they may be assigned dynamically at runtime
  - they may return other functions as their result
  - they may be keys in a dictionary or items in a sequence *mapping trick*

Basic examples

Sorting and Comparisons

- sorted() vs. list.sort()
  - sorted is general
  - returns a new sorted list from any iteratable
- list.sort() is a list only method
  - alters the list in place
- Standard comparisons (==) understands numbers, strings, etc
- What if you need to do a special comparison?

```
print sorted([5, 2, 3, 1,
   4])
[1, 2, 3, 4, 5]

print sorted(['b','a', 'c'])
['a', 'b', 'c']

print sorted([[2,2],
   [1,2]])
[[1, 2], [2, 2]]
```

Custom Comparisons

- cmp() is the built-in function that compares two objects, x and y
  - returns a negative number if x < y
  - returns 0 if they're equal
  - returns a positive number if x > y
- You can define your own arbitrary comparison function

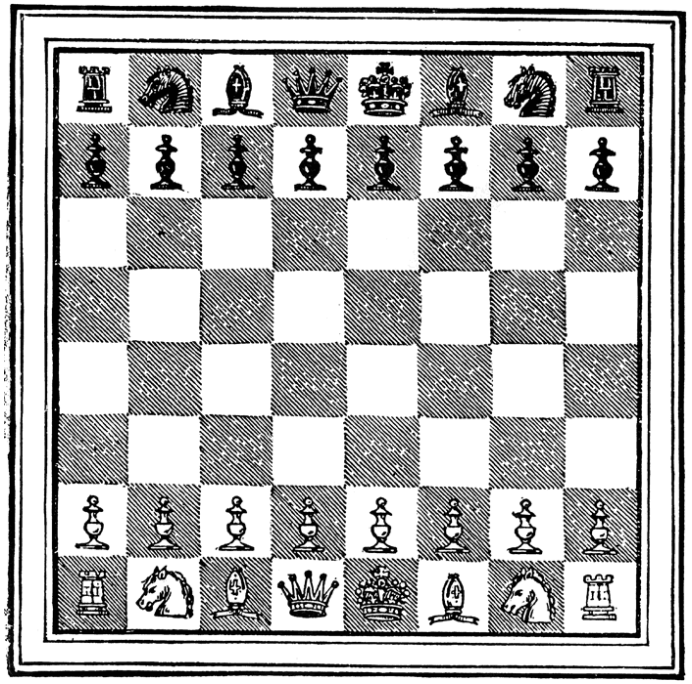
```
def food_compare(x, y):
    if 'donut' in x:
        return 1
    elif x == y: #use standard
     comparison
        return 0
    else:
        return -1

print sorted(['broccoli', 'duffbeer',
    'donut'])
['broccoli', 'donut', 'duffbeer']

print sorted(['broccoli', 'duffbeer', \
                'donut'], food_compare)
['duffbeer', 'broccoli', 'donut']
```

Using Dictionaries as Sparse Data Structures

```
def main():
    board = {}
    # a chess board
    # say we only want to
    keep track of the
    # position of the kings
    board[(0, 4)] =
    "LIGHT_KING"
```



Discussion:

- Can you compare the memory usage of this representation against a matrix using lists? (Chess boards are 8x8)
- Ignore the size of the dictionary and list support code (becomes insignificant at large sizes)

Dictionaries – methods

- Keys and values, testing for a key

```
simpsons = {'bart' : 7, 'marge' : '41', 'homer' : '42', 'lisa' : 6}

print simpsons.keys()
['homer', 'lisa', 'bart', 'marge']

print simpsons.values()
['42', 6, 7, '41']

print simpsons.items()
[('homer', '42'), ('lisa', 6), ('bart', 7), ('marge', '41')]

simpsons['barney']
KeyError: 'barney'

if 'barney' in simpsons:
    ...
```

Using Dictionaries as Records

```
import random
def main():
    homer[‘donutSupply’] = 5
    homer['hairsRemaining‘] = 4
    homer['noises'] = [‘Munch’, ‘Crunch’, ‘MMM Donut’, ‘Aghggh’]
    while homer['donutSupply'] > 0:
        homer['donutSupply'] -= 1
        print random.choice(homer['noises'])
main()
→
MMM Donut
Aghggh
```

Discussion:

Any questions on dictionaries before we move on?

