

LEC 10

CSE 122

# Nested Collections

Questions during Class?

Raise hand or send here

sli.do #cse122



*Talk to your neighbors:*

Favorite cold weather drink?  
Matcha? Latte? Hot chocolate?  
(Mine is [Atol.](#))


Music: [122 24au Lecture Tunes](#) 🍂

**Instructors:** Elba Garza and Miya Natsuhara




**TAs:**

Ayush	Heon	Harshitha	Aishah
Andrew	Izak	Marcus	Ben
Logan	Colin	Carson	Ivory
Kyle	Jessica	Jack	Cady
Maggie	Shivani	Connor	Diya
Nicole H	Ken	Cora	Katharine
Caleb	Mia	Hannah	
Nicole W	Ashley	Leo	
Jacob	Chaafen	Any	

# Agenda

- **Announcements** 
- Review/Finish: mostFrequentStart
- Recap: Nested Collections
- Practice: Search Engine

# Announcements

- Programming Assignment 2 (P2) was released on Friday!
  - Seriously, start early! This assignment is much more involved...
  - Due **Nov 7<sup>th</sup>** by 11:59 PM
- Quiz 1 on **Nov 5<sup>th</sup>**  in your registered Quiz Section
  - Topics: (Reference Semantics), Stacks and Queues, Sets, Maps
  - Practice Quiz 1 available soon along with...
- Quiz 0 grades to be released later today!
- Tomorrow, Resubmission Cycle 3 (R3) form out, due Nov 5<sup>th</sup> by 11:59 PM
  - Available assignments:  **PO** , C1, P1
  - Reminder: to use a resubmission cycle you need to
    - (1) submit your work (big blue "Submit" button on Ed)
    - **AND** (2) fill out the resubmission form (linked from Ed + course calendar)

# An aside for quizzes...

Please be legible  
and clear on your  
written answers 🙄

c. Below are methods:

Method 1:  
public  
d =  
retu  
}

Method 2:  
public  
retu  
}

Method 3:  
public  
retu  
}

Method 4:  
public  
Stri  
retu  
}

Method 5:  
public  
Syst  
Syst  
Syst  
Syst  
Syst  
}

c. Below are methods:

Method 1:  
public r  
retu  
}

Method 2:  
public r  
out.1  
}

Method 3:  
public r  
int r  
retu  
}

Method 4:  
public r  
i = i  
retu  
}

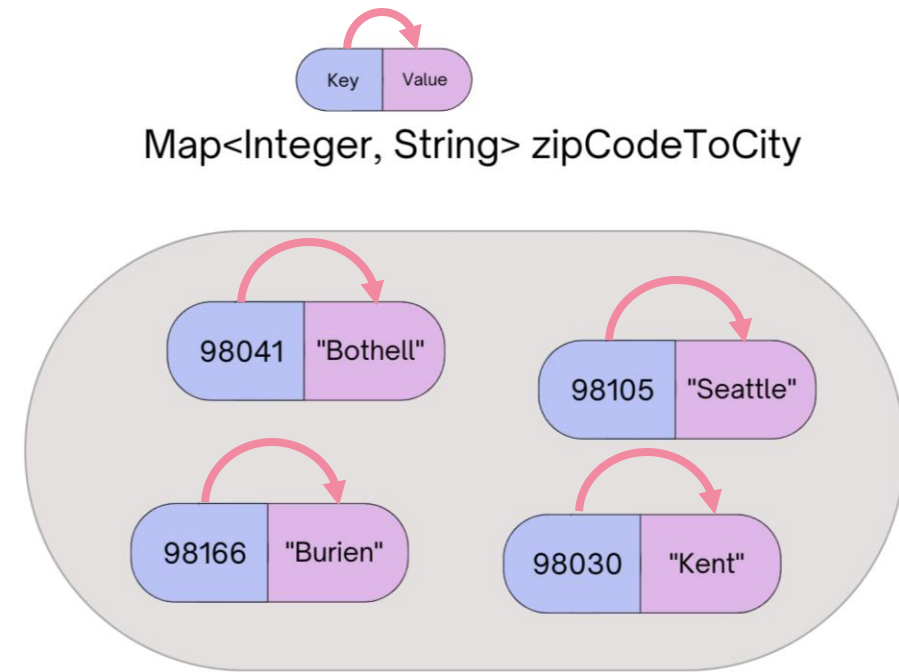
Method 5:  
public r  
Syste  
Syste  
Syste  
Syste  
Syste  
}

