

Problem Solving and Abstraction (CMPU 101)

Tom Ellman

Lecture 5

Gradebook Table

```
gradebook = table: name, NRO, asmt1, asmt2  
  row: "Allie", false, 85, 90  
  row: "Carl", false, 75, 60  
  row: "Elan", true, 95, 63  
  row: "Lavon", false, 87, 88  
  row: "Nunu", true, 70, 0  
end
```

```
>>> gradebook
```

name	NRO	asmt1	asmt2
"Allie"	false	85	90
"Carl"	false	75	60
"Elan"	true	95	63
"Lavon"	false	87	88
"Nunu"	true	70	0

Extracting Rows

```
>>> gradebook
```

name	NRO	asmt1	asmt2
"Allie"	false	85	90
"Carl"	false	75	60
"Elan"	true	95	63
"Lavon"	false	87	88
"Nunu"	true	70	0

```
>>> gradebook.row-n(0)
```

"name"	"Allie"	"NRO"	false	"asmt1"	85	"asmt2"	90
--------	---------	-------	-------	---------	----	---------	----

```
>>> gradebook.row-n(3)
```

"name"	"Lavon"	"NRO"	false	"asmt1"	87	"asmt2"	88
--------	---------	-------	-------	---------	----	---------	----

Extracting Values in Row Cells

```
>>> gradebook.row-n(0)
```

"name"	"Allie"	"NRO"	false	"asmt1"	85	"asmt2"	90
--------	---------	-------	-------	---------	----	---------	----

```
>>> gradebook.row-n(0)["name"]
```

```
"Allie"
```

```
>>> gradebook.row-n(3)
```

"name"	"Lavon"	"NRO"	false	"asmt1"	87	"asmt2"	88
--------	---------	-------	-------	---------	----	---------	----

```
>>> gradebook.row-n(3)["asmt2"]
```

```
88
```

```
>>> gradebook-nro = filter-with(gradebook,nro-ed)
```

```
>>> gradebook-nro
```

name	NRO	asmt1	asmt2
"Elan"	true	95	63
"Nunu"	true	70	0

```
fun nro-ed(r :: Row) -> Boolean:  
  r["NRO"]  
end
```

Immutability of Tables

```
>>> gradebook
```

name	NRO	asmt1	asmt2
"Allie"	false	85	90
"Carl"	false	75	60
"Elan"	true	95	63
"Lavon"	false	87	88
"Nunu"	true	70	0

The **filter-with** function creates a new table **gradebook-nro** with fewer rows. The original **gradebook** is unchanged.

Find rows of students for whom second assignment score is lower than first assignment score.

```
>>> gradebook
```

name	NRO	asmt1	asmt2
"Allie"	false	85	90
"Carl"	false	75	60
"Elan"	true	95	63
"Lavon"	false	87	88
"Nunu"	true	70	0

How should we approach this problem?

Finding rows of students for whom second assignment score is lower than first assignment score.

```
fun asmt2-lower(r :: Row) -> Boolean:  
  r["asmt1"] > r["asmt2"]  
end
```

```
>>> gradebook-lower = filter-with(gradebook,asmt2-lower)
```

```
>>> gradebook-lower
```

name	NRO	asmt1	asmt2
"Carl"	false	75	60
"Elan"	true	95	63
"Nunu"	true	70	0

Building a New Column

```
fun asmt-avg(r :: Row) -> Number:  
  doc: "Add a column with the average of the assignment grades"  
  (r["asmt1"] + r["asmt2"]) / 2  
end  
  
gradebook-avg = build-column(gradebook, "avg", asmt-avg)
```

>>> gradebook

name	NRO	asmt1	asmt2
"Allie"	false	85	90
"Carl"	false	75	60
"Elan"	true	95	63
"Lavon"	false	87	88
"Nunu"	true	70	0

>>> build-column(gradebook, "avg", asmt-avg)

name	NRO	asmt1	asmt2	avg
"Allie"	false	85	90	87.5
"Carl"	false	75	60	67.5
"Elan"	true	95	63	79
"Lavon"	false	87	88	87.5
"Nunu"	true	70	0	35

Transforming a Table

```
bonus-factor = 1.25
fun covid-bonus(n :: Number) -> Number:
  n * bonus-factor
end

fun table-covid-bonus(t :: Table) -> Table:
  transform-column(t, "avg", covid-bonus)
end

gradebook-avg-bonus = table-covid-bonus(gradebook-avg)
```

