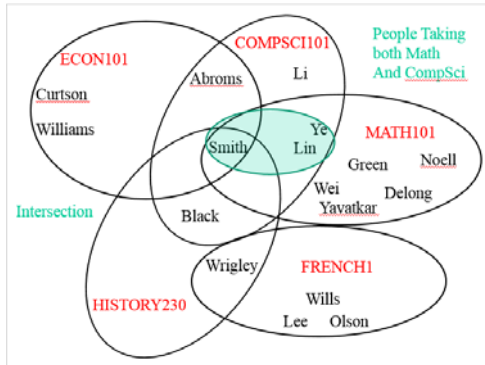


# CompSci 101

## Introduction to Computer Science



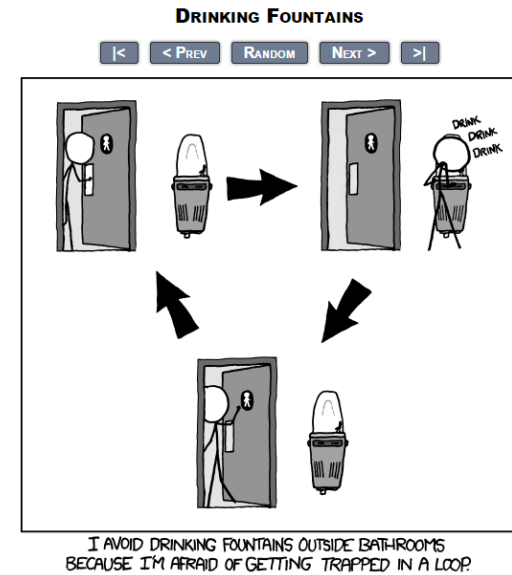
compsci101 fall17

Oct 26, 2017

Prof. Rodger

1

from  
xkcd



compsci101 fall17

2

## Announcements

- Reading and RQ15 due next time
- Assignment 5 due today, Assign 6 out
- APT 5 due Tuesday
- Today:
  - Problem solving using set operations

compsci101 fall17

3

## APT SandwichBar

### 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 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`.

### Class

```
filename: SandwichBar.py

def whichOrder(available, orders):
    """
    return zero-based index of first
    sandwich in orders, list of strings
    that can be made from ingredients
    in available, list of strings
    """
    # you write code here
```

compsci101 fall17

4

# APT SandwichBar

```
available = [ "cheese", "mustard", "lettuce" ]
orders = [ "cheese ham", "cheese mustard lettuce", "ketchup", "beer" ]
Returns: 1

They've run out of ham, but I'll consider other options now.
```

```
available = [ "cheese", "cheese", "cheese", "tomato" ]
orders = [ "ham ham ham", "water", "pork", "bread", "cheese tomato cheese", "beef" ]
Returns: 4

Ignore any duplicate elements in the lists.
```

# APT SandwichBar

[bit.ly/101f17-1026-1](http://bit.ly/101f17-1026-1)

## Step 1: work an example by hand

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

## Step 1: work an example by hand

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

- available = ["cheese", "tomato" ]
- Look orders
  - ["ham ham ham"] to ["ham"] - NO
  - ["water"] - NO
  - ["pork"] - NO
  - ["bread"] - NO
  - ["cheese", "tomato", "cheese"] to ["tomato", "cheese"] - YES!!!
- Return 4

## Step 2: write down algorithm

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

- Get the unique ingredients
  - available = [ "cheese", "tomato" ]
- Look at first order – [ "ham ham ham" ]
  - Make unique – [ "ham" ]
  - Not all ingredients are available
- Look at second order – [ "water" ]
  - Unique, not all ingredients available
- Look at third order – [ "pork" ]
  - Unique, not all ingredients available

9

## Step 2: write down the algorithm

Unique ingredients available = [ "cheese", "tomato" ]  
orders = [ "ham ham ham", "water", "pork", "bread", "cheese tomato cheese", "beef" ]

- Look at 4<sup>th</sup> order - [ "bread" ]
  - Unique, not all ingredients available
- Look at 5<sup>th</sup> - [ "cheese", "tomato", "cheese" ]
  - Make unique - [ "tomato", "cheese" ]
  - "tomato" is in available
  - "cheese" is in available
  - MATCH found return 4 (which is the 5<sup>th</sup> order since we start counting at 0)

10

## Step 3: Generalize algorithm

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

- Get the unique ingredients
- For each order
  - Make unique
  - For each ingredient in order
    - Check if ingredient is in available
  - If all ingredients are available
    - return index number of this order
- Return -1 if no orders matched

