R programming training for absolute beginners

Imran Khan

Old Dominion University

November 21, 2022 to December 06, 2022

What will you learn in this course (Nov 21 to Dec 06)?

Date/Time	Lecture Title	Contents	Resource material
Nov 21,2022 (Monday) **Time: 05:00-06:30 PM EST**	Data structures	1.What is atomic vector? 2.Types of basic atomic vectors? 3.Implicit coercion 4. Explicit coercion 5. What is list? 6. Difference between list and atomic vectors 7. Difference between matrix and dataframe 8. Arrays 9. Logical operators 10. Boolean operators 11.Meaning of vectorization in R language	Chapter # 03 from ''Hands on Programming with R'' (book will be provided) Chapter # 20 from ''R for Data Science'' https://r4ds.had.co.nz/vectors.html
Nov 22,2022 (Tuesday) **Time: 05:00-06:30 PM EST**	Subsetting (I)	1.What is subsetting? 2. Five different methods of subsetting 3. Importance of knowing dimensions of data structure for subsetting 4. Difference between [and [[5. What is Subassignment?	Chapter#04 and Chapter#05 from ''Hands on Programming with R'' (book will be provided) Chapter #04 from ''Advanced R'' https://adv-r.hadley.nz/subsetting.html
Nov 24, 2022 (Thursday) **Time: 05:00-06:30 PM EST**	Subsetting (II)	Remaining part of subsetting	
Nov 28,2022 (Monday) **Time: 05:00-06:30 PM EST**	Control flow structures	1.if statement, ifelse() and case_when() 2. Nested if statements 3. Difference between if statement and ifelse() function 4. Difference between ifelse() and case_when() 5. Difference between single and (&) and double and (&&) 6. Difference between single or (|) and double or (||).	Chapter #05 from ''Advanced R'' https://adv-r.hadley.nz/control-flow.html Chapter ''Control Structures'' from the book ''R Programming from Data Science'' (book will be provided)
Nov 29, 2022 (Tuesday) **Time: 05:00-06:30 PM EST**	Loops	1.When we need to write for loop 2. Difference between looping over the elements and looping over indexes. 3. Why we need to write less for loop in R due to vectorized nature of R language? 4 Exiting for loop (use of exit and break in loops). 5. When we write while loop in R programming? 6. Apply family of functions	Chapter#09 ''Hands on Programming with R'' (book will be provided) Chapter #05 from ''Advanced R'' https://adv-r.hadley.nz/control-flow.html Selected topics from Chapter#21 ''R for Data Science'' https://r4ds.had.co.nz/iteration.html
Dec 01, 2022 (Thursday) **Time: 05:00-06:30 PM EST**	Using functions	1.What is function in programming language? 2.How to call the function in R programming? 3.Difference between matching the arguments by name and matching the arguments by position. 4.Meaning of three dots argument in the function 5.Functions are design for data structure 6.Apply family of functions	Chapter ''Functions'' from the book ''R Programming from Data Science'' (book will be provided) Chapter # 19 from ''R for Data Science'' https://r4ds.had.co.nz/functions.html
Dec 05,2022 (Monday) **Time: 05:00-06:30 PM EST**	Writing functions	1.What is a functional programming language? 2.When should we consider writing our own functions? 3.How to write simple functions in R programming? 4. Returning the single value versus multiple values from the function. 5.Writing your own function with three dots as one of its arguments. 6. Passing anonymous function (and named function) to apply family of functions	Chapter ''Functions'' from the book ''R Programming from Data Science'' (book will be provided) Chapter # 19 from ''R for Data Science'' https://r4ds.had.co.nz/functions.html
Dec 06, 2022 (Tuesday) **Time: 05:00-06:30 PM EST**	Scoping Rules and environment	1.What is lexical scoping rules? 2.Environment basics 3.Recursing over environments 4.Special environments 5.Call stacks	Chapter #06 from ''Advanced R'' https://adv-r.hadley.nz/functions.html Chapter #07 from ''Advanced R'' https://adv-r.hadley.nz/environments.html Chapter# Environments from the book ''Hands on Programming with R''

R programming resources

Beginner tutorial and references	Intermediate References	Advanced References
https://bookdown.org/rdpeng/rprogdatascience/ by Roger Peng – A great text for R programming beginners. Discusses R from the ground up, highlighting programming details. Hands-On Programming with R by Garrett Grolemund.	http://r4ds.had.co.nz/ by Hadley Wickham and Garrett Grolemund. Similar to Advanced R, but focuses more on data analysis, while still introducing programming concepts. Especially useful for working in the tidyverse (http://tidyverse.org/)	Advanced R (http://adv-r.had.co.nz/) by Hadley Wickham. – From the author of several extremely popular R packages. Good follow-up to The Art of R Programming. (And more up-to-date material.) The R Inferno (http://www.burns-stat.com/documents/books/the-r-inferno/) by Patrick Burns.

Why you should R programming?

