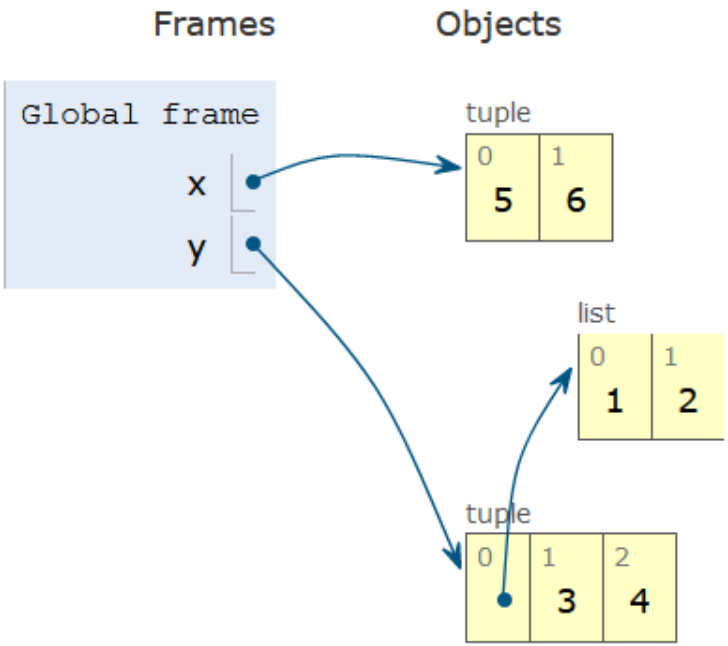
Compsci 101
Images, Tuples, Sets
Live Lecture



Susan Rodger
Nicki Washington
March 11, 2021

PFTD

- Images & Tuples cont.
- Sets and APTs
- Exam 2

Announcements

- APT-4 due today
- Assignment 3 due March 18 (in a week)

- APT-5 out today– due March 23

- Lab 6 this Friday, there is a prelab available now!

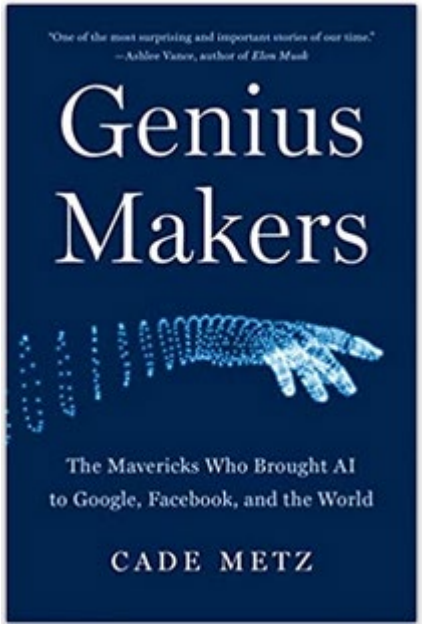
- Exam 2 Tuesday, March 16

Margot Shetterly

- Author of Hidden Figures
- Black Women NASA Scientists
  - Katherine Johnson
  - Mary Jackson
  - Dorothy Vaughn
  - Christine Darden
- Gave a talk at Duke in 2016

Cade Metz – Duke Alum

- English Major at Duke
- Took a lot of CompSci courses
- Now Tech writer for New York Times
- First book: Genius Makers



- Talk at Duke March 18, 7pm
  - Will post zoom link in Piazza

Exam 2 Topics

- Everything from Exam 1
- For loops
- While loops
- Lists
  - Parallel lists, indexing in lists
  - List of lists
  - List comprehensions
- Reading data from files
- Tuples

- NOT ON EXAM 2 - Turtles, Images, Sets

Exam 2 Rules

- This is your own work, no collaboration
- Open book, Open notes

- Do not search for answers on the internet
- Do not type in code where it can be compiled and run
  - Do not use Pycharm, textbook code boxes, Python tutor or any other place the code can be run
- Do not talk to anyone about the exam during the exam, and until it is handed back!

