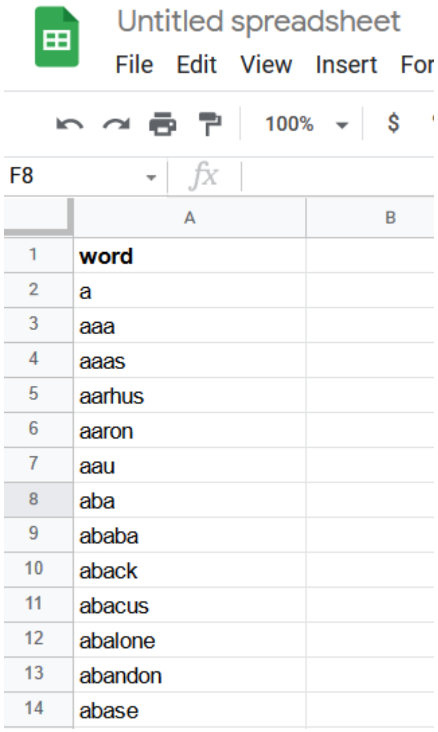
Problem Solving and Abstraction (CMPU 101)

Tom Ellman

Lecture 8

English Dictionary
~ 25K Words
No Definitions



	A	B
1	**word**	
2	a	
3	aaa	
4	aaas	
5	aarhus	
6	aaron	
7	aau	
8	aba	
9	ababa	
10	aback	
11	abacus	
12	abalone	
13	abandon	
14	abase	

English contains **over 200,000 words**, with 171,476 active words and 47,156 inactive words.

https://thelanguagedoctors.org

What questions can we answer with this table?

- Let’s work with word length.
- Find the mean word-length.
- Make a bar-chart of word-length frequencies.

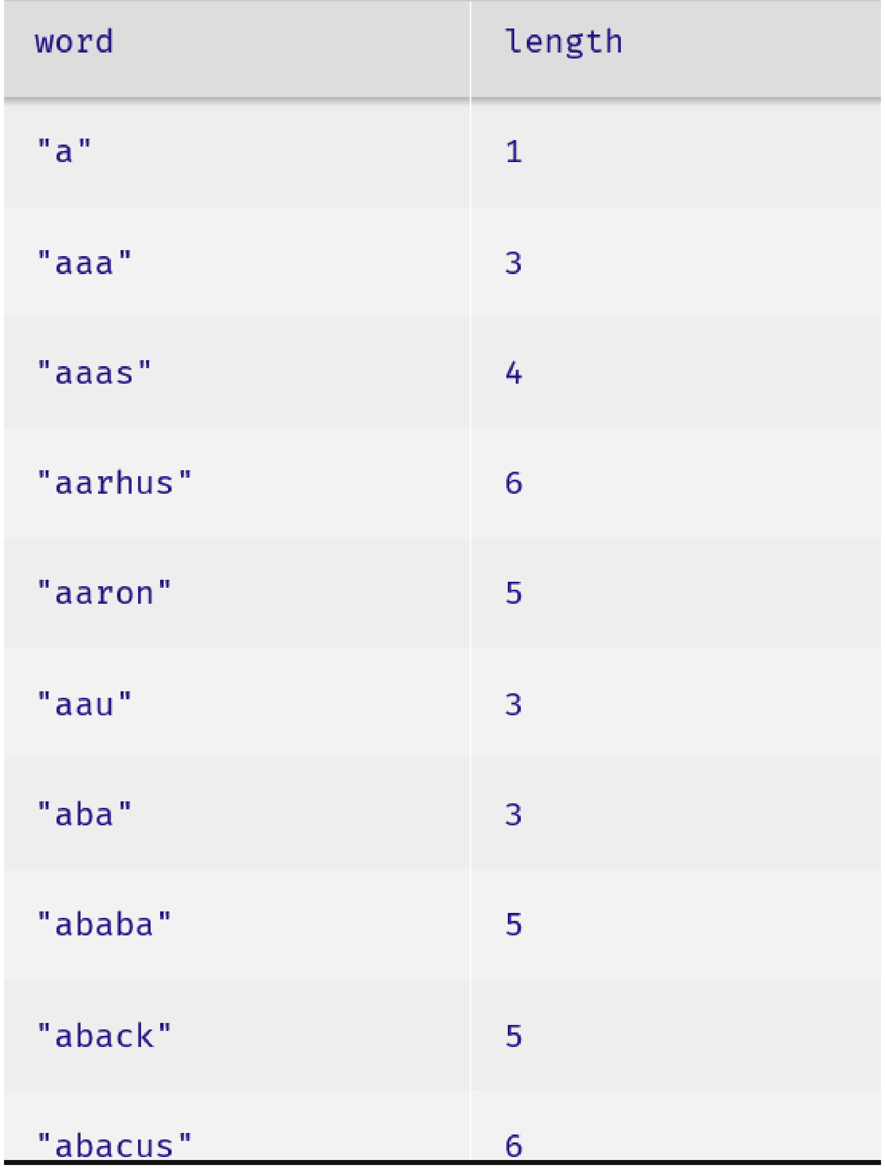
Load the English Table

```
english-ssid =
        "1ahqbMQaQinV3KvbCKKbc1K5koVSXMA7aR0UpZ9dEfuY"
eng-spreadsheet = load-spreadsheet(english-ssid)

english = load-table: word
  source: eng-spreadsheet.sheet-by-name("english", true)
  sanitize word using string-sanitizer
end
```

