stuff is {'color': 'black', 1: 2, 'cat': 100, (1, 1): 'yes', 1.5: 3}
N is for ...

• Nested Loops
  • All pairs, all pixels, all 2D structures

• None
  • Default value for functions if no return

• Newline
  • The "\n" in a line
The 21 Most Important Googlers You've Never Heard Of

Georges Harik and Noam Shazeer created the underlying data that led to AdSense

Harik and Shazeer spent years analyzing data on webpages, trying to understand clusters of words and how they worked together. The data they gather wound up being used by Google for its AdSense product, which analyzed webpages for words, and then stuck ads on them.
Announcements

• Assign 3 Transform due Today!
• Assign 4 is out today, due Thursday, March 23
• APT 4 due next Thursday, March 9
• Lab 7 tomorrow, do prelab 7 before going
  • Videos of Labs 0-6 in Sakai Resources folder
• Do not discuss APT Quiz 1 with anyone until they are handed back
• Exam 2 March 7
  • See notes from Tuesday
Exam 2 – in person – Tues, March 7

• Exam is in class on paper – 10:15am
  • Need pen or pencil
• See materials under 3/7 date
  • Exam 2 Reference sheet - part of exam
• Covers
  • topics /reading through today
  • APTs through APT4
    • APT4 – write code on paper, then type in
  • Labs through Lab 7
    • Lab 7 - Parts 1-3
  • Assignments through Assignment 3
Exam 2 topics include ...

- List, tuples, list comprehensions
- Loops – for loop, while loop, indexing with a loop
- Reading from a file
  - Converting data into a list of things
- Parallel lists
- Sets – solving problems
- Dictionaries – only reading them and understanding output, no problem solving
- No turtles on the exam!
Exam 2

• Exam 2 is your own work!
• No looking at others exam or talking to others
• You cannot use any notes, books, computing devices, calculators, or any extra paper
• Bring only a pen or pencil
• The exam has extra white space and has the Exam 2 reference sheet as part of the exam.

• Do not discuss any problems on the exam with others until it is handed back
Exam 2 – How to Study

• Practice writing code on paper!
• Rewrite an APT
• Try to write code from lecture from scratch
• Try to write code from lab from scratch
• Practice from old exams
• Put up old Sakai quizzes, but better to practice writing code
• Look at Exam 2 reference sheet when writing code!
PFTD

• Solving an APT
• Dictionaries
• Solving Problems with Dictionaries
• Practice Exam Problem
Problem Statement

We want to know how many different people have eaten at a restaurant this past week. The parameter meals has strings in the format "name:restaurant" for a period of time. Sometimes a person eats at the same restaurant often.

Return the number of different people who have eaten at the eating establishment specified by parameter restaurant.

For example, "John Doe:Moes" shows that John Doe ate one meal at Moes.

Write function `howMany` that given meals, a list of strings in the format above indicating where each person ate a meal, and restaurant, the name of a restaurant, returns the number of people that ate at least one meal at that restaurant.
APT Eating Good Example

```python
meals = ["Sue:Elmos", "Sue:Elmos", "Sue:Elmos"]

restaurant = "Elmos"

returns 1
```
WOTO-1: APT Eating Good
APT Eating Code Idea

• Make an empty list
• Loop over each meal
  • Split the meal into person and restaurant
  • If the restaurant matches
    • If person not already in list
      – Add person to the list
  • Return the length of the list
APT Eating Code Idea

• We need to count what?
  • Number of names that ate at specific restaurant
  • Unique names

• How do we do that?
  • Loop over the meals
  • Keep track of all the names that ate at that restaurant
    • Build a list of unique names.

