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.

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

Returns 3
```
**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 \( y = ([1,2], 3.14) \)
  - \( x[0] = 7 \)?
  - \( y[0].append(5) \)?

---

**Example:**

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

```
print(x)
print(y)
```

\[ y[0][0] = 5 \]

```
print(y)
```

---

**WOTO-1 Tuples**


---

**grayByPixel Function**

```python
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))
            grayscale = getGray(r,g,b)
            new_img.putpixel((x,y), grayscale)
    return new_img
```

---

3/11/21 Compsci 101, Spring 2021
getGray function

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

main

```python
if __name__ == '__main__':
    img = Image.open("images/easter_eggs.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)")
```

WOTO-2 GrayScale

Make Gray: Notice the Tuples!

```python
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)
```

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?
  ```python
  >>> for x in range(5):
  ...   for y in range(3):
  ...     print(x, y)
  ```
  Why is the first column have the number repeated like that?
What if the print became: print(y, x)?

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(list)`

Processing all Pixels at Once

• Treat `img.getdata()` as list, it's not quite a list
  • Iterable: object used in "for ... in ..." loop

  ```python
  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)
      pixels = [getGray(r,g,b) for (r,g,b) in img.getdata()]
      new_img = Image.new("RGB", img.size)
      new_img.putdata(pixels)
  ```
  Think: An image is 2D and putdata(seq) takes a 1D sequence. How did we get an image?
GrayByData

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

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:Mors" shows that John Doe ate one meal at Mors.

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

**Specification**

```python
filename: EatingGood.py
def howMany(meals, restaurant):
    # Parameter meals is 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
returns 1
```

Summary of Image functions

- Many, many more

<table>
<thead>
<tr>
<th>Image function/method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>im.show()</code></td>
<td>Display image on screen</td>
</tr>
<tr>
<td><code>im.save(&quot;foo.jpg&quot;)</code></td>
<td>Save image with filename</td>
</tr>
<tr>
<td><code>im.copy()</code></td>
<td>Return copy of im</td>
</tr>
<tr>
<td><code>im.getdata()</code></td>
<td>Return iterable pixel sequence</td>
</tr>
<tr>
<td><code>im.load()</code></td>
<td>Return Pixel collection indexed by tuple (x,y)</td>
</tr>
</tbody>
</table>
WOTO-3: APT Eating Good

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

APT Eating Code Idea

- Make an empty list
- Loop over each meal
- Split the meal into person and restaurant
- If the restaurant matches
  - If person not already in list
    - Add person to the list
- Return the length of the list

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>x.append(elt)</td>
<td>x.add(elt)</td>
</tr>
<tr>
<td>Size of collection</td>
<td>len(x)</td>
<td>len(x)</td>
</tr>
<tr>
<td>Combine collections</td>
<td>x + y</td>
<td>x</td>
</tr>
<tr>
<td>Iterate over</td>
<td>for elt in x:</td>
<td>for elt in x:</td>
</tr>
<tr>
<td>Element membership</td>
<td>elt in x</td>
<td>elt in x</td>
</tr>
<tr>
<td>Index of an element</td>
<td>x.index(elt)</td>
<td>CANNOT DO THIS</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

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