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!
def grayByPixel(img, debug=False):
    width = img.width
    height = img.height
    new_img = img.copy()

    if debug:
        print("creating %d x %d image" % (width, height))

    for x in range(width):
        for y in range(height):
            (r, g, b) = img.getpixel((x, y))
            grays = getGray(r, g, b)
            new_img.putpixel((x, y), grays)

    return new_img
getGray function

```python
def getGray(r, g, b):
    gray = int(0.21*r + 0.71*g + 0.07*b)
    return (gray, gray, gray, gray)
```
```python
if __name__ == '__main__':
    img = Image.open("images/eastereggs.jpg")
    start = time.process_time()
    gray_img = grayByPixel(img,True)
    #gray_img = grayByData(img,True)
    end = time.process_time()
    img.show()
    gray_img.show()
    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

<table>
<thead>
<tr>
<th></th>
<th>Function for List</th>
<th>Function for Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding element</td>
<td><code>x.append(elt)</code></td>
<td><code>x.add(elt)</code></td>
</tr>
<tr>
<td>Size of collection</td>
<td><code>len(x)</code></td>
<td><code>len(x)</code></td>
</tr>
<tr>
<td>Combine collections</td>
<td><code>x + y</code></td>
<td>`x</td>
</tr>
<tr>
<td>Iterate over</td>
<td><code>for elt in x:</code></td>
<td><code>for elt in x:</code></td>
</tr>
<tr>
<td>Element membership</td>
<td><code>elt in x</code></td>
<td><code>elt in x</code></td>
</tr>
<tr>
<td>Index of an element</td>
<td><code>x.index(elt)</code></td>
<td><code>CANNOT DO THIS</code></td>
</tr>
</tbody>
</table>

- 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

```python
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

```python
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
- \( A \cup B \), set union
  - Everything
- \( A \cap B \), set intersection
  - Only in both
- \( B - A \), set difference
  - In B and not A
- \( A \Delta B \), symmetric diff
  - Only in A or only in B