What if you had to sort a

- Dictionaries cannot be directly sorted – a mapping has no order

```
simpsons = {'bart' : 7, 'marge' : '41', 'homer' : '42', 'lisa' : 6}

print sorted(simpsons) #sorts by keys
['bart', 'homer', 'lisa', 'marge']

from operator import itemgetter
print sorted(simpsons.iteritems(), key=itemgetter(1),
    reverse=True)
[('homer', '42'), ('marge', '41'), ('bart', 7), ('lisa', 6)]

#if you just want the keys (sorted by value)
x = sorted(simpsons.iteritems(), key=itemgetter(1),
    reverse=True)
print [obj[0] for obj in x]
```

Tonight's theme: Files and

- Goal:
  - Produce a webpage from this CSV file
  - In class exercise shortly

```
Character,Meal,Ate,Quantity,Comment
Barney Gumble,Breakfast,Duff Beer,1,I could've gone to Harvard
Duffman,Breakfast,Power bar,4,Duffman - can't breathe!
Duffman,Lunch,Protein shake,5,Duffman - has the power!
Homer Simpson,Snack,Donut - Jelly,1,Mmm Donut
Homer Simpson,Snack,Donut - Cream,1,Mmm Donut
Homer Simpson,Snack,Donut - Strawberry,1,Mmm Donut
Homer Simpson,Dinner,Brocolli,2,So long cruel world
Lisa Simpson,Breakfast,Oranges,1,Satisfying
Lisa Simpson,Lunch,Kale,2,Einstein's favorite
Lisa Simpson,Dinner,Tofu,4,Animals are cute!
Klang,Snack,Humans,40, *how*to*cook*for*forty? *humans*
Montgomery Burns,Snack, Life Extension Elixer, 1,Excellent
```

Discussion: advanced file functionality (seeking, etc) is outside our scope

File I/O

- Say (for some inexplicable reason) you needed to parse this CSV file to extract Klang's comment
  - First, let's get that line…

```
#read the entire file into a list, one line per entry
lines = open(r"c:\simpsons_diet.csv").readlines()
#for each line / entry in the list
for line in lines:
    if line.startswith("Klang"):
        print line
```

```
…
Lisa Simpson,Dinner,Tofu,4,Animals are cute!
Klang,Snack,Humans,40, *how*to*cook*for*forty?*humans*
Montgomery Burns,Snack, Life Extension Elixer,1,Excellent…
…
```

Discussion: read only is the default behavior

Basic string manipulation

Klang,Snack,Humans,40, *how*to*cook*for*forty?

```
line = "Klang,Snack,Humans,40," \
       "*how*to*cook*for*forty?*humans*"
columns = line.split(",")
print columns
['Klang', 'Snack', 'Humans', '40', '*how*to*cook*for*forty?
   *humans*']
comment = columns [-1]
print comment
*how*to*cook*for*forty?*humans*
cleaned = comment.replace("*", " ").strip()
print cleaned [0].upper() + cleaned [1:] + "."
How to cook for forty? humans.
```

Basic File I/O – writing

- Now let's write that to a file

```
cleaned = "How to cook for forty? Humans."
output = open(r"c:\klang.txt", 'w')
output.write(cleaned + "\n") #new line
# manually close the file – this happens
# automatically when
# the file object is garbage collected
output.close()
```

Basic File I/O – writing

- Say we wanted to write one word per line

cleaned = “How to cook for forty? Humans.”

```
output = open(r"c:\klang.txt", 'w')
words = cleaned.split()
output.writelines("\n".join(columns))
output.close()
klang.txt ->
how
to
cook
…
```

Discussion: note the ‘w’, the “ “.join, and the default behavior of the split() statement

Join statements

Using iteration

```
for line in lines:
    f.write(line + “\n”)
```

Using join

```
f.write(“ “.join(lines))
```

Quick examples

```
foo = ['homer', 'lisa', 'bart']
print foo
['homer', 'lisa', 'bart']

print " ".join(foo)
homer lisa bart

x = “ loves “
print x.join(foo)
homer loves lisa loves bart
```

