Lists and Data Frames

INFO 201

Joel Ross

Winter 2017

https://slides.com/joelross/info201w17-data-frames-1/live

Today's Objectives

By the end of class, you should be able to

- Use functions and operations to work with multiple **vectors**
- Store and access data in **lists**
- Begin storing and accessing collections of data in **data frames**
- Load data sets from external `.csv` files in R

RECALL

Vectors

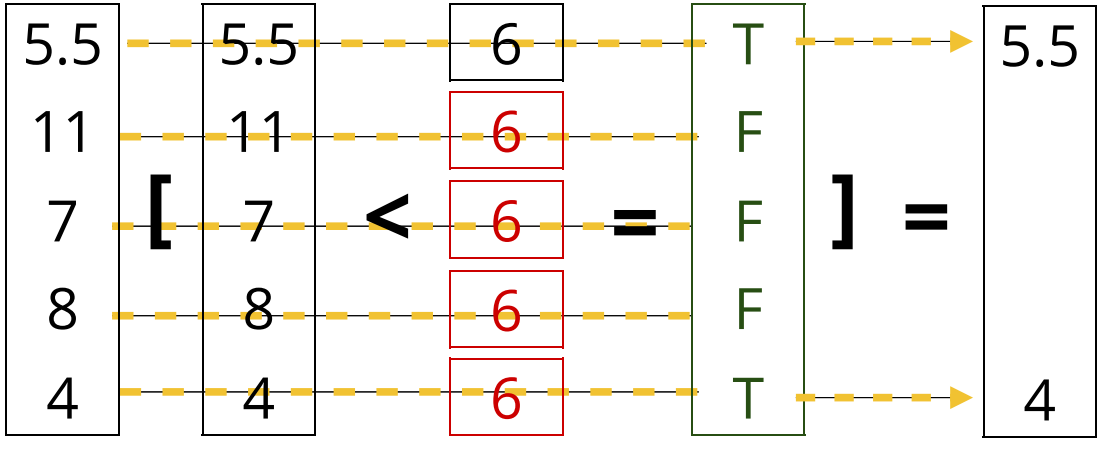
Vectors are *one-dimensional collections* of values that are all stored in a single variable. For example, a "set" of words (strings) or numbers. **Elements** must all be of the same type.

```
# combine 3 dog names into a vector
dogs <- c("Fido", "Spot", "Sparky")

# create a vector of numbers
numbers <- c(1,2,2,3,5,8,13,21,34)  # Fibonacci!



# a vector of the whole numbers from 90 to 100
nineties <- 90:99  # 90  91  92  ...  99
```

Vector Functions

Functions and operations are applied **per-member** when applied to vectors. Elements are **recycled** if the vectors are different lengths.

```
shoe.sizes <- c(5.5, 11, 7, 8, 4)


# Select shoe sizes that are smaller than 6
small.shoes <- shoe.sizes[shoe.sizes < 6]  # returns 5.5, 4
```



Warmup!

Module 7 exercise-5

New exercise; just copy and paste the code into a new file in your repo!

Lists

Lists

Lists, like vectors, are *one-dimensional collections* of values, but with some additional features:

- Lists can store elements of <u>different types</u>
- List elements can be **tagged** with a variable name

```
# List of data about a person
person <- list(name = "Ada", salary = 78000, in.union = TRUE)
```

