Lists

CMPU 101 – Problem Solving and Abstraction

Peter Lemieszewski

Introducing: lists

.row-n gives us a row in a table…

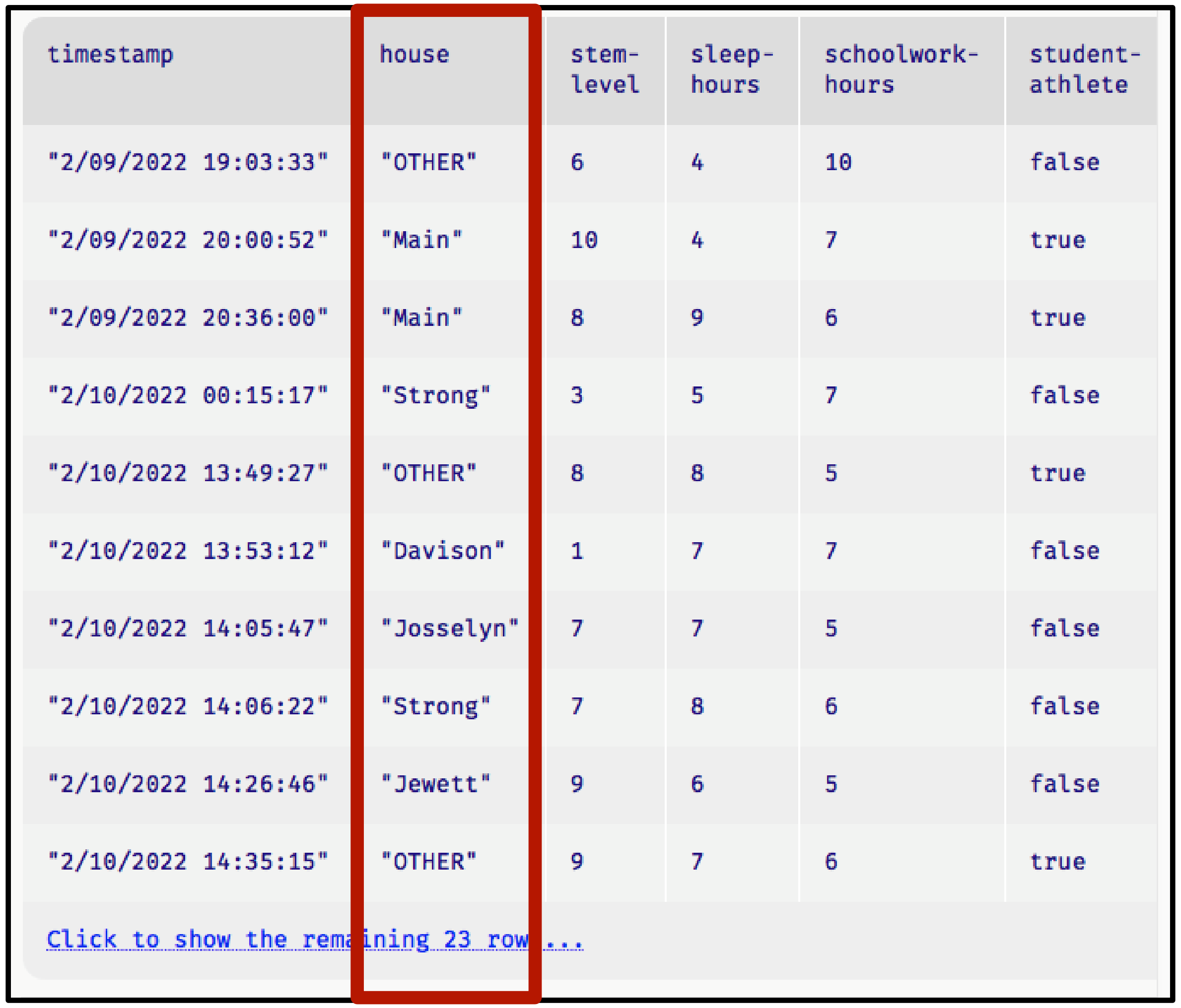
How can we access all the elements in one column?

A: get-column

Example:

student-data-cleaned.get-column("house")

[list: "OTHER", "Main", "Main", "Strong", ...]