Add a **length** Column

Add a **length** Column

```
fun word-length(r :: Row) -> Number:
  string-length(r["word"])
end

english-word-length =
            build-column(english,"length",word-length)
```

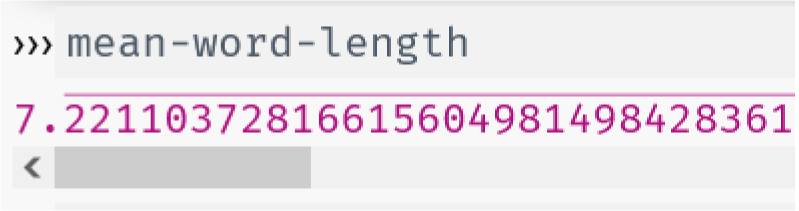


word	length
"a"	1
"aaa"	3
"aaas"	4
"aarhus"	6
"aaron"	5
"aau"	3
"aba"	3
"ababa"	5
"aback"	5
"abacus"	6

Computing Mean Length

Computing Mean Length

mean-word-length = mean(english-word-length,"length")

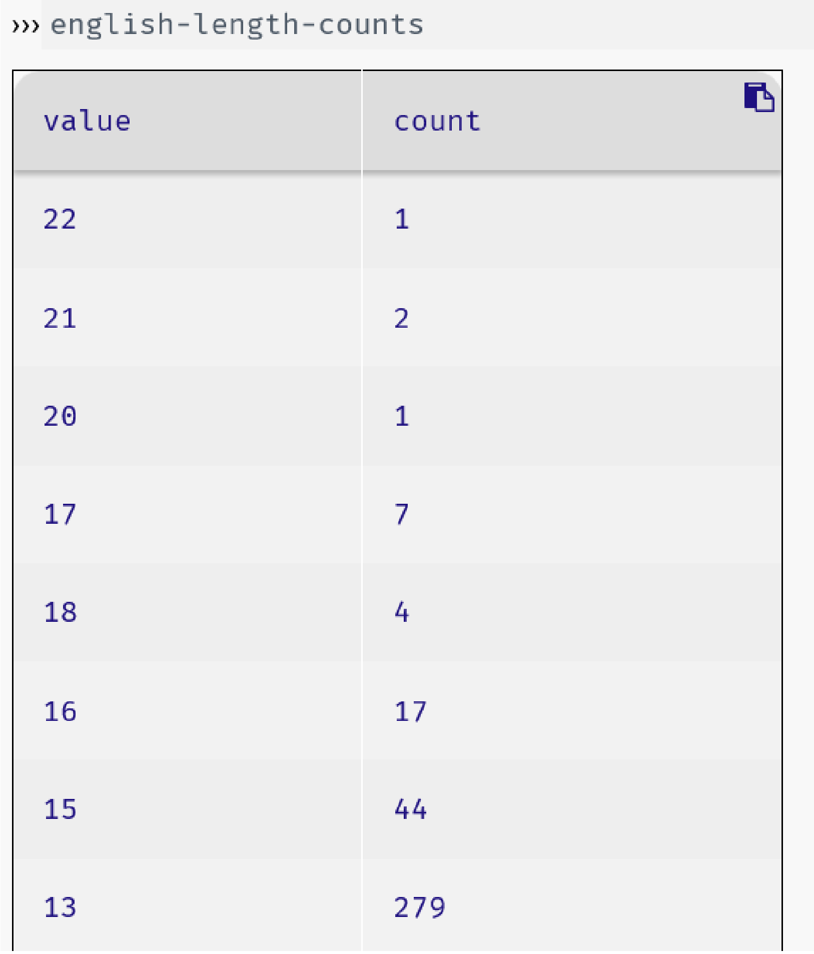


```
››› mean-word-length
7.22110372816615604981498428361
```