»» gradebook-avg

name	NRO	asmt1	asmt2	avg
"Allie"	false	85	90	87.5
"Carl"	false	75	60	67.5
"Elan"	true	95	63	79
"Lavon"	false	87	88	87.5
"Nunu"	true	70	0	35

»» table-covid-bonus(gradebook-avg)

name	NRO	asmt1	asmt2	avg
"Allie"	false	85	90	109.375
"Carl"	false	75	60	84.375
"Elan"	true	95	63	98.75
"Lavon"	false	87	88	109.375
"Nunu"	true	70	0	43.75


Adjust grades on an assignment, but only up to a bound.

```
fun adjust-entry(e :: Number) -> Number:
  m = e + n
  if (m < b): m else: b end
end
```

?

```
fun adjust-grades(asmt :: String, n :: Number, b :: Number) -> Table:
  doc: "Increase asmt grades by n, but only up to b."
  transform-column(gradebook, asmt, adjust-entry)
end
```

```
gradebook-adjusted = adjust-grades("asmt2", 15, 100)
```



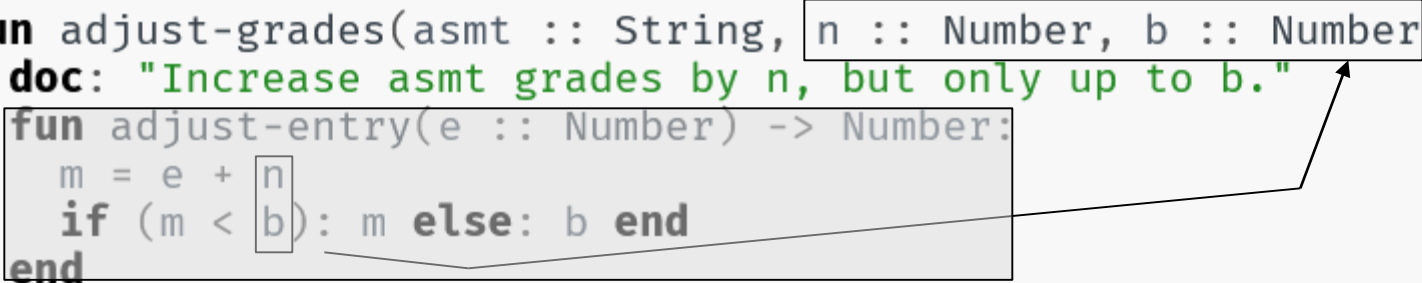
What is wrong with this program?

The **n** and **b** parameters of **adjust-grades** are visible only in the body of **adjust-grades**. In the body of the helper **adjust-entry**, the **n** and **b** parameters of **adjust-grades** are **out of scope**.

Adjust grades on an assignment, but only up to a bound.

```
fun adjust-grades(asmt :: String, n :: Number, b :: Number) -> Table:
  doc: "Increase asmt grades by n, but only up to b."
  fun adjust-entry(e :: Number) -> Number:
    m = e + n
    if (m < b): m else: b end
  end
  transform-column(gradebook, asmt, adjust-entry)
end

gradebook-adjusted = adjust-grades("asmt2", 15, 100)
```



We can put the definition of the helper **adjust-entry** inside the body of **adjust-grades**. Now the **n** and **b** parameters of **adjust-grades** are **in scope** where the helper **adjust-entry** refers to them.

```
>>> gradebook
```

name	NRO	asmt1	asmt2
"Allie"	false	85	90
"Carl"	false	75	60
"Elan"	true	95	63
"Lavon"	false	87	88
"Nunu"	true	70	0

```
>>> adjust-grades("asmt2", 15, 100)
```

name	NRO	asmt1	asmt2
"Allie"	false	85	100
"Carl"	false	75	75
"Elan"	true	95	78
"Lavon"	false	87	100
"Nunu"	true	70	15

```
>>>
```