# Agenda

- Announcements
- **Review/Finish: mostFrequentStart** ◀
- Recap: Nested Collections
- Practice: Search Engine

# Map ADT

- Data structure to map keys to values
  - Keys can be any\* type; Keys must be unique
  - Values can be any type
- Example: Mapping ticker to stock price in PO
- Operations
  - `put(key, value)`: Associate key to value
    - Overwrites duplicate keys
  - `get(key)`: Get value for key
  - `remove(key)`: Remove key/value pair



Same as Python's dict

# mostFrequentStart

Write a method called `mostFrequentStart` that takes a Set of words and does the following steps:

- Organizes words into “word families” based on which letter they start with
- Selects the largest “word family” as defined as the family with the most words in it
- Returns the starting letter of the largest word family (and *should update the Set of words to only have words from the selected family*).

# mostFrequentStart

For example, if the Set words stored the values

```
["hello", "goodbye", "library", "literary", "little", "repel"]
```

The word families produced would be

```
'h' -> 1 word ("hello")
```

```
'g' -> 1 word ("goodbye")
```

```
'l' -> 3 words ("library", "literary", "little")
```

```
'r' -> 1 word ("repel")
```

Since 'l' has the largest word family, we return 3 and modify the Set to only contain Strings starting with 'l'.

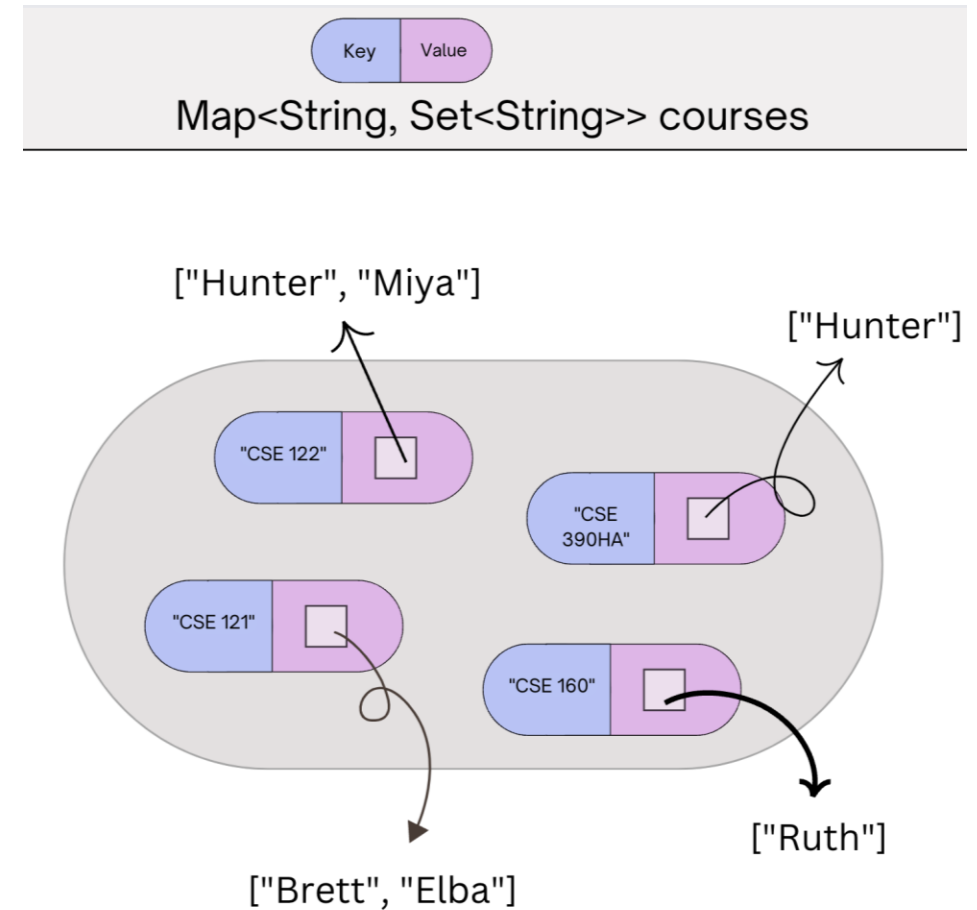


# Agenda

- Announcements
- Review/Finish: mostFrequentStart
- **Recap: Nested Collections** ◀
- Practice: Search Engine

# Nested Collections

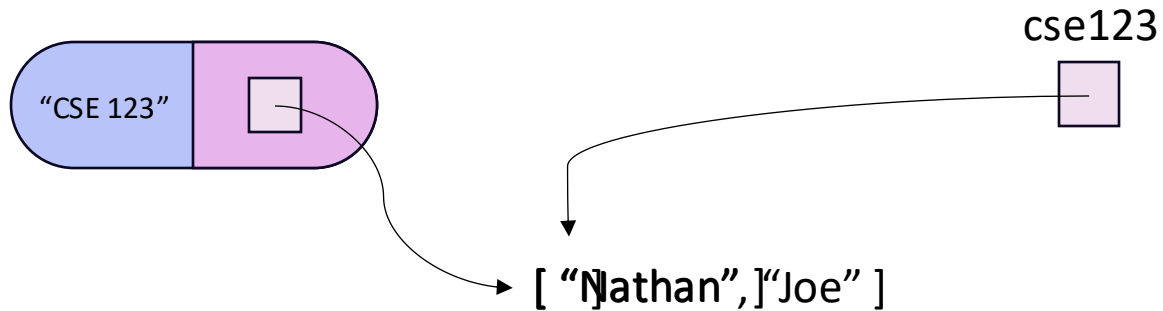
- The values inside a Map can be any type, including data structures
- Common examples:
  - Mapping: Section → Set of students in that section
  - Mapping: Recipe → Set of ingredients in that recipe
    - Or even Map<String, Map<String, Double>> for units!



