Introduction to Lists

3 October 2022
Where are we?
We’ve seen that when you want a row of a table, you use `row-n` and get a Row.

What about getting a column?
<table>
<thead>
<tr>
<th>timestamp</th>
<th>house</th>
<th>stem-level</th>
<th>sleep-hours</th>
<th>schoolwork-hours</th>
<th>student-athlete</th>
<th>extracurricular-hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;9/21/2022 21:05:52&quot;</td>
<td>&quot;OTHER&quot;</td>
<td>10</td>
<td>6</td>
<td>8</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:08:21&quot;</td>
<td>&quot;Strong House (1893)&quot;</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>false</td>
<td>2</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:09:01&quot;</td>
<td>&quot;Lathrop House (1901)&quot;</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>true</td>
<td>4</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:09:43&quot;</td>
<td>&quot;Lathrop House (1901)&quot;</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:29:32&quot;</td>
<td>&quot;Lathrop House (1901)&quot;</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>false</td>
<td>2</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:33:00&quot;</td>
<td>&quot;Cushing House (1927)&quot;</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>false</td>
<td>2</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:38:40&quot;</td>
<td>&quot;Josselyn House (1912)&quot;</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>false</td>
<td>1.5</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:41:36&quot;</td>
<td>&quot;Jewett House (1907)&quot;</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>true</td>
<td>2</td>
</tr>
<tr>
<td>&quot;9/21/2022 21:43:53&quot;</td>
<td>&quot;Main Building (1861)&quot;</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>true</td>
<td>3</td>
</tr>
<tr>
<td>&quot;9/21/2022 22:22:48&quot;</td>
<td>&quot;Davison House (1902)&quot;</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>false</td>
<td>4</td>
</tr>
</tbody>
</table>
```python
>>> student-data-cleaned.get-column("house")
["OTHER", "Strong House (1893)", "Lathrop House (1901)", "Lathrop House (1901)", ...]
```
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 really just a *List*!
Lists can be very convenient!
fun normalize-house(house :: String) -> String:

doc: "Return one of the nine Vassar houses or 'OTHER'"

if (house == "Main Building (1861)"
  or house == "Strong House (1893)"
  or house == "Raymond House (1897)"
  or house == "Lathrop House (1901)"
  or house == "Davison House (1902)"
  or house == "Jewett House (1907)"
  or house == "Josselyn House (1912)"
  or house == "Cushing House (1927)"
  or house == "Noyes House (1958)"):

  house

else:

  "OTHER"
end

where:

  normalize-house("Main Building (1861)") is "Main Building (1861)"
  normalize-house("Offcampus") is "OTHER"
houses = [list:
    "Main Building (1861)",
    "Strong House (1893)",
    "Raymond House (1897)",
    "Lathrop House (1901)",
    "Davison House (1902)",
    "Jewett House (1907)",
    "Josselyn House (1912)",
    "Cushing House (1927)",
    "Noyes House (1958)"
]

fun normalize-house(house :: String) -> String:
    doc: "Return one of the nine Vassar houses or 'Other'"
    if houses.member(house):
        house
    else:
        "OTHER"
    end

