CMPU 101 § 53 · Computer Science I

Tables and Lists

8 February 2024

Where are we?

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

Tables!

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

Rows!

stats =

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

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

stats =

n	player	team	pos	games	pts
0	"Lindsay Allen"	"MIN"	"G"	29	179
1	"Rebecca Allen"	"CON"	"G"	40	255
2	"Laeticia Amihere"	"ATL"	"F"	20	56
3	"Ariel Atkins"	"WAS"	"G"	27	311
4	"Shakira Austin"	"WAS"	"C-F"	19	190
5	"Rachel Banham"	"MIN"	"G"	32	176
6	"Kierstan Bell"	"LVA"	"G"	36	132
7	"Grace Berger"	"IND"	"G"	36	151
8	"Morgan Bertsch"	"CHI"	"F"	28	122
9	"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

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

stats =

n	player	team	pos	games	pts
0	"Lindsay Allen"	"MIN"	"G"	29	179
1	"Rebecca Allen"	"CON"	"G"	40	255
2	"Laeticia Amihere"	"ATL"	"F"	20	56
3	"Ariel Atkins"	"WAS"	"G"	27	311
4	"Shakira Austin"	"WAS"	"C-F"	19	190
5	"Rachel Banham"	"MIN"	"G"	32	176
6	"Kierstan Bell"	"LVA"	"G"	36	132
7	"Grace Berger"	"IND"	"G"	36	151
8	"Morgan Bertsch"	"CHI"	"F"	28	122
9	"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

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

```
stats.row-n(3)
```

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

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

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190

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

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

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190

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

```
fun is-guard(player :: Row)
  -> Boolean:
  doc: "Return true if the player's
primary position is guard"
  player["pos"] == "G"
where:
  is-guard(stats.row-n(0)) is true
  is-guard(stats.row-n(2)) is false
end
```

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

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

What about columns?

How can I add a new column like this?

player	team	pos	games	pts	*frequent-player*
"Lindsay Allen"	"MIN"	"G"	29	179	"somewhat"
"Rebecca Allen"	"CON"	"G"	40	255	"very"
"Laeticia Amihere"	"ATL"	"F"	20	56	"somewhat"
"Ariel Atkins"	"WAS"	"G"	27	311	"somewhat"
"Shakira Austin"	"WAS"	"C-F"	19	190	"no"
"Rachel Banham"	"MIN"	"G"	32	176	"very"
"Kierstan Bell"	"LVA"	"G"	36	132	"very"
"Grace Berger"	"IND"	"G"	36	151	"very"
"Morgan Bertsch"	"CHI"	"F"	28	122	"somewhat"
"Monique Billings"	"ATL"	"F"	39	187	"very"

Click to show the remaining 165 rows...

How can I add a new column like this?

player	team	pos	games	pts	*frequent-player*
"Lindsay Allen"	"MIN"	"G"	29	179	"somewhat"
"Rebecca Allen"	"CON"	"G"	40	255	"very"
"Laeticia Amihere"	"ATL"	"F"	20	56	"somewhat"
"Ariel Atkins"	"WAS"	"G"	27	311	"somewhat"
"Shakira Austin"	"WAS"	"C-F"	19	190	"no"
"Rachel Banham"	"MIN"	"G"	32	176	"very"
"Kierstan Bell"	"LVA"	"G"	36	132	"very"

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

player	team	pos	games	pts	frequent-player
"Lindsay Allen"	"MIN"	"G"	29	179	"somewhat"
"Rebecca Allen"	"CON"	"G"	40	255	"very"
"Laeticia Amihere"	"ATL"	"F"	20	56	"somewhat"
			27	311	"somewhat"
			19	190	"no"
			32		
			36		

```
fun how-frequent(player :: Row)
  -> String:
 if player["games"] >= 30:
    "very"
  else if player["games"] >= 20:
    "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)
```

Changing a column

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
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 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
```

```
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"

  # 1. Get the row with abbreviation `abbr`
  # 2. 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, has-abbr)
  team = matches.row-n(0)
  # 2. Return the value in the `name` column

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

# Wishlist:
# - has-abbr :: Row -> Boolean
```

```
teams = ...

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

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

# Wishlist:
# - has-abbr :: Row -> Boolean
```

```
teams = ...

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

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

fun has-abbr(r :: Row) -> Boolean:
  doc: "Return true if the row has the given abbreviation"
  r["abbr"] == abbr
where:
  has-abbr(teams.row-n(0)) is ...
end
```

*Wait – what's the **abbr** we're checking?*

```
teams = ...