Accumulator pattern!
  Initialize empty list
  Build list inside loop
APT Eating Code Algorithm
APT Eating Code Algorithm

• Make an empty list
• Loop over each meal
  • Split meal into name and restaurant
  • If the restaurant matches
    • If name not already in list
      – Add name to the list
• Return the length of the list
Code with Eating Good APT (w/list)

```python
def howMany(meals, restaurant):
    # make an empty list
    names = []
    # loop over meals
    for meal in meals:
        # split meal into name and restaurant
        data = meal.split('::')
        (name, rest) = (data[0], data[1])
        # if the restaurant matches
        if rest == restaurant:
            # If name not already in list
            if name not in names:
                # add name to the list
                names.append(name)
    # return length of names
    return len(names)
```
APT Eating Code Idea With List

• Make an empty list

• Loop over each meal
  • Split meal into name and restaurant
  • If the restaurant matches
    • If name not already in list
      – Add name to the list

• Return the length of the list
APT Eating Code – **Use set instead of list**

- **Make an empty list**
  - names = set()

- **Loop over each meal**
  - Split meal into name and restaurant
  - If the restaurant matches
    - If name not already in list
      - Add name to the list
  - Return the length of the list
    - return len(names)

Don't need this IF with sets

```python
names = set()
for meal in meals:
    name, restaurant = meal
    if restaurant == restaurant_to_match:
        if name not in names:
            names.add(name)
return len(names)
```
APT Eating Code – **Use set instead of list**

1. Make an **empty set**
   ```python
   names = set()
   ```

2. Loop over each meal
   - Split the meal into name and restaurant
   - If the restaurant matches
     - Add name to set
     ```python
     names.add(name)
     ```
   
3. Return the length of the **set**
   ```python
   return len(names)
   ```
Lists or Set?

```python
if name not in names:
    names.append(name)
```

- For EatingGood, with a list, we had to avoid adding the same element more than once
  - Lists store duplicates
  - Sets do not store duplicates, didn't need the check
Problem: Given a name, what is their favorite ice cream?

• Assume you have a lot of students

• How is the data stored?

• Assume we have parallel lists
  • students is list of names
  • icecream is list of corresponding favorite ice cream
Code might be

1 if name in students:
2   pos = students.index(name)  # find position of name
3   answer = icecream[pos]     # answer in same pos

If a billion names, this is not efficient
How does this code work?
Code might be

1  if name in students:
2       pos = students.index(name)  # find position of name
3       answer = icecream[pos]    # answer in same pos

If a billion names, this is not efficient
How does this code work?
   line 1 search through a billion names to say yes
   line 2 search through a billion names again!
   line 3 just one step access it!
How does search with `.index` work?

- **Parallel Lists**
  - Search for name first in students list
  - Use index location of name to find favorite ice cream

```
students = ['Astrachan', 'Sun', 'Rodger', 'Forbes']
    0       1       2       3
icecream = ['Chocolate', 'Chocolate Chip', 'Chocolate Chip', 'Strawberry']
     0           1           2           3
```
How does search with .index work?

• Parallel Lists
  • Search for name first in students list
  • Use index location of name to find favorite ice cream

Find Rodger’s favorite ice cream

students =
  0      1      2      3

icecream =
  0      1      2      3
How does search with `.index` work?

- **Parallel Lists**
  - Search for name first in students list
  - Use index location of name to find favorite ice cream

**Find Rodger’s favorite ice cream**

```python
students = ['Astrachan', 'Sun', 'Rodger', 'Forbes']

icecream = ['Chocolate', 'Chocolate Chip', 'Chocolate Chip', 'Strawberry']
```

```python
students
# Index of Rodger
0

// No

icecream
```
How does search with `.index` work?

- **Parallel Lists**
  - Search for name first in students list
  - Use index location of name to find favorite ice cream

**Find Rodger’s favorite ice cream**

students =

```python
['Astrachan', 'Sun', 'Rodger', 'Forbes']
```

icecream =

```python
['Chocolate', 'Chocolate Chip', 'Chocolate Chip', 'Strawberry']
```

3/2/23
How does search with .index work?

- Parallel Lists
  - Search for name first in students list
  - Use index location of name to find favorite ice cream

Find Rodger’s favorite ice cream

students =
['Astrachan', 'Sun', 'Rodger', 'Forbes']

    0       1       2       3

icecream =
['Chocolate', 'Chocolate Chip', 'Chocolate Chip', 'Strawberry']

    0       1       2       3

YES!
Use index location in other list

- Parallel Lists
  - Search for name first in students list
  - Use index location of name to find favorite ice cream

Find Rodger’s favorite ice cream

students =

0 1 2 3

icecream =

0 1 2 3

FOUND!
Use index location in other list

• Parallel Lists
  • Search for name first in students list
  • Use index location of name to find favorite ice cream

Find Rodger’s favorite ice cream

students =

['Astrachan', 'Sun', 'Rodger', 'Forbes']
0 1 2 3

icecream =

['Chocolate', 'Chocolate Chip', 'Chocolate Chip', 'Strawberry']
0 1 2 3

In same index position

FOUND!
Code was easy

- But for a lot of data could take a long time.