tag

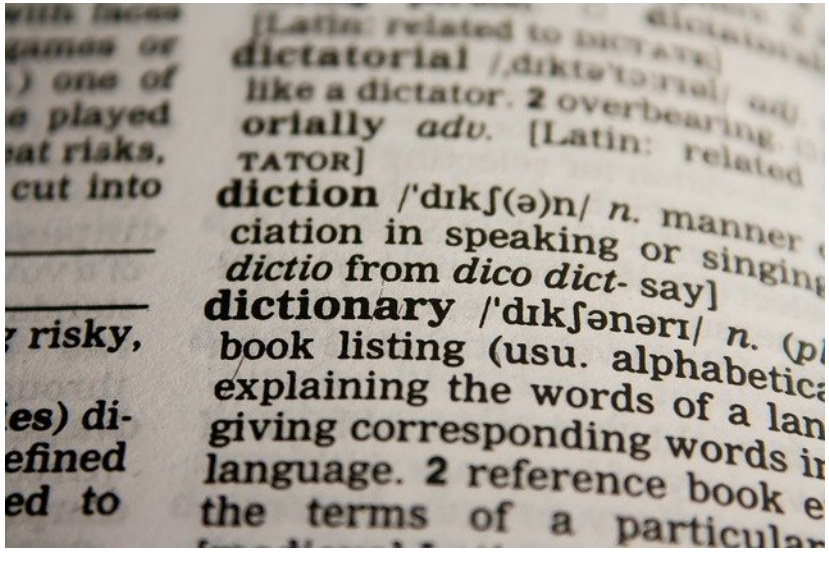
element value

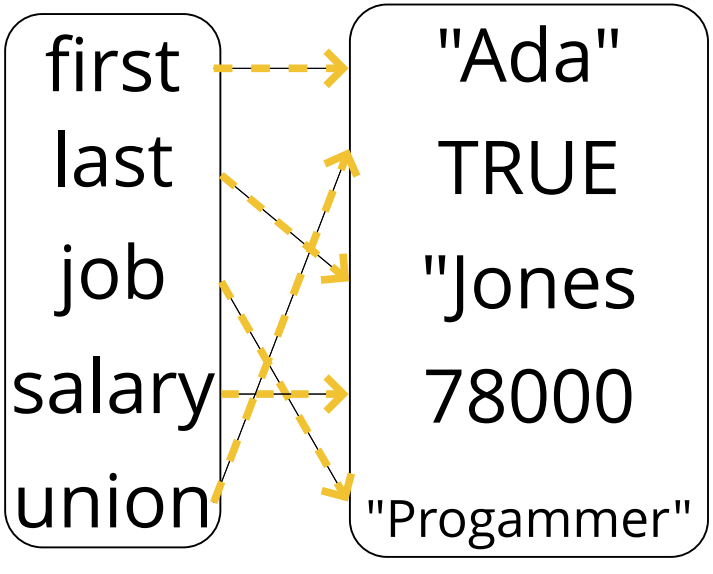
assign element to tag variable

Maps

Tagging elements in lists cause them to act like *maps*, where one value (the tag) "maps" or gives directions to another (the element).

This works a lot like a **dictionary**, allowing you to look up **values** by a **key** (tag). A tag and its element are an example of a **key-value pair**.





Accessing Lists

We access individual values in a list by using **dollar notation**. This looks up an element by its tag (key). The dollar acts like the possessive "*apostrophe s*".

```
person <- list(first.name = "Ada", job = "Programmer",
               salary = 78000, in.union = TRUE)

# access elements by tag
person$first.name  # person's first.name ("Ada")
person$salary  # person's salary (78000)

# use elements as function or operation arguments
paste(person$job, person$first.name)   # [1] "Programmer Ada"

# assign values to list element
person$job <- "Senior Programmer"  # a promotion!
print(person$job)  # [1] "Senior Programmer"

# assign value to list element from itself
person$salary <- person$salary * 1.15  # a 15% raise!
print(person$salary)  # [1] 89700
```

Double-Bracket Notation

We can also access list elements by using **double-bracket notation**, specifying either the numeric index of the element or the tag (the "*first.name-th element*")

```
person <- list(first.name = "Bob", last.name = "Wong",
               salary = 77000, in.union = TRUE)

person[["first.name"]]  # [1] "Bob"
person[["salary"]]  # [1] 77000

# can use variables in the double-brackets
name.to.use <- "last.name"  # choose name (i.e., for formality)
person[[name.to.use]]  # [1] "Wong"
name.to.use <- "first.name"  # change name to use
person[[name.to.use]]  # [1] "Bob"

# We can use indices for tagged elements as well!
person[[1]]  # [1] "Bob"
person[[4]]  # [1] TRUE
```

Vectors use ***single***-bracket notation for accessing by index

Lists use ***double***-bracket notation for accessing by index

Single vs. Double Brackets

Single-brackets are used to ***filter*** elements from a collection (whether a list or a vector), usually returning a collection of the same type.