11

## Step 4: work another example

available = [ "cheese", "mustard", "lettuce", "mustard" ]  
orders = [ "cheese ham", "ketchup mustard", "cheese mustard lettuce", "beer" ]

- available = [ "cheese", "mustard", "lettuce" ]
- Look orders
  - [ "cheese ham" ] - NO
  - [ "ketchup mustard" ] - NO
  - [ "cheese mustard lettuce" ] - YES!!!
- Return 2

12

## Step 5 – Convert to Code

compsci101 fall17

13

## Problems — snarf setExample.py

- Given a list of strings that have the **name of a course (one word)**, followed by **last names (one word each)** of people in the course:
  - Find total number of people taking any course
  - Find number of people taking just one course

```
["econ101 Abrams Curtson Williams Smith",  
"history230 Black Wrigley Smith", ... ]
```

Process data – create lists of strings of names for each course

compsci101 fall17

14

## Data for example

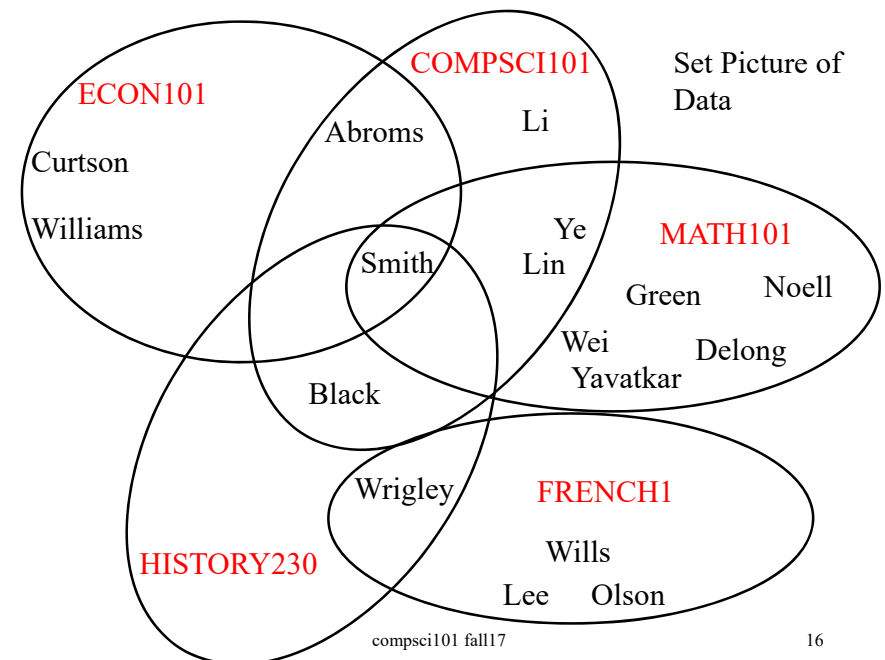
```
["compsci101 Smith Ye Li Lin Abrams Black",  
"math101 Green Wei Lin Williams DeLong Noell Ye Smith",  
"econ101 Abrams Curtson Williams Smith",  
"french1 Wills Wrigley Olson Lee",  
"history230 Black Wrigley Smith" ]
```

TO easier format to work with:

```
[ [ 'Smith', 'Ye', 'Li', 'Lin', 'Abrams', 'Black'],  
  [ 'Green', 'Wei', 'Lin', 'Williams', 'DeLong', 'Noell', 'Ye',  
    'Smith'], [ 'Abrams', 'Curtson', 'Williams', 'Smith'], .... ]
```