where:
    normalize-house("Main") is "Main Building (1861)"
    normalize-house("Offcampus") is "OTHER"
Mad Libs!
Thousands of years ago, there were calendars that enabled the ancient people to divide a year into twelve months, each month into four weeks, and each week into seven days. At first, people told time by a sun clock, sometimes known as the sundial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather clock, the pocket watch, the alarm clock, and, of course, the wristwatch. Children learn about clocks and time almost before they learn their A-B-Cs. They are taught that a day consists of 24 hours, an hour has 60 minutes, and a minute has 60 seconds. By the time they are in kindergarten, they know if the big hand is at twelve and the little hand is at three, that it is number o'clock. I wish we could continue this lesson, but we've run out of time.
Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun, each month into Plural-Noun weeks, and each week into seven Plural-Noun. At first, people told time by a sun clock, sometimes known as the Plural-Noun dial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather Plural-Noun, the pocket Plural-Noun, the alarm Plural-Noun, and, of course, the Plural-Noun watch. Children learn about clocks and time almost before they learn their A-B-Alphabet-Letter-s. They are taught that a day consists of 24 Plural-Noun, an hour has 60 Plural-Noun, and a minute has 60 Plural-Noun. By the time they are in Kindergarten, they know if the big Body-Part is at twelve and the little Body-Part is at three, that it is Number o’clock. I wish we could continue this Adjective lesson, but we’ve run out of Noun.
How can we represent a text?
Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun, each month into Number weeks, and each week into seven Plural-Noun. At first, people told time by a sun clock, sometimes known as the Noun dial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather Noun, the pocket Noun, the alarm Noun, and, of course, the Body-Part watch. Children learn about clocks and time almost before they learn their A-B- Alphabet-Letter s. They are taught that a day consists of 24 Plural-Noun, an hour has 60 Plural-Noun, and a minute has 60 Plural-Noun. By the time they are in Kindergarten, they know if the big Body-Part is at twelve and the little Body-Part is at three, that it is Number o'clock. I wish we could continue this Adjective lesson, but we’ve run out of Noun.
template = "Thousands of Plural-Noun ago, ..."

template-words = string-split-all(template, " ")

>>> template-words
[list: "Thousands", "of", "Plural-Noun", "ago", ...]
template = "Thousands of Plural-Noun ago, ...

template-words = string-split-all(template, " ")

>>> template-words
[list: "Thousands", "of", "Plural-Noun", "ago", ...]

We need to substitute a random plural noun here!
Thousands of Plural–Noun ago, ...

string-split-all

[ list: "Thousands", "of", "Plural–Noun", "ago", ... ]
"Thousands of Plural-Noun ago, ..."

string-split-all

[list: "Thousands", "of", "Plural-Noun", "ago", ...]

Something like transform-column but for lists

[list: "Thousands", "of", "gazebos", "ago", ...]
"Thousands of Plural-Noun ago, ...

string-split-all

[ list: "Thousands", "of", "Plural-Noun", "ago", ... ]

Something like transform-column but for lists

[ list: "Thousands", "of", "gazebos", "ago", ... ]

Needs a helper function!
"Thousands of Plural-Noun ago, ..."

string-split-all

[list: "Thousands", "of", "Plural-Noun", "ago", ...]

Something like transform-column but for lists using

[list: "Thousands", "of", "gazebos", "ago", ...]

**substitute-word**

"Thousands" → "Thousands"
"Plural-Noun" → "gazebos"
I’d write the helper function first!
fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
...
where:
    substitute-word("Thousands") is "Thousands"
    substitute-word("Plural-Noun") is ...
end

Uh oh! We don’t know what particular word it will be!
fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
...
where:
  substitute-word("Thousands") is "Thousands"
  substitute-word("Plural-Noun") is-not "Plural-Noun"
end

We know what it isn’t!
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    ...
where:
    substitute-word("Thousands") is "Thousands"
    substitute-word("Plural-Noun") is-not "Plural-Noun"
    plural-nouns.member(substitute-word("Plural-Noun"))
    is true
end

And we know it’s one of the right choices!
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
  ...
  where:
  substitute-word("Thousands") is "Thousands"
  substitute-word("Plural–Noun") is–not "Plural–Noun"
  plural-nouns.member(
    substitute-word("Plural–Noun"))
  is true
end

The left part of an example can be any expression!
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
  if w == "Plural-Noun":
    ...
  else:
    w
  end
else:
  w
end
where:
  ...
end
We need a random element of a list.

Time to check the Pyret documentation!
3.2.5 Random Numbers

num-random :: (max :: Number) -> Number

Returns a pseudo-random positive integer from 0 to max - 1.