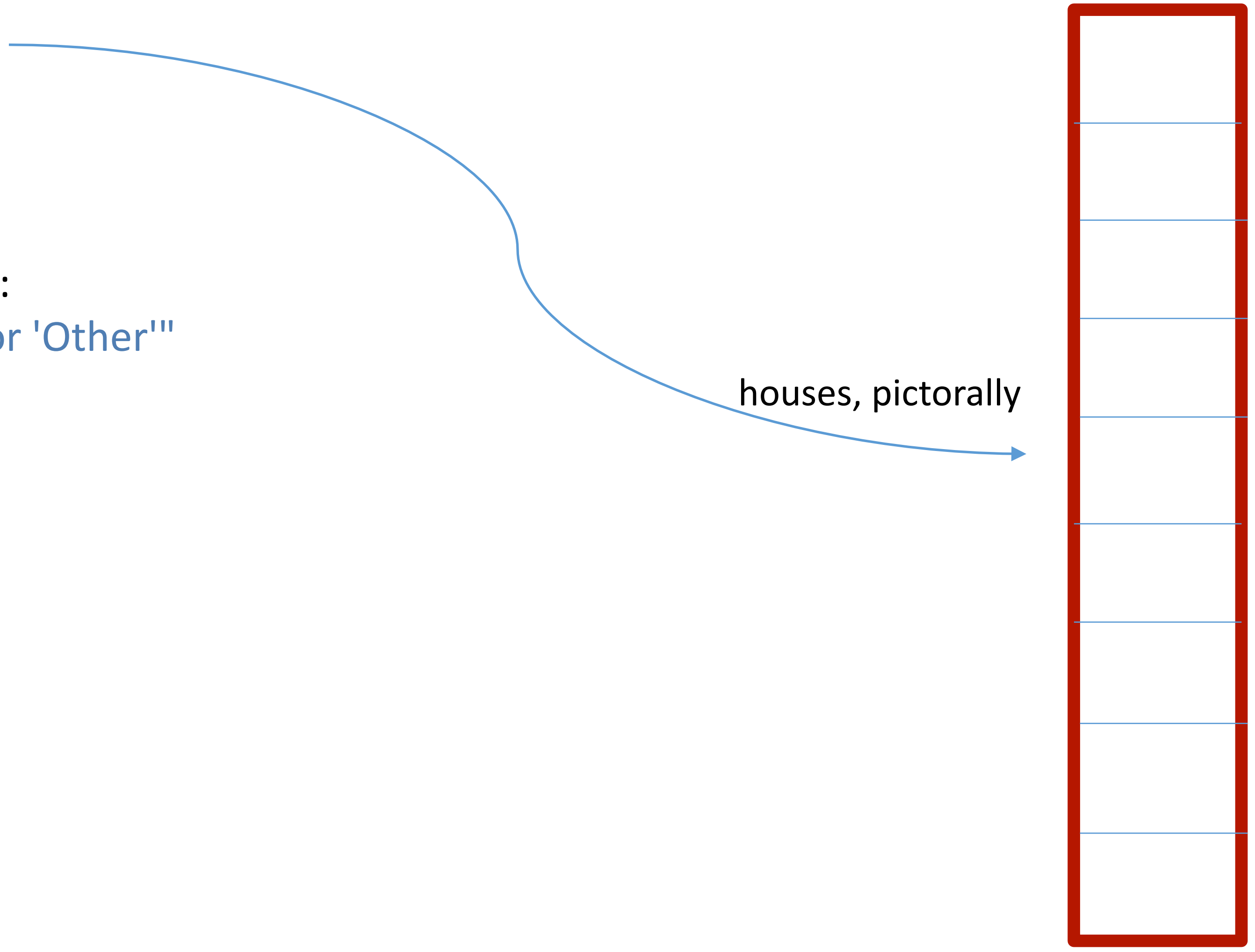


timestamp	house	stem-level	sleep-hours	schoolwork-hours	student-athlete
"2/09/2022 19:03:33"	"OTHER"	6	4	10	false
"2/09/2022 20:00:52"	"Main"	10	4	7	true
"2/09/2022 20:36:00"	"Main"	8	9	6	true
"2/10/2022 00:15:17"	"Strong"	3	5	7	false
"2/10/2022 13:49:27"	"OTHER"	8	8	5	true
"2/10/2022 13:53:12"	"Davison"	1	7	7	false
"2/10/2022 14:05:47"	"Josselyn"	7	7	5	false
"2/10/2022 14:06:22"	"Strong"	7	8	6	false
"2/10/2022 14:26:46"	"Jewett"	9	6	5	false
"2/10/2022 14:35:15"	"OTHER"	9	7	6	true

Click to show the remaining 23 rows...

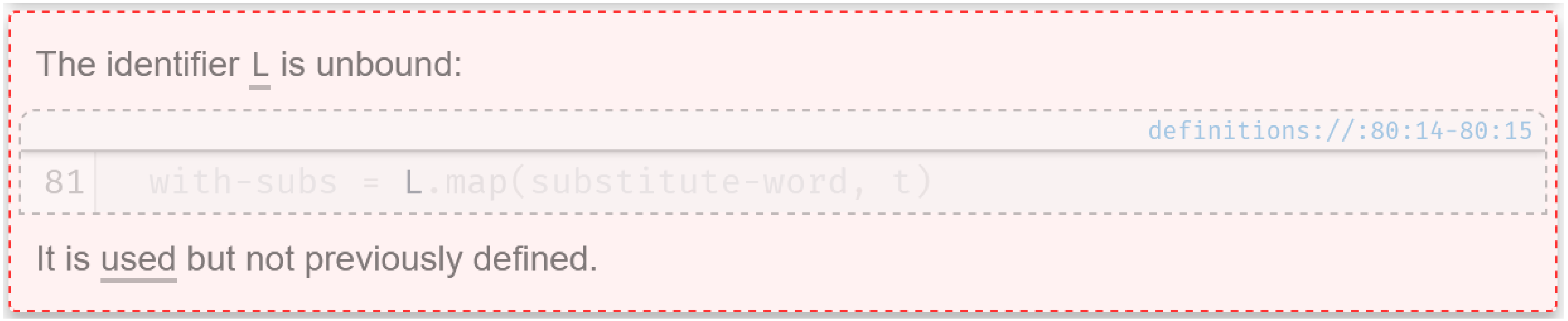
Introducing: lists for student data

```
houses = [list: "Main", "Strong", "Raymond",
  "Davison", "Lathrop", "Jewett", "Josselyn",
  "Cushing", "Noyes"]


fun normalize-house(house :: String) -> String:
  doc: "Return one of the nine Vassar houses or 'Other'"
  if L.member(houses, house):
    house
  else:
    "Other"
  end
where:
  normalize-house("Main") is "Main"
  normalize-house("Offcampus") is "Other"
end
```



