

# R Lesson 2: Objects and Data Structures

[vanderbi.it/r](http://vanderbi.it/r)

Steve Baskauf



# Preliminaries



# Common types of data

- **character**, e.g. "Fred" or "!@#ts23" (in quotes)
- **numeric**, e.g. 15 or 6.02 (no quotes)
- **logical**, TRUE or FALSE (all caps, no quotes)

# Object name recommendations

- Be descriptive (what the object is or does)
- **snake\_case** (underscores) is commonly used:
  - `ordinary_relational_processes`
- camelCase is sometimes used:
  - `bookList, alphabetizeParticipants`
- We can use the term **variable** to refer to named objects
- R doesn't know what a name "means". A meaningful name helps human readers of the code.

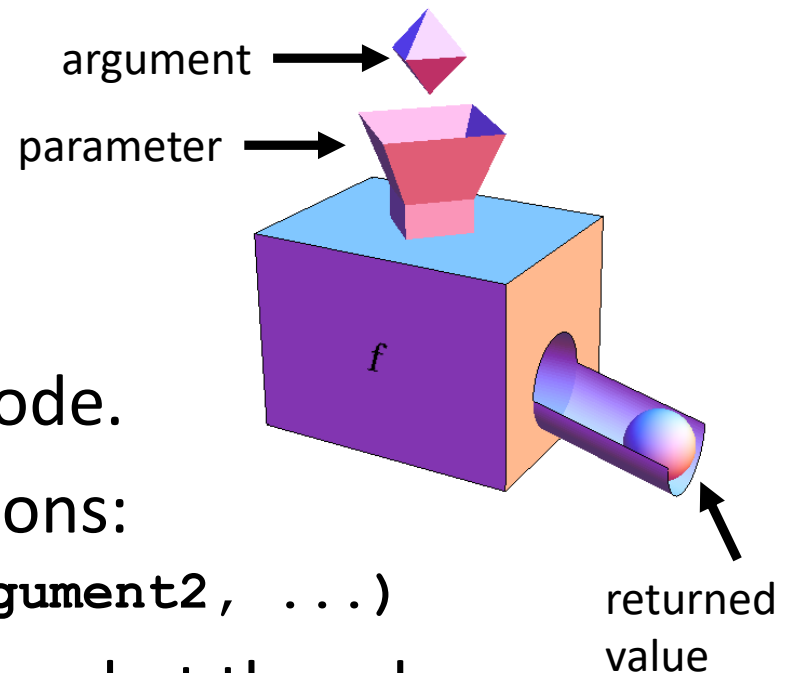
# Assigning a value to an object

- You can **assign** a value to an object using **<-** (similar to a left arrow)
- Examples:
  - `name <- "Steve"` (creating a character object)
  - `my_number <- 6.02` (creating a numeric object)
- Using the equals sign (=) is allowed, but not recommended.
- alt-minus is an RStudio shortcut to generate <-

# "Printing" the value of an object

- R does not have a "print" command.
- entering the name of an object (or expression) in the console evaluates and displays its value

# Functions



- A **function** defines a block of code.
- We pass **arguments** into functions:  
`function_name(argument1, argument2, ...)`
- Functions are usually named by what they do.  
Example:

```
my_latte <- make_latte(beans, milk, water)
```

- Functions can be:
  - built-in to R
  - defined by you in your code
  - defined by somebody else in a package

# Using a function

- We don't have to know anything about the code that makes a function work. We just need to know:
  - What the function does
  - What arguments to put into it
  - What the function will output
- Examples:
  - `sqrt(2)` (evaluate and display)
  - `x <- sqrt(3)` (evaluate and assign to an object)



# Vectors



# Vectors are king in R

vector named `animal`

"frog"	"spider"	"worm"	"bee"
--------	----------	--------	-------

`animal[1]`   `animal[2]`   `animal[3]`   `animal[4]`

- A **vector** is the most common kind of data structure in R.
- Vectors contain a sequence of the **same type** of data.