- R is an open-source language with very active community of R developers.
- Excellent at handling structured (tabular) and unstructured (videos, images, text, streaming data etc) data.
- Very popular in academia and industry for handling big data. What is big data?

Application of R programming	Level of R capabilities
Data analysis	Ahead of all other programming languages
Data visualization	Ahead of all other programming languages.
Econometrics/ Advanced statisticians	R is known as “language of econometricians/statisticians”
Traditional machine learning	At par with Python
Deep learning	Falling behind Python.

JOBS (recent) advertised on different platforms

- Linnkedin

https://www.linkedin.com/jobs/r-programming-jobs...

- Ziprecruiter

https://www.ziprecruiter.com/Jobs/R-Programmer

- Upwork (freelancing jobs)

https://www.upwork.com/freelance-jobs/r/

- Indeed

https://www.indeed.com/q-R-Programmer-jobs.html...

OUTLINE OF LECTURE 1

- **Data Structure**
- What is atomic vector in R? How we create atomic vectors in R?
- Types of atomic vectors
- What is List? How we create List in R?
- Difference between atomic vector and List
- How many dimensions atomic vector and List have?
- How can we convert the vector of one data type into the vector of another data type?
- What is implicit coercion rules?

Atomic vector

- **Vector**: A vector can only contain objects of the same class. In other words, all the elements of the atomic vector are of same data type.
- What are basic data types in R?

Character: “Imran” “SI Khan” “1” “TRUE”

Double (numeric): 1.5 2.6 5

Integer: 1L 10L

Logical: TRUE FALSE T F

Other datatypes are derived from these basic datatypes in R. For example, factor is derived from “integer”. Date time is derived from numeric(double).

- **Types of vector**: four basic types of atomic vectors

1.Character

2. numeric (real numbers or double)

3. integer

4. logical (True/False)

Examples of atomic vector

a <- c(0.5, 0.6, 0.6, 0.8) ## numeric vector

We use c() to create atomic vectors
Each element in the atomic vector is separated by the next element by a “comma”

What is this symbol “<-”: This is assignment operator. “=“ can also be used as assignment operator.

Character_vector <- c(“Andy”, “Billy”, “Cassie”, “David”) ## character vector

What do we mean by length of vector: it is the number of elements in an atomic vector

Integer_vector <- c(1L, 5L, 9L, 10L) ## integer vector

Logical_vector <- c(TRUE, FALSE, T, F) ## logical vector

What if the elements of atomic vector are of more than one data type?

We define the atomic vector as all its elements having the same data type. But what if the elements of the same vector are of different data type?

Implicit coercion in R:

P <- c(1L, 2, 2.5, TRUE, F) ##How many data types are represented in this atomic vector?

Q<- c(5L, "Imran", TRUE) ## implicitly coerced to character vector. Character is the most flexible data type.

R<- c(TRUE, "Sarah", "Mary") ## implicitly coerced to **??**

S<- c(FALSE, 1L, 2L) ## implicitly coerced to integer. Integer is more flexible than logical.

T<-c(1L, 2.5, 2L, 3) ## implicitly coerced to double (numeric) vector.

How to know the class (or data type) of atomic vector? Use typeof() or class() to determine the type of atomic vector

You can also check the class of atomic vector by "is" family of functions is.logical(), is.integer(), is.double(),is.character()

