Working with Tables

14 September 2022
Lab 2
   Due Friday

Assignment 2
   Due tonight
Where are we?
Lots of real-world data is naturally represented as tables.
Lots of real-world data is naturally represented as tables.

```plaintext
municipalities =
  table: name, kind, pop-2010, pop-2020
  row: "Adams", "Town", 5143, 4973
  row: "Adams", "Village", 1775, 1633
  row: "Addison", "Town", 2595, 2397
  row: "Addison", "Village", 1763, 1561
  row: "Afton", "Town", 2851, 2769
... end
```
Lots of real-world data is naturally represented as tables.
Recap: Accessing parts of a table
To get a particular row from a table, we use its numeric index $n$, counting from 0:

\[
\langle \text{table} \rangle . \text{row} \neg n(0)
\]
### municipalities

<table>
<thead>
<tr>
<th>name</th>
<th>kind</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Adams&quot;</td>
<td>&quot;Town&quot;</td>
<td>5143</td>
<td>4973</td>
</tr>
<tr>
<td>&quot;Adams&quot;</td>
<td>&quot;Village&quot;</td>
<td>1775</td>
<td>1633</td>
</tr>
<tr>
<td>&quot;Addison&quot;</td>
<td>&quot;Town&quot;</td>
<td>2595</td>
<td>2397</td>
</tr>
<tr>
<td>&quot;Addison&quot;</td>
<td>&quot;Village&quot;</td>
<td>1763</td>
<td>1561</td>
</tr>
<tr>
<td>&quot;Afton&quot;</td>
<td>&quot;Town&quot;</td>
<td>2851</td>
<td>2769</td>
</tr>
<tr>
<td>&quot;Afton&quot;</td>
<td>&quot;Village&quot;</td>
<td>822</td>
<td>794</td>
</tr>
<tr>
<td>&quot;Airmont&quot;</td>
<td>&quot;Village&quot;</td>
<td>8628</td>
<td>10166</td>
</tr>
<tr>
<td>&quot;Akron&quot;</td>
<td>&quot;Village&quot;</td>
<td>2868</td>
<td>2888</td>
</tr>
<tr>
<td>&quot;Alabama&quot;</td>
<td>&quot;Town&quot;</td>
<td>1869</td>
<td>1602</td>
</tr>
<tr>
<td>&quot;Albany&quot;</td>
<td>&quot;City&quot;</td>
<td>97856</td>
<td>99224</td>
</tr>
</tbody>
</table>

[Click to show the remaining 1517 rows...]

### municipalities.row-n(0)

<table>
<thead>
<tr>
<th>name</th>
<th>Adams</th>
<th>kind</th>
<th>Town</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Adams&quot;</td>
<td>5143</td>
<td>&quot;Town&quot;</td>
<td>4973</td>
<td>4973</td>
<td>4973</td>
</tr>
</tbody>
</table>
## municipalities

<table>
<thead>
<tr>
<th>name</th>
<th>kind</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Adams&quot;</td>
<td>&quot;Town&quot;</td>
<td>5143</td>
<td>4973</td>
</tr>
<tr>
<td>&quot;Adams&quot;</td>
<td>&quot;Village&quot;</td>
<td>1775</td>
<td>1633</td>
</tr>
<tr>
<td>&quot;Addison&quot;</td>
<td>&quot;Town&quot;</td>
<td>2595</td>
<td>2397</td>
</tr>
<tr>
<td>&quot;Addison&quot;</td>
<td>&quot;Village&quot;</td>
<td>1763</td>
<td>1561</td>
</tr>
<tr>
<td>&quot;Afton&quot;</td>
<td>&quot;Town&quot;</td>
<td>2851</td>
<td>2769</td>
</tr>
<tr>
<td>&quot;Afton&quot;</td>
<td>&quot;Village&quot;</td>
<td>822</td>
<td>794</td>
</tr>
<tr>
<td>&quot;Airmont&quot;</td>
<td>&quot;Village&quot;</td>
<td>8628</td>
<td>10166</td>
</tr>
<tr>
<td>&quot;Akron&quot;</td>
<td>&quot;Village&quot;</td>
<td>2868</td>
<td>2888</td>
</tr>
<tr>
<td>&quot;Alabama&quot;</td>
<td>&quot;Town&quot;</td>
<td>1869</td>
<td>1602</td>
</tr>
<tr>
<td>&quot;Albany&quot;</td>
<td>&quot;City&quot;</td>
<td>97856</td>
<td>99224</td>
</tr>
</tbody>
</table>

Click to show the remaining 1517 rows...