Using Lists

- To work with lists, we *import* the library and we give it a special name – L –
- Avoids conflicts between the names of functions that work with lists and (other) existing functions:
    - import lists as L
    - If you forget the import statement you'll see:



The identifier L is unbound:

```
definitions://:80:14-80:15
81    with-subs = L.map(substitute-word, t)
```

It is used but not previously defined.

Let’s play a game!

- Mad Libs
  - Given a part of speech (noun, verb, etc.) create a random word that fits
  - Then, a sentence requiring that part of speech is shown, with that word!
  - In doing so we create a hilarious sentence!
- An example: Plural-Noun
  - Answer: Rocks

Let’s play a game!

- Mad Libs
  - Given a part of speech (noun, verb, etc.) create a random word that fits
  - Then, a sentence requiring that part of speech is shown, with that word!
  - In doing so we create a hilarious sentence!
- An example: Plural-Noun
  - Answer: Rocks
- The sentence:
  - We saw many Plural-Noun on vacation this summer!
- Becomes:
  - We saw many Rocks on vacation this summer!

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 .

Q: How can we represent text?

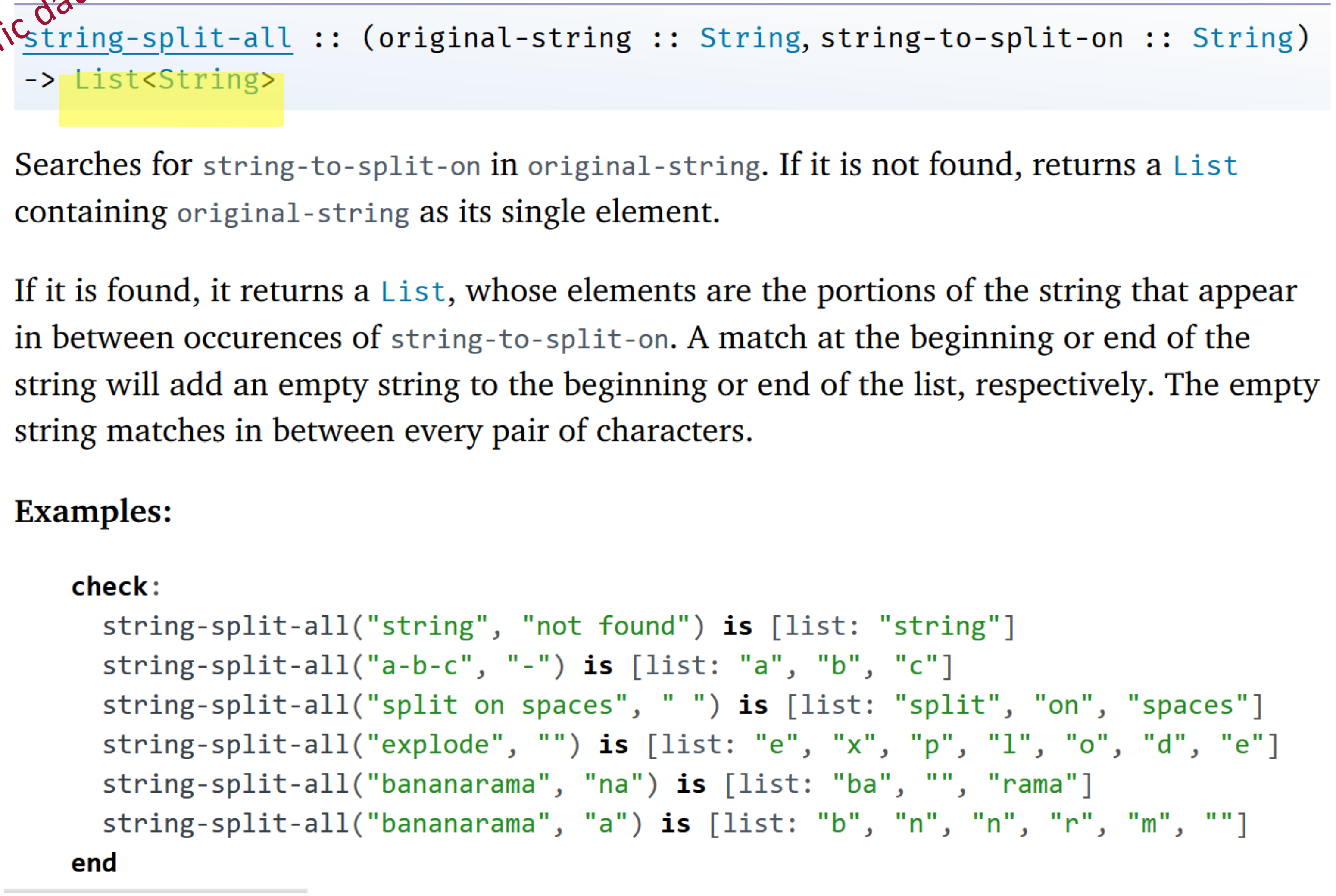
template = "Thousands of Plural-Noun ago, there were calendars that enabled the ancient Plural-Noun to divide a year into twelve Plural-Noun , each month into Number weeks, and each week into seven Plural-Noun . At first, people told time by a sun clock, sometimes known as the Noun dial. Ultimately,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