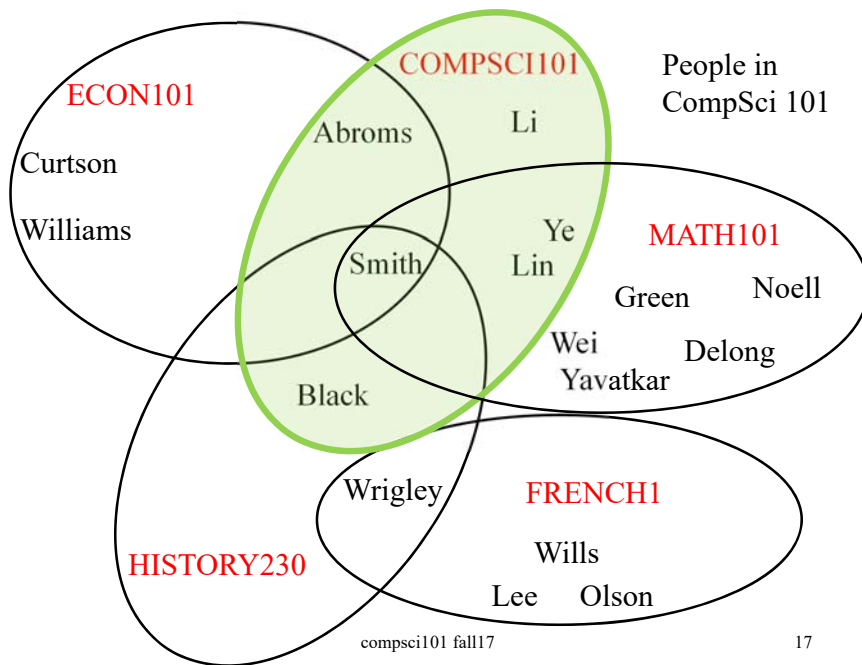
compsci101 fall17

15

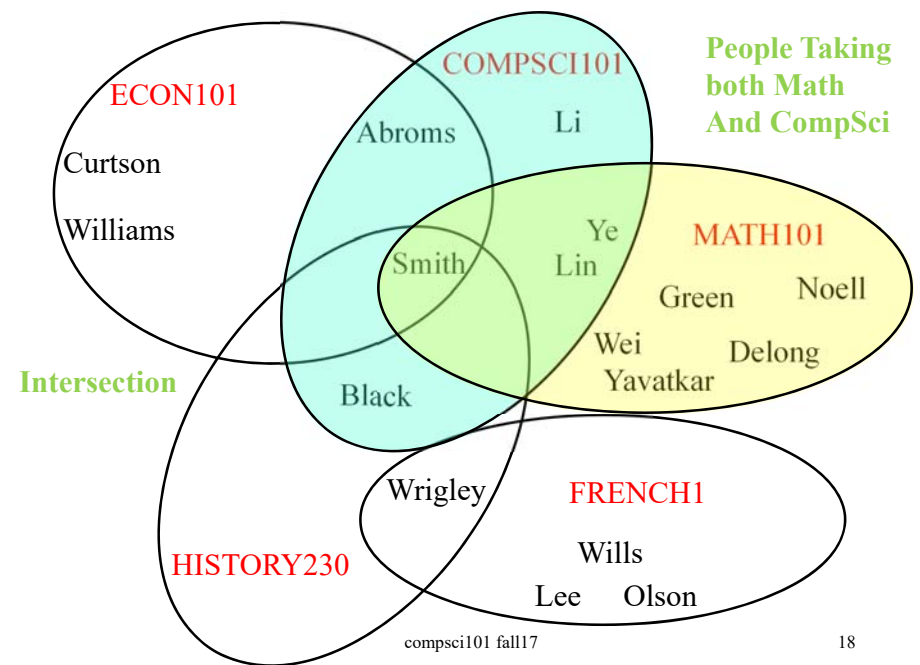


compsci101 fall17

16



17



18

## Part 1 – processList

[bit.ly/101f17-1026-2](http://bit.ly/101f17-1026-2)

- Given a list of strings that have the name of a course (one word), followed by last names of people in the course:
  - Convert list into lists of strings of names for each course

`[ "econ101 Abroms Curtson Williams Smith",  
 "history230 Black Wrigley Smith", ... ]`  
`[ [ 'Abroms', 'Curtson', 'Williams', 'Smith' ],  
 [ 'Black', 'Wrigley', 'Smith', ... ] ]`

compsci101 fall17

19

## Part 2 – peopleTakingCourses

[bit.ly/101f17-1026-3](http://bit.ly/101f17-1026-3)

