Tables

30 January 2023
Lab 2
Due Friday

Assignment 2
Due Wednesday
Where are we?
Here are some data that can be represented with what we’ve seen so far:

- A picture of a dog: Image
- The population of Azerbaijan: Number
- The complete text of the *Baghavad Gita*: String
- Whether or not I ate breakfast this morning: Boolean
What if we wanted to write a program to look up the population of any town in New York?

We can consider the last two census years – 2010 and 2020.
fun population(municipality :: String, year :: Number) -> Number:
  doc: "Return population of the municipality for the given year"
  if municipality == "New York":
    if year == 2010:
      8175133
    else if year == 2020:
      8804190
    else:
      raise("Bad year")
  end
  else if municipality == "Poughkeepsie":
    if year == 2010:
      43341
    else if year == 2020:
      45471
    else:
      raise("Bad year")
  end
  else:
    raise("Bad municipality")
  end
fun population(municipality :: String, year :: Number) -> Number:
  doc: "Return population of the municipality for the given year"
  if municipality == "New York":
    if year == 2010:
      8175133
    else if year == 2020:
      8804190
    else:
      raise("Bad year")
  end
  else if municipality == "Poughkeepsie":
    if year == 2010:
      43341
    else if year == 2020:
      45471
    else:
      raise("Bad year")
  end
  else:
    raise("Bad municipality")
  end
fun population(municipality :: String, year :: Number) -> Number:
  doc: "Return population of the municipality for the given year"
  if municipality == "New York":
    if year == 2010:
      8175133
    else if year == 2020:
      8804190
    else:
      raise("Bad year")
  end
  else if municipality == "Poughkeepsie":
    if year == 2010:
      43341
    else if year == 2020:
      45471
    else:
      raise("Bad year")
  end
  else:
    raise("Bad municipality")
  end
fun population(municipality :: String, year :: Number) -> Number:

doc: "Return population of the municipality for the given year"

if municipality == "New York":
    if year == 2010:
        8175133
    else if year == 2020:
        8804190
    else:
        raise("Bad year")
else if municipality == "Poughkeepsie":
    if year == 2010:
        43341
    else if year == 2020:
        45471
    else:
        raise("Bad year")
else:
    raise("Bad municipality")
end
end
This is not a great way to do this.

Why not?
fun population(municipality :: String, year :: Number) -> Number:
  doc: "Return population of the municipality for the given year"
  if municipality == "New York":
    if year == 2010:
      8175133
    else if year == 2020:
      8804190
    else:
      raise("Bad year")
  end
  else if municipality == "Poughkeepsie":
    if year == 2010:
      43341
    else if year == 2020:
      45471
    else:
      raise("Bad year")
  end
  else:
    raise("Bad municipality")
  end
end

What about the rest of the state?
fun population(municipality :: String, year :: Number) -> Number:
  doc: "Return population of the municipality for the given year"
  if municipality == "New York":
    if year == 2010:
      8175133
    else if year == 2020:
      8804190
    else:
      raise("Bad year")
  end
  else if municipality == "Poughkeepsie":
    if year == 2010:
      43341
    else if year == 2020:
      45471
    else:
      raise("Bad year")
  end
  else:
    raise("Bad municipality")
  end
KEY IDEA  Separate data from computations.
Tables
Tables are used for tabular data, like you might find in a spreadsheet.
To define a table in Pyret, we specify its contents like so:

```pyret
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
```
To define a table in Pyret, we specify its contents like so:

```
municipalities =
    table: name :: String, kind :: String,
           pop-2010 :: Number, pop-2020 :: Number
    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
```

As with functions, we can specify the types for parts of a 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>
</tbody>
</table>
A bit later, we’ll see how we can load tabular data from outside Pyret so we don’t need to enter it all into our program.

I’ve already made a Pyret file that has the full municipality data, which we can load:

```py
include shared-gdrive("municipalities.arr", "10LyywS8KYe0bfEHebDzCBYYq7XxDrvQn")
```
<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...
Now that we have the data in Pyret, we can write programs to answer questions.
To get a row out of a table, specify its number, beginning with 0:

```python
>>> municipalities.row_n(0)
```

```
<table>
<thead>
<tr>
<th>&quot;name&quot;</th>
<th>&quot;Adams&quot;</th>
<th>&quot;kind&quot;</th>
<th>&quot;Town&quot;</th>
<th>&quot;pop-2010&quot;</th>
<th>5143</th>
<th>&quot;pop-2020&quot;</th>
<th>4973</th>
</tr>
</thead>
</table>
```
The data type returned by .row-n is a Row.

We can access a value in the row by specifying the name of a column:

```python
>>> municipalities.row-n(0)["name"]
"Adams"
```
We can write a function that takes a row as input:

```python
fun population_decreased(r :: Row) -> Boolean:
  doc: "Return true if the municipality's population went down between 2010 and 2020"
  r["pop-2020"] < r["pop-2010"]
end
```
Filtering and ordering tables
To work with tables, we’ll use a library that goes with the textbook.

We need to tell Pyret to load it:

```py
include shared-gdrive("dcic-2021", "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
```
One thing we might want to do is to get a version of the table that only has cities where the population has decreased.
fun filter-population-decreased(t :: Table) -> Table:
    if population-decreased(t.row-n(0)):
        ...
            # Keep row 0
    if population-decreased(t.row-n(1)):
        ...
            # Keep row 1
    else:
        ...
            # Don't keep row 1
    end
else:
    ...
            # Don't keep row 0
end
end
We can use `filter-with` to return a new table of just the rows where `population-decreased` evaluates to `true`:

```
filter-with(municipalities, population-decreased)
```
We can also use `filter-with` to get just the towns:

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

filter-with(municipalities, is-town)
```
We can also order the data by the values in one column:

\[
\text{order-by(municipalities, "pop-2020", false)}
\]

This means sort descending; true means ascending.
And we can combine all of these operations.

How would we get the town with the smallest population?
order-by(
  filter-with(municipalities, is-town),
  "pop-2020",
  true).row-n(0)
Example: Population change
PROBLEM: Figure out what the fastest-growing towns are in New York.
Subtasks:

Filtering out the cities
Calculating percentage change in population
Building a column for percentage change
Sorting on that column in *descending* order
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
Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University
Doug Woos, Brown University