A: As a *list* of words!

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

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

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

From the documentation

<String> refers to a specific data type



```
string-split-all :: (original-string :: String, string-to-split-on :: String)
-> List<String>
```

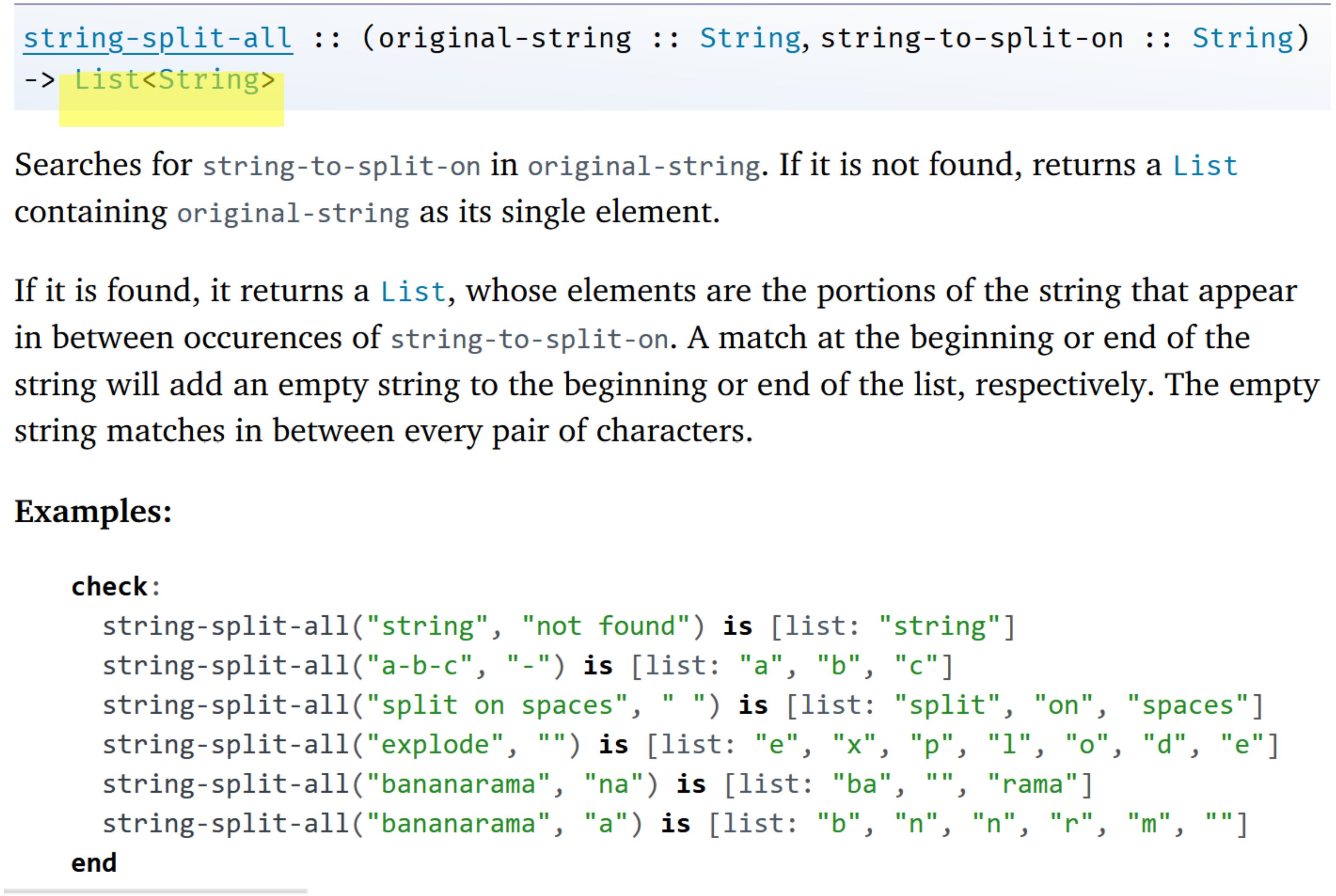
Searches for `string-to-split-on` in `original-string`. If it is not found, returns a `List` containing `original-string` as its single element.

