CSc 110, Autumn 2016

Lecture 20: Lists for Tallying; Text Processing

Adapted from slides by Marty Stepp and Stuart Reges
Value/Reference Semantics

- Variables of type int, float, boolean, store values directly:
  
  ```
  age 20
cats 3
  ```

- Values are copied from one variable to another:
  
  ```
  cats = age
  ```
  
  ```
  age 20
cats 20
  ```

- Variables of object types store references to memory:
  
  ```
  grades
  ```
  
  ```
  index
  value 89 78 93
  ```

- References are copied from one variable to another:
  
  ```
  scores = grades
  ```
  
  ```
  scores
  ```
A multi-counter problem

• Problem: Write a function `most_frequent_digit` that returns the digit value that occurs most frequently in a number.

• Example: The number 669260267 contains:
  one 0, two 2s, four 6es, one 7, and one 9.
  `most_frequent_digit(669260267)` returns 6.

• If there is a tie, return the digit with the lower value.
  `most_frequent_digit(57135203)` returns 3.
A multi-counter problem

• We could declare 10 counter variables ... 
  *counter0, counter1, counter2, counter3, counter4, 
  *counter5, counter6, counter7, counter8, counter9

• But a better solution is to use a list of size 10.
  • The element at index $i$ will store the counter for digit value $i$.
  • Example for 669260267:

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• How do we build such an list? And how does it help?
# Returns the digit value that occurs most frequently in n.
# Breaks ties by choosing the smaller value.
def most_frequent_digit(n):
    counts = [0] * 10
    while (n > 0):
        digit = n % 10  # pluck off a digit and tally it
        counts[digit] += 1
        n = n / 10

    # find the most frequently occurring digit
    best_index = 0
    for i in range(1, len(counts)):
        if (counts[i] > counts[best_index]):
            best_index = i
    return best_index
Section attendance question

• Read a file of section attendance (see next slide):

    yynyyynayayynyyayanyyyaynayyayyanayyanyyna
    ayyanyyyyayanaynaayyanaayyanaayyayyanayyyna
    yyayaynnyyayanyynnyyayyanayyanaynannnyyayyayny

• And produce the following output:

Section 1
Student points: [20, 16, 17, 14, 11]
Student grades: [100.0, 80.0, 85.0, 70.0, 55.0]

Section 2
Student points: [16, 19, 14, 14, 8]
Student grades: [80.0, 95.0, 70.0, 70.0, 40.0]

Section 3
Student points: [16, 15, 16, 18, 14]
Student grades: [80.0, 75.0, 80.0, 90.0, 70.0]

• Students earn 3 points for each section attended up to 20.
Each line represents a section.
A line consists of 9 weeks' worth of data.
  • Each week has 5 characters because there are 5 students.
  • Within each week, each character represents one student.
    • a means the student was absent (+0 points)
    • n means they attended but didn't do the problems (+1 points)
    • y means they attended and did the problems (+3 points)
def main():
    file = open("sections.txt")
    lines = file.readlines()
    section = 1
    for line in lines:
        points = [0] * 5
        for i in range(0, len(line)):
            student = i % 5
            earned = 0
            if (line[i] == 'y'):  # c == 'y' or 'n' or 'a'
                earned = 3
            elif (line[i] == 'n'):
                earned = 1
            points[student] = min(20, points[student] + earned)
        grades = [0] * 5
        for i in range(0, len(points)):
            grades[i] = 100.0 * points[i] / 20
        print("Section " + str(section))
        print("Student points: " + str(points))
        print("Student grades: " + str(grades))
        print()
        section += 1
Data transformations

• In many problems we transform data between forms.
  • Example: digits $\rightarrow$ count of each digit $\rightarrow$ most frequent digit
  • Often each transformation is computed/stored as an list.
  • For structure, a transformation is often put in its own function.

• Sometimes we map between data and list indexes.
  • by position (store the $i^{\text{th}}$ value we read at index $i$)
  • tally (if input value is $i$, store it at array index $i$)
  • explicit mapping (count 'J' at index 0, count 'X' at index 1)

• Exercise: Modify our Sections program to use functions that use lists as parameters and returns.
This program reads a file representing which students attended which discussion sections and produces output of the students' section attendance and scores.

```python
def main():
    file = open("sections.txt")
    lines = file.readlines()
    section = 1
    for line in lines:
        # process one section
        points = count_points(line)
        grades = compute_grades(points)
        results(section, points, grades)
        section += 1

    # Produces all output about a particular section.
def results(section, points, grades):
        print("Section " + str(section))
        print("Student scores: " + str(points))
        print("Student grades: " + str(grades))
        print()

    ...
```
# Computes the points earned for each student for a particular section.
def count_points(line):
    points = [0] * 5
    for i in range(0, len(line)):
        student = i % 5
        earned = 0
        if (line[i] == 'y'):
            # c == 'y' or c == 'n'
            earned = 3
        elif (line[i] == 'n'):
            earned = 2
        points[student] = min(20, points[student] + earned)
    return points

# Computes the percentage for each student for a particular section.
def compute_grades(points):
    grades = [0] * 5
    for i in range(0, len(points)):
        grades[i] = 100.0 * points[i] / 20
    return grades