- Let’s see another way, dictionaries
How the Dictionary is made

• Using a dictionary is reasonably straight-forward
  • We will be clients, not implementers
  • Efficiency not a large concern in 101
  • Our goal is to just get stuff done 😊
• To drive a car, don't have to know how it works inside
What is a Dictionary?

• A collection of (key, value) pairs (abstract view)
  • Look up key, find the value

• For list
  • $a[3]$ takes same time as $a[3000]$  
  • Finding the item is slow  
  • Fast once you know the index

• For Dictionary: $d["cake"]$
  • Finding the value associated with "cake"
  • very, very fast
Dictionaries/Maps

• Dictionaries are another way of organizing data
• Dictionaries are sometimes called maps
• Keys and Values
  • Each key maps to a value
  • Some keys can map to the same value
  • Can change the value a key maps to
Example

• Each student could be mapped to their favorite ice cream flavor

```
Students
Astrachan
Sun
Rodger
Forbes

Ice Cream Flavors
Chocolate
Chocolate Chip
Strawberry
```
How is dictionary different than a list?

• List – have to search for name first
• Dictionary – each key maps to a value
• getting name (or key) is automatic! Fast!

![Diagram showing keys and values relationship in a dictionary]
Implementing a Dictionary/Map

Keys map to values

• **Create Empty dictionary**
  
somemap = {}

• **Put in a key and its value**
  
somemap["Forbes"] = "Strawberry"

• **Get a value for a dictionary**
  
value = somemap["Forbes"]

• **Change a value for a dictionary**
  
somemap["Forbes"] = "Chocolate"
Change Astrachan's value
somemap["Astrachan"] = Coffee Mocha

Students

Astrachan
Sun
Rodger
Forbes

Ice Cream Flavors

Chocolate
Chocolate Chip
Strawberry
Change Astrachan's value

somemap["Astrachan"] = Coffee Mocha

Students

- Astrachan
- Sun
- Rodger
- Forbes

Ice Cream Flavors

- Coffee Mocha
- Chocolate Chip
- Strawberry
Value could be a set or list
How to use a Dictionary

- Create: `d = {}`
  - `d = {'a': 10, 'b': 100}`
  - `d = dict([('a', 10), ('b', 100)])`
- Insert: `d[KEY] = VALUE`
- Update/Reassign: `d[KEY] = VALUE`
- Get a value (like list indexing): `d[KEY]`
- *Key* membership (not values): `KEY in d`
  - No membership check for values
Examples

```python
stuff = {}
print(stuff)
print(type(stuff))
stuff['color'] = 'black'
stuff[1] = 2
stuff['cat'] = 100
stuff[(1, 1)] = 'yes'
stuff[1.5] = 3
print(stuff)
```

OUTPUT

```
{...
  'color': 'black',
  1: 2,
  'cat': 100,
  (1, 1): 'yes',
  1.5: 3
...
} <class 'dict'>
```
Examples

```
stuff = {}
print(stuff)  # Output: {}
print(type(stuff))  # Output: <class 'dict'>
stuff['color'] = 'black'
stuff[1] = 2
stuff['cat'] = 100
stuff[(1,1)] = 'yes'
stuff[1.5] = 3
print(stuff)
# Output: {'color': 'black', 1: 2, 'cat': 100, (1, 1): 'yes', 1.5: 3}
```

Dictionaries are unordered.
Examples

stuff is {'color': 'black', 1: 2, 'cat': 100, (1, 1): 'yes', 1.5: 3}

print(len(stuff))
stuff[3] = [6, 3, 2]

stuff[[4,7]] = 'go'
Examples

stuff is {'color': 'black', 1: 2, 'cat': 100, (1, 1): 'yes', 1.5: 3, }

print(len(stuff)) 5
stuff[3] = [6, 3, 2]

stuff is {'color': 'black', 1: 2, 'cat': 100, (1, 1): 'yes', 1.5: 3, 3: [6, 3, 2]}

stuff[[4,7]] = 'go' ERROR!!!
Examples

d = {}

d["color"] = "black"

d["color"] = "red"

d["red"] = "color"

r = d[d["red"]]

r = d["monkey"]
Examples

d={}  
d['color'] = 'black'

d['color'] = 'red'

d['red'] = 'color'

r = d[d['red']]  
r = d['monkey']

