CS 5 Black Meets Artificial Intelligence!

Just to be clear, I represent the intelligence!

Please bring this note set back to class next time.
Stuff

• True story coming in a few minutes…
• Some words about good software design...
  – Design your software “top down”
  – Design on paper starting at the top and working your way down the “design tree”
  – Write signatures for all functions before you implement
  – Implement and test “bottom up”
CS 5 Black Exam 1

• Thursday, November 2, in class
• Sample exam posted with Homework 8
• You can bring an 8.5x11 sheet with writing front-and-back
• Big ideas
  – Recursion
  – Care packages
  – Memoization
  – Higher-order functions, lambda, map, reduce, filter
  – Choosing and using the right kind of loop
  – Dictionaries and trees (this week!)
• Details syntax important not!
A short aside on Dictionaries...

```python
>>> D = {}  # initialize empty dictionary
>>> D["spam"] = 42
>>> D["spam"]
42
>>> D["cat"] = "a very clever animal"
>>> D["cat"]
"a very clever animal"
>>> list(D.keys())
["cat", "spam"]
>>> "cat" in D
True
>>> D
{ "cat" : "a very clever animal",
   "spam": 42
}
```
Keys must be immutable...

- Strings, numbers, tuples are OK
- Lists are not!

```python
>>> D = {}
>>> D[('I', 'like')] = ['spam', 'pizza', 'spam', 'cake', 'spam']
```

Key type must be immutable  
Value type can be anything!

Look! I’m mutable! (Cats are so clever.)
Tuples of size 1

(“hello”)

(“hello”,)

D[(“hello”)] == D[“hello”]
D[(“hello”, )] = [“there”, “to”]
Reverse Translation

TtoE = {"dude!": "hello", "later": "goodbye", "stoked": "happy", "ciao": "goodbye"}

>>> EtoT = reverse(TtoE)

>>> EtoT

{'hello': 'dude!', 'goodbye': 'ciao or later', 'happy': 'stoked'}

>>> EtoT['goodbye']

'ciao or later'

def reverse(inputD):
    outputD = {}
    for key in inputD.keys():
Reverse Translation

```python
def reverse(inputD):
    outputD = {}
    for key in inputD.keys():
        value = inputD[key]
        if value in outputD:
            outputD[value] = outputD[value] + " or " + key
        else:
            outputD[value] = key
    return outputD
```
Markov text generation (WMSCI 2005)

Dan Aguayo
Max Krohn
Jeremy Stribling

Demo!
We've consulted with very clear syntax. This chapter to major computational ideas that you a computer scientist is to experiment! It will tell you could do any of biology. Simply put, computing is 2. Consider, for phylogenetics the use of finding the url. The best way that Python will tell you to learn to be infectious.
To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take Arms against a Sea of troubles,
And by opposing end them: to die, to sleep
No more; and by a sleep, to say we end...

Dictionary = {

("in," ) : [ "the", "thy", "thy" ],

(" the,") : [ "heartache", "rub", "pangs", ... ]

}

Markov model
(a dictionary)
How it works...
(k = 2 in this example)

```python
>>> test = "A B C. A B B! D A A?"

>>> wordList = test.split()

>>> wordList
['A', 'B', 'C.', 'A', 'B', 'B!', 'D', 'A', 'A?']

>>> dollarify(wordList, 2)
```
test = "A B C. A B B! D A A?"

wordList =

mm2 = markov_model(wordList, 2)
{
  ('D', 'A'): ['A?'],
  ('$', 'D'): ['A'],
  ('A', 'B'): ['C.', 'B!'],
  ('$', 'A'): ['B', 'B'],
  ('$', '$'): ['A', 'A', 'D'],
  ('$', '$'): ['A', 'A', 'D']
}
This slide for your work...
What you’ll submit...

• Your code, as usual, on Gradescope
• A .txt file (e.g., RanH.txt) that you share with the class at...

https://tinyurl.com/y9nx68na

DEMO!
“I wonder about Trees” – Robert Frost

“We wonder about Robert Frost” - Trees
Phylogenetic Trees...