Exam 2 Logistics

- Take on Tues. March 16 between 7am and 11pm
- You pick the start time
  - Must start by 9:15pm
- You get 1 hour 45 min
  - Longer if you have accommodations
- Once you start, your timer starts and you must finish in 1 hour, 45 minutes
- You cannot pause the timer

Exam 2 Logistics (2)

- Go to Gradescope to start
  - login with your netid
  - select the CompSci 101 **Exam** site
    - Different site than where you turn in assignments
- Click on Exam 2 to start
- Gradescope saves answers as you type them in
  - Type 4 spaces to indent code
- Disconnected? Just log back in to Gradescope
- Question? Post a private post on Piazza

Don’t go to Gradescope site until you are ready to start!

You click it, you have started!

We do not restart it!

APT Family

APT: Family

Problem Statement

You have two lists: `parents` and `children`. The ith element in `parents` is the parent of the ith element in `children`. Count the number of grandchildren (the children of a person's children) for the person in the `person` variable.

Hint: Consider making a helper function that returns a list of a person's children.

Step 1: work an example by hand

```
parents = ['Junhua', 'Anshul', 'Junhua', 'Anshul', 'Kerry']
children = ['Anshul', 'Jordan', 'Kerry', 'Paul', 'Kai']
person =  'Junhua'

Returns 3
```

Step 1: work an example by hand

```
parents = ['Junhua', 'Anshul', 'Junhua', 'Anshul', 'Kerry']
children = ['Anshul', 'Jordan', 'Kerry', 'Paul', 'Kai']
person =  'Junhua'

Returns 3
```

- First find the children of Junhua
  - Loop over parents list
    - If name is Junhua add corresponding child to list
      - How do I do that? I need an index (parallel lists)
    - Kids are [‘Anshul’, ‘Kerry’]
  - For each kid:
    - Loop over parents list:
      - If name is kid’s name add their child to the list
        - How do I do that? I need an index (parallel lists)
    - ‘Anshul’s kids -> ‘Jordan’ and ‘Paul’
    - Kerry’s kids -> ‘Kai’
  - Return 3

Step 1: work an example by hand

```
parents = ['Junhua', 'Anshul', 'Junhua', 'Anshul', 'Kerry']
children = ['Anshul', 'Jordan', 'Kerry', 'Paul', 'Kai']
person =  'Junhua'

Returns 3
```

Notice anything?

- First find the children of Junhua
  - Loop over parents list
    - If name is Junhua add corresponding child to list
      - How do I do that? I need an index (parallel lists)
    - Kids are ['Anshul', 'Kerry']
  - For each kid:
    - Loop over parents list:
      - If name is kid's name add their child to the list
        - » How do I do that? I need an index (parallel lists)
    - 'Anshul's kids -> 'Jordan' and 'Paul'
    - Kerry's kids -> 'Kai'
  - Return 3

They are the same!

Write a helper function!

Helper function

```
def childrenOf(parents, children, name):
    <missing code to traverse parallel lists>
    return list of name's children
```

How to traverse parallel lists?

```
parents:   ['Junhua', 'Anshul', 'Junhua', 'Anshul', 'Kerry']
children:  ['Anshul', 'Jordan',  'Kerry',   'Paul',   'Kai']
              0         1         2         3         4
```

How to traverse parallel lists?

```
parents:   ['Junhua', 'Anshul', 'Junhua', 'Anshul', 'Kerry']
children:  ['Anshul', 'Jordan',  'Kerry',   'Paul',   'Kai']
              0         1         2         3         4
```

Iterate over the list – need a loop!

Need to access same position in each list

- need an index

Use a while loop with an index!

How to traverse parallel lists?

```
parents:   ['Junhua', 'Anshul', 'Junhua', 'Anshul', 'Kerry']
children:  ['Anshul', 'Jordan',  'Kerry',    'Paul',     'Kai']
              0          1          2          3          4
```

```
index = 0
while index < len(parents):
    <do something>
     index += 1                # update index
```

Tuple: What and Why?

- Similar to a list in indexing starting at 0
  - Can store any type of element
  - Can iterate over
- Immutable - Cannot mutate/change its value(s)
  - Efficient because it can't be altered
- Consider `x = (5,6)` and `y = ([1,2],3.14)`
  - `x[0] = 7`?
  - `y[0].append(5)`?

Tuple: What and Why?