Let's try reading with the CSV library

```
import csv
reader = csv.reader(open(r'c:
   \simpsons_diet.csv'), \
            delimiter=',', quotechar='"')
for row in reader:
```

```
…
[Barney Gumble,Breakfast,Duff Beer,1,I could've gone to Harvard]
[Duffman,Breakfast,Power bar,4,Duffman - can't breathe!]
[Duffman,Lunch,Protein shake,5,Duffman - has the power!]
…
```

I/O – Reading directly into variables

```
import csv
reader = csv.reader(open(r'c:
   \simpsons_diet.csv'), \
            delimiter=',', quotechar='"')
for char, meal, ate, quantity, comment in
   reader:
```

```
...
Lisa Simpson had 4 Tofu for Dinner
Klang had 40 Humans for Snack
Montgomery Burns had 1  Life Extension Elixer for
Snack
```

I/O – and now writing with

The **writer** function creates a CSV writer object, which converts values to strings and escapes them properly.

```
import csv
reader = csv.reader(open(r'c:\simpsons_diet.csv'), \
        delimiter=',', quotechar='"')
out  = open(r'c:\simpsons_new.csv', "wb")
writer = csv.writer(out, delimiter=',', quotechar='"')
for row in reader:
    writer.writerow(row)
new = ['Chronos', 'Snack', 'Klang', '1', 'The humans made
    him tasty!']
writer.writerow(new)
```

```
...
Klang,Snack,Humans,40, *how*to*cook*for*forty?*humans*
Montgomery Burns,Snack, Life Extension Elixer,1,Excellent...
Chronos, Snack, Klang, The humans made him tasty!
```

How about reading that CSV file from a network?

```
import urllib, csv

#grab our csv file from the network
url = r"http://www.cs.columbia.edu/~joshua/”
url += r“cs3101spring09/code/
   simpsons_diet.csv"
simpsons = urllib.urlopen(url)

reader = csv.reader(simpsons, delimiter=',',
   quotechar='"')
for char, meal, ate, quantity, comment in
```

In class exercise (also one of your hw problems)

- Say we needed to produce a webpage from this CSV log file.
- How would you go about it?

Goal:

```
<html><body>
<h1>Homer Simpson</
h1><p>
Ate <b>2</b> pieces of
Broccoli. <p>
So long cruel world!<p>…
```

```
Homer Simpson,Dinner,Broccoli,2,So long cruel world!
Lisa Simpson,Lunch,Kale,2,Einstein's favorite
Lisa Simpson,Dinner,Tofu,4,Animals are cute!
Klang,Snack,Humans,40, *how*to*cook*for*forty?*humans*
Montgomery Burns,Snack, Life Extension Elixer,1,Excellent…
```

Discussion: Which methods and data structures might be employed?

In class exercise (also one of your hw problems)

- We'll need
  - to read the CSV file
  - to open an output file
  - to parse the lines of the CSV and format with HTML

Goal:

```
<html><body>
<h1>Homer Simpson</
h1><p>
Ate <b>2</b> pieces of
Broccoli. <p>
So long cruel world!<p>…
```

```
Homer Simpson,Dinner,Broccoli,2,So long cruel world!
Lisa Simpson,Lunch,Kale,2,Einstein's favorite
Lisa Simpson,Dinner,Tofu,4,Animals are cute!
Klang,Snack,Humans,40, *how*to*cook*for*forty?*humans*
Montgomery Burns,Snack, Life Extension Elixer,1,Excellent…
```

Reference Code

```
#read the entire file into a list, one
    line per entry
lines = \
open(r"c:
    \simpsons_diet.csv").readlnes()
#for each line / entry in the list
for line in lines:
    if line.startswith("Klang"):
            print line
```

```
output = open(r"c:\klang.txt", 'w')
output.write(cleaned + "\n") #new
    line
# manually close the file - this
    happens
# automatically when
# the file object is garbage
    collected
        close
```

```
line = "Klang,Snack,Humans,40," \
        "*how*to*cook*for*forty?
    *humans*"
columns = line.split(",")
print columns
['Klang', 'Snack', 'Humans', '40',
    '*how*to*cook*for*forty?
    *humans*']
comment = columns [-1]
print comment
*how*to*cook*for*forty?*humans*
cleaned = comment.replace("*", "
    ").strip()
print cleaned [0].upper() + cleaned
    [1:] + "."
```

