

CMPU 101 §52 · Computer Science I

# Tables and Lists

8 February 2023



Where are we?

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*Tables!*

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*Rows!*

*stats* =

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*How do I get just this row from **stats**?*

<i>n</i>	player	team	pos	games	pts
0	"Natalie Achonwa"	"MIN"	"F"	22	116
1	"Julie Allemand"	"CHI"	"G"	25	74
2	"Lindsay Allen"	"MIN"	"G"	9	60
3	"Rebecca Allen"	"NYL"	"G"	25	174
4	"Yvonne Anderson"	"CON"	"G"	11	35
5	"Kristine Anigwe"	"PHO"	"F-C"	10	15
6	"Ariel Atkins"	"WAS"	"G"	36	527
7	"Amy Atwell"	"LAS"	"F"	4	3
8	"Shakira Austin"	"WAS"	"C-F"	36	312
9	"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*How do I get just this row from **stats**?*

*stats* =

`stats.row-n(2)`

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*How do I get just the rows for players who are guards?*

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35

```
fun is-guard(player :: Row)
  -> Boolean:
  doc: "Return true if the player's
primary position is guard"
  player["pos"] == "G"
where:
  is-guard(t.row-n(0)) is false
  is-guard(t.row-n(1)) is true
end
```

*How do I get just  
the rows for players  
who are guards?*

```
filter-with(stats,  
            is-guard)
```



player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35

```
fun is-guard(player :: Row)
  -> Boolean:
  doc: "Return true if the player's
  primary position is guard"
  player["pos"] == "G"
where:
  is-guard(t.row-n(0)) is false
  is-guard(t.row-n(1)) is true
end
```

```
filter-with(stats,
  lam(player):
    player["pos"] == "G"
end)
```

*How do I get just  
the rows for players  
who are guards?*

*This is the only place we want to use this helper function, so there's no need to name it, document it, etc.*

*We can just write it inline as a **lambda expression**.*

```
filter-with(stats, lam(player): player["pos"] == "G" end)
```

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*What about columns?*

*How can I add a new column like this?*

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*frequent-player*

"very"

"very"

"no"

"very"

"somewhat"

"somewhat"

"very"

"no"

"very"

"very"

*How can I add a new column like this?*

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527

*frequent-player*

"very"

"very"

"no"

"very"

"somewhat"

"somewhat"

"very"

*Although we'll only use this function here, it'll be too long to be conveniently written as a lambda expression.*

`build-column(stats, "frequent-player", how-frequent)`

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60

*frequent-player*

"very"

"very"

"no"

"very"

"somewhat"

```
fun how-frequent(player :: Row)
  -> String:
  if player["games"] >= 20:
    "very"
  else if player["games"] >= 10:
    "somewhat"
  else:
    "no"
  end
end
```

*Be sure to add a docstring and **where** block to make this definition complete!*

```
build-column(stats, "frequent-player", how-frequent)
```





So, we've seen that we can build a new column based on the values in each row, but what if we just want to change an existing column?

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*A fake WNBA fan like me can't remember what these team abbreviations stand for.*

*Let's fill in the actual team names.*

What *are* the team names?

## WESTERN

---

DAL



Dallas Wings

LVA



Las Vegas Aces

LAS



Los Angeles Sparks

MIN



Minnesota Lynx

PHO



Phoenix Mercury

SEA



Seattle Storm

## EASTERN

---

ATL



Atlanta Dream

CHI



Chicago Sky

CON



Connecticut Sun

IND



Indiana Fever

NYL



New York Liberty

WAS



Washington Mystics

```
fun team-name(abbr :: String) -> String:  
  doc: "Return the name of the team with the given  
  abbreviation"  
  ...  
where:  
  team-name("NYL") is "New York Liberty"  
  team-name("CHI") is "Chicago Sky"  
  ...  
end
```

```
fun team-name(abbr :: String) -> String:
  doc: "Return the name of the team with the given
abbreviation"
  if abbr == "DAL": "Dallas Wings"
  else if abbr == "LVA": "Las Vegas Aces"
  ...
end
where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end
```

*This will work, but remember what we said when we introduced tables for looking up population: We want to **separate data from computation.***

```
teams =  
  table: abbr, name  
  row: "DAL", "Dallas Wings"  
  row: "LVA", "Las Vegas Aces"  
  row: "LAS", "Los Angeles Sparks"  
  row: "MIN", "Minnesota Lynx"  
  row: "PHO", "Phoenix Mercury"  
  row: "SEA", "Seattle Storm"  
  row: "ATL", "Atlanta Dream"  
  row: "CHI", "Chicago Sky"  
  row: "CON", "Connecticut Sun"  
  row: "IND", "Indiana Fever"  
  row: "NYL", "New York Liberty"  
  row: "WAS", "Washington Mystics"  
end
```