## municipalities.row-n(1)

<table>
<thead>
<tr>
<th>name</th>
<th>kind</th>
<th>&quot;Village&quot;</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Adams&quot;</td>
<td>&quot;Village&quot;</td>
<td>1775</td>
<td>1633</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>kind</td>
<td>pop-2010</td>
<td>pop-2020</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>&quot;Adams&quot;</td>
<td>&quot;Town&quot;</td>
<td>5143</td>
<td>4973</td>
<td></td>
</tr>
<tr>
<td>&quot;Adams&quot;</td>
<td>&quot;Village&quot;</td>
<td>1775</td>
<td>1633</td>
<td></td>
</tr>
<tr>
<td>&quot;Addison&quot;</td>
<td>&quot;Town&quot;</td>
<td>2595</td>
<td>2397</td>
<td></td>
</tr>
<tr>
<td>&quot;Addison&quot;</td>
<td>&quot;Village&quot;</td>
<td>1763</td>
<td>1561</td>
<td></td>
</tr>
<tr>
<td>&quot;Afton&quot;</td>
<td>&quot;Town&quot;</td>
<td>2851</td>
<td>2769</td>
<td></td>
</tr>
<tr>
<td>&quot;Afton&quot;</td>
<td>&quot;Village&quot;</td>
<td>822</td>
<td>794</td>
<td></td>
</tr>
<tr>
<td>&quot;Airmont&quot;</td>
<td>&quot;Village&quot;</td>
<td>8628</td>
<td>10166</td>
<td></td>
</tr>
<tr>
<td>&quot;Akron&quot;</td>
<td>&quot;Village&quot;</td>
<td>2868</td>
<td>2888</td>
<td></td>
</tr>
<tr>
<td>&quot;Alabama&quot;</td>
<td>&quot;Town&quot;</td>
<td>1869</td>
<td>1602</td>
<td></td>
</tr>
<tr>
<td>&quot;Albany&quot;</td>
<td>&quot;City&quot;</td>
<td>97856</td>
<td>99224</td>
<td></td>
</tr>
</tbody>
</table>

Click to show the remaining 1517 rows...

<table>
<thead>
<tr>
<th>name</th>
<th>&quot;Addison&quot;</th>
<th>kind</th>
<th>&quot;Town&quot;</th>
<th>&quot;pop-2010&quot;</th>
<th>2595</th>
<th>&quot;pop-2020&quot;</th>
<th>2397</th>
</tr>
</thead>
</table>
To get a particular column’s value from a row, we specify the column name using square brackets:

\[
\langle row \rangle["column name"]
\]
<table>
<thead>
<tr>
<th>name</th>
<th>Adams</th>
<th>kind</th>
<th>Town</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5143</td>
<td>4973</td>
</tr>
</tbody>
</table>


```python
>>> municipalities.row-n(0)

| "name" | "Adams" | "kind" | "Town" | "pop-2010" | 5143 | "pop-2020" | 4973 |

>>> municipalities.row-n(0)["name"]
"Adams"

>>> municipalities.row-n(0)["pop-2020"]
4973
```
Recap: Ordering tables
To do more with tabular data, first include the textbook library:

```r
include shared-gdrive("dcic-2021", "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
```
We can transform tabular data to get a particular view. E.g., to order the rows from the highest 2010 population to the lowest:

```ruby
>>> order-by(municipalities, "pop-2010", false)
```

<table>
<thead>
<tr>
<th>name</th>
<th>kind</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;New York&quot;</td>
<td>&quot;City&quot;</td>
<td>8175133</td>
<td>8804190</td>
</tr>
<tr>
<td>&quot;Hempstead&quot;</td>
<td>&quot;Town&quot;</td>
<td>759757</td>
<td>793409</td>
</tr>
<tr>
<td>&quot;Brookhaven&quot;</td>
<td>&quot;Town&quot;</td>
<td>486040</td>
<td>485773</td>
</tr>
<tr>
<td>&quot;Islip&quot;</td>
<td>&quot;Town&quot;</td>
<td>335543</td>
<td>339938</td>
</tr>
<tr>
<td>&quot;Oyster Bay&quot;</td>
<td>&quot;Town&quot;</td>
<td>293214</td>
<td>301332</td>
</tr>
<tr>
<td>&quot;Buffalo&quot;</td>
<td>&quot;City&quot;</td>
<td>261310</td>
<td>278349</td>
</tr>
<tr>
<td>&quot;North Hempstead&quot;</td>
<td>&quot;Town&quot;</td>
<td>226322</td>
<td>237639</td>
</tr>
</tbody>
</table>
We can transform tabular data to get a particular view. E.g., to order the rows from the lowest 2010 population to the highest:

```
>>> order-by(municipalities, "pop-2010", true)
```