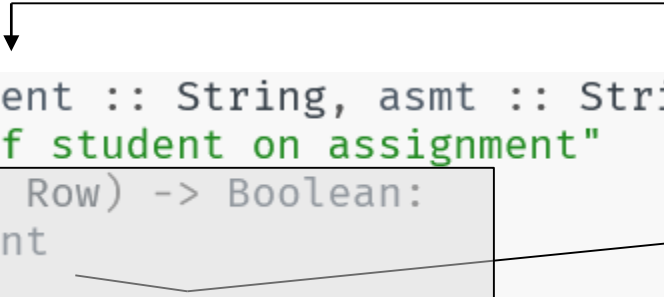
Look Up a Given Student's Score on a Given Assignment

```
fun student-row(r :: Row) -> Boolean:
  r["name"] == student
end
      ↘ ?
fun look-up-grade(student :: String, asmt :: String) -> Number:
  doc: "Return grade of student on assignment"
  s-row = filter-with(gradebook, student-row).row-n(0)
  s-row[asmt]
end
```

What is wrong with this program?

The **student** parameter of **look-up-grade** is visible only in the body of **look-up-grade**. In the body of the helper **student-row**, the **student** parameter of **look-up-grade** is **out of scope**.

Look Up a Given Student's Score on a Given Assignment



```
fun look-up-grade(student :: String, asmt :: String) -> Number:
  doc: "Return grade of student on assignment"
  fun student-row(r :: Row) -> Boolean:
    r["name"] == student
  end
  s-row = filter-with(gradebook, student-row).row-n(0)
  s-row[asmt]
end
```

The diagram illustrates the scope of the `student-row` function. A box highlights the definition of `student-row` inside the `look-up-grade` function. An arrow points from the `student` parameter in the `look-up-grade` signature to the `student` variable used in the `student-row` function body, indicating that `student` is in scope for `student-row`.

We can put the definition of **student-row** inside the body of the **look-up-grade**. Now the **student** parameter is **in scope** where the helper **non-student-row** refers to it.

```
>>> gradebook
```

name	NRO	asmt1	asmt2
"Allie"	false	85	90
"Carl"	false	75	60
"Elan"	true	95	63
"Lavon"	false	87	88
"Nunu"	true	70	0

```
>>> look-up-grade("Elan", "asmt1")
```

```
85
```

```
>>> look-up-grade("Lavon", "asmt2")
```

```
90
```

Update a Given Student's Score on a Given Assignment

Update a Given Student's Score on a Given Assignment

```
fun update-grade(s :: String, asmt :: String, g :: Number) -> Table:
  fun student-rowp(r :: Row) -> Boolean: r["name"] == s end
  fun not-student-rowp(r :: Row) -> Boolean: not(r["name"] == s) end

  sr = filter-with(gradebook, student-rowp).row-n(0)
  others = filter-with(gradebook, not-student-rowp)
  nro = sr["NRO"]
  g1 = if asmt == "asmt1": g else: sr["asmt1"] end
  g2 = if asmt == "asmt2": g else: sr["asmt2"] end
  new-row = others.row(s, nro, g1, g2)
  others.add-row(new-row)
end
```

Doable but messy!

Computations on Tables

- Look up a property value of a row.
- Find a row by number or property/column value.
- Order a table by property/column values.
- Extract rows that satisfy some criterion.
- Remove rows that satisfy some criterion.
- Build a new column, give it a name and fill the new column with data computed from properties.
- Transform a table by applying a function to each value in a column.

Missing Computations

- Modify a property in a row or some rows.
- Adding new rows to a table.
- Computations Across on Rows
- Computations Down (Up) Columns
- Delete columns.
- ... More ...

Rhode Island Population Data

```
municipalities = table:
```

```
    name, city, population-2000, population-2010
```

```
row: "Providence", true, 173618, 178042
```

```
row: "Cranston", true, 79269, 80387
```