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

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

fun has-abbr(r :: Row) -> Boolean:
  doc: "Return true if the row has the given abbreviation"
  r["abbr"] == abbr
where:
  has-abbr(teams.row-n(0)) is ...
end
```

*The name **abbr** is only defined for this function's body.*

```
teams = ...

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

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

fun has-abbr(r :: Row, abbr :: String) -> Boolean:
  doc: "Return true if the row has the given abbreviation"
  r["abbr"] == abbr
where:
  has-abbr(teams.row-n(0)) is ...
end
```

*Why can't I do this – make **abbr** another input to the **has-abbr** function?*

```
teams = ...

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

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

fun has-abbr(r :: Row, abbr :: String) -> Boolean:
  doc: "Return true if the row has the given abbreviation"
  r["abbr"] == abbr
where:
  has-abbr(teams.row-n(0)) is ...
end
```

When **`filter-with`** *calls* **`has-abbr`**, *the only input it provides is the row.*

```
teams = ...

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

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

fun has-abbr(r :: Row) -> Boolean:
  doc: "Return true if the row has the given abbreviation"
  r["abbr"] == abbr
where:
  has-abbr(teams.row-n(0)) is ...
end
```

So, what can we do so that **abbr** *is defined when we need to use it in* **has-abbr**?

```
teams = ...

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

  fun has-abbr(r :: Row) -> Boolean:
    doc: "Return true if the row has the given abbreviation"
    r["abbr"] == abbr
  end

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end
```

We can define a function inside another function. Now **`has-abbr`** *can use any of the inputs we gave to* **`team-name`**.

```
teams = ...

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

  fun has-abbr(r :: Row) -> Boolean:
    doc: "Return true if the row has the given abbreviation"
    r["abbr"] == abbr
  end

  matches = filter-with(teams, has-abbr)
  team = matches.row-n(0)
  team["name"]

where:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end
```

But, actually, **has-abbr** *is a very small function, and we only use it in one place.*

```
teams = ...

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

  fun has-abbr(r :: Row) -> Boolean:
    doc: "Return true if the row has the given abbreviation"
    r["abbr"] == abbr
  end

  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
```

This means we can just write the function body inline as a lambda expression, without giving a name to it.

We can name a number value and then use it:

```
width = 10
2 * width
```

Or we can just use it without giving a name:

```
2 * 10
```

Likewise, we can name a function value and then use it:

```
fun add-3(x): x + 3 end
add-3(10)
```

or we can just use it:

```
(lam(x): x + 3 end)(10)
```

```
teams = ...

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

  fun has-abbr(r :: Row) -> Boolean:
    doc: "Return true if the row has the given abbreviation"
    r["abbr"] == abbr
  end

  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
```

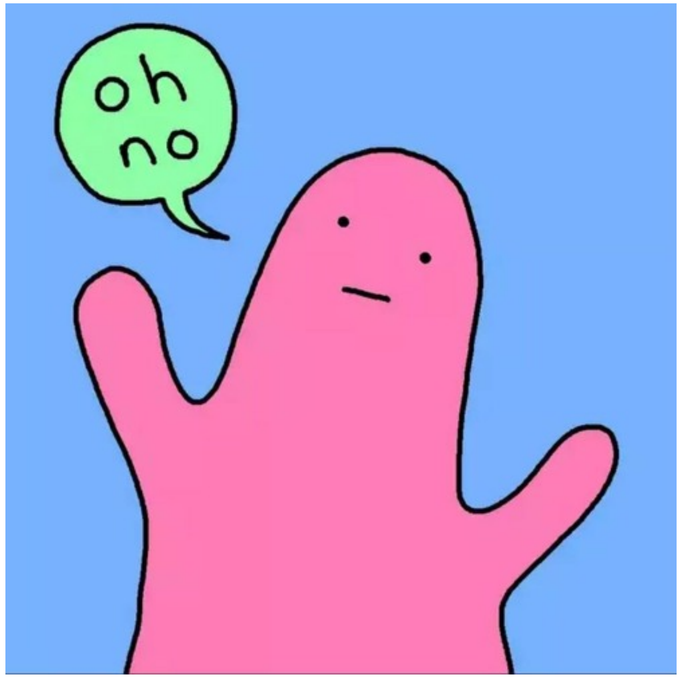
The lambda expression removes the need for the full function definition

