Working With Tables

CMPU 101 – Problem Solving and Abstraction

Peter Lemieszewski
If we skipped over conditional expressions...

• Then, please read chapter 6
  • Or review the lab from Friday
  • And, check the next slide...
If/else expressions

To form an if expression:

\[
\text{if } \langle \text{expression} \rangle:
\langle \text{expression} \rangle
\text{ else: }
\langle \text{expression} \rangle
\text{ end}
\]

- True–false question
- True ("then") answer
- False ("else") answer
Moving on to… Data Types

• Here are some data that can be represented with what we’ve seen so far:
  • A picture of a dog  \textit{Image}
  • The population of Azerbaijan  \textit{Number}
  • The complete text of the \textit{Baghavad Gita} \textit{String}
  • Whether or not I ate breakfast this morning  \textit{Boolean}
A more complex example

• 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.
  • Next slide has a way to get us started...
The Population Function (plain text)

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

Data from: Local Government 2020 Census Interactive Dashboard
wwe1.osc.state.ny.us/localgov/2020-census-interactive-dashboard.htm
use context essentials2021

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
The Population Function (pyret – nested if)

```pyret
use context essentials2021

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
```

Just pointing out nested if stmts here!
What’s all this then? (pyret: new lang. feature)

```pyret
use context essentials2021
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’s all this then? (pyret: raise)

raise...

• stops Pyret from evaluating the program and displays an error message to the user.

• This is different than returning a value, which lets Pyret continue as normal. Our "population" function returns numbers, but if it can't return a number, it will display one of these error messages.

• This is convenient when dealing with unexpected inputs.
  • Implied “contract” between caller & callee is broken
  • Data is not always “pure!”
A more complex example revisited

• What if we wanted to write a program to look up the population of any town in New York?
  • The approach used is not the best approach
    • Not at all!
    • But why?
    • Let’s take another look at the code
The Population Function (pyret)

```pyret
use context essentials2021

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
```
We have New York City...

We have Poughkeepsie...

none of the other 1528 municipalities!
How to consider functions

• KEY IDEA: Separate data from code computations.
  Then, we can reuse the data in as many functions as we want.
• Another KEY IDEA: Table-Driven Programming (my term!)
  i.e. Organize data into tables and we can tailor functions based on tables
What’s a Table?

- It is tabular data made up of rows/columns
  - similar to what you would see in a spreadsheet
Defining a Table in pyret

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
```

• Dileneate data using commas
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
```

- Dileneate data using commas

Q: What type of data makes up a single row?
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
#careful if you copy/paste from here,
#all whitespace is not the same!
end
Steps to Create the Table

1. Name the table (municipalities here) & click Run (“mi” and not “mu” here) -> table is created
2. Type in “municipalities” & press enter key -> table is displayed
3. Good idea to simply include lines 2-7 in your programs, even if they aren’t necessary right now
4. (again) be careful when doing copy/paste, tab keys and space characters have different behavior
• So much data, so little time!
  • We can share tables using the “Publish” menu button rather than typing/copying/pasting/whatever
    • Important for sharing ginormous tables instead of gathering data yourself
• End result is a sharable “link!”
  • That we can, umm, type/copy/paste/whatever.

Share or update the published copy

You can copy the link below to share the most recently published version with others.

`://code.pyret.org/editor#share=1g2BPbORjYIbScjBvPi0fr7yKGMXb-9EB&v=6d122f0`

You can copy the code below to use the published version as a library.

```python
import shared-gdrive("minicipalities", "1g2BPbORjYIbScjBvPi0fr7yKGMXb-9EB")
```

You can also click Update below to copy the current version to the published version, or click Close to exit this window.
Publish or Perish (3)

• End result is a sharable “link!”
  • That we can, umm, type/copy/paste/whatever.

Share or update the published copy

You can copy the link below to share the most recently published version with others.

://code.pyret.org/editor#share=1g2BPbORjYIbScjBvP10fr7yKGMXb-9EB6v=6d122f0

You can copy the code below to use the published version as a library.

import shared-gdrive("minicipalities", "1g2BPbORjYIbScjBvP10fr7yKGMXb-9EB"

You can also click Update below to copy the current version to the published version, or click Close to exit this window.

The juicy bits are enclosed in quotes (google identifier).

We can use “include” or “import” as with textbook data in dcc-2021
• End result is a sharable “link!”
  • That we can, umm, type/copy/paste/whatever.

Share or update the published copy

You can copy the link below to share the most recently published version with others.

://code.pyret.org/editor#share=1g2BPbORjYIbScjBvPi0fr7yKGMXb-9EB5v=6d122f0

You can copy the code below to use the published version as a library.

import shared-gdrive("minicipalities", "1g2BPbORjYIbScjBvPi0fr7yKGMXb-9EB")

You can also click Update below to copy the current version to the published version, or click Close to exit this window.

The name of the file being shared is also used. Good idea to avoid name collisions: create unique names by including (parts of) your name (unlike what I did here!)
• Here we use the table called minicipalities (as if we compiled the data ourselves)
• Let’s use the complete set of data from the NY State website!

