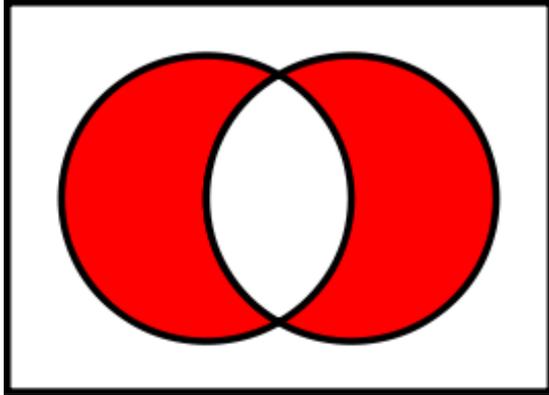


Compsci 101

Images, Tuples, Sets

Part 1 of 2

Susan Rodger
March 11, 2021



N is for ...



- **Nested Loops**
 - All pairs, all pixels, all 2D structures
- **None**
 - Default value for functions if no return
- **Newline**
 - The "\n" in a line

PFTD

- Images & Tuples cont.
- Sets and APTs

Example: Convert Color to Gray



Process each pixel
Convert to gray



First View of Image for Grayscale

- **Image is a collection of pixels**
 - Organized in rows: # rows is image height
 - Each row has the same length: image width
- **Pixels addressed by (x, y) coordinates**
 - Upper-left (0,0), Lower-right (width-1,height-1)
 - Typically is a single (x, y) entity: tuple
- **Tuple is immutable, indexed sequence (a, b, c)**

Let's run it first!



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/eastereggs.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))
```

Richard Stallman

- MacArthur Fellowship
(Genious grant)
- ACM Grace Murray Hopper
award
- Started GNU – Free Software
Foundation (1983)
 - GNU Compiler Collection
 - GNU Emacs

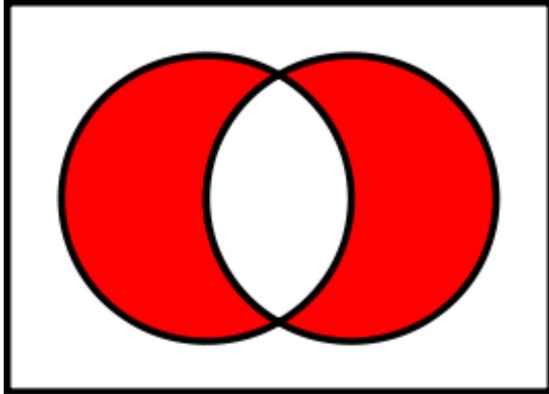


Compsci 101

Images, Tuples, Sets

Part 2 of 2

Susan Rodger
October 6, 2020



Python Sets

- **Set – unordered collection of distinct items**
 - Unordered – can look at them one at a time, but cannot count on any order
 - Distinct - one copy of each
- **Operations on sets:**
 - Modify: add, clear, remove
 - Create a new set: difference(-), intersection(&), union (|), symmetric_difference(^)
 - Boolean: issubset <=, issuperset >=
- **Can convert list to set, set to list**
 - Great to get rid of duplicates in a list

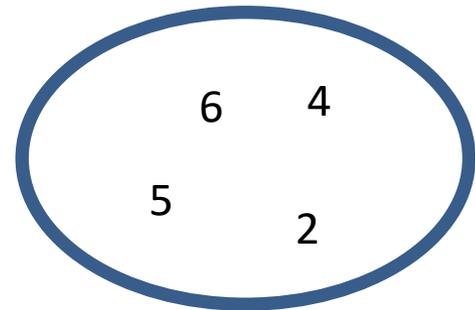
List vs Set

- **List**

- Ordered, 3rd item, can have duplicates
- Example: $x = [4, 6, 2, 4, 5, 2, 4]$

- **Set**

- No duplicates, no ordering
- Example: $y = \text{set}(x)$



- **Both**

- Add, remove elements
- Iterate over all elements

List and Set, Similarities/Differences

	Function for List	Function for Set
Adding element	<code>x.append(elt)</code>	<code>x.add(elt)</code>
Size of collection	<code>len(x)</code>	<code>len(x)</code>
Combine collections	<code>x + y</code>	<code>x y</code>
Iterate over	<code>for elt in x:</code>	<code>for elt in x:</code>
Element membership	<code>elt in x</code>	<code>elt in x</code>
Index of an element	<code>x.index(elt)</code>	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**

Creating and changing a set

