Compsci 101
List Comprehensions, Global, Parallel Lists

Susan Rodger
February 15, 2022
K is for …

• **Kernel**
  - Core of the OS, Core for Machine Learning

• **Keyboard - QWERTY or DVORAK**
  - DVORAK:

```
<table>
<thead>
<tr>
<th>~</th>
<th>!</th>
<th>@</th>
<th>#</th>
<th>$</th>
<th>%</th>
<th>^</th>
<th>&amp;</th>
<th>*</th>
<th>(</th>
<th>)</th>
<th>{</th>
<th>}</th>
<th>Backspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-----------</td>
</tr>
<tr>
<td>Caps Lock</td>
<td>A</td>
<td>O</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>D</td>
<td>H</td>
<td>T</td>
<td>N</td>
<td>S</td>
<td></td>
<td>Enter</td>
<td></td>
</tr>
<tr>
<td>Shift</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-----------</td>
</tr>
<tr>
<td>Ctrl</td>
<td>Win Key</td>
<td>Alt</td>
<td></td>
<td>Alt Gr</td>
<td>Win Key</td>
<td>Menu</td>
<td>Ctrl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

• **Key and (Key,Value) pair**
  - Heart of a dictionary
Tiffany Chen

- Duke BS - IDM CS/Biology
- Stanford PhD Biomedical Informatics (CS and Biomedicine)
- Was Director of Informatics, Cytobank
- Now Group Product Manager at Chan Zuckerberg Initiative

“If you are interested in a PhD, I would suggest doing a summer research experience as an undergraduate, but also an internship in industry. You can see how problems are solved in the real world”

“Part of the advantage of being interdisciplinary is that you can see the big picture when no one else can, and you can communicate to everyone else what that big picture is”
Announcements

• Assign 2 – Turtles due tonight!
• APT-3 due Thursday
• Assign 3-Transform out today, due Tuesday, March 1
  • There is a Sakai quiz on Assign3 – Due Feb 28
• Lab 6 Friday - Do prelab before lab
• Exam 2 – one week – in person
• APT Quiz 1 is Feb 24-Feb 27
  • Take during this time
  • Two parts – each part has two APTs
  • Each part is timed
  • More details Thursday
Exam 2 – in person – Tues, Feb 22

- Exam is in class on paper – 10:15am
  - Need pen or pencil
- See materials under 2/22 date
  - Exam 2 Reference sheet - part of exam
- Covers
  - topics /reading through today
  - APTs through APT3
  - Labs through Lab 5, Lab 6 (Part 1 – list comprehensions)
  - Assignments through Assignment 2
  - Concepts from Assign2, No turtles
PFTD

• Pancakes
• Parallel Lists
• List Comprehensions
• Transform Assignment
Pancakes!
APT Pancake

• How do you solve this (or any) problem?
  • 7 Steps!

• Some APTs are hard problems to solve (step 1-4)
  • Translating to code easy

• Some APTs have easy-to-see algorithms (step 5)
  • Translating to code is hard
APT: Pancakes

Problem Statement

You're a short-order cook in a pancake restaurant, so you need to cook pancakes as fast as possible. You have one pan that can fit capacity pancakes at a time. Using this pan you must cook numCakes pancakes. Each pancake must be cooked for five minutes on each side, and once a pancake starts cooking on a side it has to cook for five minutes on that side. However, you can take a pancake out of the pan when you're ready to flip it after five minutes and put it back in the pan later to cook it on the other side.

Write the method, minutesNeeded, that returns the shortest time needed to cook numCakes pancakes in a pan that holds capacity pancakes at once. See the examples.

<table>
<thead>
<tr>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename: Pancakes.py</td>
</tr>
</tbody>
</table>
| def minutesNeeded (numCakes, capacity):
  """
  return integer representing time to cook pancakes based on integer parameters as described below
  """
Examples

1. \(\text{numCakes} = 0\)
   \(\text{capacity} = 4\)

   Returns: 0

   It takes no time to cook 0 pancakes.

2. \(\text{numCakes} = 2\)
   \(\text{capacity} = 2\)

   Returns: 10

   You cook both pancakes on one side for five minutes, then flip them over and cook each on the other side for another five minutes.
Step 1: Solve an instance
Three pancakes in a two-cake pan

• First 5 minutes
  • 2 half cooking
  • 1 uncooked

• Second 5 minutes
  • 2 half cooking
  • 1 almost cooked
Step 1: Solve an instance
Three pancakes in a two-cake pan

• Third 5 minutes
  • 1 done
  • 2 almost cooked