If it is found, it returns a `List`, whose elements are the portions of the string that appear in between occurences of `string-to-split-on`. A match at the beginning or end of the string will add an empty string to the beginning or end of the list, respectively. The empty string matches in between every pair of characters.

Examples:

```
check:
  string-split-all("string", "not found") is [list: "string"]
  string-split-all("a-b-c", "-") is [list: "a", "b", "c"]
  string-split-all("split on spaces", " ") is [list: "split", "on", "spaces"]
  string-split-all("explode", "") is [list: "e", "x", "p", "l", "o", "d", "e"]
  string-split-all("bananarama", "na") is [list: "ba", "", "rama"]
  string-split-all("bananarama", "a") is [list: "b", "n", "n", "r", "m", ""]
end
```

From the documentation



```
string-split-all :: (original-string :: String, string-to-split-on :: String)
-> List<String>
```

Searches for `string-to-split-on` in `original-string`. If it is not found, returns a `List` containing `original-string` as its single element.

If it is found, it returns a `List`, whose elements are the portions of the string that appear in between occurences of `string-to-split-on`. A match at the beginning or end of the string will add an empty string to the beginning or end of the list, respectively. The empty string matches in between every pair of characters.

Examples:

```
check:
  string-split-all("string", "not found") is [list: "string"]
  string-split-all("a-b-c", "-") is [list: "a", "b", "c"]
  string-split-all("split on spaces", " ") is [list: "split", "on", "spaces"]
  string-split-all("explode", "") is [list: "e", "x", "p", "l", "o", "d", "e"]
  string-split-all("bananarama", "na") is [list: "ba", "", "rama"]
  string-split-all("bananarama", "a") is [list: "b", "n", "n", "r", "m", ""]
end
```

We now return you to our list of words

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


template-words = string-split-all(template, " ")
#shout out to “Plural-Noun”
››› template-words
[list: "Thousands", "of", "Plural-Noun", "ago", ...]
```

We now return you to our list of words

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


template-words = string-split-all(template, " ")
#shout out to "Plural-Noun"
››› template-words
[list: "Thousands", "of", "Plural-Noun", "ago", ...]
```

Let’s diagram what we want to do

$$\begin{tikzcd}
\text{"Thousands of Plural-Noun ago, ..."} \arrow[d, "\textit{string-split-all}"] \\
\text{[list: "Thousands", "of", "Plural-Noun", "ago", ...]}
\end{tikzcd}$$

From “Plural-Noun” to “gazebos”

"Thousands of Plural-Noun ago, ..."

string-split-all

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

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

Something like transform-column *but for lists*

From “Plural-Noun” to “gazebos”

"Thousands of Plural-Noun ago, ..."

string-split-all

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

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

Needs a helper function!

S[illegible]g like **transform-column** *but for lists*

substitute-word does what we want

"Thousands of Plural-Noun ago, ..."

string-split-all

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

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

substitute-word

"Thousands" -> "Thousands"

"Plural-Noun" -> "gazebos"

Something like **transform-column** *but for lists*

- How can we represent a text?

Let’s write the helper function substitute-word

```
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
```

Just one question – what word should we use?

```
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
```

Well, we know what word it isn’t (is-not)!

```
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
```