- Similar to a list in indexing starting at 0
  - Can store any type of element
  - Can iterate over
- Immutable - Cannot mutate/change its value(s)
  - Efficient because it can't be altered
- Consider `x = (5,6)` and `y = ([1,2],3.14)`
  - `x[0] = 7`?
  - `y[0].append(5)`?

ERROR!!!!!

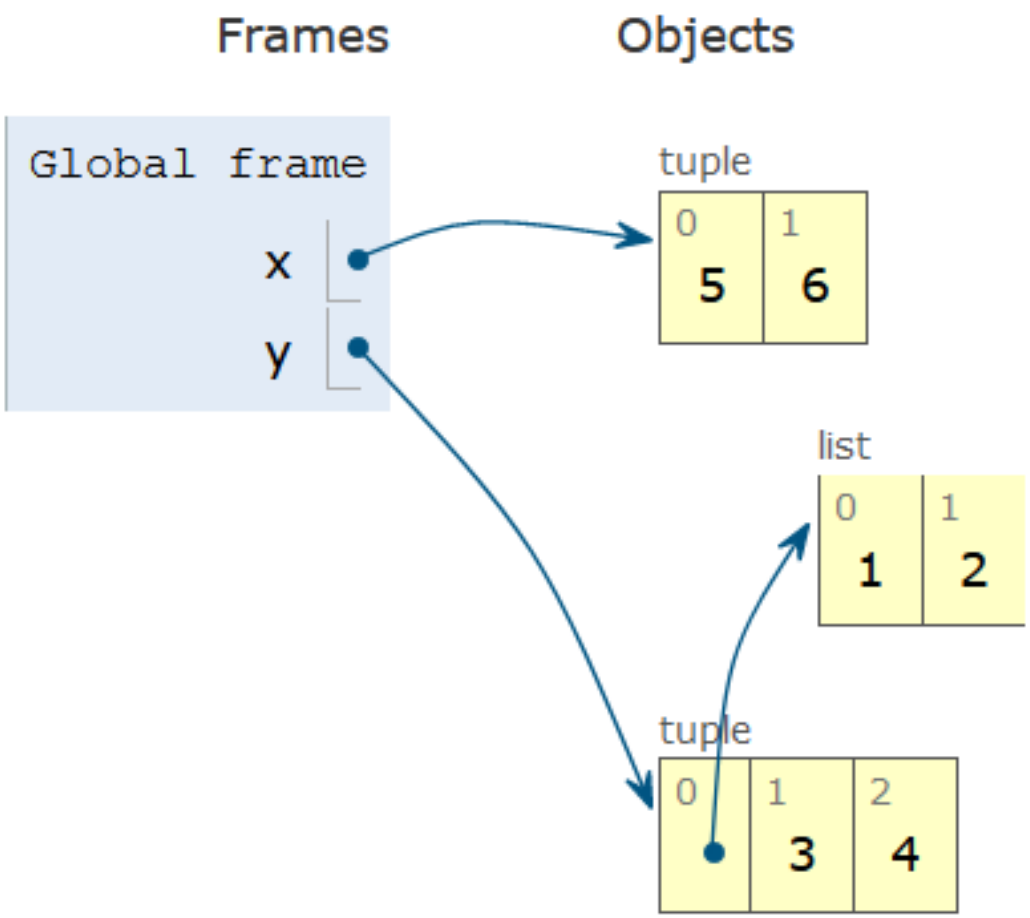
y is ([1,2,5], 3.14)

Example:

x = (5,6)
y = ([1,2], 3, 4)

print(x)
print(y)

y[0][0] = 5
print(y)

