Tables and Lists

8 February 2023
Where are we?
<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
<tr>
<td>player</td>
<td>team</td>
<td>pos</td>
<td>games</td>
<td>pts</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
</tbody>
</table>

Click to show the remaining 188 rows...
How do I get just this row from `stats`?

```plaintext
stats =
```

<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td><strong>&quot;Lindsay Allen&quot;</strong></td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
</tbody>
</table>
```

Click to show the remaining 188 rows...
How do I get just this row from `stats`?

```
stats =
```

```python
stats.row-n(2)
```
How do I get just the rows for players who are guards?

<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
</tbody>
</table>
fun is-guard(player :: Row) -> Boolean:
  doc: "Return true if the player's primary position is guard"
  player["pos"] == "G"
where:
  is-guard(t.row-n(0)) is false
  is-guard(t.row-n(1)) is true
end

filter-with(stats, is-guard)
fun is-guard(player :: Row) -> Boolean:
  doc: "Return true if the player's primary position is guard"
  player["pos"] == "G"
  where:
  is-guard(t.row-n(0)) is false
  is-guard(t.row-n(1)) is true
end
This is the only place we want to use this helper function, so there’s no need to name it, document it, etc.

We can just write it inline as a lambda expression.

```plaintext
filter-with(stats, lam(player): player["pos"] == "G" end)
```
<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
</tbody>
</table>

Click to show the remaining 188 rows...
<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
</tbody>
</table>

Click to show the remaining 188 rows...
How can I add a new column like this?

<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
</tbody>
</table>

Click to show the remaining 188 rows...
How can I add a new column like this?

<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
</tbody>
</table>

Although we’ll only use this function here, it’ll be too long to be conveniently written as a lambda expression.

`build-column(stats, "frequent-player", how-frequent)`
fun how-frequent(player :: Row) -> String:
    if player["games"] >= 20:
        "very"
    else if player["games"] >= 10:
        "somewhat"
    else:
        "no"
end

build-column(stats, "frequent-player", how-frequent)
Changing a column
So, we’ve seen that we can build a new column based on the values in each row, but what if we just want to change an existing column?
A fake WNBA fan like me can’t remember what these team abbreviations stand for.

Let’s fill in the actual team names.
What are the team names?
<table>
<thead>
<tr>
<th>WESTERN</th>
<th>EASTERN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DAL</strong></td>
<td><strong>ATL</strong> Atlanta Dream</td>
</tr>
<tr>
<td><strong>LVA</strong></td>
<td><strong>CHI</strong> Chicago Sky</td>
</tr>
<tr>
<td><strong>LAS</strong></td>
<td><strong>CON</strong> Connecticut Sun</td>
</tr>
<tr>
<td><strong>MIN</strong></td>
<td><strong>IND</strong> Indiana Fever</td>
</tr>
<tr>
<td><strong>PHO</strong></td>
<td><strong>NYL</strong> New York Liberty</td>
</tr>
<tr>
<td><strong>SEA</strong></td>
<td><strong>WAS</strong> Washington Mystics</td>
</tr>
<tr>
<td>Dallas Wings</td>
<td>Atlanta Dream</td>
</tr>
<tr>
<td>Las Vegas Aces</td>
<td>Chicago Sky</td>
</tr>
<tr>
<td>Los Angeles Sparks</td>
<td>Connecticut Sun</td>
</tr>
<tr>
<td>Minnesota Lynx</td>
<td>Indiana Fever</td>
</tr>
<tr>
<td>Phoenix Mercury</td>
<td>New York Liberty</td>
</tr>
<tr>
<td>Seattle Storm</td>
<td>Washington Mystics</td>
</tr>
</tbody>
</table>
fun team-name(abbr :: String) -> String:
  doc: "Return the name of the team with the given abbreviation"

  ...

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"

  ...

end
fun team-name(abbr :: String) -> String:
  doc: "Return the name of the team with the given abbreviation"
  if abbr == "DAL": "Dallas Wings"
  else if abbr == "LVA": "Las Vegas Aces"
  ...
  end
where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

This will work, but remember what we said when we introduced tables for looking up population: We want to separate data from computation.
teams =

<table>
<thead>
<tr>
<th>abbr</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAL</td>
<td>&quot;Dallas Wings&quot;</td>
</tr>
<tr>
<td>LVA</td>
<td>&quot;Las Vegas Aces&quot;</td>
</tr>
<tr>
<td>LAS</td>
<td>&quot;Los Angeles Sparks&quot;</td>
</tr>
<tr>
<td>MIN</td>
<td>&quot;Minnesota Lynx&quot;</td>