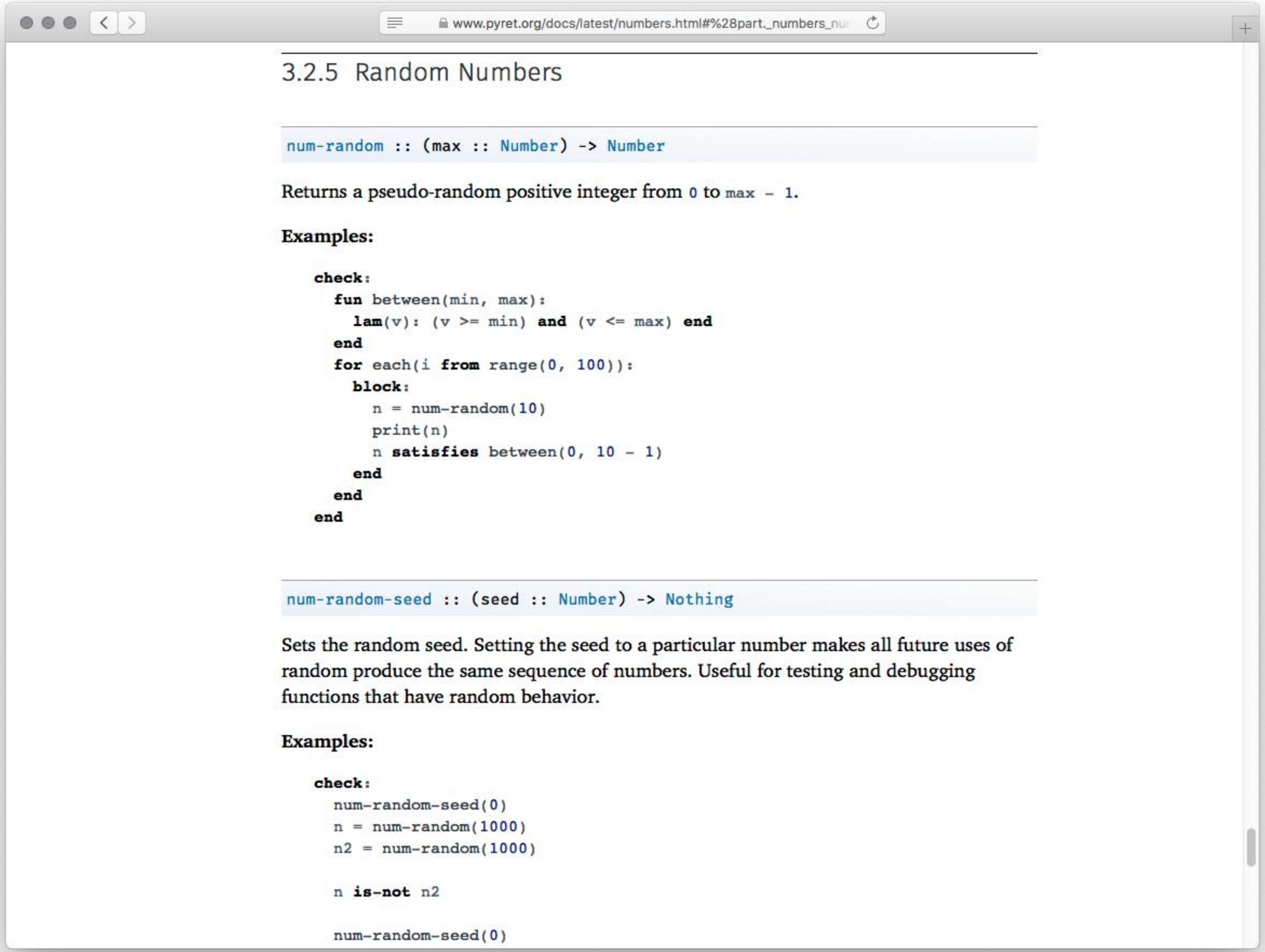
Getting closer…

```
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“
 L.member(
    plural-nouns,
    substitute-word("Plural-Noun"))
    is true
end
```

Getting closer… but we want some randomness!

```
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
```

Ripped from the documentation



www.pyret.org/docs/latest/numbers.html#%28part._numbers_nu

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
```

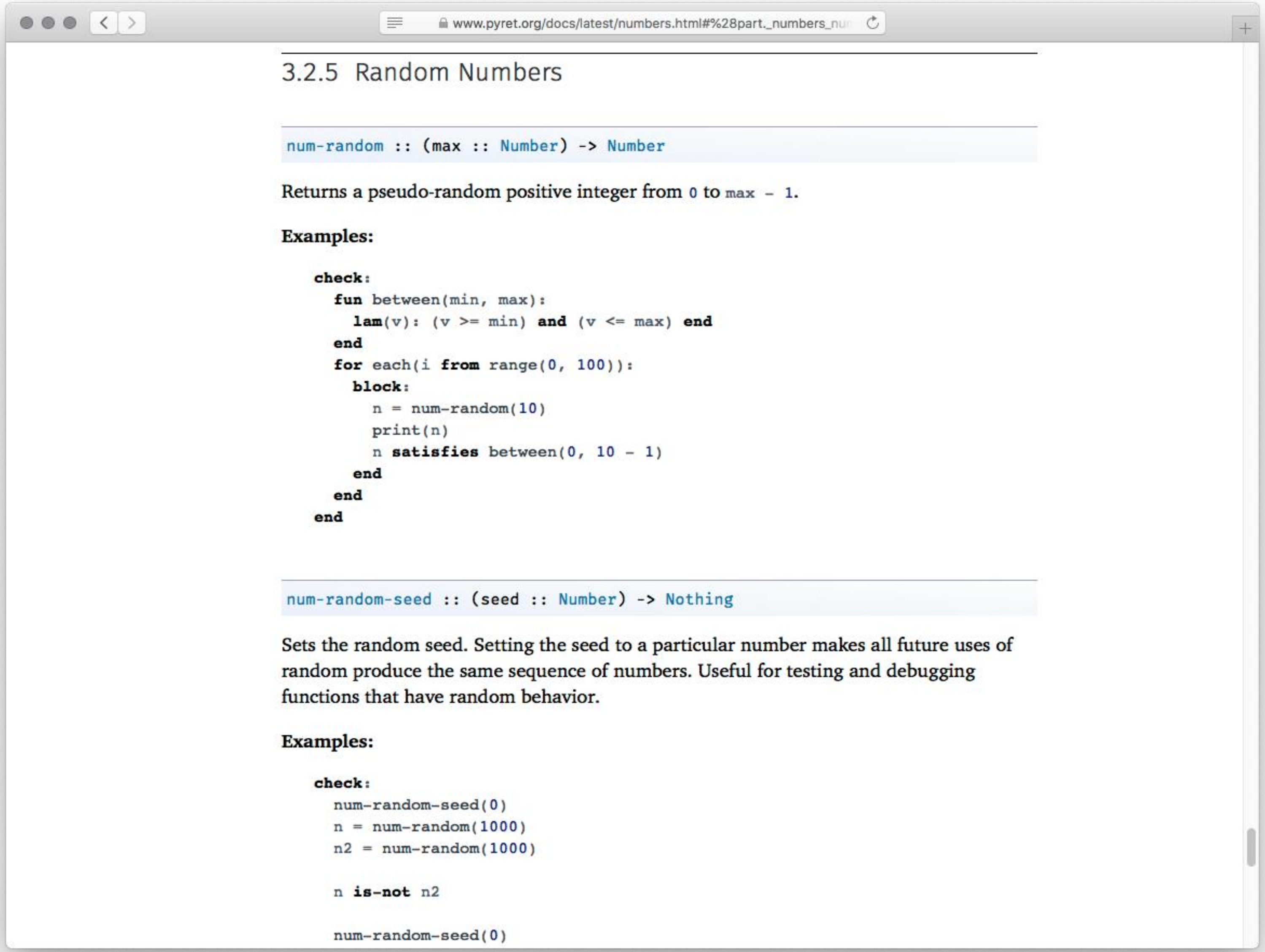
```
num-random-seed :: (seed :: Number) -> Nothing
```

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)
  n2 = num-random(1000)

  n is-not n2

  num-random-seed(0)
```