• How many minutes to cook all three pancakes?
Step 1: Solve an instance

• What kind of instances? Simple cases that are quickly solved
  • What are these in Pancake problem?

• Don’t solve for N, solve for 5 (generalize is step 3)
  • What do when there are two parameters?
    • Fix one, vary the other one
  • Helps identify cases
Step 1: Solve an instance

- Pan has capacity 8, vary # pancakes
  - Can you cook 12 in 15 minutes? Why?
  - Can you cook 13 in 15 minutes? Why?

<table>
<thead>
<tr>
<th>cakes</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2/15/22 Compsci 101, Spring 2022
Step 2: What did we just do?

- $13 - 8 = 5$
- $8/2 = 4$ # Can only take off up to half
- $10$ minutes for $8$ pancakes + $10$ minutes for $5$ more pancakes = $20$ minutes
- No, warmer trick won’t work
Step 1: Solve an instance

- Pan capacity 8, vary # pancakes, 17 pancakes?

<table>
<thead>
<tr>
<th>cakes</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 5 3 7 2 6 8
1 4 3 9 2 11 10 12
7 5 6 8
Step 2: What did we just do?

• $17 - 8 = 9$, $9 - 8 = 1$
• $8/2 = 4$
• Is $1 \leq 4$? # Yes, warmer trick will work!
• Total: 25 minutes
  • 10 minutes for 8 pancakes +
  • 5 minutes for 8 pancakes +
  • Take 1 out, start $17^{th}$ pancake
  • 5 minutes finish pancakes 8 to 15 +
  • 5 minutes finish pancake 16 and 17
Step 3: Generalize

• Pan has capacity 8, Generalize to algorithm?

<table>
<thead>
<tr>
<th>cakes</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

2/15/22 Compsci 101, Spring 2022
Step 3: Generalize

• $13 - 8 = 5$
• $8/2 = 4$
• Is $5 \leq 4$?
  • No, warmer trick won’t work
• 10 minutes for 8 pancakes + 10 minutes for 5 more pancakes = 20 minutes

• Remove as many as can with panCapacity
• Will the remainder fit in half the pan?
  • Yes, use warmer
    • 5 minutes instead of 10 for last batch
• No, don’t use warmer
  • 10 minutes for all sets of panCapacity
Step 4: Test Steps

• Remove as many as can with panCapacity

• Will the remainder fit in half the pan?
  • Yes, use warmer
    • 5 minutes instead of 10 for last batch
  • No, don’t use warmer
    • 10 minutes for all sets of panCapacity

• Case 1:
  • cap 17, cook 34
Step 4: Test Steps

- Remove as many as can with panCapacity
- Will the remainder fit in half the pan?
  - Yes, use warmer
    - 5 minutes instead of 10 for last batch
  - No, don’t use warmer
    - 10 minutes for all sets of panCapacity

- Case 1:
  - cap 17, cook 34
  - remainder = 0
  - Edge case! No need for warmer
  - Total: 20 minutes

- Case 2:
  - cap 17, cook 42
Step 4: Test Steps

- Remove as many as can with panCapacity
- Will the remainder fit in half the pan?
- Yes, use warmer
  - 5 minutes instead of 10 for last batch
- No, don’t use warmer
  - 10 minutes for all sets of panCapacity

- Case 1:
  - cap 17, cook 34
  - remainder = 0
  - Edge case! No need for warmer
  - Total: 20 minutes

- Case 2:
  - cap 17, cook 42
  - remainder = 8
  - Yes, use warmer
  - Total: 25 minutes
Step 5: Code

- Remove as many as can with panCapacity
- Will the remainder fit in half the pan?
  - Yes, use warmer
    - 5 minutes instead of 10 for last batch
  - No, don’t use warmer
    - 10 minutes for all sets of panCapacity
- N pancakes
- How many panCapacity can remove?
  - N // panCapacity
  - remainder
  - N % panCapacity
- Half of pan?
  - panCapacity / 2
Let’s code it up!

def minutesNeeded(numCakes, capacity):
    full = numCakes // capacity
    left = numCakes % capacity
    minutes = 10 * full
    if left > capacity/2:
        minutes += 10
    else:
        minutes += 5
    return minutes
Pancake flipping Video
How to teach pancake Flipping

• [http://www.youtube.com/watch?v=W_gxLKSsSIE](http://www.youtube.com/watch?v=W_gxLKSsSIE)
  • For longer, more complex robotic tasks
    • [http://www.youtube.com/watch?v=4usoE981e7I](http://www.youtube.com/watch?v=4usoE981e7I)
Problem

• Given a file of words, which word occurs the most

• For each word count how many times it occurs

• Determine which word has the highest count
Parallel Lists

- We will use parallel lists to track data
  - Each word is stored in a list named `words`
  - Word’s count is stored in a list named `counts`
  - # occurrences of `words[k]` is in `counts[k]`

