Compsci 101
Dictionaries

Susan Rodger
Yesenia Velasco
October 20, 2022

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, 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
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

**Find Rodger’s favorite ice cream**

Students =

```
0  1  2  3
```

Icecream =

```
0  1  2  3
```

Save

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
```

Save

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

students =

```python
0        1        2        3
```

icecream =

```python
0                          1                       2                           3
```

10/20/22

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
0        1        2        3
```

icecream =

```python
0                          1                       2                           3
```

10/20/22

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 =

```python
0        1        2        3
```

icecream =

```python
0                          1                       2                           3
```

10/20/22

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 =

```python
0        1        2        3
```

icecream =

```python
0                          1                       2                           3
```

10/20/22
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!

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
How to use a Dictionary

- Create: \( d = \{ \} \)
  - \( d = \{ 'a': 10, 'b': 100 \} \)
  - \( d = \text{dict}([('a', 10), ('b', 100)]) \)
- Insert: \( d[\text{KEY}] = \text{VALUE} \)
- Update/Reassign: \( d[\text{KEY}] = \text{VALUE} \)
- Get a value (like list indexing): \( d[\text{KEY}] \)
- Key membership (not values): \( \text{KEY} \in d \)
  - 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

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

Dictionaries are unordered

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

d['color'] = 'red'

d['red'] = 'color'

r = d[d['red']]

r = d['monkey']
```

10/20/22 Compsci 101, Fall 2022
Examples

d = {}
d['color'] = 'black'
d = {'color': 'black'}
d['color'] = 'red'
d = {'color': 'red'}
d['red'] = 'color'
d = {'color': 'red', 'red': 'color'}

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

10/20/22
Compsci 101, Fall 2022
33

WOTO-1 Dictionaries

Examples

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

'dog' in d
'pig' in d

'dog' in d
'pig' in d

10/20/22
Compsci 101, Fall 2022
34

in checks if ‘pig’ is a key in d

10/20/22
Compsci 101, Fall 2022
35
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)
  - 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)
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

```
8     d = {}
9     for word in names:
10            if word not in d:
11                d[word] = 1
12            else:
13                count = d[word]
14                d[word] = count + 1
15        print("d:", d)
```

Finding Largest Value in d

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

Alternative:

```
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)
```

APT Eating Good

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 `restaurant`.

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

Write function `howMany(meals, restaurant)` that given `meals`, a list of strings in the format "name:place-ate" Parameter: restaurant is a string where place-ate == restaurant.

Identified problem

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

Algorithmic

WOTO-3: APT Eating Good


meals = ["Sue:Elmos", "Sue:Elmos", "Sue:Elmos"]
restaurant = "Elmos"
returns 1
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)

```python
# 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 set  \( \text{names} = \text{set()} \)
• Loop over each meal
  • Split the meal into name and restaurant
  • If the restaurant matches
    • Add name to set
    \( \text{names.add(name)} \)
• Return the length of the set  \( \text{return len(names)} \)

Lists or Set?

• 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

\[
\text{if name not in names:} \quad \text{names.add(name)} \\
\text{names.append(name)}
\]