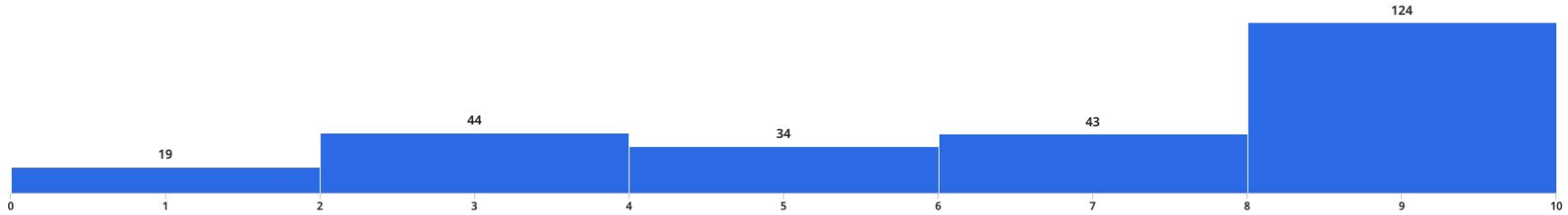
Dictionaries

15-110 – Friday 10/04

Announcements

- Semester drop deadline is Monday
- Quizlet 3 grades released
- **Hw3** due on Monday
- Sign up for code reviews!
  - Sign-ups close today at 11:59pm



Minimum	Median	Maximum	Mean	Std Dev
0.0	7.25	10.0	6.72	--

Learning Goals

- Identify the **keys** and **values** in a dictionary
- Use **dictionaries** when writing and reading code that uses pairs of data
- Use **for loops** to iterate over the parts of an **iterable** value

We can improve the efficiency of our algorithms by changing the **data structure** we use to store data.

Lists are nice when we need to store data in sequential order and want to be able to quickly lookup a piece of data stored at a specific index.

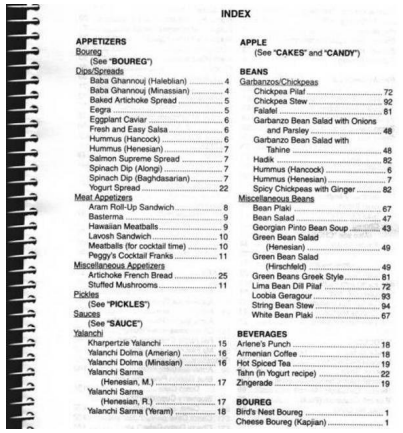
What other data structures can we use?

Dictionaries

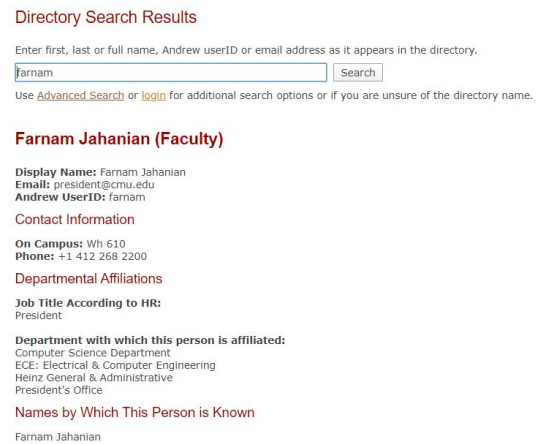
Dictionaries store data in pairs by mapping **keys** to **values**.

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



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

We use **curly brackets** to create dictionaries in Python.

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

# make a dictionary mapping strings to integers
d = { "apples" : 3, "pears" : 5,  "bananas" : 4 }
```

Each element of a dictionary is a key-value pair **separated by a colon**.

Keys in dictionaries are **unordered**, **unique**, and must be **immutable**. Values can be any data type.

```
# keys are strings, values are integers
d = { "apples" : 3, "pears" : 5,  "bananas": 4}
```

"bananas"	4
"apples"	3
"pears"	5

Keys in dictionaries are **unordered**, **unique**, and must be **immutable**. Values can be any data type.

```
# keys are strings, values are integers
d = { "apples" : 3, "pears" : 5 ,"bananas": 4 }

# keys are strings, values are strings
d = {"Pittsburgh": "PA","San Diego": "CA"}

# keys are integers, values are floats
d = {1: 3.5, 4: 7.3, 10: 12.12}
```

We use indexing to get values in a dictionary where the **index is the key** (and not a number).

```
d = { "apples": 3, "pears": 4, "bananas": 5}


d["apples"] # the value paired with this key
len(d) # number of key-value pairs
```

You get a runtime error if you try to **access a key not in the dictionary**.

```
d["ice cream"] # KeyError
d[4] # KeyError
```

Dictionaries are **mutable**.

We can update values using **index assignment** (similar to lists) and we can add new key-value pairs.