```
["apple", "fox", "vacuum", "lime"]
[  5,    2,    25,    15  ]
```
Parallel Lists

- We will use parallel lists to track data
  - Each word is stored in a list named `words`
  - Word’s count is stored in a list named `counts`
  - # occurrences of `words[k]` is in `counts[k]`

```
["apple", "fox", "vacuum", "lime"]
[5, 2, 25, 15]
```

- For example: “apple” has been seen five times
Parallel Lists

- We will use parallel lists to track data
  - Each word is stored in a list named `words`
  - Word’s count is stored in a list named `counts`
  - Number of occurrences of `words[k]` is in `counts[k]`

```
["apple", "fox", "vacuum", "lime"]
[   5,      2,      25,      15  ]
```

- For example: “vacuum” has been seen 25 times
Parallel Lists

• We will use parallel lists to track data
  • Each word is stored in a list named `words`
  • Word’s count is stored in a list named `counts`
  • # occurrences of `words[k]` is in `counts[k]`

```
["apple", "fox", "vacuum", "lime"]
[  5,    2,   25,    15  ]
```

• What happens when we read a word?

Read word “vacuum”?
Parallel Lists

• We will use parallel lists to track data
  • Each word is stored in a list named `words`
  • Word’s count is stored in a list named `counts`
  • # occurrences of `words[k]` is in `counts[k]`

```
["apple", "fox", "vacuum", "lime"]
[5, 2, 26, 15]
```

• What happens when we read a word?

Read word “vacuum”? 

Add 1 to `counts[2]`
Parallel Lists

- We will use parallel lists to track data
  - Each word is stored in a list named \texttt{words}
  - Word’s count is stored in a list named \texttt{counts}
  - \# occurrences of \texttt{words[k]} is in \texttt{counts[k]}

\begin{verbatim}
["apple", "fox", "vacuum", "lime"]
[  5,      2,      26,      15  ]
\end{verbatim}

- What happens when we read a word?

Read word “cat”?
Parallel Lists

- We will use parallel lists to track data
  - Each word is stored in a list named *words*
  - Word’s count is stored in a list named *counts*
  - # occurrences of *words[k]* is in *counts[k]*

```
["apple", "fox", "vacuum", "lime", "cat"]
[  5,     2,     26,     15  ]
```

- What happens when we read a word?
  - Read word “cat”?
Parallel Lists

• We will use parallel lists to track data
  • Each word is stored in a list named \texttt{words}
  • Word’s count is stored in a list named \texttt{counts}
  • \# occurrences of \texttt{words[k]} is in \texttt{counts[k]}

\texttt{["apple", "fox", "vacuum", "lime", "cat"]}
\texttt{[ 5, 2, 26, 15, 1 ]}

• What happens when we read a word?

Read word “cat”?
Calculate word most often in file

```python
def wordOcursTheMost(fname):
    f = open(fname)
    words = []
    counts = []
    for line in f:
        line = line.strip()  # remove newline
        data = line.split()
        for word in data:
            if word not in words:
                words.append(word)
                counts.append(1)
            else:  # update word
                pos = words.index(word)
                counts[pos] += 1
    f.close()
```
Calculate word most often in file

```python
def wordOccursTheMost(fname):
    f = open(fname)
    words = []
    counts = []
    for line in f:
        line = line.strip()  #remove newline
        data = line.split()
        for word in data:
            if word not in words:
                words.append(word)
                counts.append(1)
            else:  # update word
                pos = words.index(word)
                counts[pos] += 1
    f.close()
```

For each word
Add word if not there, with a new count of 1
word is there, update count

How do you finish the function?
Calculate word most often in file

- words is list of all the words from the file
- counts is the count of each word in the file
- Find the largest count value
  \[
  \text{maxcount} = \max(\text{counts})
  \]
- Find index location of largest count value
  \[
  \text{maxpos} = \text{counts}.\text{index}(\text{maxcount})
  \]
- Return word in same location
  \[
  \text{return words}[\text{maxpos}]
  \]
Complete function:

def wordOccursTheMost(fname):
    f = open(fname)
    words = []
    counts = []
    for line in f:
        line = line.strip()  #remove newline
        data = line.split()
        for word in data:
            if word not in words:
                words.append(word)
                counts.append(1)
            else:  # update word
                pos = words.index(word)
                counts[pos] += 1
    f.close()
    maxcount = max(counts)
    maxpos = counts.index(maxcount)
    return words[maxpos]
List Comprehension
Accumulator in one line