<table>
<thead>
<tr>
<th>name</th>
<th>kind</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Dering Harbor&quot;</td>
<td>&quot;Village&quot;</td>
<td>11</td>
<td>50</td>
</tr>
<tr>
<td>&quot;Saltaire&quot;</td>
<td>&quot;Village&quot;</td>
<td>37</td>
<td>113</td>
</tr>
<tr>
<td>&quot;Red House&quot;</td>
<td>&quot;Town&quot;</td>
<td>38</td>
<td>27</td>
</tr>
<tr>
<td>&quot;West Hampton Dunes&quot;</td>
<td>&quot;Village&quot;</td>
<td>55</td>
<td>126</td>
</tr>
<tr>
<td>&quot;Montague&quot;</td>
<td>&quot;Town&quot;</td>
<td>78</td>
<td>97</td>
</tr>
<tr>
<td>&quot;Ocean Beach&quot;</td>
<td>&quot;Village&quot;</td>
<td>79</td>
<td>153</td>
</tr>
<tr>
<td>&quot;Morehouse&quot;</td>
<td>&quot;Town&quot;</td>
<td>86</td>
<td>92</td>
</tr>
</tbody>
</table>
```python
>>> municipalities.row-n(0)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Adams</td>
<td>kind</td>
<td>Town</td>
<td>pop-2010</td>
<td>5143</td>
</tr>
</tbody>
</table>

```
```python
>>> municipalities.row-n(0)

| "name" | "Adams" | "kind" | "Town" | "pop-2010" | 5143 | "pop-2020" | 4973 |

```
```python
>>> ordered = order-by(municipalities, "pop-2010", false)
>>> biggest = ordered.row-n(0)
>>> biggest["pop-2010"]
8175133
```
Recap: Filtering tables
Make a table keeping only those municipalities with a 2010 population over 10,000:

```haskell
fun big-muni(r :: Row) -> Boolean:
    doc: "Return true if the municipality had over 10,000 people had in 2010"
    r["pop-2010"] > 10000
end

>>> filter-with(municipalities, big-muni)
```

<table>
<thead>
<tr>
<th>name</th>
<th>kind</th>
<th>pop-2010</th>
<th>pop-2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Albany&quot;</td>
<td>&quot;City&quot;</td>
<td>97856</td>
<td>99224</td>
</tr>
<tr>
<td>&quot;Alden&quot;</td>
<td>&quot;Town&quot;</td>
<td>10865</td>
<td>9706</td>
</tr>
<tr>
<td>&quot;Amherst&quot;</td>
<td>&quot;Town&quot;</td>
<td>122366</td>
<td>129595</td>
</tr>
</tbody>
</table>
Building columns
At the end of last class, we saw that we can also have functions on rows that don’t return Booleans:

```plaintext
def percent-change(r :: Row) -> Number:
    doc: "Compute the percentage change for the population of the given municipality between 2010 and 2020"
    (r["pop-2020"] - r["pop-2010"] / r["pop-2010"]
end
```
And we can use such functions to compute the values for a new column:

```
build-column(municipalities, "percent-change", percent-change)
```
And we can use such functions to compute the values for a new column:

```
build-column(municipalities, "percent-change", percent-change)
```
And we can use such functions to compute the values for a new column:

\[
\text{build-column}(\text{municipalities}, \text{"percent-change"}, \text{percent-change})
\]