From Darwin’s notebooks, 1837
Really?
From the 6th edition (1872): “The affinities of all the beings of the same class have sometimes been represented by a great tree. I believe this simile largely speaks the truth. The green and budding twigs may represent existing species; and those produced during former years may represent the long succession of extinct species.”
Trees and human evolution

- Orangutan
- Gorilla
- Human
- Common chimpanzee
- Bonobo

- ~15 MYA
- ~7 MYA
- ~6 MYA
- ~3 MYA

Trees and human evolution
Phylogenetic Trees

What about the Sontag, South, East, and Atwood Groodies?

These are called internal nodes

These are called leaf nodes or leaves or tips of the tree.

How do we represent this in Python?
A tree is either:
• None ← This is a built-in value in Python
• (Root, Left, Right)

(`"X",
 ("Y", ("W", None, None),
  ("Z", ("E", None, None), ("L", None, None))),
 ("C", None, None))`)
How many nodes are in this tree?

```python
def nodeCount(tree):
    ''' Takes a tree as input and returns total number of nodes. '''
    if tree == None:
        return 0
    else:
        root, leftTree, rightTree = tree
        return 1 + nodeCount(leftTree) + nodeCount(rightTree)

>>> nodeCount(Groodies)
7
```

It would be a shame to "leaf" out the base case!
Is my favorite species in this tree?

```python
>>> find("E", Groodies)
True
>>> find("Sontag", Groodies)
False
```

```
def find(species, tree):
    
    if tree == None:
        return False
    else:
        root, leftTree, rightTree = tree
```
Is my favorite species in this tree?

```python
def find(species, tree):
    ''' Takes a tree and species as input and returns Tree iff species in Tree.'''
    if tree == None:
        return False
    else:
        root, leftTree, rightTree = tree
        if species == root: return True
        else: return find(species, leftTree) or find(species, rightTree)
```
Four problem worksheet bonanza!

```python
>>> height(Groodies)
4
>>> height(None)
0
>>> nodeList(Groodies)
['X', 'Y', 'W', 'Z', 'E', 'L', 'C']
>>> leafList(Groodies)
['W', 'E', 'L', 'C']
>>> mrca("L", "E", Groodies) # use find as a helper!
'Z'
>>> mrca("W", "E", Groodies)
'Y'
>>> mrca("W", "C", Groodies)
'X'
>>> mrca("W", "Prof. Ran", Groodies)
None
```

```python
>>> find("L", Groodies)
True
```
```python
def height(tree):
    ''' returns the height of the given tree. '''
    if tree == None:
        return 0
    else:
        root, leftTree, rightTree = tree
        return 1 + max(height(leftTree), height(rightTree))
```

>>> height(Groodies)
4
>>> nodeList(Groodies)
['X', 'Y', 'W', 'Z', 'E', 'L', 'C']

def nodeList(tree):
    ''' returns the list of all nodes in the tree. '''
    if tree == None:
        return []
    else:
        root, leftTree, rightTree = tree
        return [root] + nodeList(leftTree) + nodeList(rightTree)
>>> leafList(Groodies)
['W', 'E', 'L', 'C']

def leafList(tree):
    ''' returns the list of leaves in the tree. '''
    if tree == None:
        return []
    else:
        root, leftTree, rightTree = tree
        if leftTree == None and rightTree == None:
            return [root]
        else:
            return leafList(leftTree) + leafList(rightTree)
def mrca(species1, species2, tree):
    ''' returns the most recent common ancestor of the two
given species in the tree, and None if there isn't one.'''
    if tree == None:
        return None
    elif not find(species1, tree) or not find(species2, tree):
        return None
    else:
        root, leftTree, rightTree = tree
        if find(species1, leftTree) and find(species2, leftTree):
            return mrca(species1, species2, leftTree)
        elif find(species1, rightTree) and find(species2, rightTree):
            return mrca(species1, species2, rightTree)
        else:
            return root