Discussion: Take 10 minutes and try to sketch it out – I'll come around and help out

Assorted utility functions

Syntax	Semantics
import os.path	Import the OS module
os.path.exists("bob.txt")	Exists test
os.path.isfile(r"c:\bob.txt")	File test
os.path.isdir("bob")	Directory test
os.curdir	Current directory
os.path.join(currentdir, "images")	System independent path
os.path.walk(rootdir,f,arg)	Function to recursively parse directories

Discussion: note the 'r' preceding c:\bob.txt – raw string support to avoid escaping \'s

Functions

- Before we go in to functions, keep in mind:
  - Pythons libraries are HUGE
  - Look before writing a function to perform a common task

Libraries Preview

http://docs.python.org/library/

- Out of the box:
  - Numeric and mathematical modules
  - Files and directory handling
  - Data persistence
  - Data compression and archiving
  - **File formats (csv, etc)**
  - Cryptographic services
  - Operating system hooks
  - Interprocess communication and threading
  - Internet data handling
  - Structured markup handling (html, xml, etc)
  - **Internet protocols (you name it)**
  - Multimedia services (presentation, manipulation)
  - Internationalization
  - GUIs
  - Debugging, profiling
  - Windows, Mac, *nix, Sun specific services

Functions 101

- Keep in mind
  - Dynamic typing and polymorphism
  - Mutable vs. Immutable parameters vs. call by reference / value

- Maximize
  - Leverage preexisting code
  - Your own code reuse
  - Procedural decomposition
  - Documentation
- Minimize
  - Redundancy
  - Errors

Discussion: Review: dynamic typing, polymorphism, by reference / by value

Functions – Polymorphism

```
def multiply(x, y):
    return x * y

print multiply (2,4)
8

print multiply (.5,2)
1.0

print multiply("foo", 3)
foofoofoo
```

```
def intersection(groupA, groupB):
    result = []
    for obj in groupA:
        if obj in groupB:
            result.append(obj)
    return result

print intersection([1,2,3], (1,3,4))
[1, 3]


foo = {"homer" : "donuts", "lisa" :
    "kale"}
bar = ["homer", "kale"]
print intersection(foo, bar)
['homer']
```

Discussion: Notice the list and tuples?

Rewriting intersection with a list comprehension

```
def intersection(groupA, groupB):
    result = []
    for obj in groupA:
        if obj in groupB:
            result.append(obj)
    return result
```

Discussion: How can we make this more elegant?

Rewriting intersection with a list comprehension

```
def intersection(groupA, groupB):
    result = []
    for obj in groupA:
        if obj in groupB:
            result.append(obj)
    return result
```

```
def intersection(groupA, groupB):
    return [obj for obj in groupA if obj in
```

Python has built in sets

```
simpsons = set(["homer", "marge", "bart", "lisa"])
philosophers = set(["aristotle", "sophocles", "homer"])


print simpsons.intersection(philosophers)
set(['homer'])

print simpsons.union(philosophers)
set(['homer', 'marge', 'aristotle', 'lisa', \
      'sophocles', 'bart'])

print set(['homer']).issubset(philosophers)
```

Discussion: Other set operators: union, add, issuperset, issubset, etc

def is a statement

Legal in Python

```
import random

if random.randint(0,1):
    def func(x):
        return x + 1
else:
    def func(x):
        return x -1

print func(1)

0
2
...
```

Why is this legal?

