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

Announcements

- Assign 3 Transform due Today!
- Assign 4 is out, due Thursday, March 17
- APT 4 due this Thursday
- APT-5 out Thursday, due March 24
- No lab this week
- A few consulting hours during spring break
- Do not discuss Exam 1 or APT Quiz 1 with anyone until they are handed back
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

```python
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?

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

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

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

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

• Very, very fast: essentially index by key
  • For list a[3] takes same time as a[3000]

• For Dictionary: d["cake"]
  • Finding the value associated with "cake"
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

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!

Example

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

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 | Chocolate
Sun | Chocolate Chip
Rodger | Strawberry
Forbes |

Value could be a set or list

Students | Ice Cream Flavors
Astrachan | Coffee Mocha
| Chocolate
| Vanilla
| Blueberry
Sun | Chocolate Chip
Rodger | Chocolate Chip
| Blueberry
| Banana
Forbes | 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

```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)
```

Examples

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

d['color'] = 'red'

d['red'] = 'color'

r = d[d['red']]

r = d['monkey']
```

Examples

```python
stuff = {'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

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

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

Examples

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

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

for t in d.items():
    print(t)
**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']

**Sandwich Bar**

**APT: SandwichBar Search**

**Problem Statement**

It's time to get something to eat and I've come across a sandwich bar. Like most people, I prefer certain types of sandwiches. In fact, I keep a list of the types of sandwiches I like.

The sandwich bar has certain ingredients available. I will list the types of sandwiches I like in order of preference and buy the first sandwich the bar can make for me. In order for the bar to make a sandwich for me, it must include all of the ingredients I desire.

Given available, a list of Strings/ingredients the sandwich bar can use, and a orders, a list of Strings that represent the types of sandwiches I like, in order of preference (most preferred first), return the 0-based index of the first sandwich in orders, list of strings that can be made from ingredients in available, list of strings.

```python
filename: SandwichBar.py
def whichOrder(available, orders):
    # you write code here
```

Given available, a list of Strings/ingredients the sandwich bar can use, and a orders, a list of Strings that represent the types of sandwiches I like, in order of preference (most preferred first), return the 0-based index of the sandwich I will buy. Each element of orders represents one type of sandwich I like as a space-separated list of ingredients in the sandwich. If the bar can make no sandwiches I like, return -1.

**Sandwich Bar Example**

- available = [ "cheese", "cheese", "cheese", "tomato" ]
- orders = [ "ham ham ham", "water", "pork", "bread", "cheese tomato cheese", "beef" ]
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!