IMPLICIT COERCION RULES

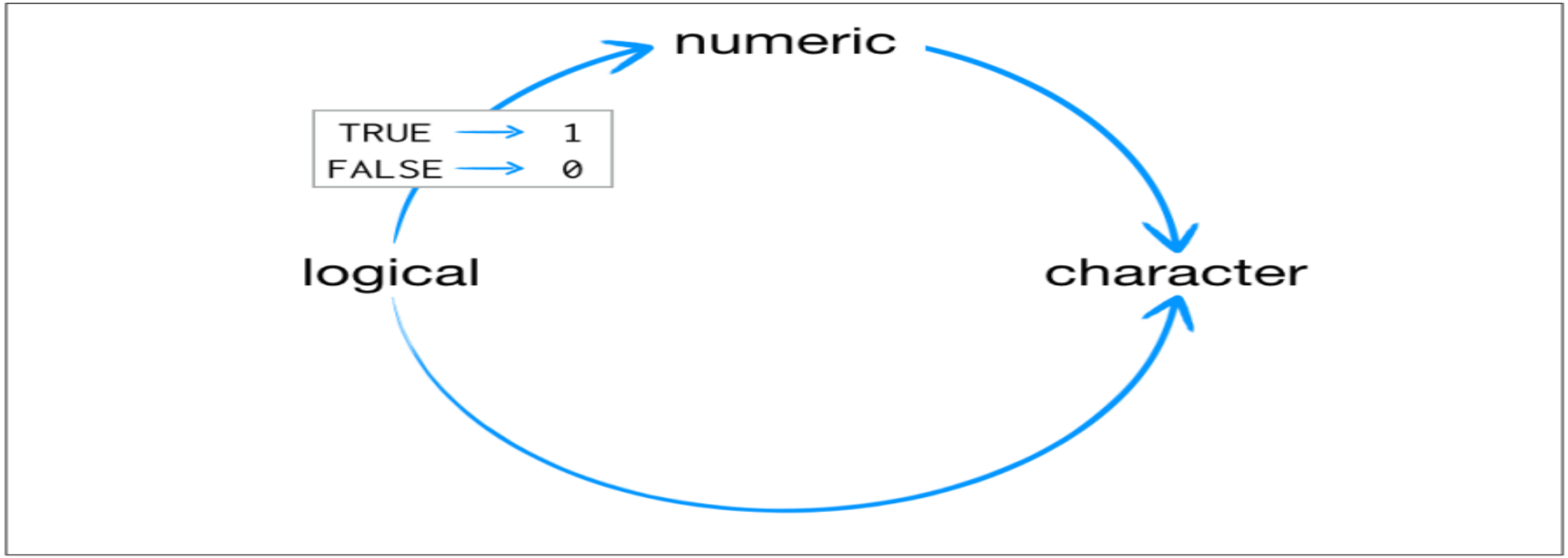


Figure 3-1. R always uses the same rules to coerce data to a single type. If character strings are present, everything will be coerced to a character string. Otherwise, logicals are coerced to numerics.

Explicit coercion

- Converting the vector of one data type into vector of another data type.

We use “as” family of functions to explicitly coerce the atomic vector of one data type into another data type.

Vector type	Checking the vector type	Explicit coercion	Converted vector
p<- c(1.5, 2.5, 3.5, 4.8, 5.99) ## double vector	Class(p) typeof(p) is.double(p)	as.integer(p) ## this will convert the double into integer	c(1, 2, 3, 4, 5)
q<- c(1, 2, 3, 4, 5) ##double vector	Class(p) typeof(p) is.double(p)	as.character(q) ## this will convert the double into character vector	c("1”, "2”, "3" ,"4”, “5”)
r<-c(TRUE, FALSE, TRUE) ##logical vector	Class(r) typeof(r) is.logical(r)	as.integer(r) ## this will convert the logical vector into integer vector.	c(1, 0, 1)
S<-c(“a”, “b”, “c”, “d”) ##character vector	Class(S) typeof(S) is.character(S)	as.integer(S) ## Try to convert but NAs will be introduced.	c(NA, NA, NA, NA)

What if we want to store elements of different data types in a data structure?

- If we try to store the elements of different data types in an atomic vector, R will implicitly coerce the atomic vector to a single data type. Therefore, R will not let you store the elements of different data types in atomic vector. What we do now?
- **Welcome to "list" data structure** : Most powerful data structure in R.
- List are very flexible. List can store elements of different data types in it.
- x <- list(1, "a", TRUE, list(5,5,6,7))

How to create list? How are the elements of list separated from each other? How many elements the list named "x" have? What is the data type of each element of list "x"?

- y<-c(1, 1.5, TRUE) ## is "y" list ??Recall previous slides material
- z<-list(1, 2, 3, 4, 5) ## we have stored data of one type in "z". Is "z" list or atomic vector?
- named_list<- list(names = c("Andy", "Billy", "Cassie", "David"), age = c(30,35,40, 45), language=c("R", "R", "R", "Python"))

it is a named element. Each element of list has a name. Name of first element is "names". Name of second element is "age" and name of third element is "language".

Matrix and Data frame

- Both Matrix and data frame have two dimensions—rows and columns.
- How to create the matrix from atomic vector?
- vec_1<-c(1,2,3,4,5,6)
- Matrix_example <- matrix(vec_1, nrow = 2)
- df <- data.frame(Name = c("ace", "two", "six"), suit = c("clubs", "clubs", "clubs"), value = c(1, 2, 3))