*Advantage: This makes it easy to add new teams or more information about these teams, in a central place.*

```
teams = ...
```

```
fun team-name(abbr :: String) -> String:  
  doc: "Return the name of the team with the given  
  abbreviation"
```

```
  # Get the row with abbreviation `abbr`  
  # Return the value in the `name` column
```

```
where:
```

```
  team-name("NYL") is "New York Liberty"
```

```
  team-name("CHI") is "Chicago Sky"
```

```
  ...
```

```
end
```



```
teams = ...
```

```
fun team-name(abbr :: String) -> String:  
  doc: "Return the name of the team with the given  
  abbreviation"  
  matches =  
    filter-with(teams, lam(r): r["abbr"] == abbr end)  
  team = matches.row-n(0)  
  # Return the value in the `name` column
```

```
where:
```

```
team-name("NYL") is "New York Liberty"
```

```
team-name("CHI") is "Chicago Sky"
```

```
...
```

```
end
```

```
teams = ...
```

```
fun team-name(abbr :: String) -> String:  
  doc: "Return the name of the team with the given  
abbreviation"  
  matches =  
    filter-with(teams, lam(r): r["abbr"] == abbr end)  
  team = matches.row-n(0)  
  team["name"]  
where:  
  team-name("NYL") is "New York Liberty"  
  team-name("CHI") is "Chicago Sky"  
  ...  
end
```

```
teams = ...
```

```
fun team-name(abbr :: String) -> String:  
  doc: "Return the name of the team with the given  
abbreviation"  
  matches =  
    filter-with(teams, lam(r): r["abbr"] == abbr end)  
  team = matches.row-n(0)  
  team["name"]
```

```
where:
```

```
team-name("NYL") is "New York Liberty"
```

```
team-name("CHI") is "Chicago Sky"
```

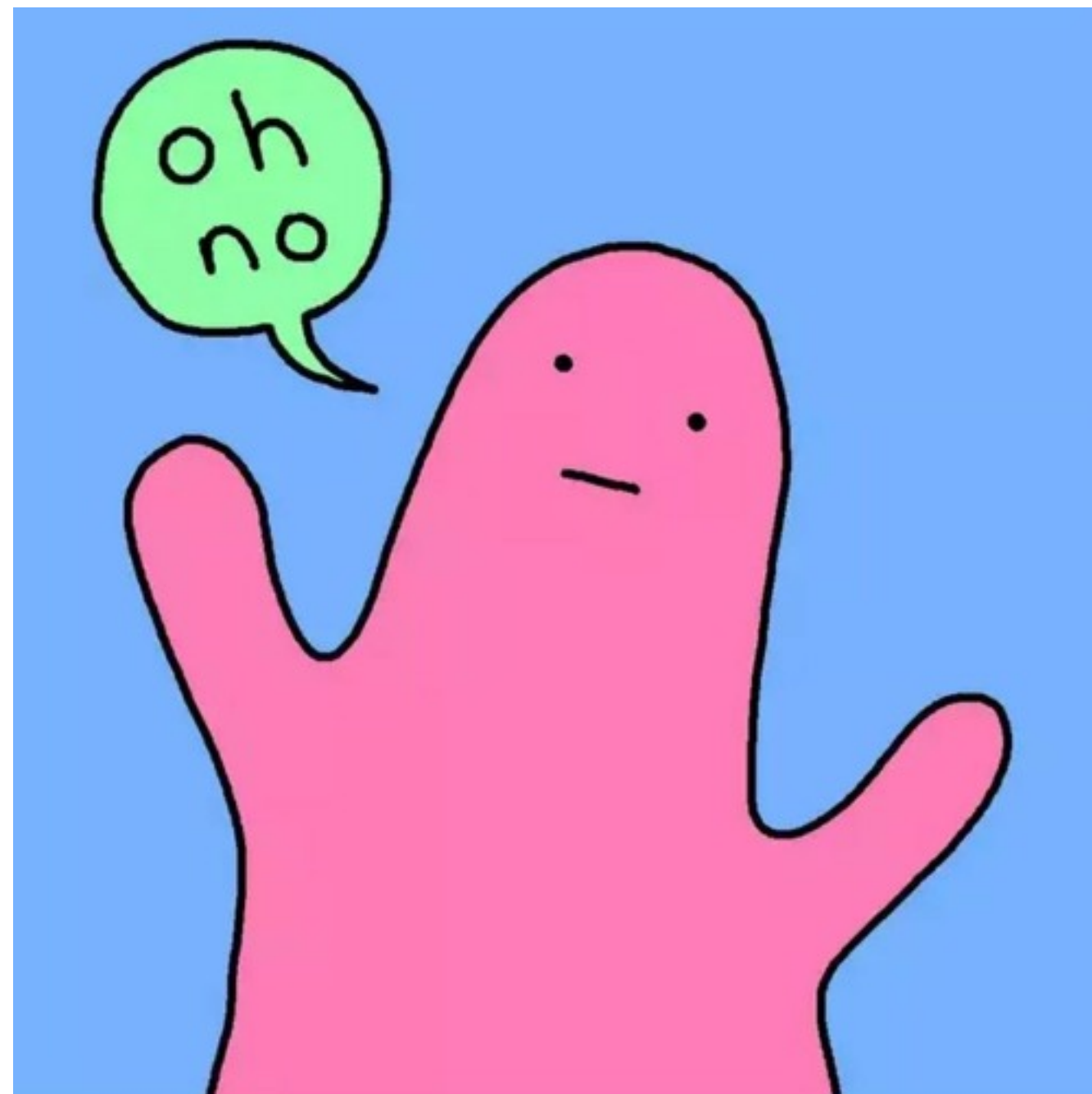
```
...
```

```
end
```

```
transform-column(stats, "team", team-name)
```