</tr>
<tr>
<td>PHO</td>
<td>&quot;Phoenix Mercury&quot;</td>
</tr>
<tr>
<td>SEA</td>
<td>&quot;Seattle Storm&quot;</td>
</tr>
<tr>
<td>ATL</td>
<td>&quot;Atlanta Dream&quot;</td>
</tr>
<tr>
<td>CHI</td>
<td>&quot;Chicago Sky&quot;</td>
</tr>
<tr>
<td>CON</td>
<td>&quot;Connecticut Sun&quot;</td>
</tr>
<tr>
<td>IND</td>
<td>&quot;Indiana Fever&quot;</td>
</tr>
<tr>
<td>NYL</td>
<td>&quot;New York Liberty&quot;</td>
</tr>
<tr>
<td>WAS</td>
<td>&quot;Washington Mystics&quot;</td>
</tr>
</tbody>
</table>

Advantage: This makes it easy to add new teams or more information about these teams, in a central place.
teams = ...

fun team-name(abbr :: String) -> String:
  doc: "Return the name of the team with the given abbreviation"
  # Get the row with abbreviation `abbr`
  # Return the value in the `name` column
where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end
fun team-name(abbr :: String) -> String:
    doc: "Return the name of the team with the given abbreviation"
    matches =
        filter-with(teams, lam(r): r["abbr"] == abbr end)
    team = matches.row-n(0)
    # Return the value in the `name` column
where:
    team-name("NYL") is "New York Liberty"
    team-name("CHI") is "Chicago Sky"
    ...
end
teams = ...

fun team-name(abbr :: String) -> String:
    doc: "Return the name of the team with the given abbreviation"
    matches =
        filter-with(teams, lam(r): r["abbr"] == abbr end)
    team = matches.row-n(0)
    team["name"]

where:
    team-name("NYL") is "New York Liberty"
    team-name("CHI") is "Chicago Sky"
    ...
end
teams = ...

fun team-name(abbr :: String) -> String:
    doc: "Return the name of the team with the given abbreviation"

    matches =
        filter-with(teams, lam(r): r["abbr"] == abbr end)
    team = matches.row-n(0)
    team["name"]

where:
    team-name("NYL") is "New York Liberty"
    team-name("CHI") is "Chicago Sky"

transform-column(stats, "team", team-name)
row-n-too-large

(Show program evaluation trace...)
teams = ...

fun team-name(abbr :: String) -> String:
  doc: "Return the name of the team with the given abbreviation"
  matches =
    filter-with(teams, lam(r): r["abbr"] == abbr end)
  
  team = matches.row-n(0)
  team["name"]

  where:
    team-name("NYL") is "New York Liberty"
    team-name("CHI") is "Chicago Sky"

  ...

end

transform-column(stats, "team", team-name)

0 is too big? That means there were no matching rows! An abbreviation not in our table. What is it?
fun team-name(abbr :: String) -> String:
  doc: "Return the name of the team with the given abbreviation"
  matches =
    filter-with(teams, lam(r): r["abbr"] == abbr end)
  if matches.length() == 0:
    abbr
  else:
    team = matches.row-n(0)
    team["name"]
  end
where:
  ...
end

transform-column(stats, "team", team-name)
<table>
<thead>
<tr>
<th>Name</th>
<th>Team</th>
<th>Position</th>
<th>Games</th>
<th>Minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A'ja Wilson&quot;</td>
<td>&quot;Las Vegas Aces&quot;</td>
<td>&quot;F&quot;</td>
<td>36</td>
<td>703</td>
</tr>
<tr>
<td>&quot;Han Xu&quot;</td>
<td>&quot;New York Liberty&quot;</td>
<td>&quot;C&quot;</td>
<td>32</td>
<td>273</td>
</tr>
<tr>
<td>&quot;Jackie Young&quot;</td>
<td>&quot;Las Vegas Aces&quot;</td>
<td>&quot;G&quot;</td>
<td>34</td>
<td>542</td>
</tr>
<tr>
<td>&quot;Li Yueru&quot;</td>
<td>&quot;Chicago Sky&quot;</td>
<td>&quot;C&quot;</td>
<td>16</td>
<td>28</td>
</tr>
<tr>
<td>&quot;Emma Cannon&quot;</td>
<td>&quot;TOT&quot;</td>
<td>&quot;F&quot;</td>
<td>24</td>
<td>164</td>
</tr>
<tr>
<td>&quot;Tina Charles&quot;</td>
<td>&quot;TOT&quot;</td>
<td>&quot;C&quot;</td>
<td>34</td>
<td>502</td>
</tr>
<tr>
<td>&quot;Crystal Dangerfield&quot;</td>
<td>&quot;TOT&quot;</td>
<td>&quot;G&quot;</td>
<td>33</td>
<td>180</td>
</tr>
<tr>
<td>&quot;Kaela Davis&quot;</td>
<td>&quot;TOT&quot;</td>
<td></td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>&quot;Rennia Davis&quot;</td>
<td>&quot;TOT&quot;</td>
<td>&quot;F&quot;</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>&quot;AD Durr&quot;</td>
<td>&quot;TOT&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Reshanda Gray&quot;</td>
<td>&quot;TOT&quot;</td>
<td>&quot;F&quot;</td>
<td>27</td>
<td>59</td>
</tr>
</tbody>
</table>

Total? Players who played for more than one team?
What’s a column anyway?
We've seen that when you want a row of a table, you use \texttt{.row-n} and get a Row.

What about getting a column?
How do I get just the points column?