- Python statements are executable – we have runtime, but not compile time in the C sense
- Functions are objects like everything else in Python
- Assignment is legal as well
  - foo = func
  - print foo(1)

Discussion: randint(0,1) --> 0 evaluates to False, 1 to True

Functions – pass by reference

Mutable (lists, dictionaries, classes) by

```
def rev(x):
    x.reverse()

def addOne(x):
    x += 1

def main():
    myList = ["a", "b"]
    rev(myList)
    print myList

main()
['b', 'a']
```

Immutable (numbers, strings) by value

```
def rev(x):
    x.reverse()

def addOne(x):
    x += 1

def main():
    x = 5
    addOne(x)
    print x

main()
5
```

Discussion: Similarity to Java

Recursion

```
# returns the factorial of x
def fact(x): return (1 if x == 0 else x *
fact(x-1))
```

```
# equivalently...
def fact2(x):
    if x == 0:
        return 1
    else:
        return x *
  fact2(x-1)
```

```
# an iterative
  solution
def fact3(x):
    result = 1
    while (x > 1):
     result = result * x
        x = x - 1
    return result
```

Discussion: Question from last week: is vs. ==. Is (is) identity function, == (is) equality

Scoping: namespace resolution

- What's a namespace?
  - Defines a variables scope, the area in which it carries meaning
- Static, lexical scope
  - Scope is a function of program text
- General rules
  - Namespace binding happens as variables are declared
  - Names defined inside a def are visible only within that def, and do not clash with outsiders
  - Each call to a function creates a new local scope
  - You can explicitly access global variables with the global statement, but this is rare

Scoping: namespace

- Global scope:
  - Each module is namespace in which variables can be created
  - These are visible to the outside world
  - Global scope is limited in span to a single file only
- Variable resolution:
  - 1. Local (function)
  - 2. Enclosing functions
  - 3. Global (module)
  - 4. Built-in (python)

```
import math
pi = 3.14
print math.pi
3.14159265359
print pi
3.14
```

```
open = 5
print open
5
open("foo.txt") #???
TypeError: 'int' object is not
    callable
```

```
# global scope
x = 8
def foo(y):
    # local scope
    z = x + y
    return z

print  foo(2)
Global names: x, fn
Local names: y, z
```

Discussion: Careful! Local scope wins: you can override built in fns

Command line arguments

- Each argument passed to the program ends up in sys.argv, which is just a list
- The first item is always the name of the module
- Arguments are separated by spaces

```
#mutiply.py
import sys
for arg in sys.argv:
    print arg
x = sys.argv[1]
y = sys.argv[2]
print x * y

$ python mutiply.py 5 2
10
```

Quick example

```
#getSyllables.py
import sys
def main():
    if len(sys.argv) != 2:
        print 'Invalid Input'
        print 'Usage: python getSyllables.py
   yourWord'
    else:
        word = sys.argv[1]
```

Exercise 1: Producing a webpage from a CSV file

- In this exercise your goal is to create an html page by opening and parsing the Simpsons CSV file:
  - http://www1.cs.columbia.edu/~joshua/cs3101spring09/code/simpsons_diet.csv
- Step 0. See the lecture slides for help!
- Step 1. You may either download this file manually and read it from disk, or open it directly using the URLLIB package.
- Step 2. Parse simpsons_diet.csv either manually by splitting or using the CSV package.
- Step 3. Open a new file for writing "simpsons.html"
- Step 4. In any way you choose, parse the simpsons_diet.csv file and format it with html. This doesn't have to be fancy, just show me you understand how to access the fields of the csv file.

```
Homer Simpson,Dinner,Broccoli,2,So long cruel world!
Lisa Simpson,Lunch,Kale,2,Einstein's favorite
Lisa Simpson,Dinner,Tofu,4,Animals are cute!
Klang,Snack,Humans,40, *how*to*cook*for*forty?*humans*
Montgomery Burns,Snack, Life Extension Elixer,1,Excellent...
```

```
<html><body>
Montgomery Burns had a
 <b> Life Extension Elixer</b>
for a <b>Snack</b>
...
```

This weeks extra credit

For a **very large** triangle in the form:

40
73 11
52 10 40
26 53 06 34
10 51 87 86 90

Compute the maximum path value beginning at the top, and proceeding by single vertical or adjacent steps to the bottom (i.e., the path in blue).

http://www1.cs.columbia.edu/~joshua/cs3101spring09/code/triangle.txt

Exercise 2

- Briefly implement two strategies to tackle the extra credit problem
- Write a script which takes a command line parameter indicating the chosen strategy
- Output the maximum path value computed with that approach
- For this exercise you will be graded only on the correctness of your program, not whether your strategies actually find the optimal answer