Designing Programs for Tables

CMPU 101 – Problem Solving and Abstraction

Peter Lemieszewski
Semester Recap:

1. We can represent complex data as tables...
   - encoded directly in a program or loaded from an external source.
   - Real data may need (automatic/manual) clean-up.

2. 
   - 
   - 
   - 
   - 

3. 
   - 

4. 

5. 

Semester Recap:

1. We can represent complex data as tables...
   • encoded directly in a program or loaded from an external source.
   • Real data may need (automatic/manual) clean-up.

2. We can use *sanitizers* for automatic data clean-up to...
   • Ensure all data in a column is of the desired type, with default values for *null data*.
   • But “real data” sets can be much harder to work with than contrived examples:
     • Missing values
     • Inconsistent entry of data
     • Differing levels of precision (dates like: 1987 vs 7 July 1987)

3. 

4. 

5. 
Semester Recap:

1. We can represent complex data as tables...
   - encoded directly in a program or loaded from an external source.
   - Real data may need (automatic/manual) clean-up.

2. We can use *sanitizers* for automatic data clean-up to...
   - Ensure all data in a column is of the desired type, with default values for null data.
   - But “real data” sets can be much harder to work with than contrived examples:
     - Missing values
     - Inconsistent entry of data
     - Differing levels of precision (dates like: 1987 vs 7 July 1987)

3. We can *modify* table data by hand
   - This begs the question: Should we modify table data by hand?

4. 

5. 
Semester Recap:

1. We can represent complex data as tables...  
   • encoded directly in a program or loaded from an external source.  
   • Real data may need (automatic/manual) clean-up.

2. We can use sanitizers for automatic data clean-up to...  
   • Ensure all data in a column is of the desired type, with default values for null data.  
   • But “real data” sets can be much harder to work with than contrived examples:  
     • Missing values  
     • Inconsistent entry of data  
     • Differing levels of precision (dates like: 1987 vs 7 July 1987)

3. We can modify table data by hand  
   • This begs the question: Should we modify table data by hand?

4. We can modify table data later using transform-column

5. 
Semester Recap:

1. We can represent complex data as tables...
   • encoded directly in a program or loaded from an external source.
   • Real data may need (automatic/manual) clean-up.

2. We can use *sanitizers* for automatic data clean-up to...
   • Ensure all data in a column is of the desired type, with default values for null data.
   • But “real data” sets can be much harder to work with than contrived examples:
     • Missing values
     • Inconsistent entry of data
     • Differing levels of precision (dates like: 1987 vs 7 July 1987)

3. We can *modify* table data by hand
   • This *begs the question*: Should we modify table data by hand?

4. We can *modify* table data later using *transform-column*

5. We can *remove* (apparent) bad data using *filter-with*.
Last Friday’s Lab

• We saw this clean-up process in our lab by looking at the student data from the form (none of) you filled out.
  • That’s because I never sent out the email to have you fill out the form.
• Let’s continue to use this data set though...
Task plans

• If you aren’t sure how to approach a problem, utilize a set of procedures to design a solution & identify code you need to write:

1. Develop a concrete example of desired output
   • Typically, a table with 4–6 rows

2. Identify functions useful to transform data
   • Functions you already know or look up in the documentation

3. Develop a sequence of steps to transform data
   • Draw as pictures, use textual descriptions, or a combination of the two
   • Use functions from previous step

4. Repeat Step 3 to further break down steps until it is easy to write expressions/functions for each step
Example: “Bin”ning

• How should we consider the distribution of responses to this question...

Would you classify your academic focus as humanities, STEM, or somewhere in the middle? *

1 2 3 4 5 6 7 8 9 10

Super humanities 〇 〇 〇 〇 〇 〇 〇 〇 〇 〇

Super STEM
• We don’t particularly care about how many students rated their STEM-iness as 2 or 8 or any particular number.
• Instead, we might want to bin the responses into a few categories.
Example: “Bin”ning
Example: “Bin”ning

1 3  
non-STEM

7
STEM

8 10  
super-STEM
task plan

• To count the number of students in these three categories

• SNAPSHOT: TBD (Let’s develop one together)
task plan

• To count the number of students in these three categories

• SNAPSHOT: TBD (Let’s develop one together)
More formally,

1. Write stem-category.
2. Add stem category to table using build-column.
3. Summarize results using count.
4. Visualize the results using pie-chart.
Let’s develop code + tests (1ˢᵗ requires a table)

```
test-table =
  table: stem-level
    row: 1
    row: 3
    row: 4
    row: 7
    row: 8
    row: 10
  end
```