```
stats =

<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
<th>pos</th>
<th>games</th>
<th>pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Natalie Achonwa&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;F&quot;</td>
<td>22</td>
<td>116</td>
</tr>
<tr>
<td>&quot;Julie Allemand&quot;</td>
<td>&quot;CHI&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>74</td>
</tr>
<tr>
<td>&quot;Lindsay Allen&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>&quot;Rebecca Allen&quot;</td>
<td>&quot;NYL&quot;</td>
<td>&quot;G&quot;</td>
<td>25</td>
<td>174</td>
</tr>
<tr>
<td>&quot;Yvonne Anderson&quot;</td>
<td>&quot;CON&quot;</td>
<td>&quot;G&quot;</td>
<td>11</td>
<td>35</td>
</tr>
<tr>
<td>&quot;Kristine Anigwe&quot;</td>
<td>&quot;PHO&quot;</td>
<td>&quot;F-C&quot;</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>&quot;Ariel Atkins&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>527</td>
</tr>
<tr>
<td>&quot;Amy Atwell&quot;</td>
<td>&quot;LAS&quot;</td>
<td>&quot;F&quot;</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>&quot;Shakira Austin&quot;</td>
<td>&quot;WAS&quot;</td>
<td>&quot;C-F&quot;</td>
<td>36</td>
<td>312</td>
</tr>
<tr>
<td>&quot;Rachel Banham&quot;</td>
<td>&quot;MIN&quot;</td>
<td>&quot;G&quot;</td>
<td>36</td>
<td>283</td>
</tr>
</tbody>
</table>
```

Click to show the remaining 188 rows...
How do I get just the points column?