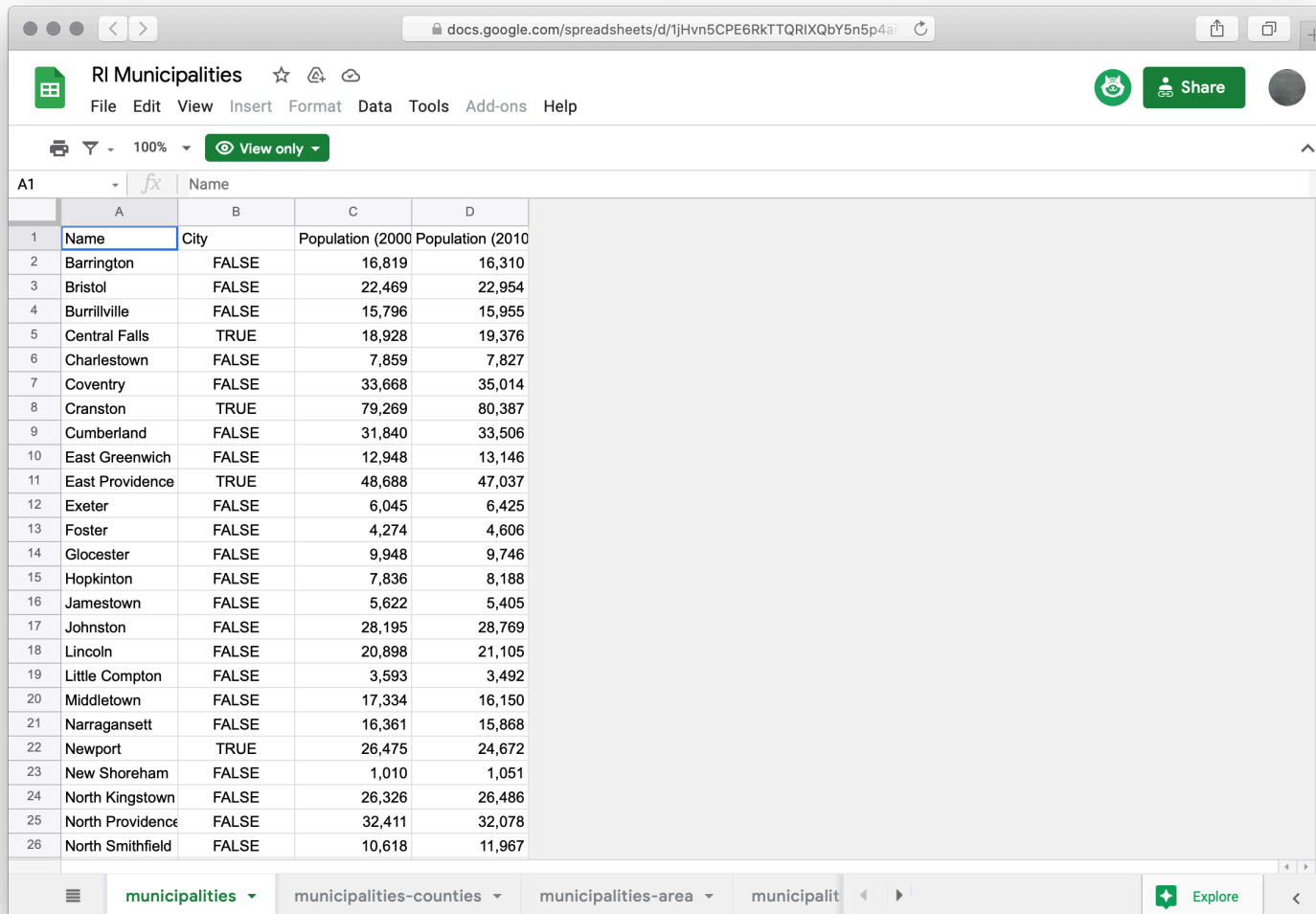
```
row: "Coventry", false, 33668, 35014
```

```
row: "Warwick", true, 85808, 82672,
```

```
row: "North Providence", false, 32411, 32078
```

```
end
```

Google Sheets



The screenshot shows a Google Sheets interface with a spreadsheet titled "RI Municipalities". The spreadsheet contains a table with 5 columns: Name, City, and Population (2000), Population (2010). The table lists 26 municipalities in Rhode Island. The interface includes a menu bar (File, Edit, View, Insert, Format, Data, Tools, Add-ons, Help), a toolbar with icons for print, filter, zoom (100%), and view mode (View only), and a bottom tab bar with tabs for "municipalities", "municipalities-counties", "municipalities-area", and "municipalities".