```
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:
  team-name("NYL") is "New York Liberty"
  team-name("CHI") is "Chicago Sky"
  ...
end

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

"Jackie Young"	"Las Vegas Aces"	"G"	40	704
"Kristine Anigwe"	"TOT"	"F-C"	13	25
"Alaina Coates"	"TOT"	"C"	12	11
"Liz Dixon"	"TOT"	"F"	14	27
"Queen Egbo"	"TOT"	"F-C"	37	178
"Destanni Henderson"	"TOT"	"G"	15	57
"Joyner Holmes"	"TOT"			113
"Ashley Joens"	"TOT"	"G"	19	39
"Evina Westbrook"	"TOT"	"G"	18	20
"Amanda Zahui B."	"TOT"	"C"	34	57
"Kristine Anigwe"	"Chicago Sky"	"F-C"	10	25

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
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122
"Monique Billings"	"ATL"	"F"	39	187

Click to show the remaining 165 rows...

How do I get just the points column?

stats =

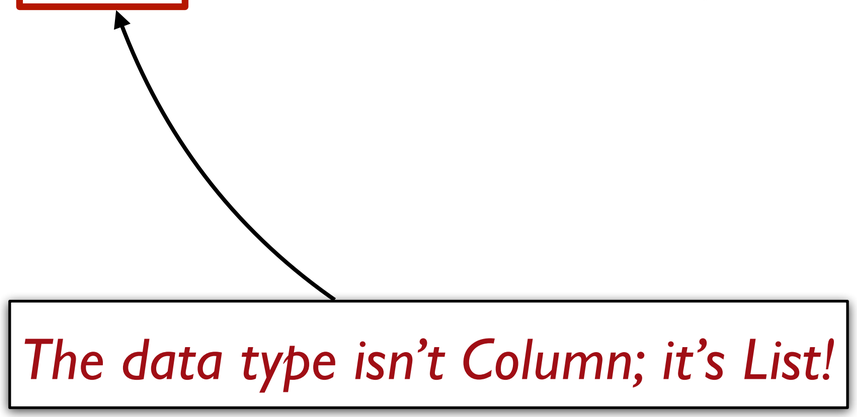
player	team	pos	games	pts
"Lindsay Allen"	"MIN"	"G"	29	179
"Rebecca Allen"	"CON"	"G"	40	255
"Laeticia Amihere"	"ATL"	"F"	20	56
"Ariel Atkins"	"WAS"	"G"	27	311
"Shakira Austin"	"WAS"	"C-F"	19	190
"Rachel Banham"	"MIN"	"G"	32	176
"Kierstan Bell"	"LVA"	"G"	36	132
"Grace Berger"	"IND"	"G"	36	151
"Morgan Bertsch"	"CHI"	"F"	28	122

How do I get just the points column?

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

```
››› stats.get-column("pts")
[list: 179, 255, 56, 311, 190, ...]
```

```
››› stats.get-column("pts")
[list: 179, 255, 56, 311, 190, ...]
```



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?

```
fun is-nearby(town :: String) -> String:
  doc: ...
  (town == "Hyde Park") or
  (town == "Pleasant Valley") or
  (town == "Poughkeepsie") or
  (town == "LaGrange")
where:
  ...
end
```

```
nearby-towns = [list:
  "Hyde Park",
  "Pleasant Valley",
  "Poughkeepsie",
  "LaGrange"
]

fun is-nearby(town :: String) -> String:
  doc: ...
  member(nearby-towns, town)
where:
  ...
end
```

Mad Libs!

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

How can we represent a text?

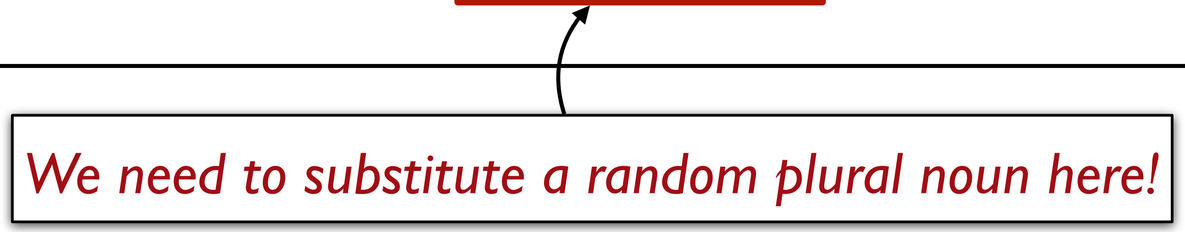
```
template = "Thousands of Plural-Noun ago, there were calendars that
enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun ,
each month into Number weeks, and each week into seven Plural-Noun . At
first, people told time by a sun clock, sometimes known as the Noun dial.
Ultimately, 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 ."
```