Find the frequencies of word-lengths.

Find the frequencies of word-lengths.

```
english-length-counts = count(english-word-length,"length")
```



```
››› english-length-counts
```

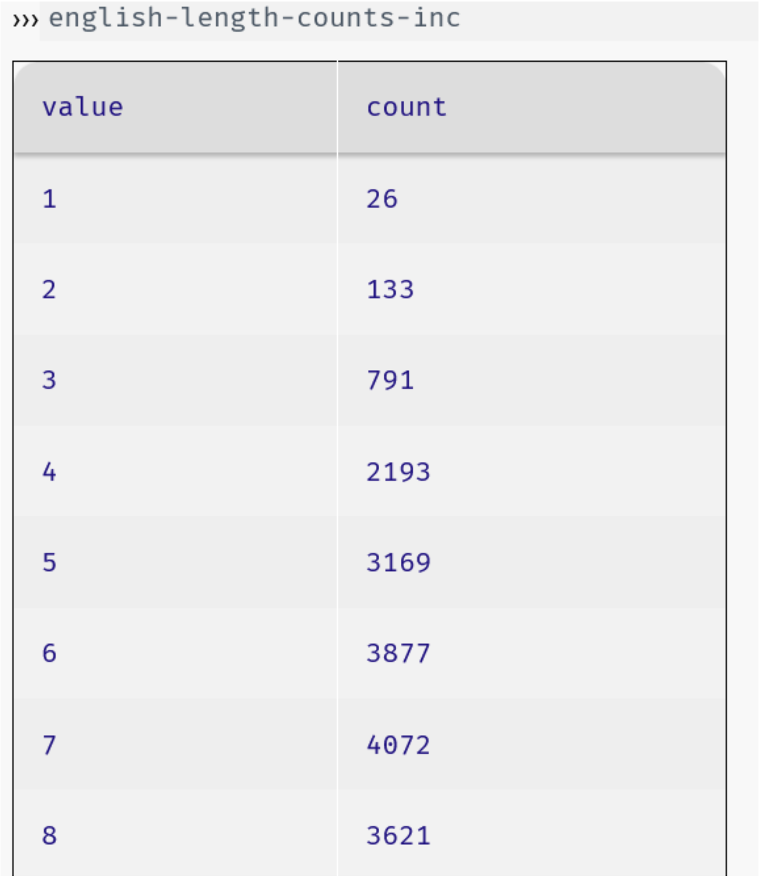
value	count
22	1
21	2
20	1
17	7
18	4
16	17
15	44
13	279

There is 1 word of length 22
There are 7 words of length 17.
Etc.

Order the table by **value**, increasing.

Order the table by **value**, increasing.

