Lists, data frames, and tibbles

Presenter: Steve Baskauf
steve.baskauf@vanderbilt.edu

Jean & Alexander Heard
LIBRARIES

CodeGraf landing page

- vanderbi.lt/codegraf

Lists

Jean & Alexander Heard
LIBRARIES

Lists

list named `thing`

name	*fruitKind*	*euler*	*vectorData*	*curse*
value	"apple"	2.71828	animal	"!@#$%"
reference value by position	`thing[[1]]`	`thing[[2]]`	`thing[[3]]`	`thing[[4]]`
reference value by name	`thing$fruitKind`	`thing$euler`	`thing$vectorData`	`thing$curse`

- Like vectors, **lists** are one-dimensional data structures.
- However, lists can be **heterogeneous** (contain more than one kind of data object)
- It is typical to give names to values of a list.

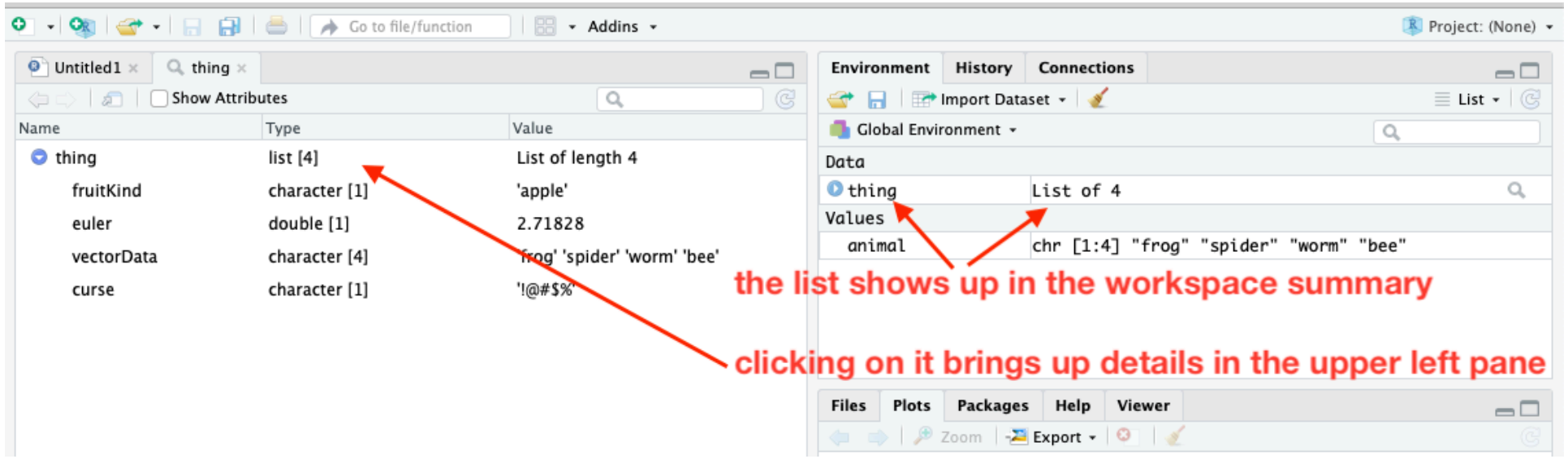
Creating a list

- Lists are created using the **`list()`** function:

```
thing <- list(fruit_kind="apple",
              euler=2.71828,
              vector_data=animal,
              curse="!@#$%")
```

- This list contains character strings, a number, and a vector.
- Values can be assigned names as they are added to the list

Viewing contents of a list



- You can see what's in a list by clicking on its name in the workspace summary in the Environment pane

Referencing list items

- List items can be referenced by:
  - **position** using double square brackets and the index number