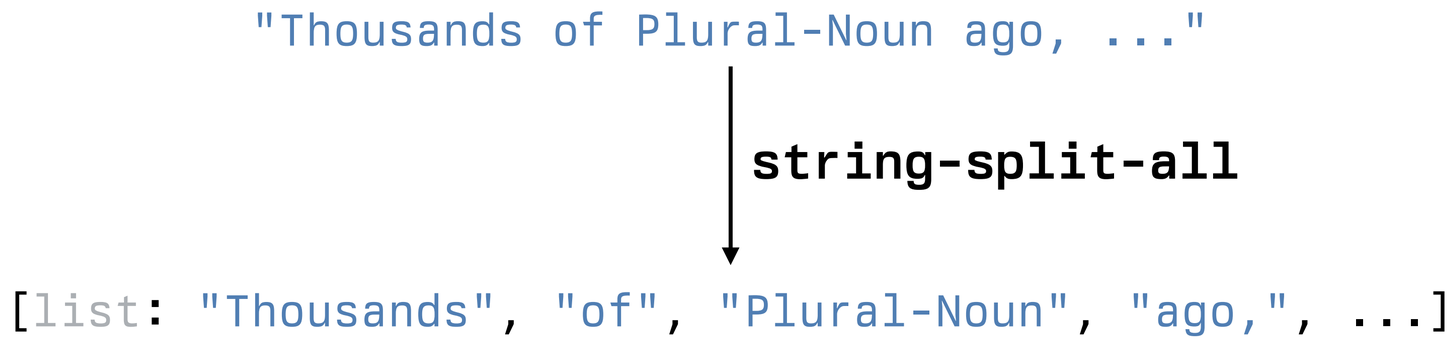
```
template = "Thousands of Plural-Noun ago, ..."

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

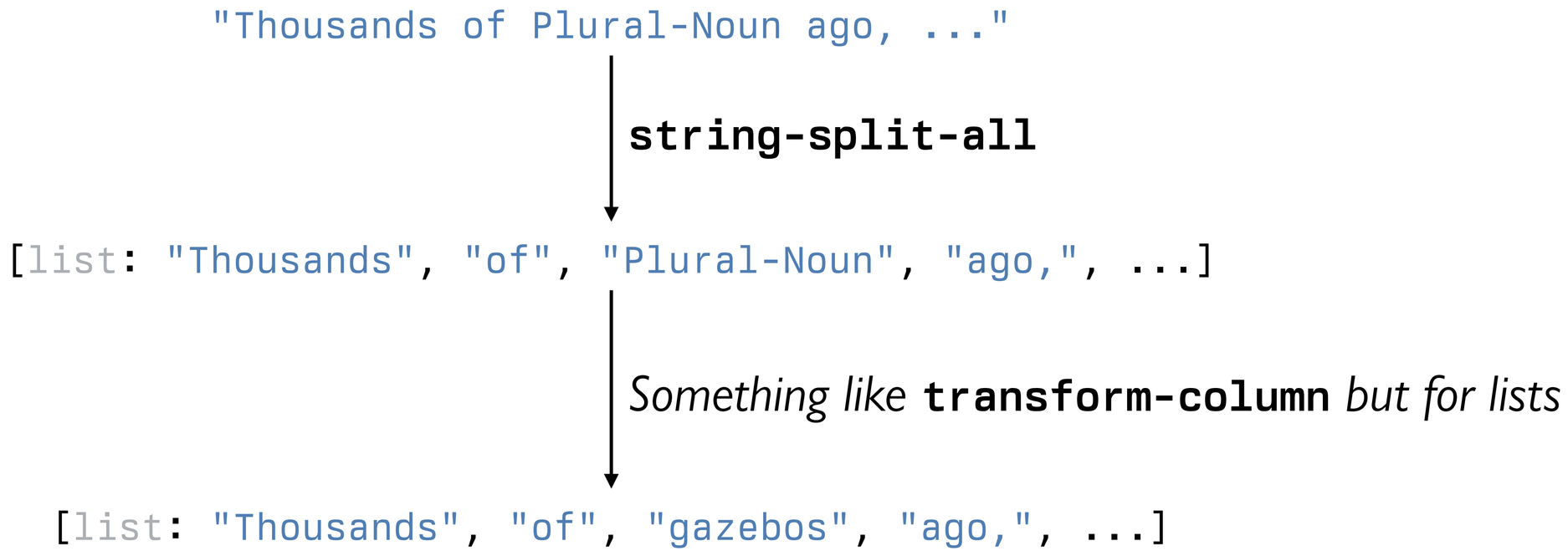