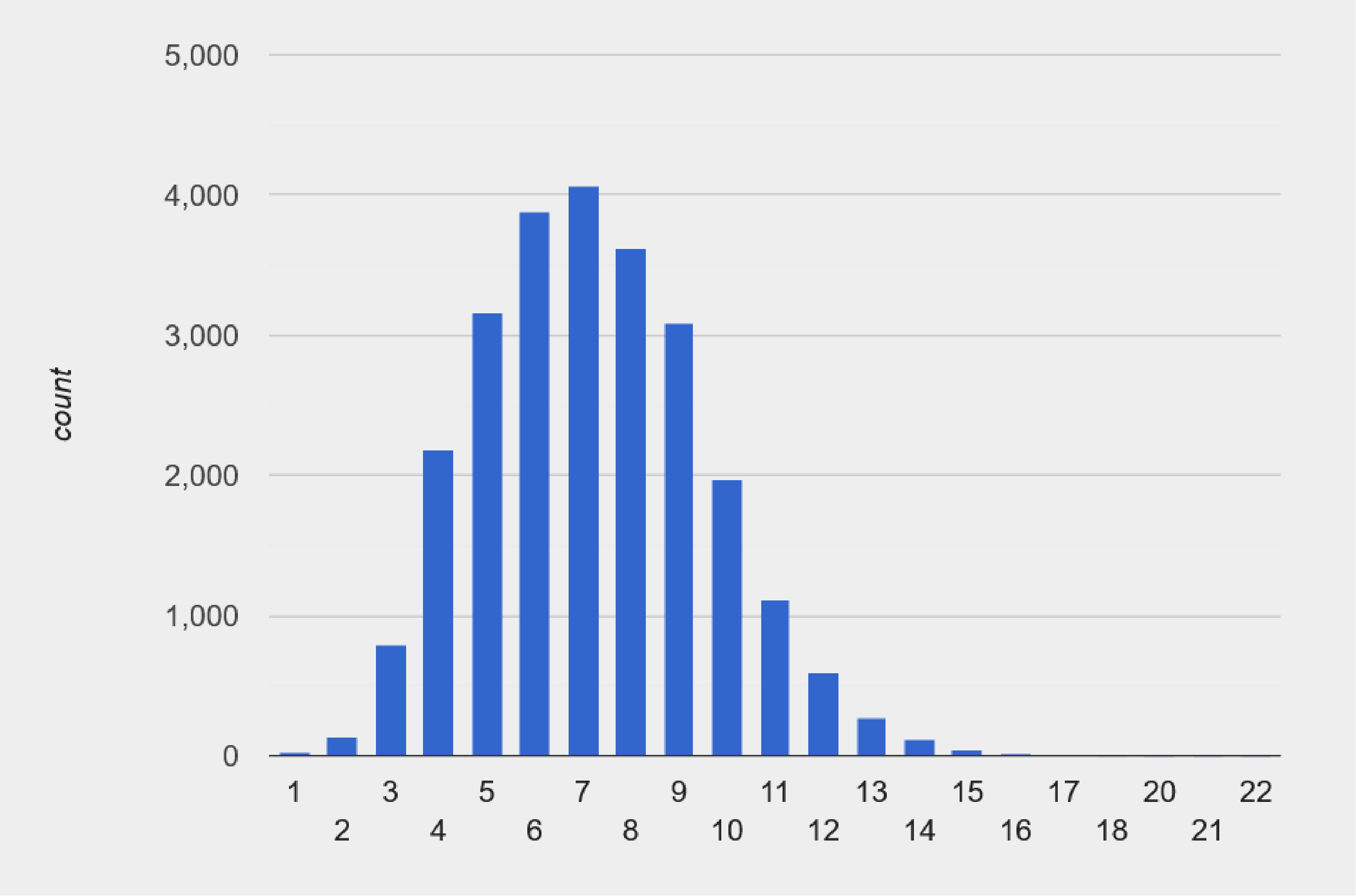
```
english-length-counts-inc
            = order-by(english-length-counts,"value",true)
```



```
››› english-length-counts-inc
```

value	count
1	26
2	133
3	791
4	2193
5	3169
6	3877
7	4072
8	3621

bar-chart(english-length-counts-inc,"value","count")



Dealing with Multiple Tables

- What if the data of interest lies in two or more different tables?
- Consider the example of RI municipalities.
- Find population density by municipality.
  - Population is on one sheet.
  - Land area is on another sheet.