```
d = { "apples": 3, "pears": 4, "bananas": 5 }
d["grapes"] = 7 # adds a new key-value pair
d["apples"] = d["apples"] + 1 # updates the key-value pair
```

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

```
d.pop("pears") # mutating remove
```

We can **search for a key** in a dictionary using the `in` operator

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

We cannot use `in` to look up the dictionary's values. We need to loop over the keys and check each key's value instead!

Activity: Trace The Code

In the following code, the keys represent student IDs and the values represent student names. **After running the code, what key-value pairs will the dictionary hold?**

```
d = { 26: "Chen", 23: "Patrick" }
d[88] = "Rosa"
d[23] = "Pat"
d[51] = d[23]
if "Chen" in d:
    d.pop("Chen")
```

Activity: Predict the Printed Values

```
d = { 26: "Chen", 23: "Patrick", 26: “Rosa” }
print(d[26])
```

[1]: `"Chen"`
[2]: `"Rosa"`
[3]: `"ChenRosa"`
[4]: This will crash with an error
[5]: I don’t know

Looping over Iterables

An **iterable** is any data type that is a ***sequence* we can loop over** using a for loop in Python.

Strings are a sequence of characters where each character is at a specific index.

Lists are a sequence of values where each element is at a specific index.

Dictionaries are a sequence of key-value pairs where each value has a specific key.

A dictionary is **unordered**, so we cannot loop over it with range.

```
d = { "apples": 3, "pears": 4, "bananas": 5 }

for i in range(len(d)):
   print(d[i])  # Key Error
```

We can **directly loop over iterables** using a `for` loop without using `range`!

```
for <itemVariable> in <iterableValue>:
    <itemActionBody>
```

Iterating over string characters:

```
s = "this is the best class"
for char in s:
    print(char.upper())
```

Iterating over list elements:

```
L = ["Hello", "World"]
for word in L:
    print(word + "!")
```

We can directly for **loop over dictionary keys** and then index into the dictionary to get the value.

```
d = { "apples" : 5, "beets" : 2, "lemons" : 1 }

for k in d:
    print("Key:", k)
    print("Value:", d[k])
```

For dictionaries, **you always have to use a For-Iterable loop** but for strings and lists you can also use a For-Range loop.

Both of these sum the values in `lst`:

```
result = 0
for item in lst:
    result = result + item
```

or

```
result = 0
for i in range(len(lst)):
    result = result + lst[i]
```

Activity: `printItems(foodCounts)`

You do: write the function `printItems(foodCounts)` that takes a dictionary mapping foods (strings) to counts (integers), loops over the key-value pairs, and print the number of each individual food type included in the input.

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

5 apples

2 beets

1 lemons

Problem Solving with Dictionaries

Example: We often need to loop over the keys and **doing something with each key-value pair**.

Algorithm: Sum all the values in a dictionary.

```
def addValues(d):
    total = 0
    for key in d:
        total = total + d[key]
    return total
```

Example: We often need to **build/create a new dictionary**.

Algorithm: Create an alphabet dictionary for a list of strings.

```
def makeAlphabetDict(words):
    d = { }
    for word in words:
        letter = word[0]
        if letter not in d:
            d[letter] = [word]
        else:
            d[letter].append(word)
    return d
```

When building a dictionary, you usually need to check if a key is already in the dictionary.

We can **nest dictionaries** by mapping each key to another dictionary.

Algorithm: Create a multiplication table.

```
def createMultDict(n):
    d = { }
    for x in range(1, n+1):
        innerD = { }
        for y in range(1, n+1):
            innerD[y] = x * y
        d[x] = innerD
    return d
```

[if time] Activity: `hasShortKeys(d, limit)`

You do: write a program that takes a dictionary mapping strings to numbers and a limit (a number) and returns `True` if all the keys are at most the limit in length, and `False` otherwise.

For example, `hasShortKeys({ "abc" : 2, "de" : 5}, 3)` would return `True`, but `hasShortKeys({ "abc" : 2, "defgh" : 2}, 4)` would return `False`.

Learning Goals

- Identify the **keys** and **values** in a dictionary
- Use **dictionaries** when writing and reading code that uses pairs of data
- Use **for loops** to iterate over the parts of an **iterable** value

Advanced Examples

Bonus slides

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 2D list of students and their colleges (a list of two-element lists of `"student"` and `"college"`), how many students are in each college?

We will create a dictionary with colleges as the keys and the student counts as the values.

```
def countByCollege(studentLst):
    collegeDict = { }
    for pair in studentLst:
        name = pair[0]
        college = pair[1]
        if college not in collegeDict:
            collegeDict[college] = 0
        collegeDict[college] += 1
    return collegeDict

countByCollege([ ["erhurst" ,"CIT"],
["neerajsa","SCS"], ["cosorio","DC"],
["dtoussai", "CIT"]])
```

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
```