```
››› template-words
[list: "Thousands", "of", "Plural-Noun", "ago,", ...]
```

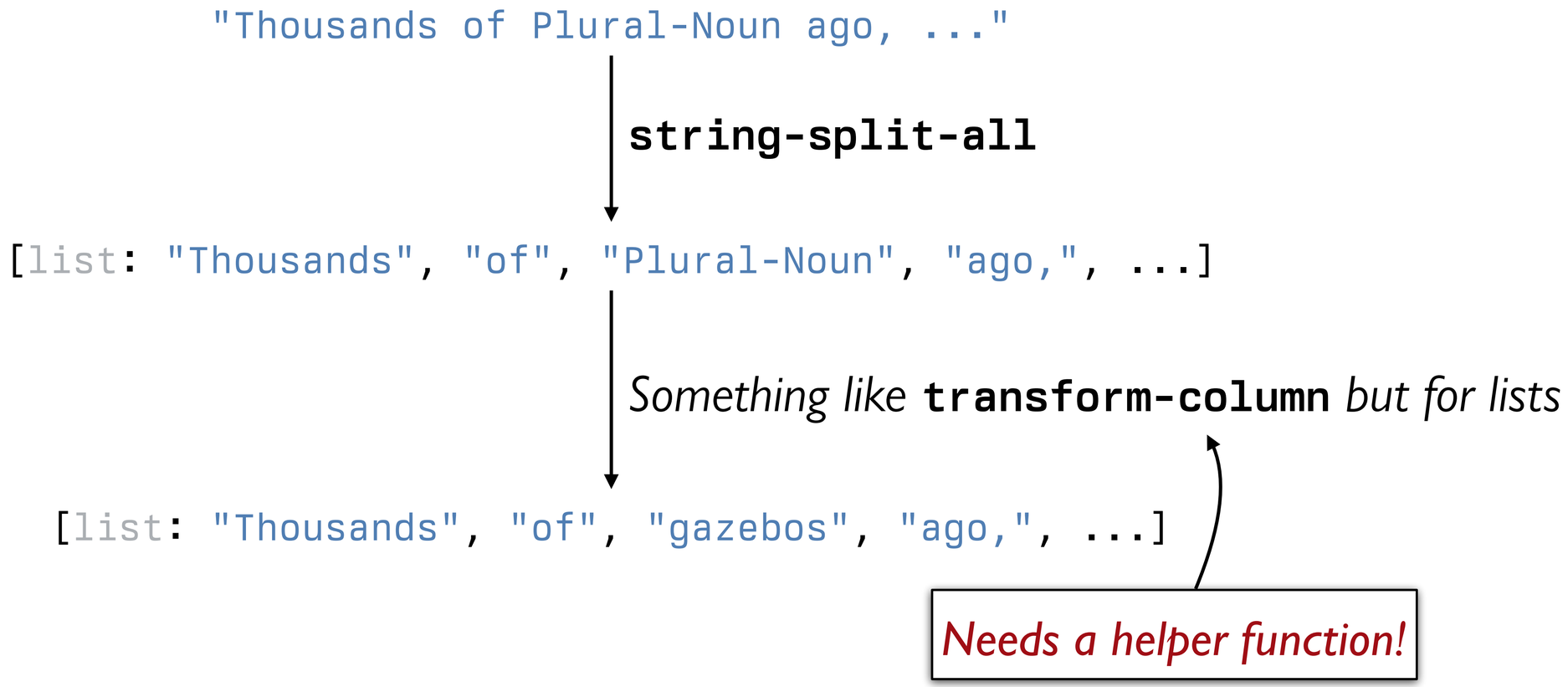
```
template = "Thousands of Plural-Noun ago, ..."

