Map
What if we want to take a recipe and make it vegan?

Let’s think about what the input and output should be. We’re starting with the list of ingredients,

```python
list: "egg", "butter", "flour", "sugar", "salt", "baking powder", "blueberries"
```

and it should become, say,

```python
list: "flax", "margarine", "flour", "sugar", "salt", "baking powder", "blueberries"
```
We need an operation that produces a list, where some of the items are different than in the input list.

We can’t do this with member, distinct, or filter.
L.map is similar to the `transform-column` function we used with tables.

It takes a function and a list as input and produces a list where each item is the result of running the function on the corresponding item of the input list.
fun veganize-ingredient(ingredient :: String) -> String:

doc: "Change a non-vegan ingredient to its vegan equivalent"

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
end
fun veganize-recipe(recipe :: List<String>) -> List<String>:
    doc: "Update a recipe to be vegan"
    L.map(veganize-ingredient, recipe)
where:
    veganize-recipe(pasta) is pasta
    veganize-recipe(dumplings) is
        [list: "flax", "wonton wrappers",
            "mushroom", "garlic", "salt", "soy sauce"]
end
Because `veganize-ingredient` is just a helper function for `veganize-recipe`, we might prefer to define it inside `veganize-recipe`:

```kotlin
fun veganize-recipe(recipe :: List<String>) -> List<String>:
    fun veganize-ingredient(ingredient :: String) -> String:
        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
    end

    L.map(veganize-ingredient, recipe)
end

where:

veganize-recipe(pasta) is pasta
veganize-recipe(dumplings) is [list: "flax", "wonton wrappers", "mushroom", "garlic", "salt", "soy sauce"]
end
```
Operation signatures
What operations have we seen so far?

L.member
List, ⟨item⟩ -> Boolean
Indicates whether the item is in the list

L.distinct
List -> List
Returns the unique values from input list

L.filter
Function, List -> List
Returns list of items from input list on which function returns true (in the same order as in the input list)

...
What operations have we seen so far?

**L.member**

List, ⟨item⟩ → Boolean

Indicates whether the item is in the list

**L.distinct**

List → List

Returns the unique values from input list

**L.filter**

Function, List → List

Returns list of items from input list on which function returns true (in the same order as in the input list)

...
What operations have we seen so far?

**L.member**
List, ⟨item⟩ → Boolean
Indicates whether the item is in the list

**L.distinct**
List → List
Returns the unique values from input list

**L.filter**
Function, List → List
Returns list of items from input list on which function returns true
(in the same order as in the input list)
What operations have we seen so far?

**L.member**

List, ⟨item⟩ → Boolean

Indicates whether the item is in the list

**L.distinct**

List → List

Returns the unique values from input list

**L.filter**

(a → Boolean), List → List

Returns list of items from input list on which function returns true (in the same order as in the input list)

...
What operations have we seen so far?

L.member

List, ⟨item⟩ → Boolean

Indicates whether the item is in the list

L.distinct

List → List

Returns the unique values from input list

L.filter

(a → Boolean), List<a> → List<a>

Returns list of items from input list on which function returns true (in the same order as in the input list)

...
What about `map`?

Function, List -> List
What about **map**?

\[(a \rightarrow b), \text{List} \rightarrow \text{List}\]

A function that takes an input of some type – call it \(a\) – and returns an output of type \(b\), which might be the same as \(a\) or might not. E.g., we might be taking a Number and converting it to a String.
What about **map**?

\[(a \rightarrow b), \text{List}<a> \rightarrow \text{List}\]

The input list needs to be made of \textit{as that we can give to that function.}
What about `map`?

\[(a \rightarrow b), \text{ List}\langle a \rangle \rightarrow \text{ List}\langle b \rangle\]

The output list will be made of the `bs` that the function returned.
For a full list of operations and their signatures, see the Pyret lists documentation.
Lists and tables
We’ve seen one way of describing a set of recipes – as a set of hardcoded lists.

This makes sense when we have a small set of recipes that doesn’t change often, but we might want something better.
Another possibility would be to use a table with one column per ingredient:

```
recipes1 = table:
    name :: String, spaghetti :: Boolean, milk :: Boolean,
    tomatoes :: Boolean, onions :: Boolean, blueberries :: Boolean,
    garlic :: Boolean, salt :: Boolean
row: "pasta", true, false, true, true, false, true, true
end
```

<table>
<thead>
<tr>
<th>name</th>
<th>spaghetti</th>
<th>milk</th>
<th>tomatoes</th>
<th>onions</th>
<th>blueberries</th>
<th>garlic</th>
<th>salt</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;pasta&quot;</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
The table would let us make plots and charts using the operations we know in Pyret

The lists are easier to write and modify

The tables could become sparse if we add more categories and ingredients
Whether you use tables or lists depends on the data you have and how you plan to use it.

For the programs we’ve written today, the lists were sufficient and lightweight, so they were the better choice.

Other programs might have benefitted from the table-shaped data.
Another possibility we’ll return to later is combining lists and tables, e.g.,

```plaintext
recipes2 = table:
  name :: String, ingredients :: List<String>
  row: "pasta", [list: "spaghetti", "tomatoes", "garlic", "onion", "salt"]
end
```

<table>
<thead>
<tr>
<th>name</th>
<th>ingredients</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;pasta&quot;</td>
<td>[list: &quot;spaghetti&quot;, &quot;tomatoes&quot;, &quot;garlic&quot;, &quot;onion&quot;, &quot;salt&quot;]</td>
</tr>
</tbody>
</table>
Lecture code:

https://code.pyret.org/editor#share=1j8jQBFc7dt04L6wqwddliK800AJzeP2a&v=1904b2c