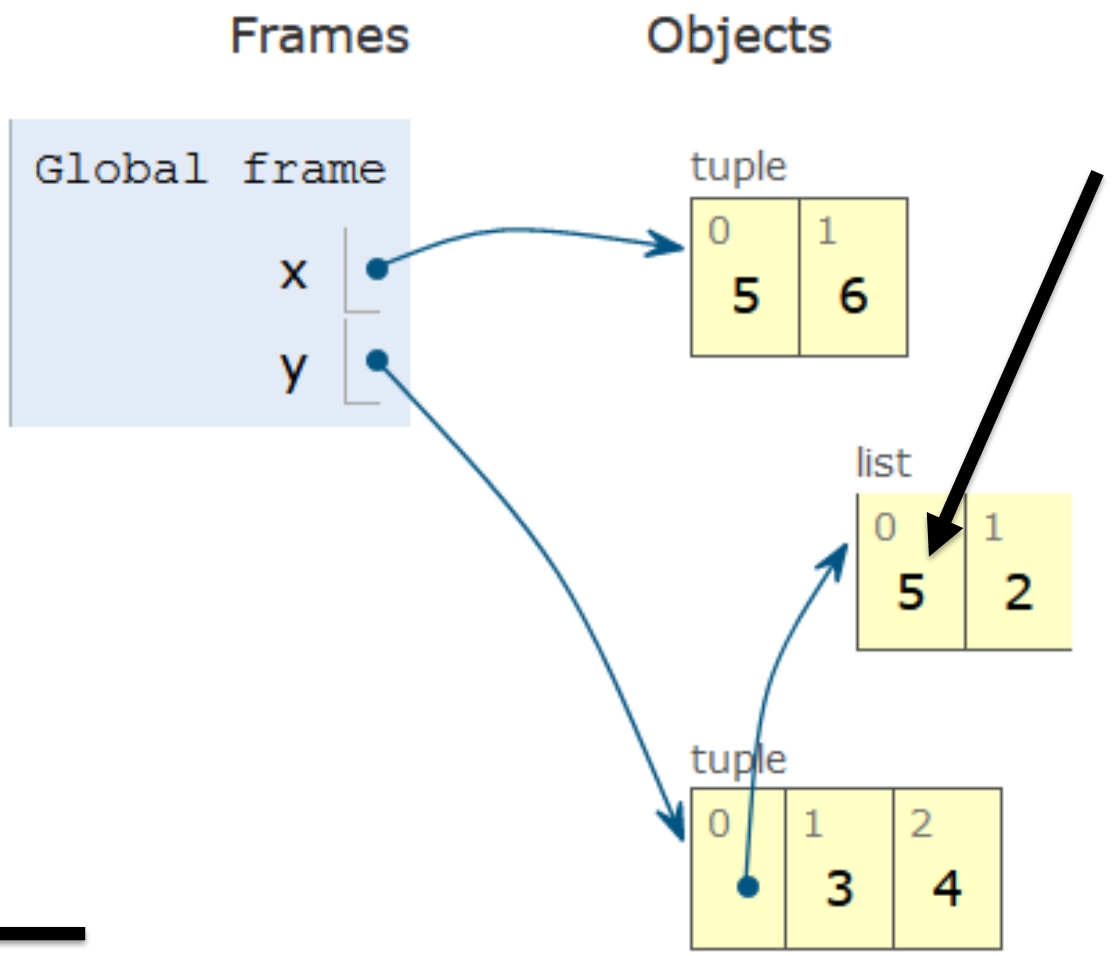


Example:

x = (5,6)
y = ([1,2], 3, 4)

print(x)
print(y)

y[0][0] = 5
print(y)



WOTO-1 Tuples
http://bit.ly/101s21-0311-1

grayByPixel Function

```
13  def grayByPixel(img, debug=False):
14      width = img.width
15      height = img.height
16      new_img = img.copy()
17      if debug:
18          print("creating %d x %d image" % (width,height))
19      for x in range(width):
20          for y in range(height):
21              (r,g,b) = img.getpixel((x,y))
22              grays = getGray(r,g,b)
23              new_img.putpixel((x,y),grays)
24      return new_img
```

getGray function

```
12  def getGray(r,g,b):
13      gray = int(0.21*r + 0.71*g + 0.07*b)
14      return (gray,gray,gray)
```

main

```
36  if __name__ == '__main__':
37      img = Image.open("images/easterеggs.jpg")
38      start = time.process_time()
39      gray_img = grayByPixel(img,True)
40      #gray_img = grayByData(img,True)
41      end = time.process_time()
42      img.show()
43      gray_img.show()
44      print("Time = %1.3f" % (end-start))
```

WOTO-2 GrayScale
http://bit.ly/101s21-0311-2

Make Gray: Notice the Tuples!

```
13  def grayByPixel(img, debug=False):
14      width = img.width
15      height = img.height
16      new_img = img.copy()
17      if debug:
18          print("creating %d x %d image" % (width,height))
19      for x in range(width):
20          for y in range(height):
21              (r,g,b) = img.getpixel((x,y))
22              grays = getGray(r,g,b)
23              new_img.putpixel((x,y),grays)
```

How does this code make a grey image?