Examples:

check:
  fun between(min, max):
    lam(v): (v >= min) and (v <= max) end end
  end
  for each i from range(0, 100):
    block:
      n = num-random(10)
      print(n)
      n satisfies between(0, 10 - 1)
    end end

num-random-seed :: (seed :: Number) -> Nothing

Sets the random seed. Setting the seed to a particular number makes all future uses of random produce the same sequence of numbers. Useful for testing and debugging functions that have random behavior.

Examples:

check:
  num-random-seed(0)
  n = num-random(1000)
We didn’t find a built-in way to get a random element of a list, but we found a way to get a random number.

How could we use this?
.get :: (n :: Number) -> a

Returns the n-th element of the given List, or raises an error if n is out of range.

Examples:

    check:
    [list: 1, 2, 3].get(0) is 1
    [list: ].get(0) raises "too large"
    [list: 1, 2, 3].get(-1) raises "invalid argument"
    end

.set :: (n :: Number, e :: a) -> List<a>

Returns a new List with the same values as the given List but with the n-th element set to the given value, or raises an error if n is out of range.

Examples:

    check:
    [list: 1, 2, 3].set(0, 5) is [list: 5, 2, 3]
    [list: ].set(0, 5) raises "too large"
    end

.foldl :: (f :: (a, Base -> Base), base :: Base) -> Base

Computes f(last-elt, ... f(second-elt, f(first-elt, base))...). For empty, returns base.

In other words, .foldl uses the function f, starting with the base value, of type Base, to
With a table, we could use `.row-n` to get a specific row by its index number.

With a list, we can use `.get` to get an item.
Get a random number

Get then list element positioned at that number
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand = num-random(3)
        plural-nouns.get(rand)
    else:
        w
    end
where:
    ...
end
plural-nouns = [list: "gazebos", "avocados", "pandas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand = num-random(3)
        plural-nouns.get(rand)
    else:
        w
    end
where:
    ...
end
fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand = num-random(3)
        plural-nouns.get(rand)
    else:
        w
    end
end

plural-nouns = [list: "gazebos", "avocados", "pandas", "quokkas"]
plural-nouns = [list: "gazebos", "avocados", "pandas", "quokkas"]

fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand = num-random(length(plural-nouns))
        plural-nouns.get(rand)
    else:
        w
    end

where:
    ...
end
Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun, each month into Number weeks, and each week into seven Plural-Noun. At first, people told time by a sun clock, sometimes known as the Noun dial. Ultimately, they invented the great timekeeping devices of today, such as the grandfather Noun, the pocket Noun, the alarm Noun, and, of course, the Body-Part watch. Children learn about clocks and time almost before they learn their A-B- Alphabet-Letter s. They are taught that a day consists of 24 Plural-Noun, an hour has 60 Plural-Noun, and a minute has 60 Plural-Noun. By the time they are in Kindergarten, they know if the big Body-Part is at twelve and the little Body-Part is at three, that it is Number o'clock. I wish we could continue this Adjective lesson, but we’ve run out of Noun.
plural-nouns =
  [list: "gazebos", "avocados", "pandas", "quokkas"]

numbers =
  [list: ",-1", "42", "a billion"]

nouns =
  [list: "apple", "computer", "borscht"]

body-parts =
  [list: "elbow", "head", "spleen"]

alphabet-letters =
  [list: "A", "C", "Z"]

adjectives =
  [list: "funky", "boring"]
fun substitute_word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand = num-random(length(plural-nouns))
        plural-nouns.get(rand)
    else if w == "Number":
        rand = ...
    else:
        w
    end
where:
    ...
end
fun substitute-word(w :: String) -> String:
  doc: "Substitute a random word if w is a category"
  if w == "Plural-Noun":
    rand = num-random(length(plural-nouns))
    plural-nouns.get(rand)
  else if w == "Number":
    rand = ...
  else:
    w
  end
end
where:
  ...
end