```python
use context essentials2021
# Load textbook functions for working with tables
include shared-gdrive("dcic-2021",
"iwyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
# Load the full municipalities table
include shared-gdrive("municipalities",
"18eBAC9Rc8fDpQjUkBQRj7JjvAaXFs")

#Click "Run" then
#To see the tabular data in pyret,
#Type in "municipalities" sans quotes (the name of the table) on RHS

"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 1580 rows...
Turning the Tables (2)

• You should be able to copy/paste these lines into pyret to get the same results:
  • (It worked on my machine at home!)

# Load textbook functions for working with tables
include shared-gdrive("dcic-2021",
     "lwyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
# Load the full municipalities table
include shared-gdrive("municipalities",
     "18eBAc9RcBfDQDpgjUkBQRj7LjgvAaXF"")

#Click "Run" then
#To see the tabular data in pyret,
#Type in "municipalities" sans quotes (the name of the table) on RHS
Ok, I’ve got a table. Now what?

• Now that we have the data in Pyret, we can write programs to “crunch the numbers” i.e. analyze the data!

• We’ll need to learn some basic table manipulation functions first...
Extracting Rows

To get a row out of a table, specify its number, beginning with 0:

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

<table>
<thead>
<tr>
<th>name</th>
<th>Adams</th>
<th>kind</th>
<th>Town</th>
<th>pop-2010</th>
<th>5143</th>
<th>pop-2020</th>
<th>4973</th>
</tr>
</thead>
</table>
Row Data

• 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"
• A note about the format of the above statement
  • The parentheses ( ) are saying that row-n is a function
  • The square brackets [ ] are saying to look up or extract the value of a particular column (the column named “name” here)
Row Data as input to a function

- We can write a function that takes a row as input:

```haskell
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
```

- If you remember Friday’s lab, we can safely omit the explicit checks using if statements when returning a Boolean.

```python
if r["pop-2020"] < r["pop-2010"]:
    true
else:
    false
end
```
Defining a Table in pyret – adding data types

```pyret
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

#careful if you copy/paste from here,
#all whitespace is not the same!
end
```
Filtering data and (re)Ordering Tables

From this point on, we will need to include the textbook functions via:

```
# Load textbook functions for working with tables
include shared-gdrive("dcic-2021", "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
```

(I’ll provide it with sample code; you’ll just need to remember to copy/paste into your programs)
Filtering data and (re)Ordering Tables

From this point on, we will need to include the textbook functions via:

```python
# Load textbook functions for working with tables
include shared-gdrive("dcic-2021", "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
```

(I’ll provide it with sample code; you’ll just need to remember to copy/paste into your programs)
Can we synthesize the data in municipalities to create a new table showing only cities where the population decreased between 2010 and 2020?
Filtering data and (re)Ordering Tables

Can we synthesize the data in municipalities to create a new table showing only cities where the population decreased between 2010 and 2020?

Spoiler Alert: YES, we can do that!
Brainstorming ways to do this: Table as parameter

# Create function that accepts a table and finds all municipalities with pop. decrease
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
Brainstorming ways to do this (2)

```kotlin
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
```

We would need 1500+ if statements? Noooooooo…
Good idea, but awful implementation. We don’t really need to write code like this!
We can write general, all-purpose code to handle this.
filter-with can be used as a function to create a table with the desired set of rows...

filter-with (municipalities, population-decreased)

Two parameters
1. Our (municipalities) table
2. A function that filter-with uses. It will accept a row as a parameter and return a Boolean

- In other words, filter-with will iterate through the rows in our table, keeping what fits its criterion
  - A place with a decrease in population!
filter-with(t :: Table, keep :: (Row -> Boolean))

-> Table

Read this as: Given a table and a predicate on rows, returns a table with only the rows for which the predicate returns true.