Data frame

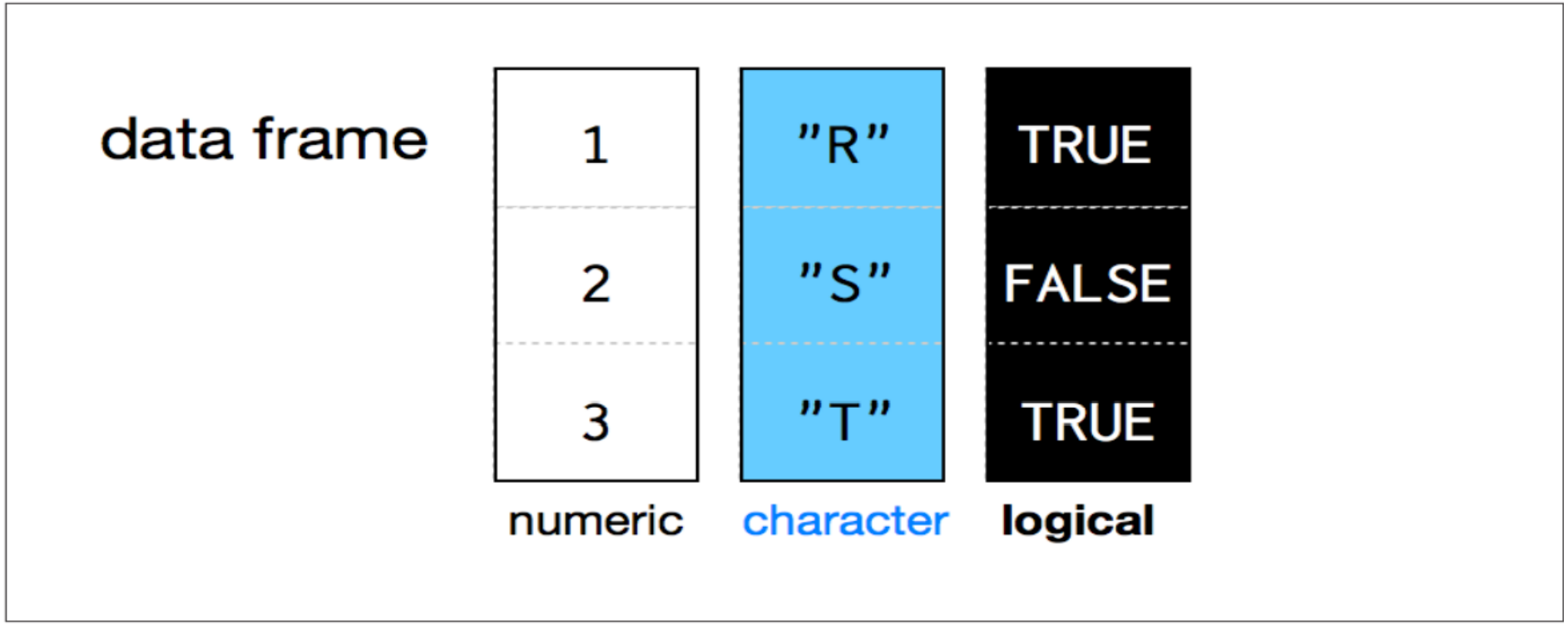


Figure 3-2. Data frames store data as a sequence of columns. Each column can be a different data type. Every column in a data frame must be the same length.

SUMMARIZING R's COMMON DATA STRUCTURES

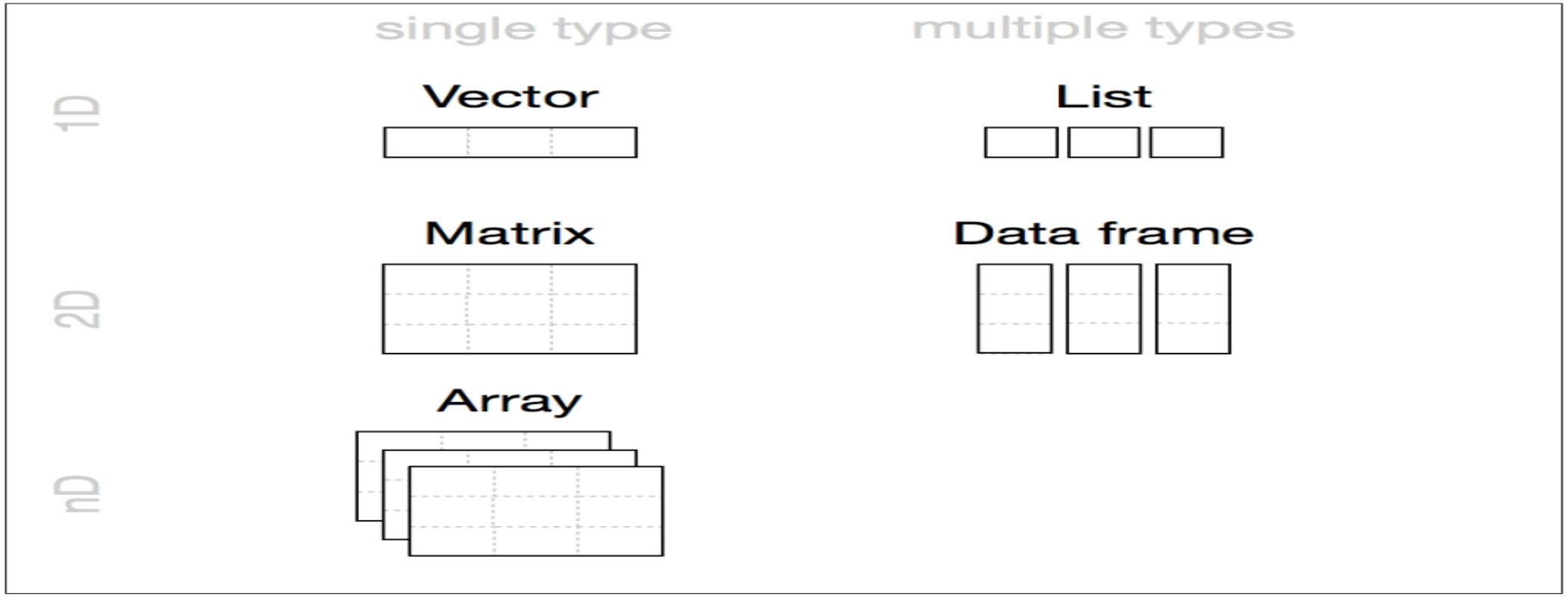


Figure 3-6. R's most common data structures are vectors, matrices, arrays, lists, and data frames.