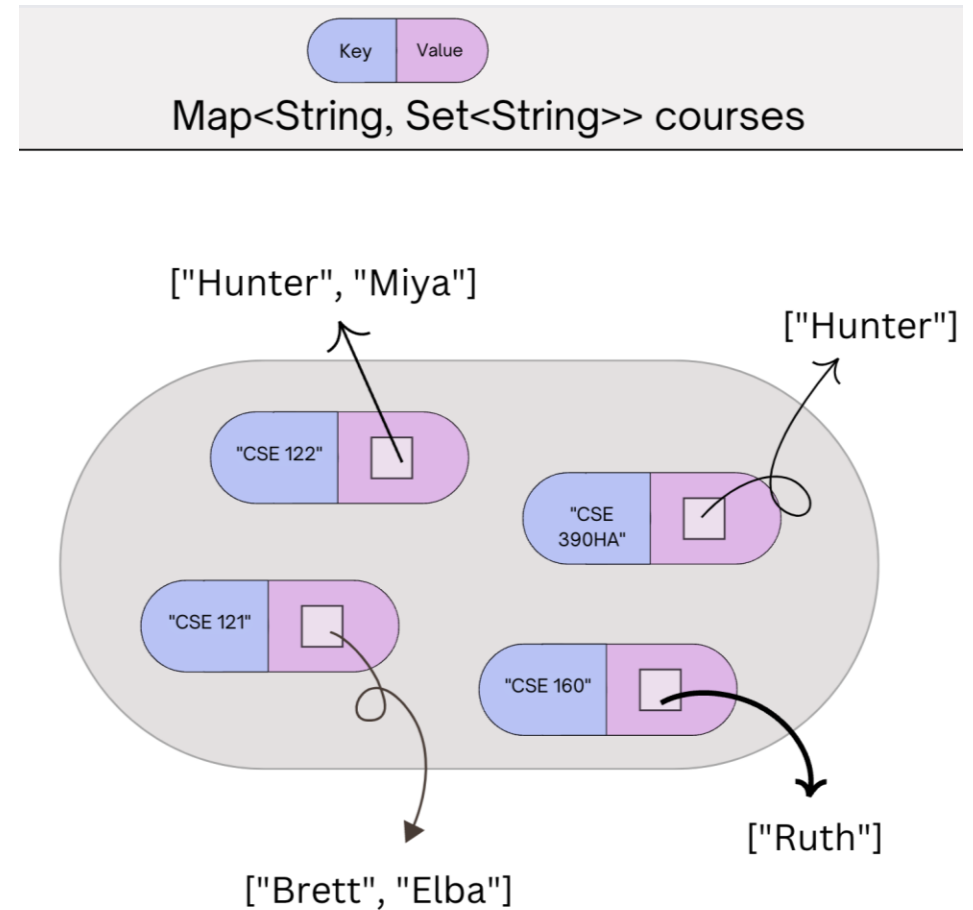
# Updating Nested Collections

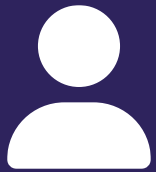
The “value” inside the Map is a reference to the data structure!

- Think carefully about number of references to a particular object



```
courses.put("CSE 123", new HashSet<String>());  
courses.get("CSE 123").add("Nathan");  
Set<String> cse123 = courses.get("CSE 123");  
cse123.add("Joe");
```





# Practice : Think



sli.do #cse122

**Suppose map had the following items. What would its items be after running this code?**

```
map: {"KeyA"=[1, 2], "KeyB"=[3], "KeyC"=[4, 5, 6]}
```

```
Set<Integer> nums = map.get("KeyA");  
nums.add(7);  
map.put("KeyB", nums);  
map.get("KeyA").add(8);  
map.get("KeyB").add(9);
```

- A. {"KeyA"=[1, 2], "KeyB"=[1, 2, 7], "KeyC"=[4, 5, 6]}
- B. {"KeyA"=[1, 2, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}
- C. {"KeyA"=[1, 2, 7, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}
- D. {"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9], "KeyC"=[4, 5, 6]}








# Practice : Pair

[sli.do](#) [#cse122](#)

## Suppose map had the following items. What would its items be after running this code?

```
map: {"KeyA"=[1, 2], "KeyB"=[3], "KeyC"=[4, 5, 6]}
```



```
Set<Integer> nums = map.get("KeyA");  
nums.add(7);  
map.put("KeyB", nums);  
map.get("KeyA").add(8);  
map.get("KeyB").add(9);
```

nums

A: [1, 2, 7, 8, 9]

B: [3]

C: [4, 5, 6]

- A. {"KeyA"=[1, 2], "KeyB"=[1, 2, 7], "KeyC"=[4, 5, 6]}
- B. {"KeyA"=[1, 2, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}
- C. {"KeyA"=[1, 2, 7, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}
- D. {"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9], "KeyC"=[4, 5, 6]}