    `thing[[2]]`

  - **name** using a dollar sign and the name string

    `thing$curse`

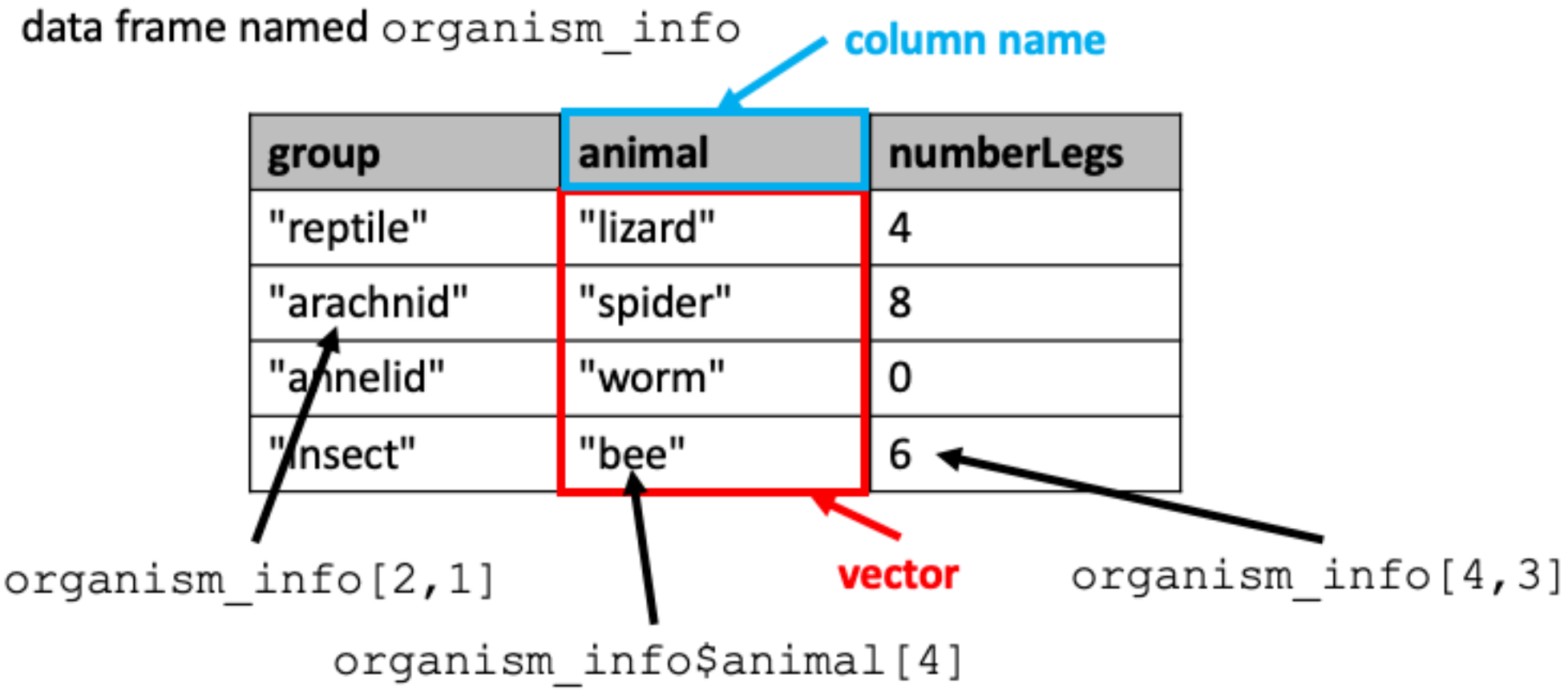
Differences between vectors and lists

- vectors are homogeneous, lists can contain different types of items
- list items are commonly named, vector items can be named but usually aren't
- vector items are usually referenced by their index number (position), list items are commonly reference by their name using the $ notation

Data frames

Jean & Alexander Heard
LIBRARIES

Data frames



- The **column values** are like **vectors**
- The **set of columns** is like a **list**

Making a data frame from vectors

- First make the named vectors

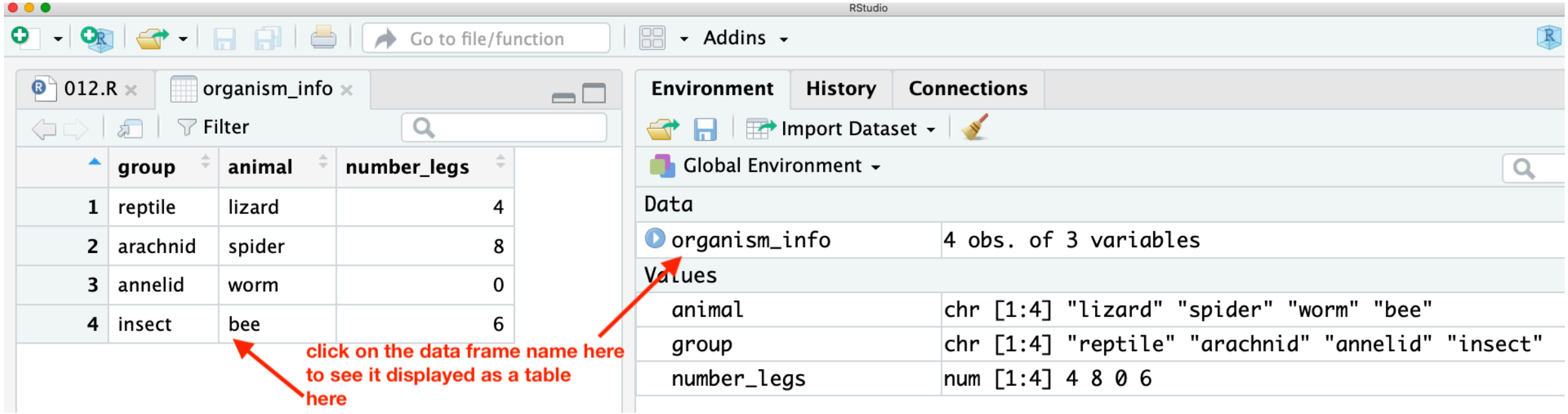
```
group <- c("reptile", "arachnid", "annelid", "insect") # vector of strings

animal <- c("lizard", "spider", "worm", "bee")

number_legs <- c(4,8,0,6) # vector of numbers
```