R’s logical operators

Table 5-1. R’s logical operators

Operator	Syntax	Tests
`>`	`a > b`	Is a greater than b?
`>=`	`a >= b`	Is a greater than or equal to b?
`<`	`a < b`	Is a less than b?
`<=`	`a <= b`	Is a less than or equal to b?
`==`	`a == b`	Is a equal to b?
`!=`	`a != b`	Is a not equal to b?
`%in%`	`a %in% c(a, b, c)`	Is a in the group c(a, b, c)?

Boolean operators in R

- Boolean operators are things like and (&) and or(|). They collapse the results of multiple logical tests into a single TRUE or FALSE. R has six Boolean operators.

Table 5-2. R's Boolean operators

Operator	Syntax	Tests
&	`cond1 & cond2`	Are both `cond1` and `cond2` true?
|	`cond1 pipe cond2`	Is one or more of `cond1` and `cond2` true?
xor	`xor(cond1, cond2)`	Is exactly one of `cond1` and `cond2` true?
!	`!cond1`	Is `cond1` false? (e.g., `!` flips the results of a logical test)
any	`any(cond1, cond2, cond3, ...)`	Are any of the conditions true?
all	`all(cond1, cond2, cond3, ...)`	Are all of the conditions true?

Basic calculation in R

Math	R	Result
$3+2$	`3 + 2`	5
$3-2$	`3 - 2`	1
$3 \cdot 2$	`3 * 2`	6
$3/2$	`3 / 2`	1.5

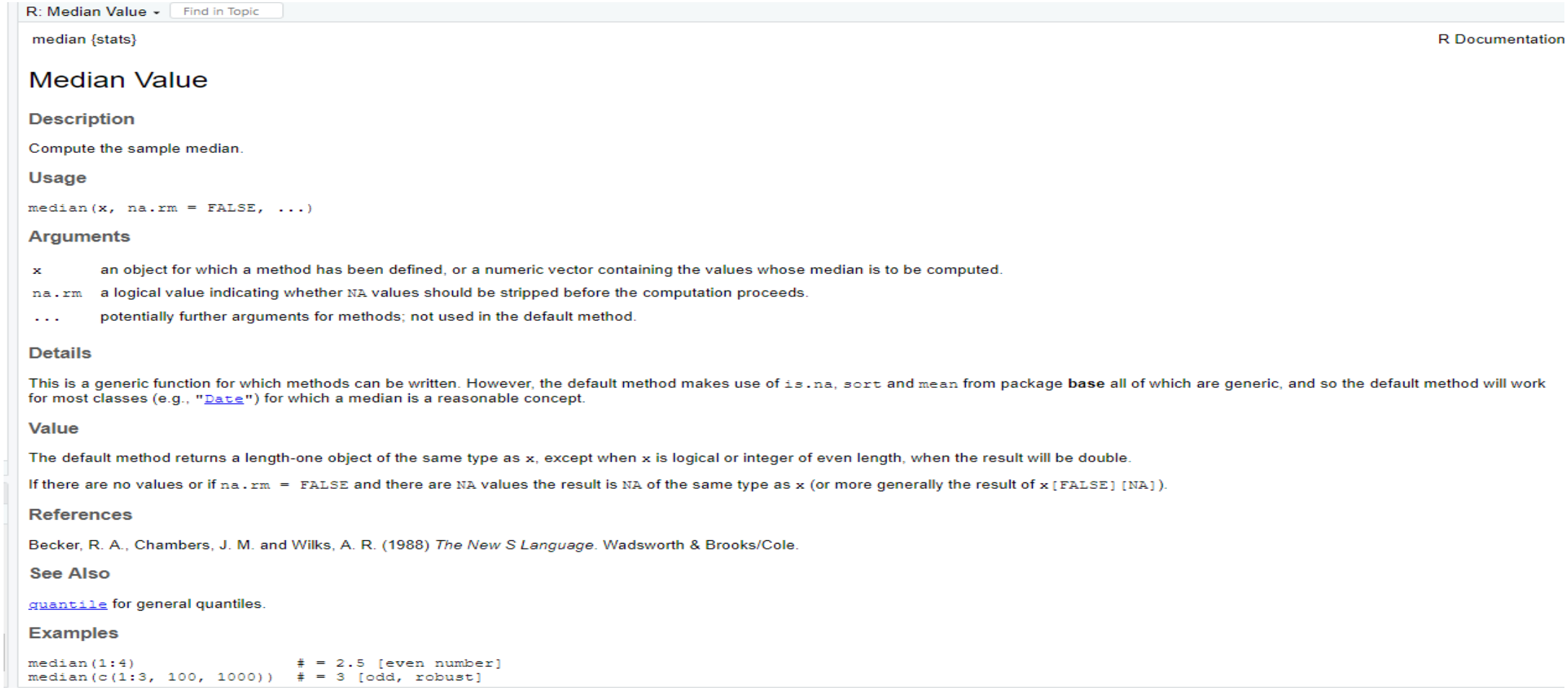
Math	R	Result
3^2	`3 ^ 2`	9
$2^{(-3)}$	`2 ^ (-3)`	0.125
$100^{1/2}$	`100 ^ (1 / 2)`	10
$\sqrt{100}$	`sqrt(100)`	10

