Lab 4 Part 2

- Can revise **Part 1 voting.py** module
  - Have returned initial feedback as **part1_feedback.txt**
  - We will run additional tests after Part 2 is submitted
- Use your **voting** module, which contains useful functions for working with election data, to experiment with different election algorithms
- Implement **plurality**, **borda**, and **ranked choice voting**
  - **Heads up**: **rankedChoice** you need a while loop
  - Think about what should the "continuation condition" be of the loop
- Try the different algorithms on our class’s ice cream preferences!
  - Find out the winning flavor?
  - Is there consensus between voting rules?
Understanding Lab Feedback

• +, −, ~ in Gradesheet.txt

• + requirement meets all expectations: is perfectly addressed

• ~ is sort of like +/-, some requirements met expectations, but some things did not meet expectations (can be improved)

• − did not meet our expectations

• Look for inline comments in Gradesheet.txt in your python scripts: they start with #$
Viewing In Line Comments

Can choose side by side view or in line view
### View In Line Comments:

#### In Line View:
- **goodbye.py**
  
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+ # $ no user input requested</td>
<td>+1 -0</td>
</tr>
<tr>
<td>2</td>
<td><code>print(&quot;goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Side by Side View:
- **hello.py**
  
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+ # $ well done</td>
<td>+2 -0</td>
</tr>
<tr>
<td>2</td>
<td><code># Write your first program in this file.</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>print(&quot;Hello, World!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>

---

- **goodbye.py**
  
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>print(&quot;goodbye!&quot;)</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>print(&quot;goodbye!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>

- **hello.py**
  
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+ # $ well done</td>
<td>+2 -0</td>
</tr>
<tr>
<td>2</td>
<td><code># Write your first program in this file.</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>print(&quot;Hello, World!&quot;)</code></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>print(&quot;Hello, World!&quot;)</code></td>
<td></td>
</tr>
</tbody>
</table>
Simplify Common Coding Patterns

- Simplifying Boolean expressions:
  
  ```python
  if isIsogram(word) == True:
  
  Can just say
  
  if isIsogram(word):
  
  Similarly,
  
  if len(word) == len(uniques(word)):
    return True
  
  else:
    return False
  
  Can just say return len(word) == len(uniques(word))
  ```
Simplify Common Coding Patterns

• **Don't:** Iterate over a function call returning a sequence
  for word in sized(6, readWords(filename)):
    for word in sized(7, readWords(filename)):
      ...

• **Do:** Define sequence before for loop
  
  words6 = sized(6, readWords(filename))
  words7 = sized(7, readWords(filename))
  for word in words6:
    for word in words7:
      ...
  ...
Avoid Inefficient Coding Practices

- Consecutive for loops for performing multi-step operations
  - Can perform multiple operations inside a single for loop

```python
newWordList = []
for word in wordList1:
    newWordlist.append(word + 'r')
for word1 in newWordList:
    for word2 in wordList2:
        if(canon(word1) == canon(word2):
            do something
```
Avoid Inefficient Coding Practices

• Consecutive for loops for performing multi-step operations
  • Can perform multiple operations inside a single for loop

```python
for word1 in wordList1:
    newWord = word1 + 'r'
    for word2 in wordList2:
        if (canon(newWord) == canon(word2)):
            do something
```
Functions Should Have a Default Return

- If your return statement is within a conditional be careful
  - Think: is that return statement always reached?
  - What happens if it is not reached?
  - Is your function sometimes returning None?

def foo(word, wordList):
    if word in wordList:
        return canon(word)
    return ''  # return the default type
Functions Should Have a Default Return

- If your return statement is within a conditional be careful
  - Think: is that return statement always reached?
  - What happens if it is not reached?
  - Is your function sometimes returning None?

```python
def foo(word, wordList):
    result = '' # use variables
    if word in wordList:
        result = canon(word)
    return result
```
Other Style Issues

- Common style issues:
  - Use camelCase for multi-word variable names
  - 80 character limit per line for code and comments
  - Use descriptive variable names
  - Make good use of vertical and horizontal white space for readability
    - Leave a space around operators like +, -
    - Space after # in comments
    - Space around commas, etc
    - Line breaks also help
Testing Your Code

• **Doctests**
  
  • Think about writing tests for common cases first
    
    • Use pencil and paper to think through elections so you know what the expected output should be
  
  • Also think about edge cases (empty lists, ties, etc)

• Expected output in doctests must be formatted correctly (strings need ‘ ’, check your []s, etc)
Getting Help

• Herd TAs will contact you if they need to reschedule
• TA hours will be held next week
• Check course webpage for instructor office hours
• **Good luck and enjoy reading days!**