New stuff here, what and where?

Revisiting nested Loops

- What is printed here? y varies first
  - Value of x as inner loop iterates?

```
>>> for x in range(5):
...     for y in range(3):
...         print(x, y)
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
3 0
3 1
3 2
4 0
4 1
4 2
```

Why is the first column have the number repeated like that?
What if the print became:
`print(y, x)`?

Make Gray cont.

```
13  def grayByPixel(img, debug=False):
14      width = img.width
15      height = img.height
16      new_img = img.copy()
17      if debug:
18          print("creating %d x %d image" % (width,height))
19      for x in range(width):
20          for y in range(height):
21              (r,g,b) = img.getpixel((x,y))
22              grays = getGray(r,g,b)
23              new_img.putpixel((x,y),grays)
```

If stop code halfway, what half of image is gray?

Tuple

Tuple

Nested Loops

Tuple

How many parameters does putpixel have?

Accessing Individual Pixels is Inefficient

- Accessing each one one-at-a-time is inefficient
  - Python can do better "under the hood"

- PIL provides a function **`img.getdata()`**
  - Returns list-like object for accessing all pixels
  - Similar to how file is a sequence of characters
  - Symmetry: **`img.putdata(sequence)`**

Processing all Pixels at Once

- Treat **`img.getdata()`** as list, it's not quite a list
  - Iterable: object use in "`for … in …`" loop

```
27  def grayByData(img, debug=False):
28      pixels = [getGray(r,g,b) for (r,g,b) in img.getdata()]
29      new_img = Image.new("RGB", img.size)
30      new_img.putdata(pixels)
```

Think: An image is 2D and putdata(seq) takes a 1D sequence. How did we get an image?

Hint: What type are the elements in the list comprehension?

Hint: What do we know about the length of that sequence and the sequence putdata(…) needs?

GrayByData

```
27  def grayByData(img, debug=False):
28      pixels = [getGray(r,g,b) for (r,g,b) in img.getdata()]
29      new_img = Image.new("RGB", img.size)
30      new_img.putdata(pixels)
31      if debug:
32          print("created %d x %d gray image" % (img.width,img.height))
33      return new_img
```

Summary of Image functions

- Many, many more
  - http://bit.ly/pillow-image

Image function/method	Purpose
`im.show()`	Display image on screen
`im.save("foo.jpg")`	Save image with filename
`im.copy()`	Return copy of im
`im.getdata()`	Return iterable pixel sequence
`im.load()`	Return Pixel collection indexed by tuple (x,y)

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 parameter `restaurant`.

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

Specification

```
 filename: EatingGood.py

def howMany(meals, restaurant):
      """
      Parameter meals a list of strings with each in the format
      "name:place-ate". Parameter restaurant is a string
      return # unique name values where place-ate == restaurant
      """

      # you write code here
      return 0
```

Write function `howMany` that given `meals`, a list of strings in the format above indicating where each person ate a meal, and `restaurant`, the name of a restaurant, returns the number of people that ate at least one meal at that restaurant.