Math	R	Result
π	`pi`	3.1415927
e	`exp(1)`	2.7182818

Math	R	Result
$\log(e)$	`log(exp(1))`	1
$\log_{10}(1000)$	`log10(1000)`	3
$\log_2(8)$	`log2(8)`	3
$\log_4(16)$	`log(16, base = 4)`	2

Calling “help” in RStudio

- ?median



R: Median Value Find in Topic

median {stats} R Documentation

Median Value

Description

Compute the sample median.

Usage

```
median(x, na.rm = FALSE, ...)
```

Arguments

`x`	an object for which a method has been defined, or a numeric vector containing the values whose median is to be computed.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	potentially further arguments for methods; not used in the default method.

Details

This is a generic function for which methods can be written. However, the default method makes use of `is.na`, `sort` and `mean` from package **base** all of which are generic, and so the default method will work for most classes (e.g., "`Date`") for which a median is a reasonable concept.

Value

The default method returns a length-one object of the same type as `x`, except when `x` is logical or integer of even length, when the result will be double.

If there are no values or if `na.rm = FALSE` and there are `NA` values the result is `NA` of the same type as `x` (or more generally the result of `x[FALSE][NA]`).

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

`quantile` for general quantiles.

Examples

```
median(1:4)                # = 2.5 [even number]
median(c(1:3, 100, 1000))  # = 3 [odd, robust]
```

LEVEL I SESSION II

Outline of Today's lecture

- What is subsetting?
- **Examples**: Subsetting with one dimensional and two-dimensional data structure
- Five different methods of subsetting
- Preserving versus simplifying the dimensionality in higher dimensional data structure.
- Difference between subsetting operators [, [[and $
- Sub-assignment (combining subsetting with assignment to modify the elements of data structure in place)

SUBSETTING

- Subsetting is pulling/accessing the subset of values/elements from the data structure. Subsetting does not modify (remove the values/elements) the original R object. Subsetting returns the copy of values.
- **Important**: For subsetting you need to know the **dimensions** of data structure. For example, atomic vectors and list have one dimension, therefore, we need to give a single index to subset them. Data frame and Matrix have two dimensions, therefore we need two indexes (one for row and the other for column) to subset dataframe/matrix. And the indexes are separated by a comma.
- Remember the counting starts from 1 in the R programming language. It is different from Python. In Python counting starts from 0.

Subsetting examples

- How we subset one dimensional R data structures (vectors and list)?
- How we subset two-dimensional R data structures (matrices and data frame)?

If I want to extract/access more than one value simultaneously from the following vector, what should I do?
vec[1:3] ## let's suppose you want to access the values at position 1, 2 and 3