*Name of the new column*

*Name of the function to use*
So, if we have this table, $t$, 

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;dog&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;cat&quot;</td>
<td>3</td>
</tr>
</tbody>
</table>

then the result of calling `build-column(t, "c", builder)` is:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;dog&quot;</td>
<td>2</td>
<td>builder(&quot;dog&quot;, 2)</td>
</tr>
<tr>
<td>&quot;cat&quot;</td>
<td>3</td>
<td>builder(&quot;cat&quot;, 3)</td>
</tr>
</tbody>
</table>
For example, if we have

```scala
fun builder(r :: Row) -> Number:
    string-length(row["a"]]) + row["b"]
end
```

Then we end up with the following table:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;dog&quot;</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>&quot;cat&quot;</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
The values that the builder function returns will be the values in the new column that we’re adding to each row.
build-column ::
  (t :: Table,
   colname :: String,
   builder :: (Row -> A))
  -> Table
build-column ::
  (t :: Table,
   colname :: String,
   builder :: (Row -> A))
-> Table

What's this argument?
This is the second time we’ve seen a function that takes a function as one of its inputs!

Both `filter-with` and `build-column` need a helper function that tells them how to do what we want.

Just as a function is an abstraction over specific computations, `filter-with` and `build-column` are abstractions over more specific functions. They provide the common functionality and the arguments we give provide the specifics.
Interlude: Functional programming
We can sort the rows a table with `order-with`, select certain rows using `filter-with`, and add a new column of values with `build-column` but none of these functions change the original table!
Just as the expression $2 + 3$ doesn’t change the value of 2 or of 3, functions that take a table as input don’t change the original table.

Instead, they return a new table.
This is a paradigm called *functional programming*. If you have experience working in other languages, this may seem strange, but it can be extremely useful! We'll explore the idea of functional programming more in the coming weeks.
Loading Google Sheets into Pyret
We’ve seen that it’s inconvenient to type a large table into a Pyret program. Last time, we loaded the municipalities table from a separate Pyret file that I prepared ahead of time.

More often, we’ll want to load our data from outside of Pyret.
<table>
<thead>
<tr>
<th>Municipality</th>
<th>Class</th>
<th>2010</th>
<th>2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adams</td>
<td>Town</td>
<td>5,143</td>
<td>4,973</td>
</tr>
<tr>
<td>Adams</td>
<td>Village</td>
<td>1,775</td>
<td>1,633</td>
</tr>
<tr>
<td>Addison</td>
<td>Town</td>
<td>2,698</td>
<td>2,097</td>
</tr>
<tr>
<td>Addison</td>
<td>Village</td>
<td>1,783</td>
<td>1,561</td>
</tr>
<tr>
<td>Affton</td>
<td>Town</td>
<td>2,851</td>
<td>2,769</td>
</tr>
<tr>
<td>Affton</td>
<td>Village</td>
<td>822</td>
<td>794</td>
</tr>
<tr>
<td>Airmont</td>
<td>Village</td>
<td>8,628</td>
<td>10,166</td>
</tr>
<tr>
<td>Akron</td>
<td>Village</td>
<td>2,888</td>
<td>2,888</td>
</tr>
<tr>
<td>Alabama</td>
<td>Town</td>
<td>1,860</td>
<td>1,602</td>
</tr>
<tr>
<td>Albany</td>
<td>City</td>
<td>97,856</td>
<td>99,224</td>
</tr>
<tr>
<td>Albany</td>
<td>Town</td>
<td>8,468</td>
<td>7,639</td>
</tr>
<tr>
<td>Albion</td>
<td>Town</td>
<td>2,073</td>
<td>2,009</td>
</tr>
<tr>
<td>Albion</td>
<td>Village</td>
<td>6,056</td>
<td>5,637</td>
</tr>
<tr>
<td>Alden</td>
<td>Town</td>
<td>10,865</td>
<td>9,706</td>
</tr>
<tr>
<td>Alden</td>
<td>Village</td>
<td>2,605</td>
<td>2,804</td>
</tr>
<tr>
<td>Alexander</td>
<td>Town</td>
<td>2,534</td>
<td>2,491</td>
</tr>
<tr>
<td>Alexander</td>
<td>Village</td>
<td>609</td>
<td>618</td>
</tr>
<tr>
<td>Alexandria</td>
<td>Town</td>
<td>4,061</td>
<td>3,741</td>
</tr>
<tr>
<td>Alexandria</td>
<td>Town</td>
<td>1,078</td>
<td>924</td>
</tr>
<tr>
<td>Alfred</td>
<td>Town</td>
<td>5,237</td>
<td>5,157</td>
</tr>
<tr>
<td>Alfred</td>
<td>Village</td>
<td>4,174</td>
<td>4,086</td>
</tr>
<tr>
<td>Allegany</td>
<td>Town</td>
<td>8,004</td>
<td>7,493</td>
</tr>
<tr>
<td>Allegany</td>
<td>Village</td>
<td>1,816</td>
<td>1,644</td>
</tr>
<tr>
<td>Allen</td>
<td>Town</td>
<td>448</td>
<td>494</td>
</tr>
<tr>
<td>Alma</td>
<td>Town</td>
<td>842</td>
<td>781</td>
</tr>
<tr>
<td>Almont</td>
<td>Town</td>
<td>1,653</td>
<td>1,612</td>
</tr>
<tr>
<td>Almond</td>
<td>Town</td>
<td>466</td>
<td>416</td>
</tr>
<tr>
<td>Altamont</td>
<td>Village</td>
<td>1,798</td>
<td>1,874</td>
</tr>
</tbody>
</table>
include gdrive-sheets