```python
stats = stats.get_column("pts")
```
The data type isn’t Column; it’s List!
A **List** is an ordered sequence of data.

For example,

```plaintext
grades = [0.96, 0.73, 1.0, 0.5]
```

```plaintext
fellowship = [
    "Frodo", "Sam", "Merry", "Pippin", "Gandalf",
    "Legolas", "Gimli", "Aragorn", "Boromir"
]
```
So, what good is a List?
Mad Libs!
Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun, each month into Number, and each Plural-Noun into seven Plural-Noun. At first, people told time by a sun clock, sometimes known as the Noun dial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather Noun, the pocket Noun, the alarm Noun, and, of course, the Body-Part watch. Children learn about clocks and time almost before they learn their A-B- Alphabet-Letter. They are taught that a day consists of 24 Plural-Noun, an hour has 60 Plural-Noun, and a minute has 60 Plural-Noun. By the time they are in Kindergarten, they know if the big Body-Part is at twelve and the little Body-Part is at three, that it is Number o'clock. I wish we could continue this Adjective lesson, but we've run out of Noun.
Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Number, each month into Number weeks, and each week into seven Number. At first, people told time by a sun clock, sometimes known as the Noun dial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather Noun, the pocket Noun, the alarm Noun, and, of course, the Body-Part watch. Children learn about clocks and time almost before they learn their A-B-Alphabet-Letter s. They are taught that a day consists of 24 Number, an hour has 60 Number, and a minute has 60 Number. By the time they are in Kindergarten, they know if the big Body-Part is at twelve and the little Body-Part is at three, that it is Number o’clock. I wish we could continue this Adjective lesson, but we’ve run out of Noun.
How can we represent a text?
Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun, each month into Number weeks, and each week into seven Plural-Noun. At first, people told time by a sun clock, sometimes known as the Noun dial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather Noun, the pocket Noun, the alarm Noun, and, of course, the Body-Part watch. Children learn about clocks and time almost before they learn their A-B- Alphabet-Letter s. They are taught that a day consists of 24 Plural-Noun, an hour has 60 Plural-Noun, and a minute has 60 Plural-Noun. By the time they are in Kindergarten, they know if the big Body-Part is at twelve and the little Body-Part is at three, that it is Number o' clock. I wish we could continue this Adjective lesson, but we’ve run out of Noun.
template = "Thousands of Plural-Noun ago, ..."

template-words = string-split-all(template, " ")

>>> template-words
[list: "Thousands", "of", "Plural-Noun", "ago", ...]
template = "Thousands of Plural-Noun ago, ..."

template-words = string-split-all(template, " ")

>>> template-words
[list: "Thousands", "of", "Plural-Noun", "ago", ...]

We need to substitute a random plural noun here!
"Thousands of Plural-Noun ago, ..."

string-split-all

[list: "Thousands", "of", "Plural-Noun", "ago", ...]

Something like transform-column but for lists

[list: "Thousands", "of", "gazebos", "ago", ...]

Needs a helper function!
"Thousands of Plural-Noun ago, ..."

string-split-all

[ list: "Thousands", "of", "Plural-Noun", "ago", ... ]

Something like transform-column but for lists using

[ list: "Thousands", "of", "gazebos", "ago", ... ]

substitute-word

"Thousands" -> "Thousands"
"Plural-Noun" -> "gazebos"
I’d write the helper function first!
fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
...
where:
    substitute-word("Thousands") is "Thousands"
   substitute-word("Plural-Noun") is ...
end

Uh oh! We don’t know what particular word it will be!
fun substitute-word(w :: String) -> String:
   doc: "Substitute a random word if w is a category"
...
where:
   substitute-word("Thousands") is "Thousands"
   substitute-word("Plural-Noun") is-not "Plural-Noun"
end

We know what it isn’t!
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    ...

where:
    substitute-word("Thousands") is "Thousands"
    substitute-word("Plural-Noun") is-not "Plural-Noun"
    plural-nouns.member(
        substitute-word("Plural-Noun"))
    is true
    end

And we know it’s one of the right choices!
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
...

where:
    substitute-word("Thousands") is "Thousands"
    substitute-word("Plural-Noun") is-not "Plural-Noun"
    plural-nouns.member(
        substitute-word("Plural-Noun"))
    is true

end

The left part of an example can be any expression!
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        ...
    else:
        w
    end
where:
    ...
end
We need a random element of a list.

Time to check the Pyret documentation!
3.2.5 Random Numbers

\texttt{num-random} :: (\texttt{max} :: \texttt{Number}) \to \texttt{Number}

Returns a pseudo-random positive integer from \(0\) to \(\text{max} - 1\).

Examples:

\begin{verbatim}
check:
  fun between(min, max):
    lam(v): (v >= min) and (v <= max) end
  end
for each i from range(0, 100):
  block:
    n = num-random(10)
    print(n)
    n satisfies between(0, 10 - 1)
  end
end
\end{verbatim}

\texttt{num-random-seed} :: (\texttt{seed} :: \texttt{Number}) \to \texttt{Nothing}

Sets the random seed. Setting the seed to a particular number makes all future uses of
random produce the same sequence of numbers. Useful for testing and debugging
functions that have random behavior.

Examples:

\begin{verbatim}
check:
  num-random-seed(0)
  n = num-random(1000)
\end{verbatim}
We didn’t find a built-in way to get a random element of a list, but we found a way to get a random number.

How could we use this?
.get :: (n :: Number) -> a

Returns the nth element of the given List, or raises an error if n is out of range.

Examples:

check:
