CMPU 101 § 51 · Computer Science I

Tables

12 September 2022

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:

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	*Image*
The population of Azerbaijan	*Number*

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*

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
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
end
```

We can nest "if" statements!

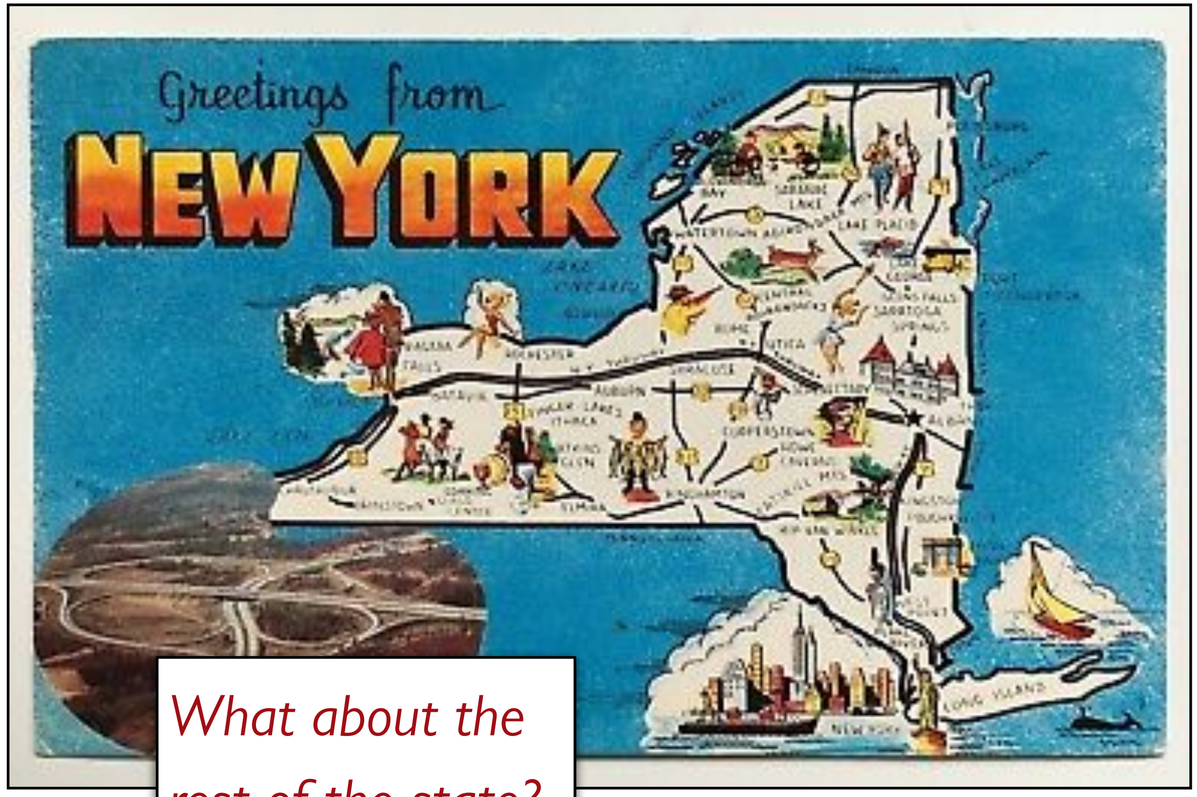
```
fun population(municipality :: String, year :: Number) -> Number:
  doc: "Return population of the municipality for the given year"
  if municipality == "New York":
    if year == 2010:
      8175133
    else if year == 2020:
      8804190
    else:
      raise("Bad year")
    end
  else if municipality == "Poughkeepsie":
    if year == 2010:
      43341
    else if year == 2020:
      45471
    else:
      raise("Bad year")
    end
  else:
    raise("Bad municipality")
  end
end
```

```
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
```

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
```

