Dictionaries

15-110 – Monday 10/12

Learning Goals

- Identify the **keys** and **values** in a dictionary
- Use **dictionaries** when writing and reading code

Data Structures Organize Data

So far, we've talked about **efficiency** in terms of algorithm design. We can solve the same problem multiple ways, and some approaches are more efficient than others

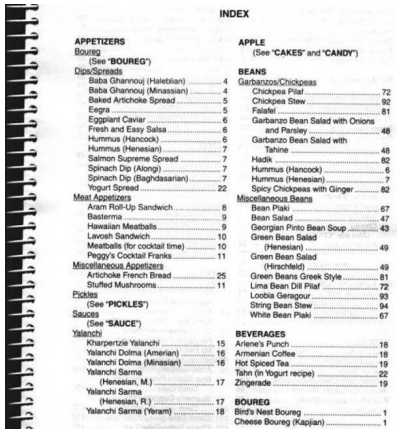
We can also improve the efficiency of an algorithm by changing how the data is stored. Formats for storing data are called **data structures**. For example, a **list** is a basic data structure that stores values in sequential (and indexed) order. What other structures exist?

Dictionaries

Python Dictionaries Map Keys to Values

The first data structure we'll discuss is the **dictionary**, or **hashmap**. Dictionaries store data in **pairs** by mapping **keys** to **values**.

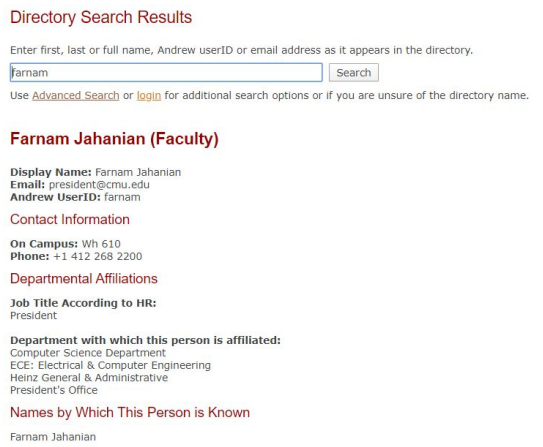
We use dictionary-like data in the real world all the time! Examples include phonebooks (which map names to phone numbers), the index of a book (which maps terms to page numbers), or the CMU directory (which maps andrewIDs to information about people).



INDEX

APPETIZERS
Boureg
(See "BOUREG")
Dips/Spreads
Baba Ghannouj (Halebian) 4
Baba Ghannouj (Minassian) 4
Baked Artichoke Spread 5
Eegra 5
Eggplant Caviar 6
Fresh and Easy Salsa 6
Hummus (Hancock) 6
Hummus (Henesian) 7
Salmon Supreme Spread 7
Spinach Dip (Along) 7
Spinach Dip (Baghdasarian) 7
Yogurt Spread 22
Meat Appetizers
Aram Roll-Up Sandwich 8
Basterma 9
Hawaiian Meatballs 9
Lavosh Sandwich 10
Meatballs (for cocktail time) 10
Peggy's Cocktail Franks 11
Miscellaneous Appetizers
Artichoke French Bread 25
Stuffed Mushrooms 11
Pickles
(See "PICKLES")
Sauces
(See "SAUCE")
Yalanchi
Kharpertzie Yalanchi 15
Yalanchi Dolma (Amerian) 16
Yalanchi Dolma (Minasian) 16
Yalanchi Sarma
(Henesian, M.) 17
Yalanchi Sarma
(Henesian, R.) 17
Yalanchi Sarma (Yeram) 18

APPLE
(See "CAKES" and "CANDY")
BEANS
Garbanzos/Chickpeas
Chickpea Pilaf 72
Chickpea Stew 92
Falafel 81
Garbanzo Bean Salad with Onions and Parsley 48
Garbanzo Bean Salad with Tahine 48
Hadik 82
Hummus (Hancock) 6
Hummus (Henesian) 7
Spicy Chickpeas with Ginger 82
Miscellaneous Beans
Bean Pilaki 67
Bean Salad 47
Georgian Pinto Bean Soup 43
Green Bean Salad
(Henesian) 49
Green Bean Salad
(Hirschfeld) 49
Green Beans Greek Style 81
Lima Bean Dill Pilaf 72
Loobia Geragour 93
String Bean Stew 94
White Bean Pilaki 67
BEVERAGES
Arlene's Punch 18
Armenian Coffee 18
Hot Spiced Tea 19
Tahn (in Yogurt recipe) 22
Zingerade 19
BOUREG
Bird's Nest Boureg 1
Cheese Boureg (Kapjian) 1



Directory Search Results

Enter first, last or full name, Andrew userID or email address as it appears in the directory.

farnam Search

Use Advanced Search or login for additional search options or if you are unsure of the directory name.

Farnam Jahanian (Faculty)

Display Name: Farnam Jahanian
Email: president@cmu.edu
Andrew UserID: farnam

Contact Information

On Campus: Wh 610
Phone: +1 412 268 2200

Departmental Affiliations

Job Title According to HR:
President

Department with which this person is affiliated:
Computer Science Department
ECE: Electrical & Computer Engineering
Heinz General & Administrative
President's Office

Names by Which This Person is Known

Farnam Jahanian

Key-Value Pairs

In a dictionary, a **key-value pair** is two values that have been paired together for organizational purposes. We'll be able to access the value by looking up the key, like how we can access a list value using its index.