- Then put the vectors into a data frame

```
organism_info <- data.frame(group, animal, number_legs)
```

- The vector names will be used for the column names

Viewing contents of a data frame



- Click on the name of the data frame in the Environment pane
- The contents will be displayed as a table

Referring to parts of a data frame

- Since the columns are like list items, we can refer to them by name:

```
organism_info$animal
```

- Individual cells can be referenced by:
  - row and column

```
organism_info[2,1]
```

  - column name and position in column

```
organism_info$animal[4]
```

Tibbles

Jean & Alexander Heard
LIBRARIES

What are tibbles?

- Tibbles are a special kind of data frame
- They have all of the characteristics of data frames, but have fewer restrictions about their structure
- They are a key component of the *tidyverse*, a paradigm developed by Hadley Wickham, developer of RStudio. The tidyverse is a collection of related R packages that are commonly used in data science.

How are tibbles different from regular data frames?

- The rules for tibble column names are relaxed. For example, spaces can be included.
- Character data are not automatically imported into data frames as *factors*. Factors are important for experimental analysis, but in many other circumstances we don't care about them.
- Viewing tibbles provides more information that data frames.
- In many cases it does not matter whether your table is a generic data frame or a tibble. However, for statistical analysis of experiments, the distinction may be important.

Loading data frames and tibbles from files

Jean & Alexander Heard
LIBRARIES

Tabular data in delimited files

- **Delimited files** are text files where values are separated by some text character and lines are separated by **newline** characters (i.e. "hard returns").
- Most common type of delimited file: **CSV** (comma separated values)
- Also used: TSV (tab separated values)
- Delimited files are much simpler than Excel files and are commonly used for archiving data.
- CSV files can be made by exporting from Excel

Reading delimited files into data frames

- There are several ways to read data from CSV files into R:
  - by a **file path** (platform-dependent)

```
my_data_frame <- read.csv("~/test.csv")
```
(Mac)

```
my_data_frame <- read.csv("c:\temp\test.csv")
```
(Windows)

  - by a **file-choosing dialog**

```
my_data_frame <- read.csv(file.choose())
```

  - by a **URL**

```
my_data_frame <-
read.csv("https://gist.githubusercontent.com/baskaufs/1a7
a995c1b25d6e88b45/raw/4bb17ccc5c1e62c27627833a4f25380f27d
30b35/t-test.csv")
```

Controlling the import process

- You can specify if the file has a **header row** (labels) using the **`header`** key (default value is TRUE)
- You can specify the **separator** if it's different from comma using the **`sep`** key (default value is comma)
- **`\t`** is the escaped value for a tab character
- Example:

```
nls_ds1 <- read.csv(file.choose(),
                    header = TRUE,
                    sep = "\t")
```

Examining a data frame

- **`head()`** shows the first 6 rows
- **`tail()`** shows the last 6 rows
- **`names()`** returns the column names
- **`str()`** describes the structure of the data frame with information about each column

Other import options

- Excel spreadsheets (openxlsx package)
  - `read.xlsx()`

- Reading files as tibbles (readr package)
  - `read_csv()`
  - Tab separated values: `read_tsv()`

Basic operations on data frames

Jean & Alexander Heard
LIBRARIES

Operating on columns of a data frame

- Since data frame columns are essentially vectors, vectorized operations can be performed on them.
- Column properties:
  - `length(df$column)`
  - `mode(df$column)`
- The output of operations on data frame columns are generally a vector whose length is the number of rows in a column.
- Vectorized operations:
  - one-vector `df$column * 7`
  - two-vector `df$col1 + df$col2`

Vector recycling

- If an operation requires two vectors to be the same length, R automatically repeats the shorter one until it is long enough to complete the operation.
- Example:

```
a <- c(1, 2)
b <- c(10, 15, 17, 5, 1)
a + b
```

a will be extended to a length of 5: **1, 2, 1, 2, 1**

resulting in: **11 17 18 7 2**

Mixing operations on data frames and vectors

- A vector and a data frame column can part of two-vector operations

```
number_wings <- c(0, 0, 0, 4)
number_appendages <- number_wings + organism_info$number_legs
```

- The vector may be shorter than the column and be recycled

```
weekdays <- c(0, 1, 1, 1, 1, 1, 0)
work_week_calls <- calendar$number_calls * weekdays
```