Ok… how do we get from random number to…



www.pyret.org/docs/latest/numbers.html#%28part._numbers_nu

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)
  n2 = num-random(1000)

  n is-not n2

  num-random-seed(0)
```

…a random list item?

- With a table, we use **.row-n** to get a specific row by its index number.
- With a list, we can use **L.get** to get an item.

…a random list item?

- With a table, we use **.row-n** to get a specific row by its index number.

- With a list, we can use **L.get** to get an item.
- So…
  - Get a random number. Then,
  - <u>Get list element(item) positioned at that number</u>

Adding randomness to our code

```
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": #we want a Plural Noun!
    ...
  else:
    w
  end
where:
  ...
end
```

Adding randomness to our code

```
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) #we have 3 items in our plural-nouns list
    L.get(plural-nouns, rand)
  else:
    w
  end
where:
  else:
    w
  end
where:
  ...
end
```

Q:Do we have to know how many plural-nouns we have?

```
plural-nouns = [list: "gazebos", "avocados", "umiaks", "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) #we have 3 items in our plural-nouns list… oops, no we don’t
    L.get(plural-nouns, rand)
  else:
    w
  end
where:
  else:
    w
  end
where:
  ...
end
```

A: No, we don’t!

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

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

The other parts of speech (data) for our madlib

```
plural-nouns =
  [list: "gazebos", "avocados", “umiaks", “pandas"]

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"]
```

Getting the rest of the random words

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

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

  end
where:
  else:
    w
  end
where:
  …
end
```

Getting the rest of the random words

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

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

  end
where:
  else:
    w
  end
where:
  …
end
```

Can we generalize this code even further? Speficially those calls to num-random?

Yes we can!

Generalizing the call to num-random

```
#address need for general utility that gives us a random word.
fun rand-word(l :: List<String>) -> String:
  doc: "Return a random word in the given list"
  rand = num-random(L.length(l))
  L.get(l, rand)
where:
  L.member(plural-nouns, rand-word(plural-nouns))
    is true
end
```

Completing the helper function…