Program Outline

1. Write a function add-area-column to add an area column to a table using its name column and the table ri-areas.
    a. Write a function mun-name-to-area to find area given the name of a municipality using the table ri-areas.
    b. Use the function with build-column to make an area column in a table using the row’s name column and the mun-name-to-area function.
2. Write a function add-density-column that takes a table and adds a new column density using the population-2010 and area columns.
    a. Write a function density that takes a row and uses the population-2010 and area columns to compute density.
    b. Use build-column and density functions to make a density column in the latest table.
3. Write a function muns-by-density that takes tables municipalities and areas and uses add-area-column and add-density column to extend municipalities with area and density columns.

Program Outline

```
fun muns-by-density(municipalities :: Table, areas :: Table) -> Table:

          fun add-area-column(municipalities :: Table, areas :: Table) -> Table:
                    fun mun-name-to-area(areas :: Table, name :: String) -> Number:
                    build-column

          fun add-density-column(municipalities :: Table) -> Table:
                    fun density(r :: Row) -> Number:
                    build-column
```

Open the spreadsheet.

```
include shared-gdrive("dcic-2021",
        "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")
include gdrive-sheets
ssid = "1jHvn5CPE6RkTTQRIXQbY5n5p4aiOH7fZsnwK2s6s6tc"
spreadsheet = load-spreadsheet(ssid)
```

Load the municipality data.

```
ri-municipalities = load-table:
            name :: String, city :: Boolean,
            population-2000 :: Number,
            population-2010 :: Number
  source: spreadsheet.sheet-by-name("municipalities", true)
end
```

Load the land area data.

```
ri-areas = load-table: name :: String, area :: Number
  source: spreadsheet.sheet-by-name("municipalities-area", true)
end
```

Write a function to compute area for a municipality.

Write a function to compute area for a municipality.

```
fun mun-name-to-area(areas :: Table, name :: String) -> Number:
  fun row-matches-namep(r :: Row) -> Boolean:
    r["name"] == name
  end
  filter-with(areas, row-matches-namep).row-n(0)["area"]
end
```

A nested predicate (**row-matches-namep**) checks whether a row contains the data of a named municipality. We use **filter-with** to make a table including only the row(s) of the named municipality. Finally, we get the first row and return the datum in the **area** column

Anonymous Function (Lambda Expression)

Instead of:

```
 fun row-matches-namep (r :: Row) -> Boolean:
   r["name"] == name
 end
```

Named Function

we can write:

```
lam(r :: Row): r["name"] == name end
```

Anonymous Function

Since **row-matches-namep** is used only in **mun-name-to-area**, it does not need a name for future reference. An anonymous function is more concise, but perhaps less clear without a name.

Area for a Municipality Using an Anonymous Function

Area for a Municipality Using an Anonymous Function

```
fun mun-name-to-area(areas :: Table, name :: String) -> Number:
  filter-with(areas, lam(r :: Row): r["name"] == name end).row-n(0)["area"]
end
```

Here we use an anonymous function (predicate) as a parameter to the **filter-with** function.

Add an area column to the municipalities table.

Add an area column to the municipalities table.

```
fun add-area-column(municipalities :: Table, areas :: Table) -> Table:
  build-column(municipalities, "area",
    lam(r :: Row): mun-name-to-area(areas, r["name"]) end)
end
```

Here we use an anonymous function as a parameter to the **build-column** function.

Add a density column to the municipalities table.

Add a density column to the municipalities table.

```
fun density(r :: Row) -> Number:
  r["population-2010"] / r["area"]
end

fun add-density-column(municipalities :: Table) -> Table:
  build-column(municipalities, "density", density)
end
```

Putting it all Together

Putting it all Together

```
fun muns-by-density(municipalities :: Table, areas :: Table) -> Table:
  with-area = add-area-column(municipalities, areas)
  with-density = add-density-column(with-area)
  order-by(with-density, "density", false)
End

ri-muns-by-density = muns-by-density (ri-municipalities, ri-areas)

densest-municipality = mun-by-density.row-n(0)
```

Starting with the municipalities table, we first add the area column and then add the density column. Finally we order the table by density, decreasing.