Coming Attractions
Lambdas & Lists

CMPU 101 — Problem Solving and Abstraction

Peter Lemieszewski
Function Call vs. Inline Function

- A **function call** is, essentially, a “break in the action” for a CPU
  - Such that it might take a moment to find out where that function actually is:
  - They could be built-in or user written, like the textbook functions we have to include

- An **inline function** is code that the CPU can execute line-by-line
  - Similar to how one would read a book (no skipping around!)
fun percent-true(t :: Table, col :: String) -> Number:
    doc: "Return the percentage of rows that are true in column 'col'"

fun true-filter(r :: Row) -> Boolean:
    r[col]
end

filter-with(t, true-filter).length() / t.length()
end

• # The nested function true-filter is only used (called) in one location
• Do we have to name it and call it if we’re only going to do this once?
  • Spoiler alert: No, we don’t!
fun percent-true(t :: Table, col :: String) -> Number:
  doc: "Return the percentage of rows that are true in column 'col'"
  filter-with(t, \r: \[r[col]] end).length() / t.length()

filter-with(t, true-filter).length() / t.length()
end

• # We can instruct pyret to use an unnamed function!
• It will only ever be executed in-line (and from within filter-with)
Definition: $\lambda$

- A **lambda expression** defines an **anonymous function**
  - i.e. a function that can be passed as an argument but doesn’t have an associated name.
  - A lambda expression is executed as an in-line function
    - And can improve application performance (why?)
  - They are a common feature in modern programming languages

- Recognize them, but use them as you become comfortable using them.
  - Useful as “helper functions”
  - Nothing wrong with named functions!
Rows are easy to access.

.row-n gives us a row in a table...

<table>
<thead>
<tr>
<th>timestamp</th>
<th>house</th>
<th>stem-level</th>
<th>sleep-hours</th>
<th>schoolwork-hours</th>
<th>student-athlete</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;2/9/2022 19:03:33&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/9/2022 20:00:52&quot;</td>
<td>&quot;Main&quot;</td>
<td>10</td>
<td>4</td>
<td>7</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/9/2022 20:36:00&quot;</td>
<td>&quot;Main&quot;</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/10/2022 00:15:17&quot;</td>
<td>&quot;Strong&quot;</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 13:49:27&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/10/2022 13:53:12&quot;</td>
<td>&quot;Davison&quot;</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:05:47&quot;</td>
<td>&quot;Josselyn&quot;</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:06:22&quot;</td>
<td>&quot;Strong&quot;</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:26:46&quot;</td>
<td>&quot;Jewett&quot;</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:35:15&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>true</td>
</tr>
</tbody>
</table>

Click to show the remaining 23 rows...
Rows are easy to access. But what about columns?

.row-n gives us a row in a table...
How can we access all the elements in one column?

<table>
<thead>
<tr>
<th>timestamp</th>
<th>house</th>
<th>stem-level</th>
<th>sleep-hours</th>
<th>schoolwork-hours</th>
<th>student-athlete</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;2/09/2022 19:03:33&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/09/2022 20:00:52&quot;</td>
<td>&quot;Main&quot;</td>
<td>10</td>
<td>4</td>
<td>7</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/09/2022 20:36:00&quot;</td>
<td>&quot;Main&quot;</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/10/2022 00:15:17&quot;</td>
<td>&quot;Strong&quot;</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 13:49:27&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>true</td>
</tr>
<tr>
<td>&quot;2/10/2022 13:53:12&quot;</td>
<td>&quot;Davison&quot;</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:05:47&quot;</td>
<td>&quot;Josselyn&quot;</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:06:22&quot;</td>
<td>&quot;Strong&quot;</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:26:46&quot;</td>
<td>&quot;Jewett&quot;</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>false</td>
</tr>
<tr>
<td>&quot;2/10/2022 14:35:15&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>true</td>
</tr>
</tbody>
</table>

Click to show the remaining 23 rows...
Introducing: lists

.row-n gives us a row in a table...

How can we access all the elements in one column?

A: get-column

Example:

```
student-data-cleaned.get-column("house")
[list: "OTHER", "Main", "Main", "Strong", ...]
```
Introducing: lists

The concept is similar to Zeyu Zheng’s solution from earlier in the lecture!

• in that solution, there was one big string with all the house names. (a kind-of list!)
• string-contains was used to find the desired string in “list” of house names
• What if we want to use the “substrings” independently.
  • It is messy to separate each house name!
• What if we wanted to do something similar with numbers or Booleans or...
  • a general all-purpose solution for all data types besides strings is needed
Introducing: lists for student data

```

fun normalize-house(house :: String) -> String:
  doc: "Return one of the nine Vassar houses or 'Other'"
  if member(houses, house):
    house
  else:
    "Other"
end
```

where:

- `normalize-house("Main")` is "Main"
- `normalize-house("Offcampus")` is "Other"
Link to code

• https://code.pyret.org/editor#share=1WXx7yJvtOKJtXjza0CdCi8g dtozF8ZnR&v=31c9aaf
Acknowledgements

• This lecture incorporates material from:
  • Kathi Fisler, Brown University,
  • Jason Waterman, Vassar College
  • And, Jonathan Gordon, Vassar College