Problem Solving and Abstraction
(CM PU 101)

Tom Ellman
Lecture 10
## Table Troubles

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>NumTickets</th>
<th>DiscountCode</th>
<th>Delivery</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Josie&quot;</td>
<td>&quot;<a href="mailto:jo@gmail.com">jo@gmail.com</a>&quot;</td>
<td>2</td>
<td>&quot;BIRTHDAY&quot;</td>
<td>&quot;email&quot;</td>
</tr>
<tr>
<td>&quot;Sam&quot;</td>
<td>&quot;<a href="mailto:s@apple.com">s@apple.com</a>&quot;</td>
<td>1</td>
<td>&quot;STUDENT&quot;</td>
<td>&quot;pickup&quot;</td>
</tr>
<tr>
<td>&quot;Bart&quot;</td>
<td>&quot;<a href="mailto:bart@simpson.org">bart@simpson.org</a>&quot;</td>
<td>5</td>
<td>&quot;BRIBERY&quot;</td>
<td>&quot;email&quot;</td>
</tr>
<tr>
<td>&quot;Ernie&quot;</td>
<td>&quot;<a href="mailto:ernie@heaven.org">ernie@heaven.org</a>&quot;</td>
<td>0</td>
<td>&quot;EARLYBIRD&quot;</td>
<td>&quot;pickup&quot;</td>
</tr>
<tr>
<td>&quot;Alvina&quot;</td>
<td>&quot;<a href="mailto:alvie@school.edu">alvie@school.edu</a>&quot;</td>
<td>3</td>
<td>&quot;&quot;</td>
<td>&quot;pickup&quot;</td>
</tr>
<tr>
<td>&quot;Zander&quot;</td>
<td>&quot;<a href="mailto:zandaman@hell.com">zandaman@hell.com</a>&quot;</td>
<td>10</td>
<td>&quot;BIRTHDAY&quot;</td>
<td>&quot;email&quot;</td>
</tr>
<tr>
<td>&quot;Shweta&quot;</td>
<td>&quot;<a href="mailto:snc@this.org">snc@this.org</a>&quot;</td>
<td>3</td>
<td>&quot;STUDENT&quot;</td>
<td>&quot;email&quot;</td>
</tr>
</tbody>
</table>
Does every discount in the table appear in the set of valid discount codes?
Does every discount in the table appear in the set of valid discount codes?

At the moment, we might write:

```plaintext
fun check-discounts1(t :: Table) -> Table:  
  doc: "Find the rows with invalid discount codes."
  fun invalid-code(r :: Row) -> Boolean:  
    not((r["DiscountCode"] == "STUDENT") or 
    (r["DiscountCode"] == "BIRTHDAY") or 
    (r["DiscountCode"] == "EARLYBIRD") or 
    (r["DiscountCode"] == ""))
  end
  filter-with(t, invalid-code)
end
```

Cumbersome!
Every time the set of discount codes changes, we need to change our function.

But how you check the discount-codes shouldn’t change. So let’s write a function that need not change when the data changes.
How can we rewrite this function so the set of valid discount codes is specified outside the function?

Make a table with one column to hold the valid codes?

This would work, but we really don’t need a table if each row has only one datum.
Lists

• Lists are a fundamental type of data structure.
• A list is a container type, i.e. a list contains data.
• A datum on a list is called a “member” or an “element” of the list.
• A list can hold any number of elements.
• A list holds elements in a specific order.
• Normally all elements of a list have the same data type.
A list is like a column, but without the header.

```python
>>> digits = [list: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> digits
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> valid_discounts = [list: "STUDENT", "BIRTHDAY", "EARLYBIRD ", "]

>>> valid_discounts
["STUDENT", "BIRTHDAY", "EARLYBIRD ", "]
```
import lists as L

To work with lists, we import the lists library and we give it a special name — L — to avoid conflicts between the names of functions that work with lists and existing function.