ERROR!!!!!!!
Examples

d = {'a':'cat', 'e':'dog'}

'pig' in d
'a' in d
'dog' in d
Examples

d = {'a': 'cat', 'e': 'dog'}

'pig' in d  False
'a' in d    True
'dog' in d  False

in checks if 'dog' is a key in d
More on Dictionary

• Like lists, but with keys
• KEY – immutable type, unique within dictionary
• VALUE – any type, not unique within dictionary
• Dictionary is unordered collection of (KEY, VALUE) pairs
More on using a Dictionary/Map

• **Assume** `somemap` **is a dictionary**

• **Get all the keys (as a list)**
  
  • `listKeys = somemap.keys()`

• **Get all the values (as a list)**
  
  • `listValues = somemap.values()`

• **Other methods**
  
  • `clear` – empty dictionary
  
  • `items` – return (key,value) pairs
  
  • `update` – update with another dictionary
Examples

d = {'a':4, 'e': 3, 'b':4 }

v = d.values()
k = d.keys()
p = d.items()

for t in d.items():
    print(t)
Examples

d = {'a':4, 'e':3, 'b':4 }

v = d.values()  
v is [4, 3, 4]

k = d.keys()  
k is ['a','e','b']

p = d.items()  
p is [('a',4), ('e',3), ('b',4)]

for t in d.items():
    print(t)

    ('a', 4)
    ('e', 3)
    ('b', 4)
Problem

• Given a list of names of people who ate at a restaurant, who ate there the most?

• A name appears more than once if they ate there more than once

• Example input:

• `names = ['Sarah', 'Beth', 'Sarah', 'Purnima', 'Beth', 'Beth', 'Purnima']`
WOTO-3 Problem Solving
Counting Dictionary

d = {}

for word in names:
    if word not in d:
        d[word] = 1
    else:
        count = d[word]
        d[word] = count + 1

print("d:", d)
Finding Largest Value in d

```python
17 val = 0
18 for key in d:
19     if d[key] > val:
20         val = d[key]
21 print("val:", val)
```

Alternative:

```python
23 maxval = max(d.values())
24 print("maxval:", maxval)
```
Find key goes with largest value

```python
maxname = ""

for key in d:
    if d[key] == maxval:
        maxname = key

print("maxname", maxname)
```
Possible Exam Questions
PROBLEM 3:  
(Wins and Losses)

Consider the following data file of information on club basketball teams. Each line in the file represents two teams playing each other and their scores. The format of each line in the file is team1, followed by a hyphen, followed by the number of points team1 made, followed by a colon, followed by team2, followed by a hyphen, and followed by the number of points team2 made. The first team on each line is the home team, where the game was played.

An example of the data file is shown below. For example, in the first line, duke was the home team and duke played against unc, with duke scoring 78 points and unc scoring 76 points, so duke won the game.

duke-78:unc-76
unc-87:virginia tech-80
wake forest-73:duke-92
miami-82:unc-79
wake forest-67:miami-77
ncsu-68:unc-70
unc-80:gatech-65
ncsu-77:virginia tech-73
virginia tech-83:wake forest-79
gatech-75:ncsu-81
gatech-81:wake forest-70
duke-76:ncsu-74
virginia tech-75:miami-74
A. Write the function `processinfo` that has one parameter `filename` which represents the name of the file. This function returns a list of lists of items in which each inner list has four items and represents one line from the file. The first item is a string of team1’s name, the second item is the integer number of points team1 scored, the third item is a string of team2’s name, and the fourth item is the integer number of points team2 scored.