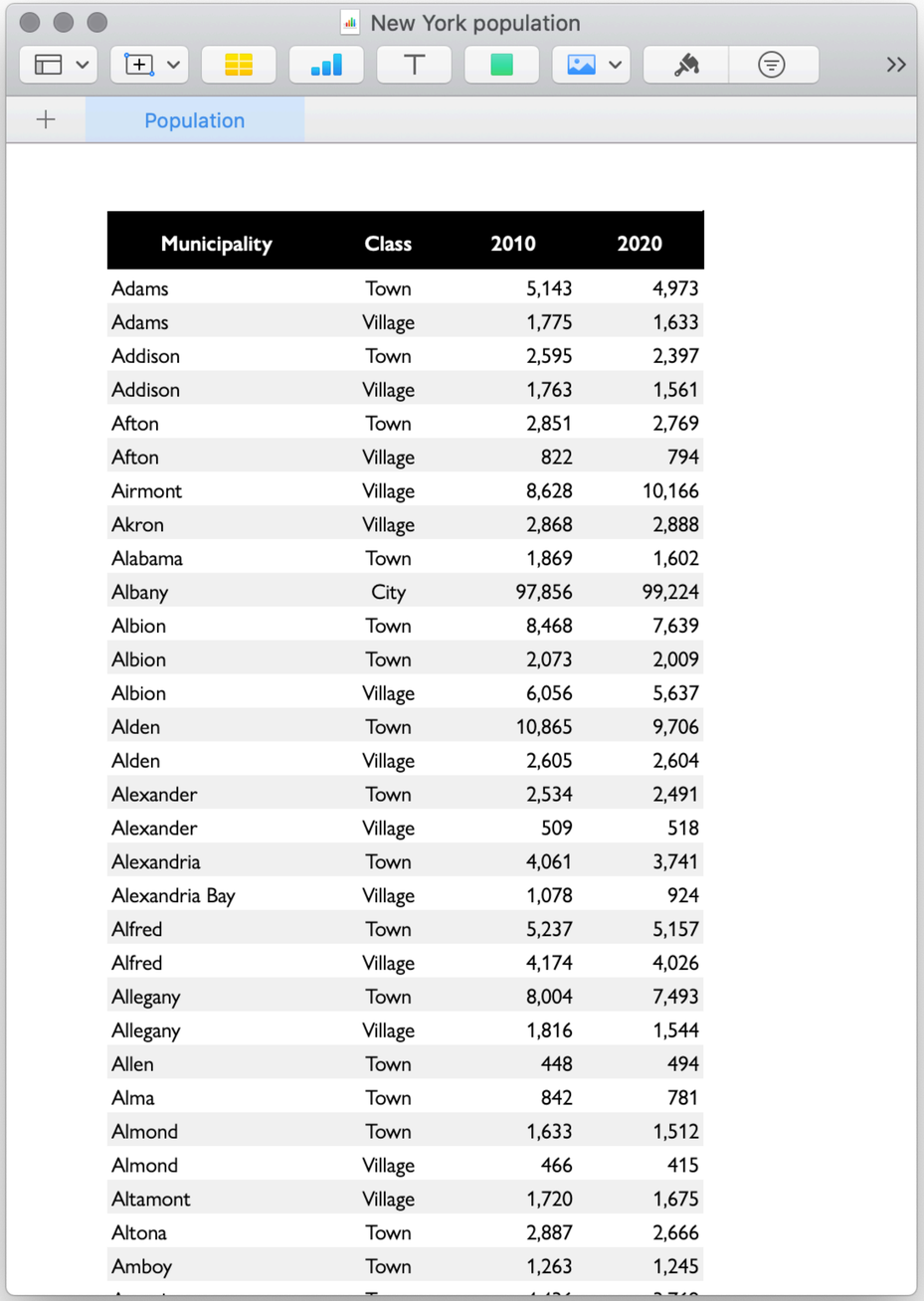


```
fun population(municipality :: String, year :: Number) -> Number:
  doc: "Return population of the municipality for the given year"
  if municipality == "New York":
    if year == 2010:
      8175133
    else if year == 2020:
      8804190
    else:
      raise("Bad year")
    end
  else if municipality == "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 computations.

Tables

Tables are used for tabular data, like you might find in a spreadsheet.



Municipality	Class	2010	2020
Adams	Town	5,143	4,973
Adams	Village	1,775	1,633
Addison	Town	2,595	2,397
Addison	Village	1,763	1,561
Afton	Town	2,851	2,769
Afton	Village	822	794
Airmont	Village	8,628	10,166
Akron	Village	2,868	2,888
Alabama	Town	1,869	1,602
Albany	City	97,856	99,224
Albion	Town	8,468	7,639
Albion	Town	2,073	2,009
Albion	Village	6,056	5,637
Alden	Town	10,865	9,706
Alden	Village	2,605	2,604
Alexander	Town	2,534	2,491
Alexander	Village	509	518
Alexandria	Town	4,061	3,741
Alexandria Bay	Village	1,078	924
Alfred	Town	5,237	5,157
Alfred	Village	4,174	4,026
Allegany	Town	8,004	7,493
Allegany	Village	1,816	1,544
Allen	Town	448	494
Alma	Town	842	781
Almond	Town	1,633	1,512
Almond	Village	466	415
Altamont	Village	1,720	1,675
Altona	Town	2,887	2,666
Amboy	Town	1,263	1,245

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

```
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
```

name	kind	pop-2010	pop-2020
"Adams"	"Town"	5143	4973
"Adams"	"Village"	1775	1633
"Addison"	"Town"	2595	2397
"Addison"	"Village"	1763	1561
"Afton"	"Town"	2851	2769

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:

```
include shared-gdrive("municipalities.arr",
  "1G9AkiPLQSKcc3kUn_8MjnPu29z8ny1wA")
```

```
>>> municipalities
```

name	kind	pop-2010	pop-2020
"Adams"	"Town"	5143	4973
"Adams"	"Village"	1775	1633
"Addison"	"Town"	2595	2397
"Addison"	"Village"	1763	1561
"Afton"	"Town"	2851	2769
"Afton"	"Village"	822	794
"Airmont"	"Village"	8628	10166
"Akron"	"Village"	2868	2888
"Alabama"	"Town"	1869	1602
"Albany"	"City"	97856	99224

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:

```
>>> 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:

```
>>> 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
```

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`:

```
filter-with(municipalities, population-decreased)
```

We can also use **`filter-with`** to get just the towns:

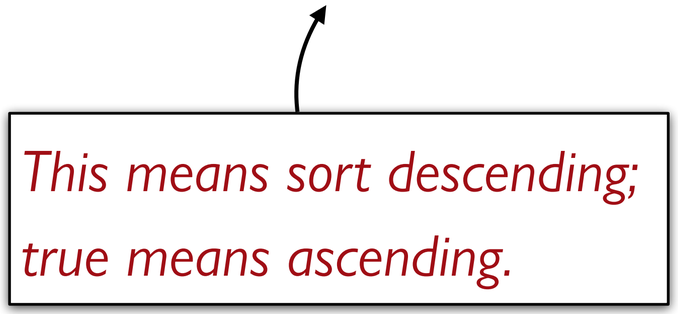
```
fun is-town(r :: Row) -> Boolean:
  doc: "Check if a row is for a town"
  r["kind"] == "Town"
end

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)
```



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