# Creating vectors

- We commonly use the **construct** function to make vectors:

```
number_vector <- c(1, 3, 6, 10, 15)
animal <- c("frog", "spider", "worm", "bee")
```

- We can also generate a **sequence** of numbers:

```
number_range <- 3:9
count_down <- 10:0
go_negative <- 5:-3
```

- The generated sequence is just another vector!
- (Python users: note the range includes the final value)

# Knowing what's going on with a vector

- display it in console
- examine its value in the environment data pane
- examine its properties:
  - `length(animal)` (how many items)
  - `mode(animal)` (type of data in vector)

# Referencing parts of vectors

- Referencing a **single item**:  
`animal[3]` (displays the third item)  
`animal[2] <- "arachnid"` (assigns "arachnid" to the 2<sup>nd</sup> item)
- Referencing a **range of items** (subvector):  
`animal[2:4]` (the range 2:4 is actually a vector itself)
- (Python users: R vectors are "1 based"; the first item is numbered 1, not 0. Also, the range includes the final value.)

# Single item objects are vectors, too.

- Surprisingly, a single data item assigned to an object is also a vector. We can see this if we ask its length as if it were a vector:

```
an_item <- "some character string"  
length(an_item)
```

- We can reference the single item using vector notation:

```
an_item[1]
```

# Operations on vectors

- Many functions work equally well for a single item or a multi-item vector (since they are both vectors):  
`number_vector <- c(1, 3, 6, 10, 15)`  
`sqrt(number_vector)`
- When operations are performed on vectors, they generally are performed on **all items** in the vectors.

```
> a <- c(10, 30, 100)
> b <- c(5, 10, 20)
> c <- a/b
> c
[1] 2 3 5
```

# More complicated things are also vectors

- A **matrix** is a vector that has been assigned two dimensions
- An **array** is a vector that has been assigned any number of dimensions
- As forms of vectors, matrices and arrays can only consist of one kind of data.
- Example:

```
a_vector <- c(1.1, 1.2, 2.1, 2.2, 3.1, 3.2)
```

```
a_matrix <- matrix(a_vector, 2, 3)
```



# Missing data indicators

- R's built-in indicators for **missing data**:
  - NA** ("not available") means there is a value, but it's missing; length =1
  - NULL means no value; length=0

```
vector_with_missing <- c(1, 2, NA, 3)
```

- NA will prevent some calculations. Example:

```
mean(vector_with_missing)
```

- NA can be used for missing data in tables instead of blank cells

# Other important data structures



# Lists

list named thing

name	<i>fruitKind</i>	<i>euler</i>	<i>vectorData</i>	<i>curse</i>
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

The screenshot shows the RStudio interface. The upper left pane displays a table of objects in the workspace:

Name	Type	Value
thing	list [4]	List of length 4
fruitKind	character [1]	'apple'
euler	double [1]	2.71828
vectorData	character [4]	'frog' 'spider' 'worm' 'bee'
curse	character [1]	'!@#\$\$'

The upper right pane shows the Environment summary with the following data:

Global Environment	
thing	List of 4
Values	
animal	chr [1:4] "frog" "spider" "worm" "bee"

Red arrows and text annotations explain the workflow:

- A red arrow points from the 'thing' row in the upper left pane to the 'thing' row in the Environment pane.
- Another red arrow points from the 'thing' row in the Environment pane to the 'Values' row in the Environment pane.
- Red text says: "the list shows up in the workspace summary".
- Red text says: "clicking on it brings up details in the upper left pane".

- 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`

# Clearing the contents of a pane

- Click on the little broom near the top of the pane
- The view in the pane will be cleared
- In the case of the Environment pane, the values will also be cleared.

# Data frames

data frame named `organismInfo`

column name

group	animal	numberLegs
"reptile"	"frog"	4
"arachnid"	"spider"	8
"annelid"	"worm"	0
"insect"	"bee"	6

`organismInfo[2,1]`

`organismInfo$animal[4]`

vector

`organismInfo[4,3]`

- **Data frames** are essentially tables
- 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("frog", "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

The screenshot shows the RStudio interface. The main editor displays a table with 4 rows and 3 columns: 'group', 'animal', and 'numberLegs'. The table contains the following data:

	group	animal	numberLegs
1	reptile	frog	4
2	arachnid	spider	8
3	annelid	worm	0
4	insect	bee	6

The Environment pane on the right shows the 'organismInfo' data frame with 4 observations and 3 variables. A red arrow points from the 'organismInfo' name in the Environment pane to the table view in the main editor. Below the arrow, red text reads: "click on the data frame name here to see it displayed as a table here".