Using single-brackets with a list will *return* a list, which is not usually what we want.

```
my.list <- list('A', 201, TRUE, 'rhinoceros')

# SINGLE brackets returns a list
my.list[1]
            # [[1]]
            # [1] "A"

# DOUBLE brackets returns a vector
my.list[[1]]  # [1] "A"

# can use any vector as the argument to single brackets
my.list[1:3]
```

Module 8 exercise-1

Data Frames

Data Frames

Data frames are *two-dimensional collections* of values, organized into **rows** and **columns** (like a table)

feature

record or observation

	name	height	weight
1	Ada	58	115
2	Bob	59	117
3	Chris	60	120
4	Diya	61	123
5	Emma	62	126

Text

Data Frames

Data frames are actually ***lists of vectors***, where each vector is the same length (but can be different types).

```
# vector of names
name <- c('Ada','Bob','Chris','Diya','Emma')

# Vector of heights
height <- 58:62

# Vector of weights
weight <- c(115, 117, 120, 123, 126)

# Combine the vectors into a data.frame
# Note the names of the variables become the names of the columns!
my.data <- data.frame(name, height, weight, stringsAsFactors=FALSE)


# Retrieve weights (the `weight` element of the list: a vector!)
my.weights <- my.data$weight

# Retrieve heights (the whole column: a vector!)
my.heights <- my.data[['height']]
```

for string vectors
(we'll talk about this later)

Describing Data Frames

R includes functions for describing data frames:

Function	Description
`nrow(my.data.frame)`	Number of rows in the data frame
`ncol(my.data.frame)`	Number of columns in the data frame
`dim(my.data.frame)`	Dimensions (rows, columns) in the data frame
`colnames(my.data.frame)`	Names of the columns of the data frame
`rownames(my.data.frame)`	Names of the row of the data frame
`head(my.data.frame)`	Extracts the first few rows of the data frame (as a new data frame)
`tail(my.data.frame)`	Extracts the last few rows of the data frame (as a new data frame)
`View(my.data.frame)`	Opens the data frame in as spreadsheet-like viewer (only in RStudio)

Use `?FunctionName` *to look up a function!*

https://www.rdocumentation.org/

Module 9 exercise-1

Fork and clone module!
(start fresh if you did it earlier)

Accessing Data Frame Data

Data frames are ***lists of vectors*** so we can use **dollar** and **double-bracket** notation, but we can also use **single-brackets** with a **comma** to select elements by row and column:

```
# access element at given row, column
my.data.frame[row, col]
```

`row` and `col` may be either *numerical indices* or row/column *names*:

```
# element in the second row, third column
my.data[2,3]

# element in row named 'Ada' and column named 'height'
my.data['Ada','height']

# second element in the 'height' column
my.data[2,'height']
```

Access Multiple Elements

Leave off either the ***row*** or ***column*** (**but keep the comma**) in order to select multiple rows or columns!

```
# extract the second row
my.data[2, ]  # comma

# extract the second column AS A VECTOR
my.data[, 2]  # comma

# extract the 'height' column AS A VECTOR
my.data[, height]  # comma

# extract the second column AS A DATA FRAME (list filtering)
my.data[2]  # no comma

# Get the 'height' and 'weight' columns
my.data[, c('height', 'weight')]

# Perform filtering
my.data[my.data$height > 60, ] # rows for which 'height' > 60
```

Extracting from 1+ columns returns a sub-data frame
Extracting from just 1 column returns a vector