```
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
```

Back to our task plan

- We’ve completed our helper,
- Now we need to run it on every word in the list, the same way
- **transform-column**
- runs a function on every row of a table.

Back to our task plan

- We’ve completed our helper, **substitute-word**!
- Now we need to run it on every word in the list, the same way
- **transform-column**
- runs a function on every row of a table.

- This is the way: **L.map**

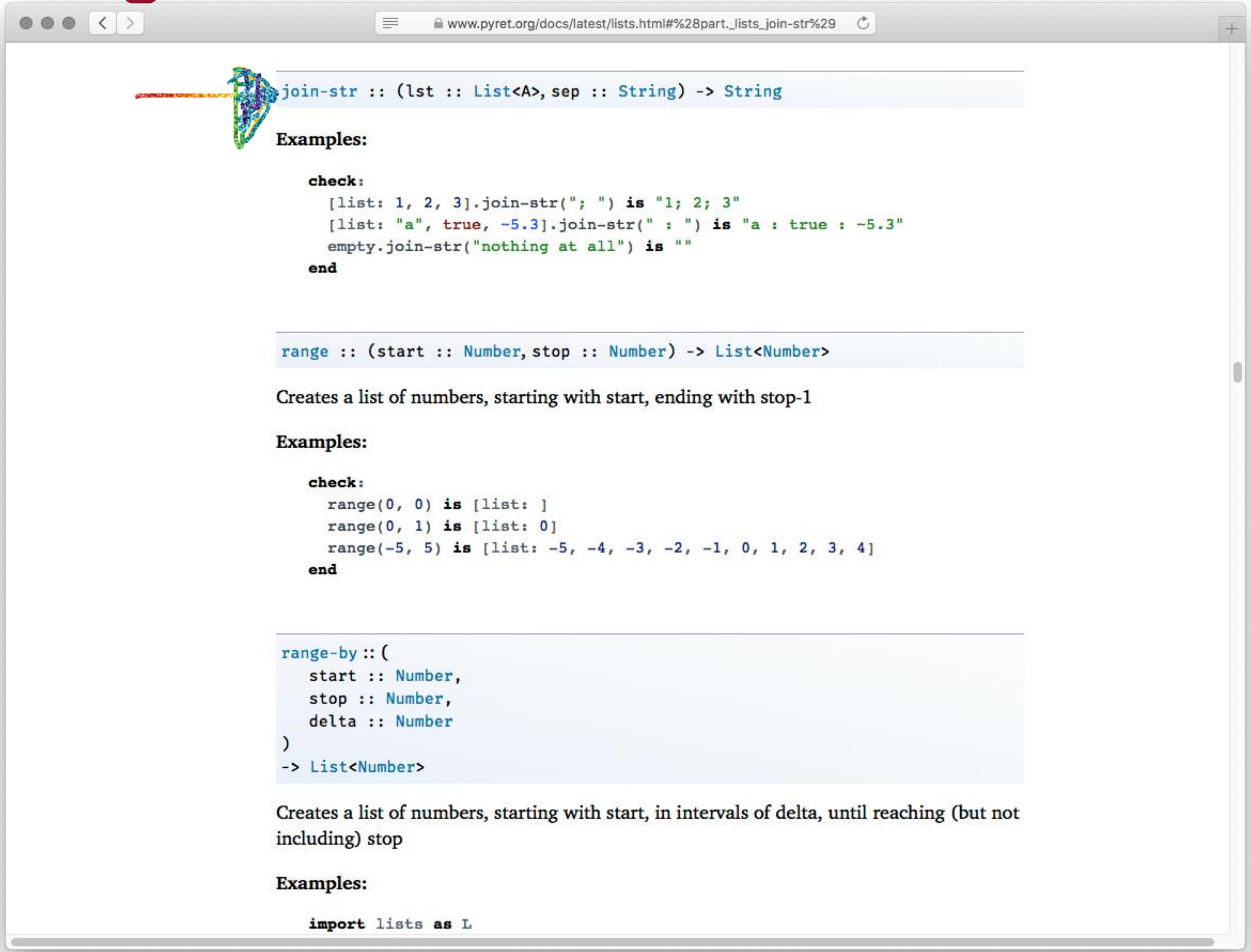
Mad-libs so far…

```
fun mad-libs(t :: List<String>) -> String:
  doc: "Randomly fill in the blanks in the mad libs template"
  L.map(substitute-word, t) #like transform-column
end
```

Mad-libs so far… actually…

```
fun mad-libs(t :: List<String>) -> String:
  doc: “Actually… This only returns a list of strings "
  L.map(substitute-word, t) #like transform-column
end
```

… to the string documentation!



www.pyret.org/docs/latest/lists.html#%28part._lists_join-str%29

```
join-str :: (lst :: List<A>, sep :: String) -> String
```

Examples:

```
check:
  [list: 1, 2, 3].join-str("; ") is "1; 2; 3"
  [list: "a", true, ~5.3].join-str(" : ") is "a : true : ~5.3"
  empty.join-str("nothing at all") is ""
end
```

```
range :: (start :: Number, stop :: Number) -> List<Number>
```

Creates a list of numbers, starting with start, ending with stop-1

Examples:

```
check:
  range(0, 0) is [list: ]
  range(0, 1) is [list: 0]
  range(-5, 5) is [list: -5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
end
```

```
range-by :: (
   start :: Number,
   stop :: Number,
   delta :: Number
)
-> List<Number>
```

Creates a list of numbers, starting with start, in intervals of delta, until reaching (but not including) stop

Examples:

```
import lists as L
```

Mad-libs: final version

```
fun mad-libs(t :: List<String>) -> String:
  doc: "Randomly fill in the blanks in the mad libs template"
  # L.map(substitute-word, t) used on next line.
with-subs = L.map(substitute-word, t)
  L.join-str(with-subs, " ")
end
```

Acknowledgements

- This lecture incorporates material from:
- Kathi Fisler, Brown University,
- Jason Waterman, Vassar College
- And, Jonathan Gordon, Vassar College