- 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]`**

# Loading data from files



# 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(file="https://gist.githubusercontent.com/baskauf
s/1a7a995c1b25d6e88b45/raw/4bb17ccc5c1e62c27627833a4f2538
0f27d30b35/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")
```

# Practice: Nashville schools data

1. What does R do when column headers have spaces in them?
2. Display the values in the zip code column
3. How many values are there in the zip code column?
4. Calculate the number of students in each school by adding the values in the male and female columns
5. Calculate the fraction of students that are white in each school
6. Calculate the average fraction of white students by school



# Factors in data frames



# Factors

- A **factor** is a data structure for categorizing data.
- Its origin comes from **experimental design** terminology.
- In an experiment, each **category** into which an experimental trial can fall is called a **level**.
- Factors are sometimes called **grouping variables** because they are used to group observations.
- Factors may be required for some statistical tests and visualizations.

# Factor example: science fair

water factor	height (cm)
wet	25
wet	21
dry	14
wet	13
dry	10
wet	18

- The water factor has two levels: wet and dry
- The height observations can be grouped by whether the experimental treatment was wet or dry

# Factor example: creating factor values

- Create a vector of character strings and a vector of number values:

```
water_conditions <- c("wet", "wet", "dry",  
"wet", "dry", "wet")
```

```
height <- c(25, 21, 14, 13, 10, 18)
```

- Convert the strings into a factor

```
water_factor <- factor(water_conditions)
```

- Display the values of each data structure

```
water_conditions
```

```
water_factor
```

```
height
```

# How to tell that a data structure is a factor

```
> water_conditions
[1] "wet" "wet" "dry" "wet" "dry" "wet"
> water_factor
[1] wet wet dry wet dry wet
Levels: dry wet
> height
[1] 25 21 14 13 10 18
> |
```



The screenshot shows the R Studio Environment pane. At the top are tabs for 'Environment', 'History', and 'Connections'. Below the tabs is a toolbar with icons for file operations and a search bar. The main area is titled 'Global Environment' and contains a table of variables. The table has two columns: the variable name and its R representation. The variables listed are 'height' (a numeric vector), 'water\_conditions' (a character vector), and 'water\_factor' (a factor with two levels, 'dry' and 'wet').

Values	
height	num [1:6] 25 21 14 13 10 18
water_conditions	chr [1:6] "wet" "wet" "dry" "wet" "dry" "wet"
water_factor	Factor w/ 2 levels "dry","wet": 2 2 1 2 1 2

- The main clue is that the **values of the levels** are listed.

# Data frames and factors

- **character strings** imported from CSV files are automatically turned into **factors**
- **numbers** imported from CSV files are imported as **number vectors**
- This automatic behavior takes place because of the historical orientation of R towards statistics.
- The same behavior happens when data frames are built from individual vectors. (Investigate **organism\_info** example)
- This can be good or bad depending on how you want to use the data.

# Questions about the schools data

1. Is zip code a vector or a factor?
  2. Should zip code be a vector or a factor?
  3. Is school name a vector or a factor?
  4. Should school name be a vector or a factor?
- Convert these data to the correct form using:  
**factor()**    turn a vector into a factor  
**as.character()**    turn a factor into a character vector
  - How many levels of zip codes are there (vs. rows)?

# Tibbles

- **Tibbles** are a special kind of data frame
- Tibbles are not built into R.
- Tibbles are part of the **tidyverse** (can be installed separately).
- Tibbles:
  - do not automatically convert character strings to factors when loading from files
  - are more relaxed about column headers
  - plus several other features



# Loading tibbles

- Create a tibble from vectors:

```
library("tibble")
```

```
organism_tibble <- tibble(group, animal,  
number_legs)
```

- Load using **readr** package from tidyverse:

```
library(readr)
```

```
tibble_from_csv <- read_csv(file.choose())
```

- Load an **Excel** file into a tibble

```
library(readxl)
```

```
tibble_from_xl <- read_excel(file.choose())
```

# Homework

- Homework problems related to the Nashville Schools data are in the R script for the lesson