For example, the line `data = processinfo("teamdata.txt")` where "teamdata.txt" is the file above would result in `data` having the value on the next page.

duke-78:unc-76
unc-87:virginia tech-80
wake forest-73:duke-92
miami-82:unc-79
wake forest-67:miami-77
ncsu-68:unc-70
unc-80:gatech-65
ncsu-77:virginia tech-73
virginia tech-83:wake forest-79
gatech-75:ncsu-81
gatech-81:wake forest-70
duke-76:ncsu-74
virginia tech-75:miami-74

data = [ ['duke', 78, 'unc', 76],
        ['unc', 87, 'virginia tech', 80],
        ['wake forest', 73, 'duke', 92],
        ['miami', 82, 'unc', 79],
        ['wake forest', 67, 'miami', 77],
        ['ncsu', 68, 'unc', 70],
        ['unc', 80, 'gatech', 65],
        ['ncsu', 77, 'virginia tech', 73],
        ['virginia tech', 83, 'wake forest', 79],
        ['gatech', 75, 'ncsu', 81],
        ['gatech', 81, 'wake forest', 70],
        ['duke', 76, 'ncsu', 74],
        ['virginia tech', 75, 'miami', 74] ]
Complete the function `processinfo` below.

```python
def processinfo(filename):
    f = open(filename)
```
How to Solve
How to Solve

• Loop over lines in a file
  • "process" each line
• Build a new list
  • Append each line that is converted into a list
How to solve one line
How to solve one line

• "duke-78:unc-76"
  Split into list of two strings
  • ["duke-78", "unc-76"]
  Split into list of two strings
  • ["duke", "78"]  ["unc","76"]
  Needs to be integer
  ["duke",78,"unc",76]

Need to put all together
Final result
def processInfo(filename):
    f = open(filename)
    biglist = []
    for line in f:
        line = line.strip()
        listboth = line.split(':')
        lista = listboth[0].split('-')
        listb = listboth[1].split('-')
        smalllist = [lista[0], int(lista[1]), listb[0], int(listb[1])]
        biglist.append(smalllist)
    return biglist

"duke-78:unc-76"
["duke-78", "unc-76"]
["duke", 78]
["unc", 76]
["duke", 78, "unc", 76]
B. Write the function `schoolsBeat` that has two parameters, `data` and `team`, where `data` is the list of lists in the format from Part A, and `team` is a string.

This function returns a list of tuples, where each tuple is information about a game that `team` won. Each tuple has the name of the team beat, followed by the number of points they won by.

For example, assume `data` is the lists of lists of four items on the previous page. The two examples below show the result of calling `schoolsBeat` with this filename and a team name. For example, duke beat three teams, ncsu by 2 points, unc by 2 points and wake forest by 19 points, wake forest did not beat any teams, and unc beat three teams.

<table>
<thead>
<tr>
<th>call</th>
<th>returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>schoolsBeat(data, &quot;duke&quot;)</code></td>
<td>[('ncsu', 2), ('unc', 2), ('wake forest', 19)]</td>
</tr>
<tr>
<td><code>schoolsBeat(data, &quot;wake forest&quot;)</code></td>
<td>[]</td>
</tr>
<tr>
<td><code>schoolsBeat(data, &quot;unc&quot;)</code></td>
<td>[('gatech', 15), ('ncsu', 2), ('virginia tech', 7)]</td>
</tr>
</tbody>
</table>

```python
def schoolsBeat(data, team):
    data = [...
    return [...
```
How to solve
Ex: "duke-78:unc-76" and team duke
→ ("unc", 2) (duke beat unc by 2)
How to solve
Ex: "duke-78:unc-76" and team duke
→ ("unc", 2)  (duke beat unc by 2)

• Build a new list (accumulator pattern!)
• Loop over each list in the big list (for loop)
  • Is the team in the list? yes
    • Is the team the first team?
      – Did the team win?
      – How much did they win by
      – Create a tuple and add it to new list
    • Or is the team the second team?
      – Process in a similar way
def schoolsBeat(data, team):  

    team is "duke"
def schoolsBeat(data, team):
    newlist = []
    for lista in data:
        if team in lista:
            score1 = lista[1]
            score2 = lista[3]
            if team == lista[0]:  # first team
                if score1 > score2:
                    newlist.append( (lista[2], score1-score2) )
            else:  # second team
                if score2 > score1:
                    newlist.append( (lista[0], score2-score1) )
    return newlist
This problem was Fall 2016 Problem 3

- Check out the other solutions!