  [list: 1, 2, 3].get(0) is 1
  [list: ].get(0) raises "too large"
  [list: 1, 2, 3].get(-1) raises "invalid argument"
end

.set :: (n :: Number, e :: a) -> List<a>

Returns a new List with the same values as the given List but with the nth element set to the given value, or raises an error if n is out of range.

Examples:

check:
  [list: 1, 2, 3].set(0, 5) is [list: 5, 2, 3]
  [list: ].set(0, 5) raises "too large"
end

.foldl :: (f :: (a, Base -> Base), base :: Base) -> Base

Computes \( f(last-elt, ... f(second-elt, f(first-elt, base))...) \). For empty, returns \( base \).

In other words, .foldl uses the function \( f \), starting with the base value, of type \( Base \), to
With a table, we could use `.row-n` to get a specific row by its index number.

With a list, we can use `.get` to get an item.
Get a random number

Get then list element positioned at that number
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
  if w == "Plural-Noun":
    rand = num-random(3)
    plural-nouns.get(rand)
  else:
    w
  end
end
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
  if w == "Plural-Noun":
    rand = num-random(3)
    plural-nouns.get(rand)
  else:
    w
  end
where:
  ...
end
plural-nouns = [list: "gazebos", "avocados", "pandas", "quokkas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand = num-random(3)
        plural-nouns.get(rand)
    else:
        w
    end
end

where:
    ...
end
plural-nouns = [list: "gazebos", "avocados", "pandas", "quokkas"]

fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
  if w == "Plural-Noun":
    rand = num-random(length(plural-nouns))
    plural-nouns.get(rand)
  else:
    w
  end
end
Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun, each month into weeks, and each week into seven Plural-Noun. At first, people told time by a sun clock, sometimes known as the Noun dial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather Noun, the pocket Noun, the alarm Noun, and, of course, the Body-Part watch. Children learn about clocks and time almost before they learn their A-B-Alphabet-Letters. They are taught that a day consists of 24 Plural-Noun, an hour has 60 Plural-Noun, and a minute has 60 Plural-Noun. By the time they are in Kindergarten, they know if the big Body-Part is at twelve and the little Body-Part is at three, that it is Number o'clock. I wish we could continue this Adjective lesson, but we’ve run out of Noun.
plural-nouns =
    [list: "gazebos", "avocados", "pandas", "quokkas"]

numbers =
    [list: "-1", "42", "a billion"]

nouns =
    [list: "apple", "computer", "borscht"]

body-parts =
    [list: "elbow", "head", "spleen"]

alphabet-letters =
    [list: "A", "C", "Z"]

adjectives =
    [list: "funky", "boring"]
fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand = num-random(length(plural-nouns))
        plural-nouns.get(rand)
    else if w == "Number":
        rand = ...  
    else:
        w
    end
end

where:
...
fun `rand-word` (l :: List<String>) \rightarrow String:
  doc: "Return a random word in the given list"
  rand = num-random(length(l))
  l.get(rand)

where:
  plural-nouns.member(rand-word(plural-nouns)) is true
  numbers.member(rand-word(numbers)) is true

end
fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
  if w == "Plural-Noun":
    rand-word(plural-nouns)
  else if w == "Number":
    rand-word(numbers)
  else if w == "Noun":
    rand-word(nouns)
  else if w == "Body-Part":
    rand-word(body-parts)
  else if w == "Alphabet-Letter":
    rand-word(alphabet-letters)
  else if w == "Adjective":
    rand-word(adjectives)
  else:
    w
end
end

This is still a bit repetitious – but it’s good enough for today!
Go back to the task plan.

We’ve completed our helper, and now we need to

split the input into words

run the helper on every word in the list

Similar to how transform-column runs a function on every row of a table.
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    ...
end
Go back to the task plan.

We’ve completed our helper, and now we need to

✓ split the input into words

run the helper on every word in the list

Similar to how transform-column runs a function on every row of a table.

This is called map!
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    map(substitute-word, t)
    ...
end
Go back to the task plan.

We’ve completed our helper, and now we need to

☑️ split the input into words
☑️ run the helper on every word in the list

Similar to how `transform-column` runs a function on every row of a table.

*Ok – are we done?*
fun mad-libs(t :: String) -> String:
  doc: "Randomly fill in the blanks in the mad libs template"
  words = string-split-all(t, " ")
  map(substitute-word, t)
  ...
end

This gives us a list of strings. How can we join it back into a single string?
fun mad-libs(t :: String) -> String:
  doc: "Randomly fill in the blanks in the mad libs template"
  words = string-split-all(t, " ")
  words-sub = map(substitute-word, words)
  join-str(with-sub, " ")
end
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs
template"
    words = string-split-all(t, " ")
    words-sub = map(substitute-word, t)
    join-str(words-sub, " ")

end
fun mad-libs(t :: String) -> String:
  doc: "Randomly fill in the blanks in the mad libs template"
  words = string-split-all(t, " ")
  words-sub = map(substitute-word, t)
  join-str(words-sub, " ")
where:
  mad-libs(template) is-not template
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    words-sub = map(substitute-word, t)
    join-str(words-sub, " ")
where:
    mad-libs(template) is-not template
    string-contains(mad-libs(template), "Plural-Noun")
        is false
end
Preview: Lists and recursion
What if `join-str` didn’t already exist for our convenience?

To write a function that processes a list element by element, we need to understand the real nature of lists.
A list consists of two parts: a first element and the rest of the list.