To use a function from the library, we pre-pend the function name with “L.”.
Tools for Working with Lists

? -> List
- [list: ...]
- get-column

List -> List
- L.distinct
- L.filter
- L.append

List -> ?
- L.length
- L.member
- L.any
- L.all
Does every discount in the table appear in the set of valid discount codes?
Does every discount in the table appear in the set of valid discount codes?

```plaintext
valid-discounts = [list: "STUDENT","BIRTHDAY","EARLYBIRD"]

fun check-discounts2(t :: Table) -> Table:
  doc: "Filter out rows whose discount code is not valid."

  fun valid-code(r :: Row) -> Boolean:
    L.member(valid-discounts, r["discount"])
  end

  filter-with(t, valid-code)
end
```

If the valid discounts change, we need only to update the list: valid-discounts. The function code stays the same.
When we’ve been working with tables we’ve been using the data type Row, but we never saw a Column data type!

Why not? Well, a column consists of an ordered collection of values, of unbounded length.

A column is a lot like a list!

```python
>>> event-data.get-column("name")
```

The `get-column` function returns a list.
Find the names of people who got a specific discount type.
Find the names of people who got a specific discount type.

```plaintext
fun people-with-discount(t :: Table, d :: String) -> List:
    rows = filter-with(t, lam(r): r["DiscountCode"] == d end)
    rows.get-column("Name")
end

>>> people-with-discount(event-data-clean,"STUDENT")
["Sam", "Shweta"]
```
# Recipes

pancakes = [list: "egg", "butter", "flour", "sugar", "salt", "baking powder", "blueberries"]
dumplings = [list: "egg", "wonton wrappers", "pork", "garlic", "salt", "gf soy sauce"]
pasta = [list: "spaghetti", "tomatoes", "garlic", "onion"]

A recipe is a list of required ingredients.
A dietary restriction is a list of restricted ingredients.

An inventory is a list of ingredients on hand.
meal-ingredients = L.append(pancakes, L.append(dumplings, pasta))

All ingredients needed for a (high-carb) meal, possibly with duplicates.

A shopping list is a list of distinct (L.distinct) ingredients that are not members (L.member) of the inventory list.
All ingredients needed for a (high-carb) meal, possibly with duplicates.

```haskell
meal-ingredients = L.append(pancakes, L.append(dumplings, pasta))
```

A shopping list is a list of distinct (L.distinct) ingredients that are not members (L.member) of the inventory list.

```haskell
fun shopping-list(ingredients :: List, inventory :: List) -> List:
    d = L.distinct(ingredients)
    L.filter(lam(i): not(L.member(inventory, i)) end, d)
end
```
Check whether a recipe is gluten-free.
**L.filter**(<predicate>,<input-list>) returns a new list containing just the members of the input list for which the predicate returns **true**.

In this function we use **L.filter** to make a list of all ingredients in the recipe that are on the gluten list. If the new list is empty (**L.length** is zero), the recipe is gluten-free (return **true**).
Alternate solution using: \texttt{L.any}

\texttt{L.any(<predicate>,<list>)} returns \texttt{true} if the predicate returns \texttt{true} when applied to at least one member of the list.
L.any(<predicate>,<list>) returns true if the predicate returns true when applied to at least one member of the list.

In this function, we use L.any to ensure that there are not any members of recipe that are also members of gluten.
Alternate alternate solution using: \texttt{L.all}

\texttt{L.all(<predicate>,<list>)} is \texttt{true} if the predicate returns \texttt{true} for each member of the list.
L.all(⟨predicate⟩,⟨list⟩) is **true** if the predicate returns **true** for each member of the list.