```python
def onlyPos(nums):
    ret = []
    for n in nums:
        if n > 0:
            ret.append(n)
    return ret

print(onlyPos([1,2,3,-1,-2,-3]))
```

return \([n \text{ for } n \text{ in } \text{nums if } n > 0]\)

• List Comprehension
  • We will use a complete, but minimal version of list comprehensions, much more is possible
List Comprehension Syntax

- \( V \) is any variable: all list elements in order
- \( V_{\text{EXP}} \) is any expression, often use \( V \)

```python
ret = []
for V in LIST:
    ret.append(V_EXP)
```

```python
ret = [V_EXP for V in LIST]
```

```python
ret = []
for V in LIST:
    if BOOL_EXP:
        ret.append(V_EXP)
```

```python
ret = [V_EXP for V in LIST if BOOL_EXP]
```
List Comprehension Syntax

- if part optional - **BOOL_EXP** is a Boolean expression usually using **V**

```
ret = []
for V in LIST:
    ret.append(V_EXP)
```

```
ret = []
for V in LIST:
    if BOOL_EXP:
        ret.append(V_EXP)
```

```
ret = [V_EXP for V in LIST]
```

```
ret = [V_EXP for V in LIST if BOOL_EXP]
```
List Comprehension Examples

```python
print( [n*2 for n in range(6)] )
```

```python
print( [n for n in range(10) if n % 2 == 1] )
```
List Comprehension Examples

print([n*2 for n in range(6)])

[0, 2, 4, 6, 8, 10]

print([n for n in range(10) if n % 2 == 1])

[1, 3, 5, 7, 9]
List Comprehension Examples

print( [n/2 for n in range(10) if n % 2 == 0] )

lst = ['banana', 'pineapple', 'apple']
print( [c for c in lst if 'n' in c] )
List Comprehension Examples

print([n/2 for n in range(10) if n % 2 == 0])

[0, 1, 2, 3, 4]

lst = ['banana', 'pineapple', 'apple']
print([c for c in lst if 'n' in c])

['banana', 'pineapple']
WOTO-3 List Comprehension Examples
WOTO-3 List Comprehension Example

```python
words = ['giraffe', 'zebra', 'ant', 'lion', 'elephant']
x = [2*x for x in [len(w) for w in words if len(w)>3] if x%2==0]
```

Don’t do this!!!

```python
words = ['giraffe', 'zebra', 'ant', 'lion', 'elephant']
y = [len(w) for w in words if len(w) > 3]
x = [2*x for x in y if x%2==0]
```

Break it up to two list comprehensions

```
y is [7, 5, 4, 8]  
x is [8, 16]
```
Assignment 3: Transform

• Reading and writing files
  • We've seen how to read, writing is similar
  • Open, read, and close
  • Open, write, and close - .write(…)

• Apply a function to every word in a file
  • Encrypt and decrypt
  • Respect lines, so resulting file has same structure
Encrypting and Decrypting

• We give you:
  • Transform.py
  • Vowelizer.py - Removes vowels, then re-vowelize

• You implement
  • Pig Latin
  • Caesar cipher

• Challenge: Shuffleizer
Concepts in Starter Code

- Global variables
  - Generally avoided, but very useful
  - Accessible in all module functions

- FileDialog and tkinter
  - API and libraries for building UI and UX

- Docstrings for understanding!

Look at code
Transform – Remove Vowels

• First line of twain.txt:
1 The Notorious Jumping Frog of Calaveras County

• Run Transform.py on twain.txt

• Set as:
  
  doTransform("-nvw", Vowelizer.encrypt)
  
  #doTransform("-rvw", Vowelizer.decrypt)

• Results in new file: twain-nvw.txt

• First line of twain-nvw.txt is:
1 Th Ntrs Jmpng Frg f Clvrs Cnty
Transform – Get vowels back?

- First line of twain-nvw.txt:

| 1 | Th Ntrs Jmpng Frg f Clvrs Cnty |

- Run Transform.py on twain-nvw.txt

- Set as:

```python
#doTransform("-nvw", Vowelizer.encrypt)
doTransform("-rvw", Vowelizer.decrypt)
```

- Results in new file: twain-nvw-rvw.txt
- First line of twain-nvw-rvw.txt is:

| 1 | oath antares jumping fargo fe cleavers county |
Transform – Vowels summary

• First line in twain.txt

1. The Notorious Jumping Frog of Calaveras County

• After removing vowels – “encrypt”

1. Th Ntrs Jmpng Frg f Clvrs Cnty

• After trying to re-vowelize – “decrypt”

1. oath antares jumping fargo fe cleavers county