	A	B	C	D
1	Name	City	Population (2000)	Population (2010)
2	Barrington	FALSE	16,819	16,310
3	Bristol	FALSE	22,469	22,954
4	Burrillville	FALSE	15,796	15,955
5	Central Falls	TRUE	18,928	19,376
6	Charlestown	FALSE	7,859	7,827
7	Coventry	FALSE	33,668	35,014
8	Cranston	TRUE	79,269	80,387
9	Cumberland	FALSE	31,840	33,506
10	East Greenwich	FALSE	12,948	13,146
11	East Providence	TRUE	48,688	47,037
12	Exeter	FALSE	6,045	6,425
13	Foster	FALSE	4,274	4,606
14	Glocester	FALSE	9,948	9,746
15	Hopkinton	FALSE	7,836	8,188
16	Jamestown	FALSE	5,622	5,405
17	Johnston	FALSE	28,195	28,769
18	Lincoln	FALSE	20,898	21,105
19	Little Compton	FALSE	3,593	3,492
20	Middletown	FALSE	17,334	16,150
21	Narragansett	FALSE	16,361	15,868
22	Newport	TRUE	26,475	24,672
23	New Shoreham	FALSE	1,010	1,051
24	North Kingstown	FALSE	26,326	26,486
25	North Providence	FALSE	32,411	32,078
26	North Smithfield	FALSE	10,618	11,967

Loading Google Sheets into Pyret

```
# Textbook library
```

```
include shared-gdrive("dcic-2021",  
    "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
```

```
# Google Sheets library
```

```
include gdrive-sheets
```

```
# The ID of the Google Sheets file, which appears
```

```
# in the URL
```

```
ssid = "1jHvn5CPE6RkTTQRIXQbY5n5p4aiOH7fZsnwK2s6s6tc"
```

```
spreadsheet = load-spreadsheet(ssid)
```


A spreadsheet may have multiple sheets.

```
>>> spreadsheet  
spreadsheet("municipalities",  
            "municipalities-counties",  
            "municipalities-area",  
            "municipalities-full")
```

Converting a Google Sheet into a Pyret Table

```
municipalities = load-table:  
  name :: String,  
  city :: Boolean,  
  population-2000 :: Number,  
  population-2010 :: Number  
source: spreadsheet.sheet-by-name("municipalities", true)  
# true because the sheet has a "header" row  
end
```


>>> municipalities

name	city	population-2000	population-2010 
"Barrington"	false	16819	16310
"Bristol"	false	22469	22954
"Burrillville"	false	15796	15955
"Central Falls"	true	18928	19376
"Charlestown"	false	7859	7827
"Coventry"	false	33668	35014
"Cranston"	true	79269	80387
"Cumberland"	false	31840	33506
"East Greenwich"	false	12948	13146
"East Providence"	true	48688	47037
Click to show the remaining 29 rows...			

How can we find the fastest growing towns in Rhode Island?



Find the Fastest Growing Towns in Rhode Island

- Filter out the cities, i.e., keep the towns.
- Calculate the percent change in population for each town.
- Build a column from the percent changes.
- Sort the table on that column – descending.

```
fun is-town(r :: Row) -> Boolean:
  not(r["city"])
end

fun percent-change(r :: Row) -> Number:
  (r["population-2010"] - r["population-2000"]) /
  r["population-2000"]
end

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

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

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

growing-fastest = ordered-towns.row-n(0)["name"]
```

```
>>> growing-fastest
"West Greenwich"
```

name	city	population-2000	population-2010	percent-change
"West Greenwich"	false	5085	6135	0.20648967551622
"North Smithfield"	false	10618	11967	0.12704840836315
"South Kingstown"	false	27921	30639	0.09734608359299
"Foster"	false	4274	4606	0.07767898923724
"Richmond"	false	7222	7708	0.06729437828856
"Exeter"	false	6045	6425	0.06286186931348
"Cumberland"	false	31840	33506	0.05232412060301
"Hopkinton"	false	7836	8188	0.04492087799897
"New Shoreham"	false	1010	1051	0.04059
"Coventry"	false	33668	35014	0.03997861470832
Click to show the remaining 21 rows...				

Computations on Tables

- Look up a property value of a row.
- Find a row by number or property/column value.
- Order a table by property/column values.
- Extract rows that satisfy some criterion.
- Remove rows that satisfy some criterion.
- Build a new column, give it a name and fill the new column with data computed from properties.
- Transform a table by applying a function to each value in a column.

Missing Computations

- Modify a property in a row or some rows.
- Adding new rows to a table.
- Computations Across on Rows
- Computations Down (Up) Columns
- Delete columns.
- ... More ...