In this function, we use L.all to ensure that each member of recipe is not a member of gluten.
Write a function to check whether a recipe is gluten-free, using \texttt{L.any}.
fun is-vegan1(recipe :: List<String>) -> Boolean:
    doc: "Return true if none of the ingredients are non-vegan"
    not(
        L.any(
            lam(i):
                L.member(meat, i) or
                L.member(dairy, i) or
                L.member(eggs, i)
            end,
            recipe)
    where:
        is-vegan1(pasta) is true
        is-vegan1(dumplings) is false
    end
Write a function to check whether a recipe is gluten-free, using \texttt{L.all}.
fun is-vegan2(recipe :: List<String>) -> Boolean:
  doc: "Return true if all the ingredients are vegan"
  L.all(
    lam(i):
      not(L.member(meat, i)) or
      L.member(dairy, i) or
      L.member(eggs, i))
    end,
    recipe)

where:
  is-vegan2(pasta) is true
  is-vegan2(dumplings) is false
end
Veganize a Meal

pancakes =
[list: "egg", "butter", "flour", "sugar",
  "salt", "baking powder", "blueberries"]

Vegan-pancakes =
[list: "flax", "margarine", "flour",
  "sugar", "salt", "baking powder", "blueberries"]
Veganize a Meal

1. Write a function **veganize-ingredient** that takes a non-vegan ingredient and returns something vegan to replace it.

2. Use **veganize-ingredient** repeatedly to replace each non-vegan ingredient on the recipe list. But how can we do this for a recipe list of any length?
Veganize a Meal (Step 1)

Write a function `veganize-ingredient` that takes a non-vegan ingredient and returns something vegan to replace it.
fun veganize-ingredient1(ingredient :: String) -> String:
    doc: "Replace an ingredient if it isn't vegan"
    if ingredient == "egg":
        "flax"
    else if ingredient == "pork":
        "mushroom"
    else if ingredient == "beef":
        "tofu"
    else if ingredient == "chicken":
        "chick'n"
    else if ingredient == "butter":
        "margarine"
    else:
        ingredient
    end
where:
    veganize-ingredient1("chicken") is "chick'n"
    veganize-ingredient1("apple") is "apple"
end

Cumbersome! Not easy to update to handle other non-vegan ingredients.
Let’s separate the data from the code, using a table.

```ruby
replacements = table: ingredient, replacement
  row: "egg", "flax"
  row: "pork", "mushroom"
  row: "beef", "tofu"
  row: "chicken", "chick'in"
  row: "butter", "margarine"
end
```

Each row indicates how a non-vegan ingredient should be replaced by a vegan ingredient.

Could we use a list, or two lists instead of a table?
Rewrite veganize-ingredient1 to use the replacements table.
fun veganize-ingredient2(ingredient :: String) -> String:
    doc: "Replace an ingredient if it isn't vegan"
    temp = filter-with(replacements,
        lam(r): r["ingredient"] == ingredient end)
    if (temp.length() > 0):
        temp.row-n(0)["replacement"]
    else: ingredient
end
end

Here we find the replacement by locating a row whose ingredient column matches the ingredient parameter.
Here we find the replacement by locating a row whose ingredient column matches the ingredient parameter.
Veganize a Meal (Step 2)

Use **veganize-ingredient** repeatedly to replace each non-vegan ingredient on the recipe list. **But how can we do this for a recipe list of any length?**
L.Map takes a function and a list as parameters. It applies the function to each list element and replaces the element with the function result.

```python
list-of-numbers = [list: 0,1,2,3,4,5,6,7,8,9]
increment-function = lam(n): n + 1 end

>>> increment-function(41)
42

>>> L.map(increment-function, list-of-numbers)
[list: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```
Use `L.map` and `veganize-ingredient2` repeatedly to replace each non-vegan ingredient on the recipe list.
L.Map takes a function and a list as parameters. It applies the function to each list element and replaces the element with the function result.
Acknowledgments

This class incorporates material from:

• Marc Smith, Vassar College
• Jason Waterman, Vassar College
• Jonathan Gordon, Vassar College
• Kathi Fisler, Brown University
• Doug Woos, Brown University