row-n-too-large

(Show program evaluation trace...)



```
teams = ...
```

```
fun team-name(abbr :: String) -> String:  
  doc: "Return the name of the team with the given  
  abbreviation"  
  matches =  
    filter-with(teams, lam(r): r["abbr"] == abbr end)  
  team = matches.row-n(0)  
  team["name"]
```

```
where:
```

```
team-name("NYL") is "New York Liberty"
```

```
team-name("CHI") is "Chicago Sky"
```

```
...
```

```
end
```

```
transform-column(stats, "team", team-name)
```

*0 is too big? That means there were no matching rows! An abbreviation not in our table. What is it?*

```
teams = ...
```

```
fun team-name(abbr :: String) -> String:  
  doc: "Return the name of the team with the given  
  abbreviation"  
  matches =  
    filter-with(teams, lam(r): r["abbr"] == abbr end)  
  if matches.length() == 0:  
    abbr  
  else:  
    team = matches.row-n(0)  
    team["name"]  
  end  
where:  
  ...  
end
```

```
transform-column(stats, "team", team-name)
```

"A'ja Wilson"	"Las Vegas Aces"	"F"	36	703
"Han Xu"	"New York Liberty"	"C"	32	273
"Jackie Young"	"Las Vegas Aces"	"G"	34	542
"Li Yueru"	"Chicago Sky"	"C"	16	28
"Emma Cannon"	"TOT"	"F"	24	164
"Tina Charles"	"TOT"	"C"	34	502
"Crystal Dangerfield"	"TOT"	"G"	33	180
"Kaela Davis"	"TOT"			34
"Rennia Davis"	"TOT"	"F"	8	10
"AD Durr"	"TOT"	"G"	25	174
"Reshanda Gray"	"TOT"	"F"	27	59

*Total? Players who played for more than one team?*

What's a column anyway?



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?

*stats* =

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*How do I get just  
the points column?*

*stats* =

player	team	pos	games	pts
"Natalie Achonwa"	"MIN"	"F"	22	116
"Julie Allemand"	"CHI"	"G"	25	74
"Lindsay Allen"	"MIN"	"G"	9	60
"Rebecca Allen"	"NYL"	"G"	25	174
"Yvonne Anderson"	"CON"	"G"	11	35
"Kristine Anigwe"	"PHO"	"F-C"	10	15
"Ariel Atkins"	"WAS"	"G"	36	527
"Amy Atwell"	"LAS"	"F"	4	3
"Shakira Austin"	"WAS"	"C-F"	36	312
"Rachel Banham"	"MIN"	"G"	36	283

[Click to show the remaining 188 rows...](#)

*How do I get just the points column?*

`stats.get-column("pts")`

```
>>> stats.get-column("pts")  
[list: 116, 74, 60, 174, 35, ...]
```

*The data type isn't Column; it's List!*

A *List* is an ordered sequence of data.

For example,

```
grades = [list: 0.96, 0.73, 1.0, 0.5]
```

```
fellowship = [list:  
    "Frodo", "Sam", "Merry", "Pippin", "Gandalf",  
    "Legolas", "Gimli", "Aragorn", "Boromir"  
]
```

So, what good is a List?



*Plural-Noun*

*Plural-Noun*

*Plural-Noun*

*Number*

*Plural-Noun*

*Noun*

*Noun*

*Noun*

*Noun*

*Body-Part*

*Alphabet-Letter*

*Plural-Noun*

*Plural-Noun*

*Plural-Noun*

*Body-Part*

*Body-*

*Part*

*Adjective*

*Noun*



Thousands of years ago, there were calendars that enabled the ancient Egyptians to divide a year into twelve months, each month into thirty days, and each week into seven days. At first, people told time by a sun clock, sometimes known as the obelisk dial. 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.

How can we represent a text?

*template* = "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", ...]

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  
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
  for each(i from range(0, 100)):
    block:
      n = num-random(10)
      print(n)
      n satisfies between(0, 10 - 1)
    end
  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
```

```
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(3)  
    plural-nouns.get(rand)  
  else:  
    w  
  end  
where:  
  ...  
end
```

```
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
```

*template* = "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
```

*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.

*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?*

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:
  mad-libs(template) is-not template

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:
  mad-libs(template) is-not template
  string-contains(mad-libs(template), "Plural-Noun")
  is false
end
```

# Preview: Lists and recursion

What if **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.

```
>>> 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:

```
>>> 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:

```
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 if the rest of the list is 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" + ""
end

```

Class code:

[tinyurl.com/101-2023-02-08](https://tinyurl.com/101-2023-02-08)