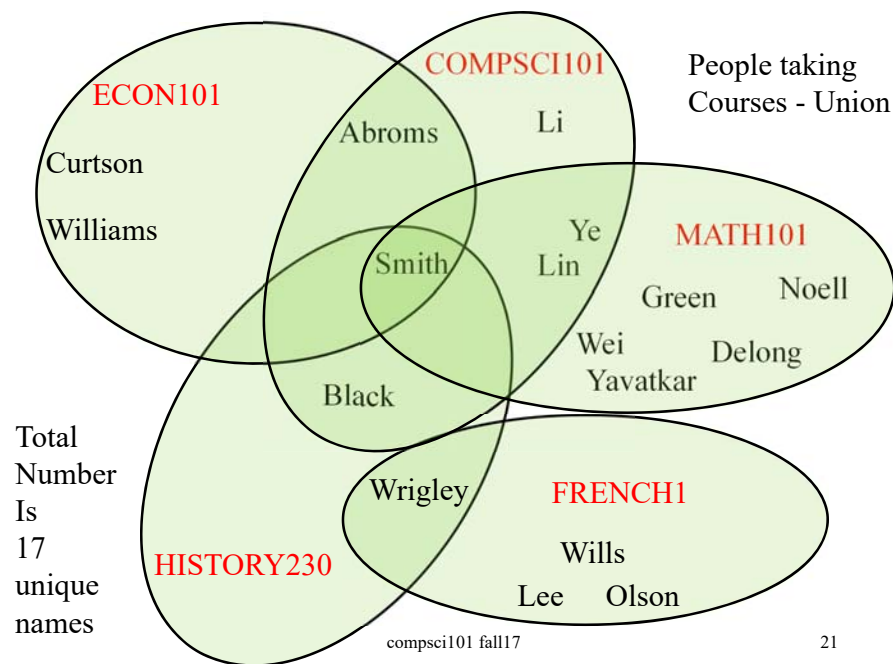
- Given a list of lists of names, each list represents the people in one course:
  - Find total number of people taking any course
  - peopleTakingCourses should return unique list of names
- Small Example
 

```
[[ 'Abroms', 'Curtson', 'Williams', 'Smith' ],  
 [ 'Black', 'Wrigley', 'Smith' ]]
```

Answer is 6 unique names

compsci101 fall17

20

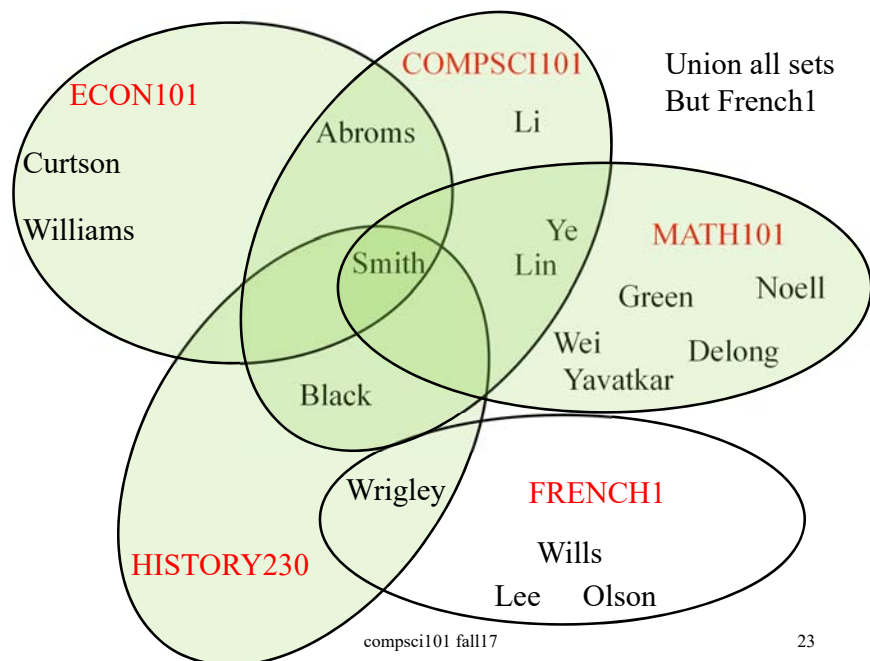


21

Next, find the number of people taking just one course

compsci101 fall17

22



23

To solve this problem

- First let's write a helper function

compsci101 fall17

24

## Part 3 — unionAllSetsButMe

[bit.ly/101f17-1026-4](http://bit.ly/101f17-1026-4)

- Given example, a list of sets of strings, and the index of one of the sets, return the union of all the sets but that one

```
example = [set(["a", "b", "c"]), set(["b", "c",  
"d", "g"]), set(["e", "d", "a"])]
```

unionAllSetsButMe(example,1) is

```
set(["a", "b", "c", "e", "d" ])
```

## Part 4 — peopleTakingOnlyOneCourse

[bit.ly/101f17-1026-5](http://bit.ly/101f17-1026-5)

- Given a list of lists of strings of names representing people from courses
  - Find number of people taking just one course

```
[[ 'Abroms', 'Curtson', 'Williams', 'Smith'],  
 [ 'Black', 'Wrigley', 'Smith', 'Abroms' ]]
```

4

