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

Compsci 101
Images, Tuples, Sets
Live Lecture

Susan Rodger
Nicki Washington
March 11, 2021

PFTD
• Images & Tuples cont.
• Sets and APTs
• Exam 2
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 = ['Jinhua', 'Anshul', 'Jinhua', 'Anshul', 'Kerry']
children = ['Anshul', 'Jordan', 'Kerry', 'Paul', 'Kai']
person = 'Jinhua'

Returns 3
```
Step 1: work an example by hand

- 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

Helper function

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

How to traverse parallel lists?

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

Notice anything?

- They are the same!

Write a helper function!
How to traverse parallel lists?

parents: ['Junhua', 'Anshul', 'Junhua', 'Anshul', 'Kerry']
children: ['Anshul', 'Jordan', 'Kerry', 'Paul', 'Kai']

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!

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)?

ERROR!!!!!
y is ([1,2,5], 3.14)
Example:

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

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

```python
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))
            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)
```

main

```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))
```

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(sequence)`

Processing all Pixels at Once

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

```python
def grayByPixel(img, debug=False):
    # Other code...
    if debug:
        print("creating %d x %d image" % (width, height))
    else:
        for x in range(width):
            for y in range(height):
                (r, g, b) = img.getpixel((x, y))
                gray = getGray(r, g, b)
                new_img.putpixel((x, y), gray)
```

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
```

### 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>

### 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:Elmos" shows that John Doe ate one meal at Elmos.

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.

```python
meals = ["Sue:Elmos", "Sue:Elmos", "Sue:Elmos"]
restaurant = "Elmos"
returns 1
```
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
  \[
  \text{names} = \text{set}()
  \]
- 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
  - Return the length of the list
    \[
    \text{return} \ \text{len(names)}
    \]
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?

- 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

<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>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
WOTO-4 Sets