# Agenda

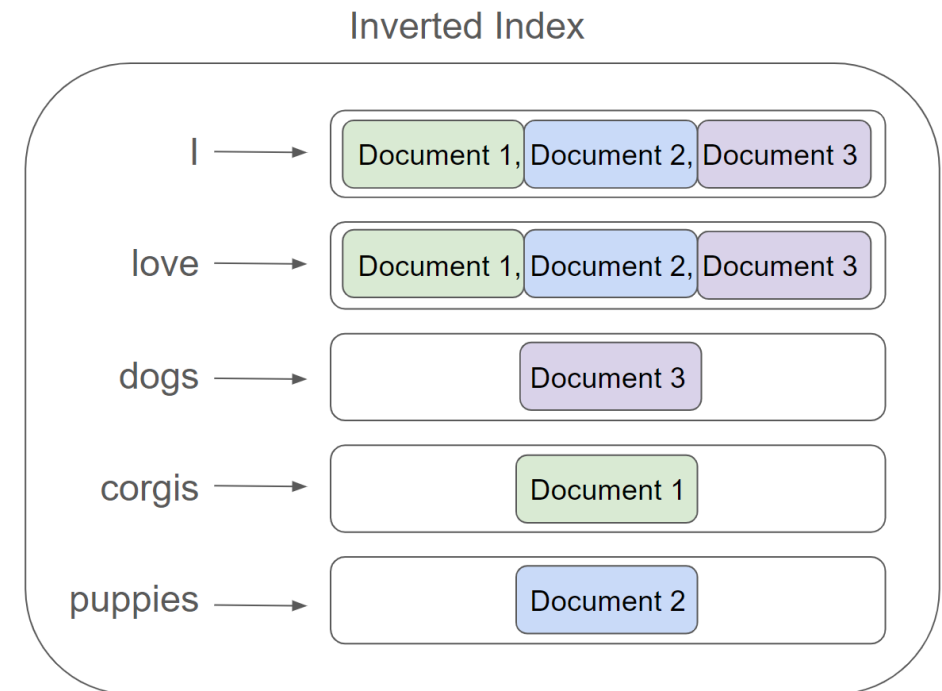
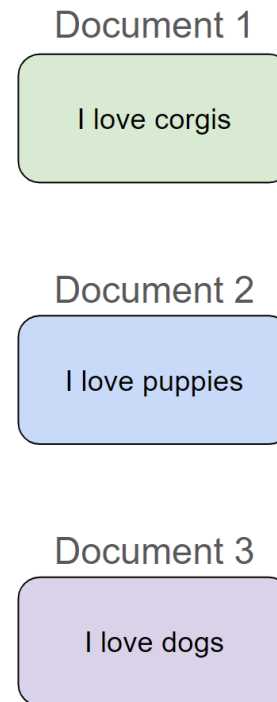
- Announcements
- Review/Finish: mostFrequentStart
- Recap: Nested Collections
- **Practice: Search Engine** ◀

# Background: Search Engines

- A **search engine** receives a **query** and returns a set of relevant **documents**. Examples: Google.com, Mac Finder, more.
  - Queries often can have more
- A search engine involves two main components
  - An **index** to efficiently find the set of documents for a query
    - Will focus on “single word queries” for today’s example
  - A **ranking algorithm** to order the documents from most to least relevant
    - Not the focus of this example
- Goal: Precompute a data structure that helps find the relevant documents for a given query

# Inverted Index

- An **inverted index** is a Mapping from possible query words to the set of documents that contain that word
  - Answers the question: “What documents contain the word ‘corgis’?”





# Ranking Results

- There is no one right way to define which documents are “most relevant”  
There are approximations, but make decisions about what relevance means
- Idea 1: Documents that have more hits of the query should come first
  - Pro: Simple
  - Con: Favors longer documents (query: “the dogs” will favor long documents with lots of “the”s)
- Idea 2: Weight query terms based on their “uniqueness”. Often use some sort of score for “Term Frequency – Inverse Document Frequency ([TF-IDF](#))”
  - Pro: Doesn’t put much weight on common words like “the”
  - Cons: Complex, many choices in how to compute that yield pretty different rankings
- Idea 3: Much more! Most companies keep their ranking algorithms very very secret 😊