APT Eating Good Example

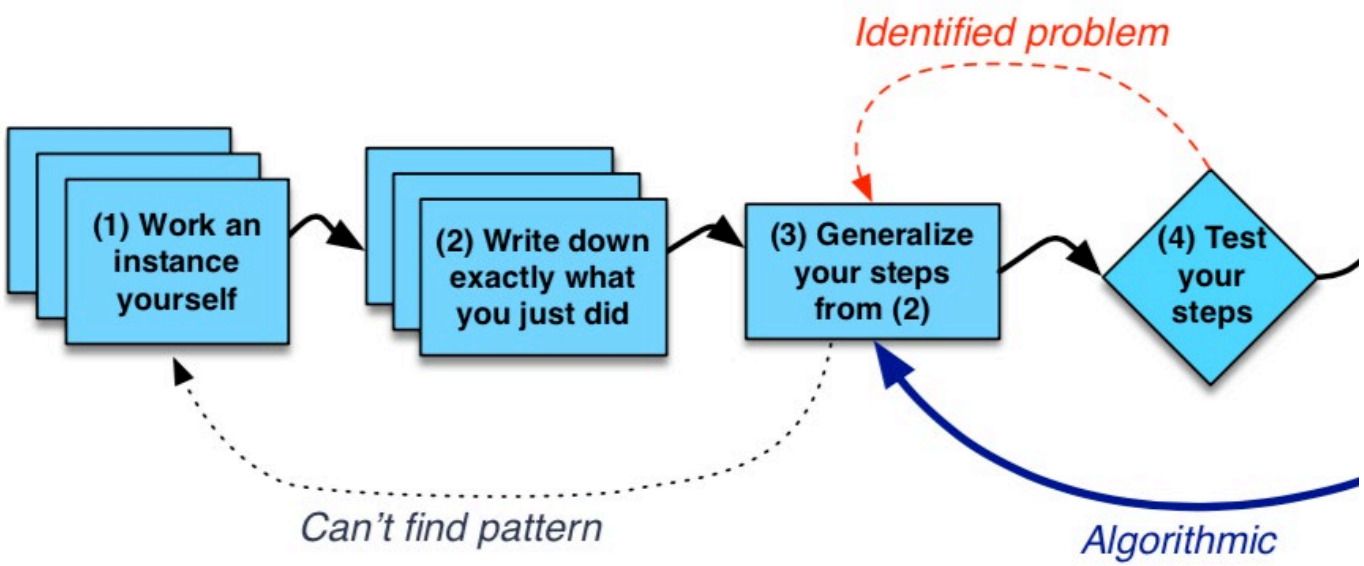
```
meals = ["Sue:Elmos", "Sue:Elmos", "Sue:Elmos"]

restaurant = "Elmos"

returns 1
```

WOTO-3: APT Eating Good
http://bit.ly/101s21-0311-3

- https://www2.cs.duke.edu/csed/pythonapt/eatinggood.html



APT Eating Code Idea

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

APT Eating Code – Use set instead of list

- Make an empty list ← names = set()
- 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

  } names.add(name)
- Return the length of the list ← return len(names)

APT Eating Code – Use set instead of list

- Make an empty set ← names = set()
- Loop over each meal
  - Split the meal into name and restaurant
  - If the restaurant matches
    - Add name to set

  } names.add(name)
- Return the length of the set ← return len(names)

Lists or Set?

```
if name not in names:
    names.append(name)
```

```
names.add(name)
```

- For EatingGood we had to avoid adding the same element more than once
  - Lists store duplicates
  - Sets do not store duplicates

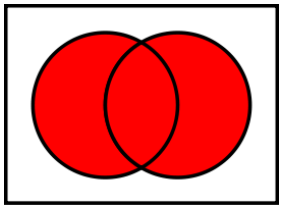
List and Set, Similarities/Differences

	Function for List	Function for Set
Adding element	`x.append(elt)`	`x.add(elt)`
Size of collection	`len(x)`	`len(x)`
Combine collections	`x + y`	`x | y`
Iterate over	`for elt in x:`	`for elt in x:`
Element membership	`elt in x`	`elt in x`
Index of an element	`x.index(elt)`	`CANNOT DO THIS`

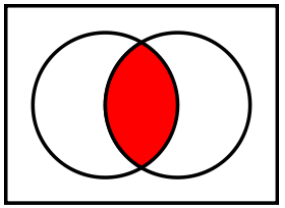
- Lists are ordered and indexed, e.g., has a first or last
- Sets are **not** ordered, very fast, e.g., **`if elt in x`**

Python Set Operators

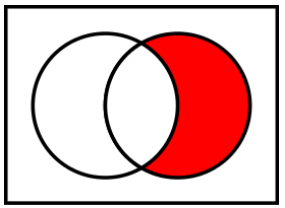
- Using sets and set operations often useful
- A | B, set union
  - Everything



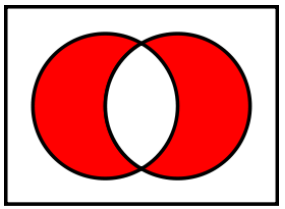
- A & B, set intersection
  - *Only* in both



- B – A, set difference
  - In B *and* not A



- A ^ B, symmetric diff
  - Only in A *or* only in B



WOTO-4 Sets
http://bit.ly/101s21-0311-4