```
colorList = ['red', 'blue', 'red', 'red', 'green']
colorSet = set(colorList)
smallList = list(colorSet)
colorSet.clear()
colorSet.add("yellow")
colorSet.add("red")
colorSet.add("blue")
colorSet.add("yellow")
colorSet.add("purple")
colorSet.remove("yellow")
```

smallList =

Creating and changing a set

```
colorList = ['red', 'blue', 'red', 'red', 'green']
colorSet = set(colorList)
smallList = list(colorSet)
colorSet.clear()
colorSet.add("yellow")
colorSet.add("red")
colorSet.add("blue")
colorSet.add("yellow")
colorSet.add("purple")
colorSet.remove("yellow")
```

```
smallList = ['red', 'green', 'blue']
```

order?

```
colorSet =
```

Creating and changing a set

```
colorList = ['red', 'blue', 'red', 'red', 'green']
colorSet = set(colorList)
smallList = list(colorSet)
colorSet.clear()
colorSet.add("yellow")
colorSet.add("red")
colorSet.add("blue")
colorSet.add("yellow")
colorSet.add("purple")
colorSet.remove("yellow")
```

```
smallList = ['red', 'green', 'blue']    order?
colorSet = set(["purple", "red", "blue"]) order?
```

Set Operations – Union and Intersection

```
UScolors = set(['red', 'white', 'blue'])  
dukeColors = set(['blue', 'white', 'black'])  
  
print dukeColors | UScolors  
print dukeColors & UScolors
```

Set Operations – Union and Intersection

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print dukeColors | UScolors
print dukeColors & UScolors
```

```
set(['blue', 'black', 'white', 'red'])
set(['blue', 'white'])
```

Set Operations - Difference

```
UScolors = set(['red', 'white', 'blue'])  
dukeColors = set(['blue', 'white', 'black'])  
  
print dukeColors - UScolors  
print UScolors - dukeColors
```

Set Operations - Difference

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print dukeColors - UScolors
print UScolors - dukeColors
```

set(['black'])

set(['red'])

Set Operations – Symmetric Difference

```
UScolors = set(['red', 'white', 'blue'])  
dukeColors = set(['blue', 'white', 'black'])  
  
print dukeColors ^ UScolors  
print UScolors ^ dukeColors
```

Set Operations – Symmetric Difference

```
UScolors = set(['red', 'white', 'blue'])
dukeColors = set(['blue', 'white', 'black'])

print dukeColors ^ UScolors
print UScolors ^ dukeColors
```

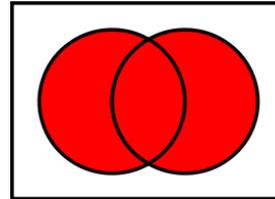
```
set(['black', 'red'])
set(['black', 'red'])
```

Python Set Operators

- Using sets and set operations often useful

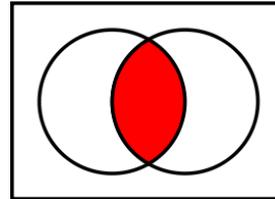
- $A | B$, set union

- Everything



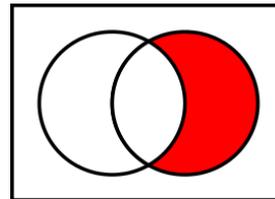
- $A \& B$, set intersection

- *Only* in both



- $B - A$, set difference

- In *B* and not *A*



- $A \wedge B$, symmetric diff

- Only in *A* or only in *B*

