Compsci 101
Dictionaries

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, due Thursday, Nov 3
• APT 4 due Thursday, Oct 27
• Lab 6 tomorrow, do prelab 6 before going
  • Videos of Labs 4 and 5 in Sakai Resources folder
• Mid-Semester Survey complete by 10/23
  • Extra credit for Exam 2
• Do not discuss APT Quiz 1 with anyone until they are handed back
• Exam 2 Nov 1
PFTD

- Guess Word
- Dictionaries
- Solving Problems with Dictionaries
- Solving an APT
Assignment 4: Guess Word

- We give you most of the functions to implement
  - Partially for testing, partially for guiding you
- But still more open ended than prior assignments
- If the doc does not tell you what to do:
  - Your chance to decide on your own!
    - Okay to get it wrong on the first try
  - Discuss with TAs and friends, brainstorm!
- Demo!
Problem: Given a name, what is their favorite ice cream?

- Assume you have a lot of people, over 1 million.

- 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']
```

Find Rodger’s favorite ice cream:

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

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

- Rodger's name is at index 2, so:
  - Ice cream is at index 2: 'Chocolate Chip'

No match found.
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']

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

Find Rodger’s favorite ice cream:

```
students =
    0        1        2        3

icecream =
    0        1        2        3
```

**NO**
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']  
            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 =

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

0 1 2 3

icecream =

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

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 =
  0       1       2       3

icecream =
  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 😊
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
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!

```
<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrachan</td>
<td>Chocolate</td>
</tr>
<tr>
<td>Sun</td>
<td>Chocolate Chip</td>
</tr>
<tr>
<td>Rodger</td>
<td>Strawberry</td>
</tr>
<tr>
<td>Forbes</td>
<td></td>
</tr>
</tbody>
</table>
```

Compsci 101, Fall 2022 22
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

Ice Cream Flavors

Astrachan
Sun
Rodger
Forbes

Chocolate
Chocolate Chip
Strawberry
Change Astrachan’s value
somemap[“Astrachan”] = Coffee Mocha
Value could be a set or list

Students
- Astrachan
- Sun
- Rodger
- Forbes

Ice Cream Flavors
- Coffee Mocha
- Chocolate
- Vanilla
- Blueberry
- Chocolate Chip
- Blueberry
- Banana
- Strawberry
- Coffee Mocha
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

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)
Examples

```python
stuff={}  # Example of an empty dictionary
print(stuff)  # Output: {}
print(type(stuff))  # Output: <class 'dict'>

# Dictionary assignment examples
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!!!
```

**Keys can only be immutable types!**
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'

\[ r = d[d['red']] \]

r = d['monkey']

ERROR!!!!!!

\[ d \text{ is } \{ \} \]

\[ d \text{ is } \{ 'color': 'black' \} \]

\[ d \text{ is } \{ 'color': 'red' \} \]

\[ d \text{ is } \{ 'color': 'red', 'red': 'color' \} \]

\[ r \text{ is 'red'} \]
Examples

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

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

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

'dog' in d  \[\text{False}\]
'a' in d     \[\text{True}\]
'pig' in d   \[\text{False}\]

\text{in checks if 'pig' 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

• Get all the keys (as a list)
  • \texttt{listKeys = somemap.keys()}

• Get all the values (as a list)
  • \texttt{listValues = somemap.values()}

• Other methods
  • \texttt{clear} — empty dictionary
  • \texttt{items} — return (key,value) pairs
  • \texttt{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 their more than once

• names = ['Sarah', 'Beth', 'Sarah', 'Purnima', 'Beth', 'Beth', 'Purnima']
Counting Dictionary

def count_dict(names):
    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
val = 0
for key in d:
    if d[key] > val:
        val = d[key]
print("val:", val)
```

Alternative:

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

```python
maxname = ""
for key in d:
    if d[key] == maxval:
        maxname = key
print("maxname", maxname)
```
# APT: EatingGood

## 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.

### Specification

```python
filename: EatingGood.py

def howMany(meals, restaurant):
    """
    Parameter meals a list of strings with each in the format
    "name:place-ate". Parameter restaurant is a string
    return # unique name values where place-ate == restaurant
    """

    # you write code here
    return 0
```
APT Eating Good Example

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

restaurant = "Elmos"

returns 1
```
WOTO-3: APT Eating Good

- https://www2.cs.duke.edu/csed/pythonapt/eatinggood.html

1. Work an instance yourself
2. Write down exactly what you just did
3. Generalize your steps from (2)
4. Test your steps

Can’t find pattern

Identified problem
Algorithmic
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

• Make an empty list

• Loop over each meal
  • Split the 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
Goes over Eating Good APT (w/list)

# make an empty list
names = []

# loop over meals
for meal in meals:
    # split every element in meals
    data = meal.split(':')
    (name, rest) = (data[0], data[1])

    # if this is the restaurant
    if rest == restaurant:
        # count number of times see a person
        # add person to list only if never seen before
        if name not in names:
            names.append(name)

# return length of names
APT Eating Code Idea With List

• Make an empty list
• Loop over each meal
  • Split the 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 the 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)
APT Eating Code – Use set instead of list

• Make an empty set
  ```python
  names = set()
  ```

• Loop over each meal
  • Split the meal into name and restaurant
  • If the restaurant matches
    • Add name to set
      ```python
      names.add(name)
      ```

• Return the length of the set
  ```python
  return len(names)
  ```
APT Eating Code – Use set instead of list

- Make an empty set

- Loop over each meal
  - Split the meal into name and restaurant
  - If the restaurant matches
    - Add name to set

- Return the length of the set

```python
names = set()
for meal in meals:
    name, restaurant = split_meal(meal)
    if restaurant matches:
        names.add(name)
return len(names)
```
Lists or Set?

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

names.add(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