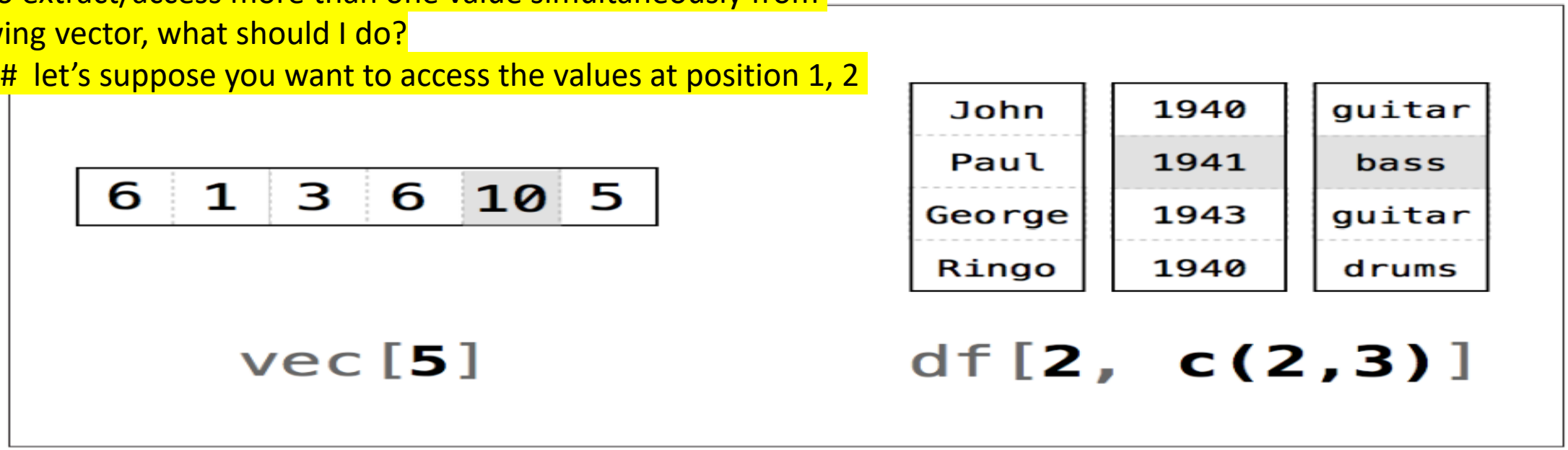


Figure 4-1. R uses the ij notation system of linear algebra. The commands in this figure will return the shaded values.

Different Subsetting methods

1.**Positive integer:**

```
vec<-c(2,10,15,21,5,6)
vec[c(1,2,3)]
##2 10 15
```

2.**Negative integer:**

```
vec<-c(2,10,15,21,5,6)
vec[-c(2,4)]
vec[c(-2,-4)]
## 2 15 5 6
```

3.**Blank spaces**

```
vec[]
##2 10 15 21  5  6
```

For the convenience, I have explained these subsetting method here for atomic vector (double). These subsetting methods works with all data structures.

4.**Logical values:**

```
vec[vec>5]
##10 15 21  6
```

5. **Names**

```
named_vector <-c(name ="David", age="40", language="R")
named_vector[c("name", "language")]
## name     language
##"David"       "R"
```

We also have named atomic vectors like named lists. Both atomic vectors and the list could be named or unnamed. However, names give you additional subsetting method

Subsetting dataframe

- We can subset the dataframe by a single index as well as two indexes. Is it not surprising for you? Yesterday we studied that the dataframe has two dimensions therefore we should need two indexes to subset values from dataframe.

- Under the hood, dataframe is a list but it is a special kind of list where all the elements of list are of equal length. Therefore, we can also treat the dataframe as a list. And we can subset the dataframe just like we subset the list (with single dimension).

Subsetting dataframe

```r
df <- data.frame(x = 1:3, y = 3:1, z = letters[1:3])
```

```r
# There are two ways to select columns from a data frame
# Like a list
df[c("x", "z")]
#>   x z
#> 1 1 a
#> 2 2 b
#> 3 3 c
# Like a matrix
df[, c("x", "z")]
#>   x z
#> 1 1 a
#> 2 2 b
#> 3 3 c
```

When you are subsetting the data frame as a list, you will provide one index. You will provide the name of elements as a character vector to access the elements (columns). If we are accessing the single element (column) or multiple elements(multiple columns), the output data structure would always be a dataframe if we are subsetting the data frame as a list.

When you are subsetting the dataframe with two dimensions, you will get sometimes dataframe and sometimes vector. In other words, when you are subsetting the dataframe with two dimensions, sometimes the dimension of output data structure is preserved and sometimes the dimension of output data structure is simplified.

Preserving versus simplifying dimensions of output data structure in higher dimensional data structure

- Preserving versus simplifying : What does it mean?
- When will the dimensionality of output data structure preserved and when simplified (Remember we are subsetting the data frame with two dimensions?
- If you are subsetting the single column you will get back the vector: In other words, the dimensionality of your output data structure is simplified (from 2-dimension to 1-dimension). Solution: Use drop = FALSE argument to preserve the dimensionality.
- However, if you are subsetting multiple columns, the return data structure will still be the data frame. In other words, the dimensionality of your output data structure is preserved (your input data structure is 2-dimensional and your output data structure is also two dimensional).

Sub-setting List: [] and[[]]

We need one index to subset list whether we are sub-setting it with single square bracket or double square bracket. Why? Because list (like vector) have one dimension.