The test table can omit the columns we’re not using!
test-table =
    table: stem-level
    row: 1
    row: 3
    row: 4
    row: 7
    row: 8
    row: 10
end

fun stem-category(r :: Row) -> String:
    doc: "Return a stem category (non-stem, stem, or super-stem) for a given stem-level"
    #tbd...
where:
    stem-category(test-table.row-n(0)) is "non-stem"
    stem-category(test-table.row-n(1)) is "non-stem"
    stem-category(test-table.row-n(2)) is "stem"
    stem-category(test-table.row-n(3)) is "stem"
    stem-category(test-table.row-n(4)) is "super-stem"
    stem-category(test-table.row-n(5)) is "super-stem"
end

The test table can omit the columns we’re not using!

If the survey data changes, our tests will still pass!
fun stem-category(r :: Row) -> String:
  doc: "Return a stem category (non-stem, stem, super-stem) for a given stem-level"
  s = r["stem-level"]
  if s < 4:
    "non-stem"
  else if s < 8:
    "stem"
  else:
    "super-stem"
end
where:
#tests on previous slide...
(3rd build column called stem-category++)

data-stem-category = 
    build-column(student-data-cleaned, 
        "stem-category", stem-category)

# the ++ part
# count the population in each category
counts = 
    count(data-stem-category, "stem-category")

# then provide visual representation
pie-chart(counts, "value", "count")
Nested Functions

(Optional) Part 3: Going further

Congratulations! You have reached the end of lab. Here is an optional exercise in case you are looking for a challenge:

**TASK:** Write a function `percent-true` that takes a table and column name as input and returns the percent of rows that are `true` for the column specified.

```haskell
fun percent-true :: Table, col :: String -> Number
  doc: "Returns the percentage of rows that are true in column 'col'"
  ...
end
```

What’s neat about this function is it will work on any table that has a column of type `Bool`!

**TASK:** Use this helper function to find the percentage of survey responders who are student-athletes. Check to see if it’s the same answer you got for Part 2.1.
One approach to student-athletes question

fun percent-true(t :: Table, col :: String) -> Number:

    doc: "Return the percentage of rows that are true in column 'col"

    ...

end
fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col"
  filter-with(t, ???).length() / t.length()
end

#?? -> need a helper function here to get us the columns with true
A(n incorrect) helper function for precent-true

(A few students ran into this exact problem on Friday!)

fun true-filter(r :: Row) -> Boolean:
  doc: "Return true if 'col' is true in this row"
  r[col] #more like return value of column!
end

fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col"
  filter-with(t, true-filter).length() / t.length()
end
What is wrong with this approach

- `col` is undefined in `true-filter`
- Pyret only knows the value for `col` when you’re “inside” `percent-true`
- This means we need to define `true-filter “inside” percent-true`
  - i.e. nest the helper function
A (correct) helper function for precent-true

fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col’’"
  #nest true filter within percent-true & before actual code
  fun true-filter(r :: Row) -> Boolean:
    r[col]
  end

  filter-with(t, true-filter).length() / t.length()
end
Add a test table to complete the solution

• As usual, let’s test our function using a simple test table:
  
  \[ \text{test-table-student-athlete} = \]
  
  \[
  \text{table:}
  
  \text{student-athlete}
  
  \text{row: true}
  
  \text{row: false}
  
  \text{end}
  
  \]

  \[
  \text{fun percent-true(t :: Table, col :: String) -> Number:}
  
  \text{# ...}
  
  \text{where:}
  
  \text{percent-true(test-table-student-athlete, "student-athlete") is 0.5}
  
  \text{end}
  
  \]
Q: When do you need to nest a helper function?

A: if that function needs data that can’t be passed in directly to the function.
Access to the code from this lecture

Includes virtually all suggested lab solutions!

https://code.pyret.org/editor#share=1WXx7yJvtOKJtXjza0CdCi8gdtozF8ZnR&v=31c9aaf
Acknowledgements

• This lecture incorporates material from:
  • Kathi Fisler, Brown University,
  • Gregor Kiczales, University of British Columbia,
  • And, Jonathan Gordon