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

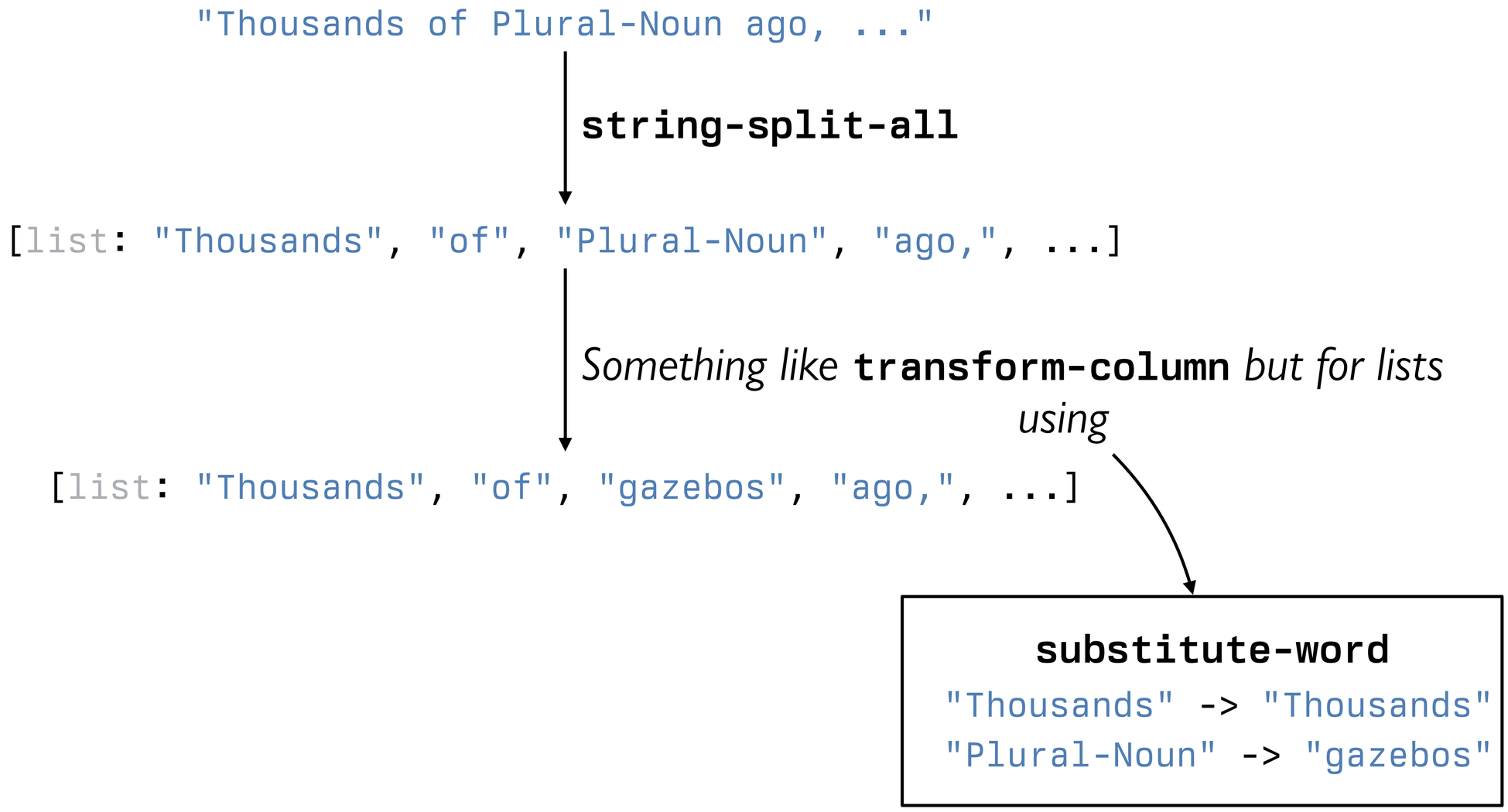
```
››› template-words
[list: "Thousands", "of", "Plural-Noun", "ago,", ...]
```




"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"
  member(plural-nouns, 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"
  member(plural-nouns, 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 the 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
```

```
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:
  member(plural-nouns, rand-word(plural-nouns)) is true
  member(numbers, 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, words)
  ...
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, words)
  ...
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(words-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, words)
  join-str(words-sub, " ")
where:
  ...


end
```

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, words)
  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, words)
  join-str(words-sub, " ")
where:
  mad-libs(template) is-not template
  string-contains(mad-libs(template), "Plural-Noun")
    is false
end
```

Class code:

tinyurl.com/101-2024-02-08