Again, two parameters
1. A data type of table
2. A keep function (the predicate) that filter-with uses. It will accept a row as a parameter and return a Boolean
A similar example with municipalities

We can also use `filter-with` to get a table made up of 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)
```
Expanding our options

We can also order the data by the values in one column:

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

**order-by**(*t :: Table, colname :: String, sort-up :: Boolean*)

-> Table

Given a table and the name of a column in that table, return a table with the same rows but ordered based on the named column. If `sort-up` is `true`, the table will be sorted in ascending order, otherwise (false) it will be in descending order.
We can combine all of these too!

How do we create a function that gives us the town with the smallest population?
We can combine all of these too!

How do we use the order-by function to give us the town with the smallest population?

order-by(
    filter-with(municipalities, is-town),
    "pop-2020",
    true).
row-n(0)
Using what we have seen

• **PROBLEM**: We want to know the fastest-growing *towns* in New York.
Using what we have seen (2)

• **PROBLEM**: We want to know the fastest-growing *towns* in New York.
• i.e. we want a table containing only towns, sorted by the *percent change* in population.
• Let’s break the problem statement into manageable parts
Using what we have seen (3)

- **PROBLEM**: We want to know the fastest-growing *towns* in New York.
- i.e. we want a table containing only towns, sorted by the *percent change* in population.
- Let’s break the problem statement into manageable parts
  - Make a new table and...
    1. Filter out the cities, etc. (i.e. only towns)
    2. Calculate percentage change in population
    3. Build a (new) column for percentage change
    4. Sort the table based on that new column in *descending* order
Building a solution (1)

• **PROBLEM**: We want to know the fastest-growing *towns* in New York.
• i.e. we want a table containing only towns, sorted by the *percent change* in population.
• Let’s break the problem statement into manageable parts
  • Make a new table and...
    1. **Filter out the cities, etc. (i.e. only towns)**
       • towns = filter-with(municipalities, is-town)
    2. Calculate percentage change in population
    3. Build a (new) column for percentage change
    4. Sort the table based on that new column in *descending* order
Building a solution (2)

- **PROBLEM**: We want to know the fastest-growing towns in New York.
- i.e. we want a table containing only towns, sorted by the *percent change* in population.
- Let’s break the problem statement into manageable parts
  - Make a new table and...
    1. Filter out the cities, etc. (i.e. only towns)
      - towns = filter-with(municipalities, is-town)
    2. Calculate percentage change in population
      ```kotlin
      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
      ```
    3. Build a (new) column for percentage change
    4. Sort the table based on that new column in *descending* order
Building a solution (3)

- **PROBLEM**: We want to know the fastest-growing *towns* in New York.
- i.e. we want a table containing only towns, sorted by the *percent change* in population.
- Let’s break the problem statement into manageable parts
  - Make a new table and...
  1. Filter out the cities, etc. (i.e. only towns)
     - *towns* = filter-with(municipalities, is-town)
  2. Calculate percentage change in population
     - 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
  3. Build a (new) column for percentage change
     - *towns-with-percent-change* =
       - build-column(towns, "percent-change", percent-change)
  4. Sort the table based on that new column in *descending* order
Building a solution (4)

- **PROBLEM**: We want to know the fastest-growing towns in New York.
- i.e. we want a table containing only towns, sorted by the percent change in population.
- Let’s break the problem statement into manageable parts
  - Make a new table and...
    1. Filter out the cities, etc. (i.e. only towns)
      - towns = filter-with(municipalities, is-town)
    2. Calculate percentage change in population
      - 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
    3. Build a (new) column for percentage change
      - towns-with-percent-change = build-column(towns, "percent-change", percent-change)
  4. Sort the table based on that new column in descending order
     - fastest-growing-towns = order-by(towns-with-percent-change, "percent-change", false)
Full solution... almost (see how it runs!)

- **PROBLEM**: We want to know the fastest-growing towns in New York.

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

• This lecture incorporates material from:
• Kathi Fisler, Brown University,
• Gregor Kiczales, University of British Columbia,
• And, Jonathan Gordon