Module 9 exercise-2

Loading Data

R Practice Data

R comes with a number of built-in data sets that can be used for practice, experimentation, and testing.

```
# view a list of included data sets
data()  # will open in new window

# Load e.g., the "Seatbelts" data set into memory
data("Seatbelts")  # quotes optional, but use them!

# The loaded data set is now available as a variable
print(Seatbelts)


# You may need to convert the data set into a data frame
# from another data type
seatbelts <- data.frame(Seatbelts)
```

CSV Files

R can also load data from external files, such as **comma-separated value** files (**.csv**).

```
Ada, 58, 115
Bob, 59, 117
Chris, 60, 120
Diya, 61, 123
Emma, 62, 126
```

record or observation

feature

Use **read.csv()** to read in a file at a given location (path)

```
# Read data from the file `data/my_file.csv`
# into a data frame `my.data`
my.data <- read.csv('data/my_file.csv', stringsAsFactors=FALSE)
```

relative path!

Paths

`/absolute/path/to/file`

leading slash

How to get there *starting from the root*

`relative/path/to/file`

How to get there *starting from here*

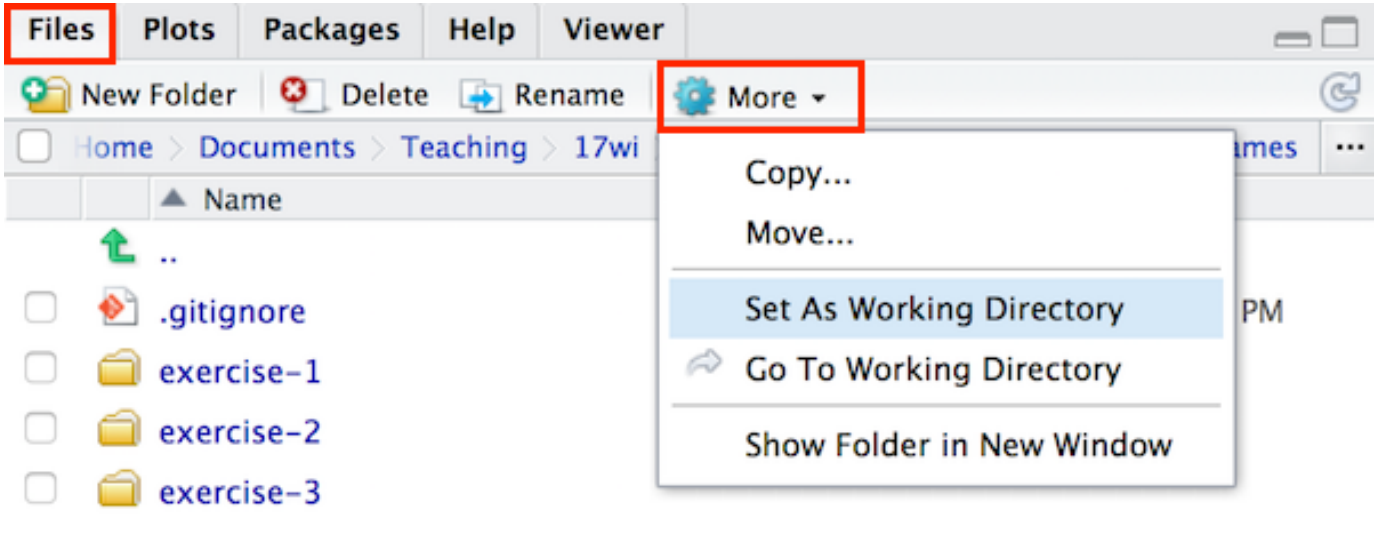
ALWAYS USE RELATIVE PATHS!

Working Directory

RStudio's **working directory** has nothing to do with which script you currently have open (if any!)

```
# get R's current working directory
getwd()  # like pwd, but not
```

Change the working directory through the executing environment: i.e., RStudio!



Module 9 exercise-3

Action Items!

- Be comfortable with **modules 8 - 9** by Thu
- Assignment 3 due ***Tuesday before class***

Thu: Data frame practice; Factors