Tables

11 September 2023
Assignment 1  Due Wednesday
Assignment 2  Out on Thursday
Lab 2  Due Friday
Where are we?
Here are some data that can be represented with what we’ve seen so far:
Here are some data that can be represented with what we’ve seen so far:

A picture of a dog

Image
Here are some data that can be represented with what we’ve seen so far:

- A picture of a dog
- The population of NYC
Here are some data that can be represented with what we’ve seen so far:

A picture of a dog
The population of NYC
The complete text of Beowulf

Image
Number
String
Here are some data that can be represented with what we’ve seen so far:

- A picture of a dog: Image
- The population of NYC: Number
- The complete text of *Beowulf*: 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:
      return 8175133
    else if year == 2020:
      return 8804190
    else:
      raise("Bad year")
  end
  else if municipality == "Poughkeepsie":
    if year == 2010:
      return 43341
    else if year == 2020:
      return 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")
  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
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

This isn’t 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
end
KEY IDEA  Separate data from computation.
Tables
Tables are used for tabular data, like you might find printed in a book or in a spreadsheet on a computer.
Tables are used for tabular data, like you might find printed in a book or in a spreadsheet on a computer.
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
```
### 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>
Next class, we’ll see how we can load tabular data from outside Pyret so we don’t need to enter it all into our program.

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

```py
include shared-gdrive("municipalities", "1M0SSJpz4E4D1YEAdDrKCBeWaQaxTR721")
```
<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 at zero:

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

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

```
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
```
fun population-decreased(r :: Row) -> Boolean:
  doc: "Return true if the municipality's population went down between 2010 and 2020"
  if r["pop-2020"] < r["pop-2010"]: 
    true
  else:
    false
  end
end

Why don’t we write it like this?
fun population-decreased(r :: Row) -> Boolean:
    doc: "Return true if the municipality's population went down between 2010 and 2020"
    if r["pop-2020"] < r["pop-2010"]:  
        true
    else:
        false
    end
end
Consider calling it on a particular input.

$$r = \begin{array}{cccccc}
\text{name} & \text{Adams} & \text{kind} & \text{Town} & \text{pop-2010} & 5143 \\
\text{pop-2020} & 4973 \\
\end{array}$$

if \ r["pop-2020"] < r["pop-2010"]:  
    \text{true}  
else:  
    \text{false}  
end
if 4973 < r["pop-2010"]:
    true
else:
    false
end

Consider calling it on a particular input.
\[
\begin{array}{rrrrrr}
\text{name} & \text{Adams} & \text{kind} & \text{Town} & \text{pop-2010} & \text{pop-2020} \\
5143 & 4973 & & & & \\
\end{array}
\]

\[r = \] 

If \(4973 < 5143\):

true

else:

false

done
if true:
    true
else:
    false
end
if true:
    true
else:
    false
end

xkcd.com/703
if true:
    true
else:
    false
end

This is equivalent to just writing true
if 4973 < 5143:
    true
else:
    false
end

This is equivalent to just writing 4973 < 5143
if r["pop-2020"] < r["pop-2010"]:
    true
else:
    false
end

This is equivalent to just writing
r["pop-2020"] < r["pop-2010"]
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

Illustration by
Gemma Correll
fun population-decreased(r :: Row) -> Boolean:
  doc: "Return true if the municipality's population went down between 2010 and 2020"
  if r["pop-2020"] < r["pop-2010"]:
    true
  else:
    false
end
end
Illustration by Gemma Correll
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:

```
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`:

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

```plaintext
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:

```order-by(municipalities, "pop-2020", false)```
We can also order the data by the values in one column:

```
order-by(municipalities, "pop-2020", false)
```

This means we want to sort in descending order; `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)
Pyret code from class:

tinyurl.com/101-2023-09-11
Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University
Doug Woos, Brown University