# The ID of the Google Sheets file, which appears
# in the URL
ssid = "1B0GB-MZ5wKZL7Xg0kKZITYW4bLsUMn2XJclnRE8WUqY"

spreadsheet = load-spreadsheet(ssid)
A spreadsheet might have more than one sheet (the tabs at the bottom of Google Sheets). But, in this case, we just have one:

```python
>>> spreadsheet
spreadsheet("municipalities")
```
To load a table from a spreadsheet, we need to tell Pyret which sheet to load it from and what we want the columns to be called (which can be different from what is in the spreadsheet):

```py
municipalities =
    load-table:
        name :: String, kind :: String,
        pop-2010 :: Number, pop-2020 :: Number
    source:
        spreadsheet.sheet-by-name("municipalities",
            true)
end
```
To load a table from a spreadsheet, we need to tell Pyret which sheet to load it from and what we want the columns to be called (which can be different from what is in the spreadsheet):

```
municipalities =
load-table:
  name :: String, kind :: String,
  pop-2010 :: Number, pop-2020 :: Number
source:
  spreadsheet.sheet-by-name("municipalities",
true)
end
```

This means there’s a header row that Pyret should skip.
Using our table loaded from Google Sheets, let’s revisit our code from last class for finding the fastest growing towns.
fun is-town(r :: Row) -> Boolean:
    doc: "Check if a row is for a town"
    r["kind"] == "Town"
end

fun percent-change(r :: Row) -> Number:
    doc: "Compute the percentage change for the population of the
given municipality between 2010 and 2020"
    (r["pop-2020"] - r["pop-2010"]) / r["pop-2010"]
end

towns = filter-with(municipalities, is-town)

towns-with-percent-change =
    build-column(towns, "percent-change", percent-change)

fastest-growing-towns =
    order-by(towns-with-percent-change, "percent-change", false)

fastest-growing-towns
fun is-town(r :: Row) -> Boolean:
    doc: "Check if a row is for a town"
    r["kind"] == "Town"
end

fun percent-change(r :: Row) -> Number:
    doc: "Compute the percentage change for the population of the given municipality between 2010 and 2020"
    (r["pop-2020"] - r["pop-2010"]) / r["pop-2010"]
end

towns = filter-with(municipalities, is-town)
towns-with-percent-change =
    build-column(towns, "percent-change", percent-change)

fastest-growing-towns =
    order-by(towns-with-percent-change,
        "percent-change", false)

fastest-growing-towns

Let's take these loose expressions and put them in a function!
fun is-town(r :: Row) -> Boolean:
  doc: "Check if a row is for a town"
  r["kind"] == "Town"
end

fun percent-change(r :: Row) -> Number:
  doc: "Compute the percentage change for the population of the
given municipality between 2010 and 2020"
  (r["pop-2020"] - r["pop-2010"]) / r["pop-2010"]
end

fun fastest-growing-towns(munis :: Table) -> Table:
  doc: "Return a table of towns ordered by their growth"

  towns = filter-with(munis, is-town)

  towns-with-percent-change =
    build-column(towns, "percent-change", percent-change)

  order-by(towns-with-percent-change, "percent-change", false)
end
We’ve done a bit of a bad thing here: We’ve written three functions, but we don't have tests for any of them!