For example, if we stored a phonebook in a dictionary, a **key** might be the string `"CMU"`, and its **value** would be the string `"412-268-2000"`. It wouldn't make sense to switch the roles, because our default action is to look up a phone number based on a name, not vice versa.

Sidebar: keys must be **immutable**, but values can be any type of data. Why? We'll explain next week, when we talk about dictionary search.

Python Dictionaries

Dictionaries have already been implemented for us in Python.

```
# make an empty dictionary
d = { }

# make a dictionary mapping strings to integers
d = { "apples" : 3, "pears" : 4 }
```

Python Dictionaries – Getting Values

Dictionaries are similar to lists. Instead of indexing by position, index by key:

```
d = { "apples" : 3, "pears" : 4 }
d["apples"] # the value paired with this key
len(d)      # number of key-value pairs
```

We can also access all the keys or all the values separately:

```
d.keys()
d.values()
```

Python Dictionaries – Adding and Removing

How do we add a new key-value pair? Use **indexed assignment** with the key. This works whether or not that key has been assigned a value yet. If the key is already in the dictionary, the value for the key is updated; it does not add a new key-value pair.

```
d["bananas"] = 7 # adds a new key-value pair
d["apples"] = d["apples"] + 1 # updates the value
```

To remove a key-value pair, use **pop**, with just the key as a parameter.

```
d.pop("pears") # destructively removes
```

Python Dictionaries – Search

We can **search** for a key in a dictionary using the built-in `in` operation.

```
d = { "apples" : 3, "pears" : 4 }
"apples" in d # True
"kiwis" in d  # False
```

We mainly use dictionaries because it's very fast to search for keys. In fact, searching a dictionary for a key takes **constant time**! How is this possible? We'll discuss more in the Search Algorithms II lecture.

We can't do constant-time lookups of the dictionary's values; we need to do a linear-time loop of the keys and check each key's value instead.

Activity: Trace the code

After running the following code, what key-value pairs will the dictionary hold?

```
d = { "PA" : "Pittsburgh", "NY" : "New York City" }
d["WA"] = "Seattle"
d["NY"] = "Buffalo"
if "Pittsburgh" in d:
    d.pop("Pittsburgh")
```

For Loops on Dictionaries

When we need to loop over a dictionary, we can't use a for-range loop, because there are no numeric indices to loop over. That means we need to use a for loop over the iterable value instead.

When we run a for loop directly over a dictionary, the loop visits all key-value pairs in some order. The loop control variable is set to the **key** of each key-value pair. To access the value, you must index into the dictionary with that key.

```
d = { "apples" : 5, "beets" : 2, "lemons" : 1 }
for k in d:
    print("Key:", k)
    print("Value:", d[k])
```

Activity: `countItems(foodCounts)`

You do: write the function `countItems(foodCounts)` that takes a dictionary mapping foods (strings) to counts (integers) and returns the total amount of food stored in the dictionary. The function should also print the number of each individual food type as it counts up the total.

For example, if `d = { "apples" : 5, "beets" : 2, "lemons" : 1 }`, the function might print

```
5 apples
2 beets
1 lemons
```

then return `8`.

Coding with Dictionaries

Coding with Dictionaries – Track Information

We often use dictionaries when problem-solving. One common use of dictionaries is to **track information** about a list of values.

For example, given a list of students and their college (represented as "student,college"), how many students are in each college?

We will create a dictionary with college as the key and the student count as the value.

```
def countByCollege(studentLst):
    collegeDict = { }
    for student in studentLst:
        name = student.split(",")[0]
        college = student.split(",")[1]
        if college not in collegeDict:
            collegeDict[college] = 0
        collegeDict[college] += 1
    return collegeDict
```

Coding with Dictionaries – Find Most Common

We also use dictionaries to find the most common element of a list, by mapping elements to counts.

For example, given the dictionary returned by the previous function, which college is the most popular?

```
def mostPopularCollege(collegeDict):
    best = None
    bestScore = -1
    for college in collegeDict:
        if collegeDict[college] > bestScore:
            bestScore = collegeDict[college]
            best = college
    return best
```

Coding with Dictionaries – Has Duplicates

We might also want to check whether a list contains duplicate values. One approach is to use nested loops. For each item in the list, search through the rest of the items to see if there is a duplicate. The nested loop has $O(n^2)$ runtime.

We can use a dictionary to check if any student appears more than once in list of students in O(n) runtime.

```
def hasDuplicates(students):
    studentDict = { }
    for student in students:
        name = student.split(",")[0]
        college = student.split(",")[1]
        if name in studentDict:
            return True
        else:
            studentDict[name] = college
    return False
```

Activity: `mostCommonFirstLetter(s)`

You do: rearrange the lines of code in this Parsons Problem to make a program `mostCommonFirstLetter(s)` that takes a sentence (string), tracks the letters that occur as the first letters of words in the string, and returns the character that most often occurs as a first letter. In the case of a tie, return any of the letters that tied.

For example, `mostCommonFirstLetter("do you have a voting plan for the election happening next month?")` would return `"h"`, since `"h"` occurs twice (`"have"` and `"happening"`) while each other word occurs once.

Puzzle link: http://bit.ly/110-firstletter

Note: define and update `bestLetter` before `bestCount` (otherwise the puzzle will mark your answer as wrong).

Learning Goals

- Identify the **keys** and **values** in a dictionary
- Use **dictionaries** when writing and reading code

- **Feedback:** https://bit.ly/110-feedback