```python
>>> l = [list: 1, 2, 3]
>>> l.first
1
>>> l.rest
[list: 2, 3]
```
The first element is linked to the rest and so on until we reach the empty list:

```python
>>> link(1, empty)
["list": 1]
>>> link(1, link(2, link(3, empty)))
["list": 1, 2, 3]
```
When we write a function that recursively processes a list, we deal with these two cases – linking an element or being empty:

```plaintext
fun add-nums(l :: List<Number>) -> Number:
  cases (List) l:
    | empty => 0
    | link(f, r) => f + add-nums(r)
  end

where:
  add-nums([list: ]) is 0
  add-nums([list: 1]) is 1 + 0
  add-nums([list: 2, 1]) is 2 + 1 + 0
end
```
In the case of joining strings, we need to know not just if the current list is empty but is the rest of the rest empty. This is how we know whether to add a space or not.
fun join-with-spaces(l :: List<String>) -> String:
    doc: "Join the strings in l with a space between each one"
    cases (List) l:
        | empty => ""
        | link(f, r) =>
            cases (List) r:
                | empty => f
                | link(fr, rr) =>
                    f + " " + join-with-spaces(r)
            end
    end
    where:
        join-with-spaces([list: ]) is ""
        join-with-spaces([list: "y"] ) is "y" + ""
        join-with-spaces([list: "x", "y"] ) is "x" + " " + "y" + ""
    end
Class code:

tinyurl.com/101-2023-02-08