Don’t repeat yourself!
fun rand-word(l :: List<String>) -> String:
    doc: "Return a random word in the given list"
    rand = num-random(length(l))
    l.get(rand)
where:
    plural-nouns.member(rand-word(plural-nouns)) is true
    numbers.member(rand-word(numbers)) is true
end
fun substitute-word(w :: String) -> String:
    doc: "Substitute a random word if w is a category"
    if w == "Plural-Noun":
        rand-word(plural-nouns)
    else if w == "Number":
        rand-word(numbers)
    else if w == "Noun":
        rand-word(nouns)
    else if w == "Body-Part":
        rand-word(body-parts)
    else if w == "Alphabet-Letter":
        rand-word(alphabet-letters)
    else if w == "Adjective":
        rand-word(adjectives)
    else:
        w
    end
end

This is still a bit repetitious – but it’s good enough for today!
Go back to the task plan.

We’ve completed our helper, and now we need to

split the input into words

run the helper on every word in the list

Similar to how `transform-column` runs a function on every row of a table.
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    ...
end
Go back to the task plan.

We’ve completed our helper, and now we need to

✔️ split the input into words

run the helper on every word in the list

Similar to how transform-column runs a function on every row of a table.
Go back to the task plan.

We’ve completed our helper, and now we need to

- split the input into words
- run the helper on every word in the list

Similar to how `transform-column` runs a function on every row of a table.

*This is called map!*
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    map(substitute-word, t)
    ...
end
Go back to the task plan.

We’ve completed our helper, and now we need to

✅ split the input into words

✅ run the helper on every word in the list

Similar to how `transform-column` runs a function on every row of a table.

Ok – are we done?
fun mad-libs(t :: String) -> String:
  doc: "Randomly fill in the blanks in the mad libs template"
  words = string-split-all(t, " ")
  map(substitute-word, t)
  ...
end

This gives us a list of strings. How can we join it back into a single string?
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    words-sub = map(substitute-word, words)
    join-str(with-sub, " ")
end
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    words-sub = map(substitute-word, t)
    join-str(words-sub, " ")
where:
    ...
What do we know is true about the output?
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    words-sub = map(substitute-word, t)
    join-str(words-sub, " ")

where:
    mad-libs(template) is-not template
fun mad-libs(t :: String) -> String:
    doc: "Randomly fill in the blanks in the mad libs template"
    words = string-split-all(t, " ")
    words-sub = map(substitute-word, t)
    join-str(words-sub, " ")
where:
    mad-libs(template) is-not template
    string-contains(mad-libs(template), "Plural-Noun")
    is false
end
Preview: Lists and recursion
What if \texttt{join-str} didn’t already exist for our convenience?

To write a function that processes a list element by element, we need to understand the real nature of lists.
A list consists of two parts: a `first` element and the `rest` of the list.

```python
>>> l = [list: 1, 2, 3]
>>> l.first
1
>>> l.rest
[list: 2, 3]
```
The first element is linked to the rest and so on until we reach the empty list:

```python
>>> link(1, empty)
[list: 1]
>>> link(1, link(2, link(3, empty)))
[list: 1, 2, 3]
```
When we write a function that recursively processes a list, we deal with these two cases – linking an element or being empty:

```plaintext
fun add-nums(l :: List<Number>) -> Number:
    cases (List) l:
        | empty => 0
        | link(f, r) => f + add-nums(r)
    end
where:
    add-nums([list:    ]) is 0
    add-nums([list:    1]) is 1 + 0
    add-nums([list: 2, 1]) is 2 + 1 + 0
end
```
In the case of joining strings, we need to know not just if the current list is empty but is the rest of the rest empty. This is how we know whether to add a space or not.
fun join-with-spaces(l :: List<String>) -> String:
    doc: "Join the strings in l with a space between each one"
    cases (List) l:
        | empty => ""
        | link(f, r) =>
            cases (List) r:
                | empty => f
                | link(fr, rr) =>
                    f + " " + join-with-spaces(r)
            end
        end
    end
where:
    join-with-spaces([list: ] ) is ""
    join-with-spaces([list: "y"] ) is "y" + ""
    join-with-spaces([list: "x", "y"] ) is "x" + " " + "y" + ""
Class code:

https://tinyurl.com/101-2022-10-03