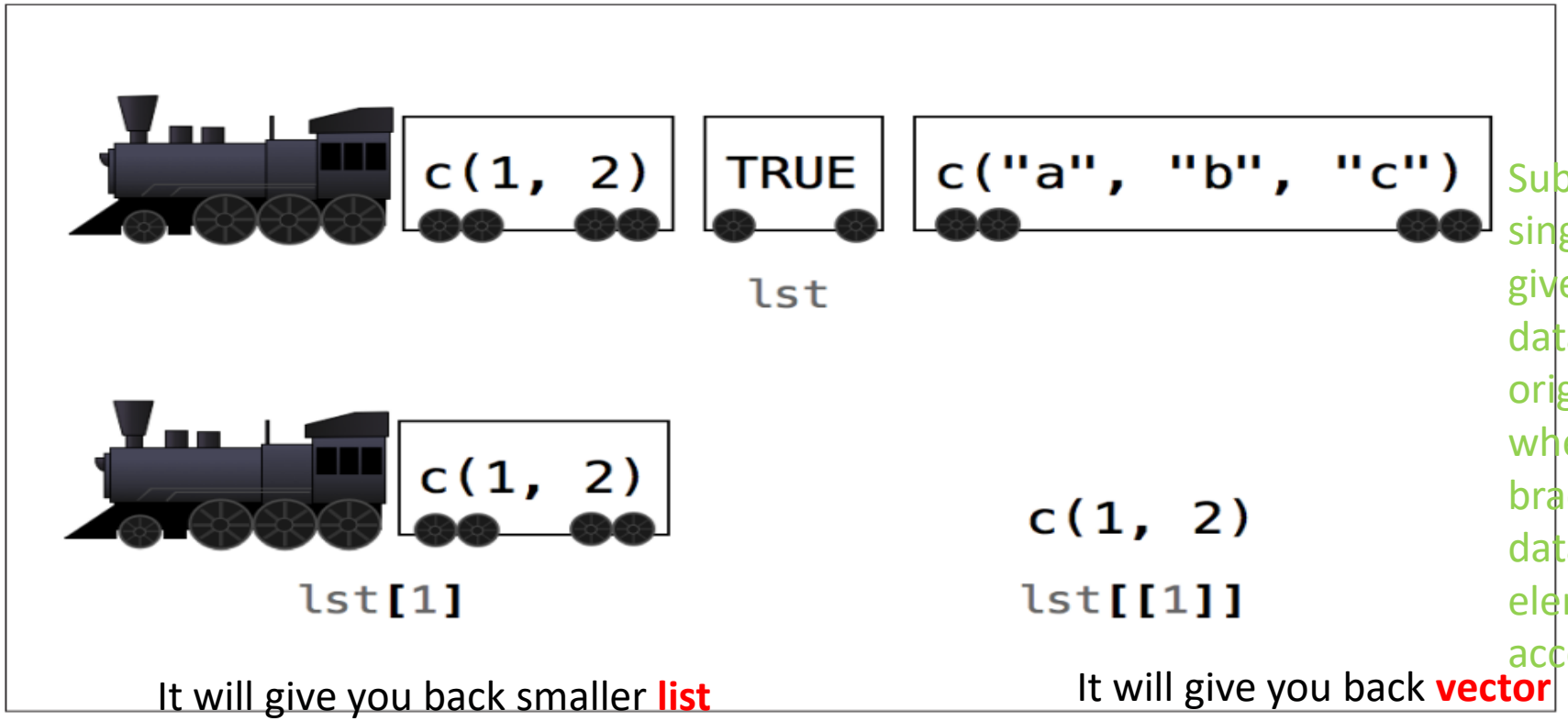


Figure 4-3. It can be helpful to think of your list as a train. Use single brackets to select train cars, double brackets to select the contents inside of a car.

Sub-setting through single square bracket give you back the same data structure as the original data structure whereas double square bracket return you the data structure of the element being accessed/pulled.

Subsetting sub element(s) of an element of list

- If I want to access/pull/extract the second sub-element of the first element in the following list. How would you do it?
- lst <-list(c(1,2), TRUE,   c("a","b","c"))

Two ways to do that:

lst[[c(1,2)]]

lst[[1]][[2]] I prefer this method for subsetting sub elements

If I want to access/pull/extract the second and third sub-elements of the third element in the above list. How would you do it?

lst[[3]] [c(2,3)]

Subsetting a dataframe and list with $ operator

```
a <- list(a = 1:3, b = "a string", c = pi, d = list(-1, -5))
a$a  ## you will get back the data structure of first element
a$b ## What is the data structure you will get back?
a$d ## ??
```

name_of_list$name_of_element

Dataframe: Subsetting with $ operator gives you back atomic vector (single column).

$ operator lets you pull/access the SINGLE element from a list or dataframe.

For example, run the following command in RSTudio: mtcars$mpg

And also run typeof(mtcars$mpg) ## you will get back double.

Subassignment

What if we want to replace (modify in place) the single or multiple values in the data structure?

```
vec<- c(100, 200, 300, 400, 450, 600)
vec[5]<-500
vec
#100 200 300 400 500 600
```

What will this command do? Can you guess the position of elements we are replacing?

```
vec<- c(100, 200, 300, 400, 450, 600)
vec[c(1,6)]<-c(1000,1000)
vec
```

Don’t forget to practice code at home

Let us proceed to RStudio for hands-on training to understand different methods of sub-setting. Don’t worry! We will conquer sub-setting in 2nd R Session.