# Data Bias

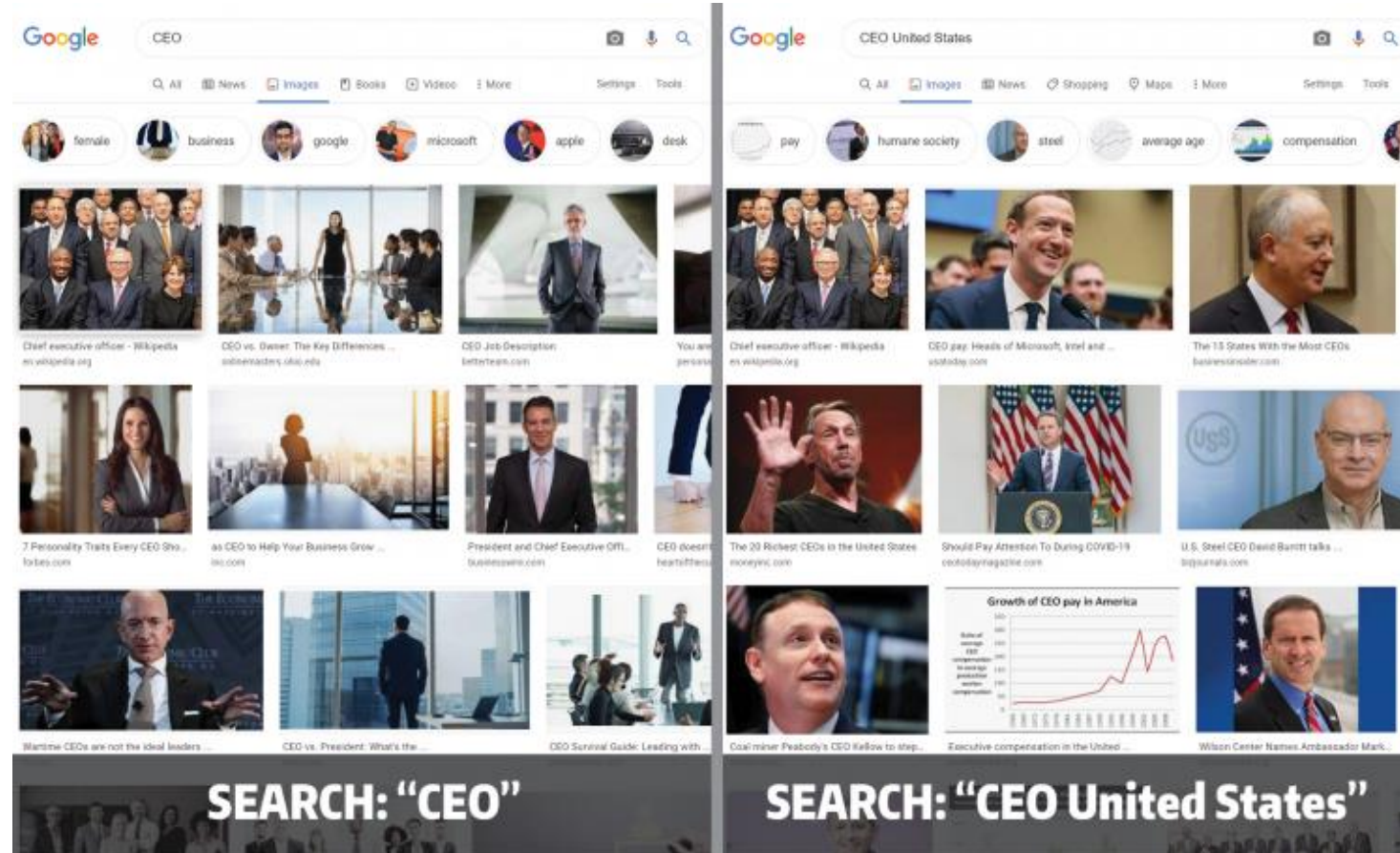
- Image results for searching the term “CEO” on Google (2015)
  - Notice anything about the results?



<https://www.washington.edu/news/2015/04/09/whos-a-ceo-google-image-results-can-shift-gender-biases/>

# Data Bias

- Fix: Image results for searching “CEO” and “CEO United States” (2022)



<https://www.washington.edu/news/2022/02/16/googles-ceo-image-search-gender-bias-hasnt-really-been-fixed>

# Data Bias

- Google's autocomplete recommendations used to actually look like this
  - Fix: Don't display autocomplete results for phrases like "why are [group] \_\_\_\_\_"

Are these changes fixing the right thing?

*Btw, this is a great book that you should check out if you're interested ->*