Let’s see how we can rectify this.
Testing table functions
We can test table program by using *test tables*.

These are tables that have the same *structure* as the table for our real data, but which are *smaller* and contain data that are useful for testing.
test_municipalities =
  table: name :: String, kind :: String,
         pop_2010 :: Number, pop_2020 :: Number
  row: "Osgiliath", "City", 100, 101
  row: "Lake-town", "Town", 100, 102
  row: "Bee", "Village", 100, 99
  row: "Hobbiton", "Town", 50, 54
end
Let’s see how we use these test data to write examples for our table functions.
test-municipalities =
  table: name :: String, kind :: String, pop-2010 :: Number, pop-2020 :: Number
  row: "Osgiliath", "City", 100, 101
  row: "Lake-town", "Town", 100, 102
  row: "Bee", "Village", 100, 99
  row: "Hobbiton", "Town", 50, 54
end

fun is-town(r :: Row) -> Boolean:
  doc: "Check if a row is for a town"
  r["kind"] == "Town"
end
test-municipalities =
  table: name :: String, kind :: String, pop-2010 :: Number, pop-2020 :: Number
  row: "Osgiliath", "City", 100, 101
  row: "Lake-town", "Town", 100, 102
  row: "Bee", "Village", 100, 99
  row: "Hobbiton", "Town", 50, 54
end

fun is-town(r :: Row) -> Boolean:
  doc: "Check if a row is for a town"
  r["kind"] == "Town"
where:
  is-town(test-municipalities.row-n(0)) is false
  is-town(test-municipalities.row-n(1)) is true
  is-town(test-municipalities.row-n(2)) is false
end
test-municipalities =
  table: name :: String, kind :: String, pop-2010 :: Number, pop-2020 :: Number
    row: "Osgiliath", "City", 100, 101
    row: "Lake-town", "Town", 100, 102
    row: "Bee", "Village", 100, 99
    row: "Hobbiton", "Town", 50, 54
  end

fun percent-change(r :: Row) -> Number:
  doc: "Compute the percentage change for the population of the given municipality between 2010 and 2020"
  (r["pop-2020"] - r["pop-2010"]) / r["pop-2010]
end
test-municipalities =
    table: name :: String, kind :: String, pop-2010 :: Number, pop-2020 :: Number
    row: "Osgiliath", "City", 100, 101
    row: "Lake-town", "Town", 100, 102
    row: "Bee", "Village", 100, 99
    row: "Hobbiton", "Town", 50, 54
end

fun percent-change(r :: Row) -> Number:
    doc: "Compute the percentage change for the population of the given municipality between 2010 and 2020"
    (r["pop-2020"] - r["pop-2010"]) / r["pop-2010"]
where:
    percent-change(test-municipalities.row-n(0)) is 0.01
    percent-change(test-municipalities.row-n(1)) is 0.02
    percent-change(test-municipalities.row-n(2)) is -0.01
end
test-municipalities =
  table: name :: String, kind :: String, pop-2010 :: Number, pop-2020 :: Number
    row: "Osgiliath", "City", 100, 101
    row: "Lake-town", "Town", 100, 102
    row: "Bee", "Village", 100, 99
    row: "Hobbiton", "Town", 50, 54
end

fun fastest-growing-towns(munis :: Table) -> Table:
  doc: "Return a table of towns ordered by their growth"
  towns = filter-with(munis, is-town)
  towns-with-percent-change =
    build-column(towns, "percent-change", percent-change)
  order-by(towns-with-percent-change, "percent-change", false)
end
test-municipalities =
  table: name :: String, kind :: String, pop-2010 :: Number, pop-2020 :: Number
    row: "Osgiliath", "City", 100, 101
    row: "Lake-town", "Town", 100, 102
    row: "Bee", "Village", 100, 99
    row: "Hobbiton", "Town", 50, 54
end

fun fastest-growing-towns(munis :: Table) -> Table:
  ...
where:
  test-municipalities-after =
    table: name :: String, kind :: String, pop-2010 :: Number, pop-2020 :: Number, percent-change :: Number
      row: "Hobbiton", "Town", 50, 54, 0.08
      row: "Lake-town", "Town", 100, 102, 0.02
    end

  fastest-growing-towns(test-municipalities) is test-municipalities-after
end
Pyret code from class:

https://tinyurl.com/101-2022-09-14
Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University
Doug Woos, Brown University