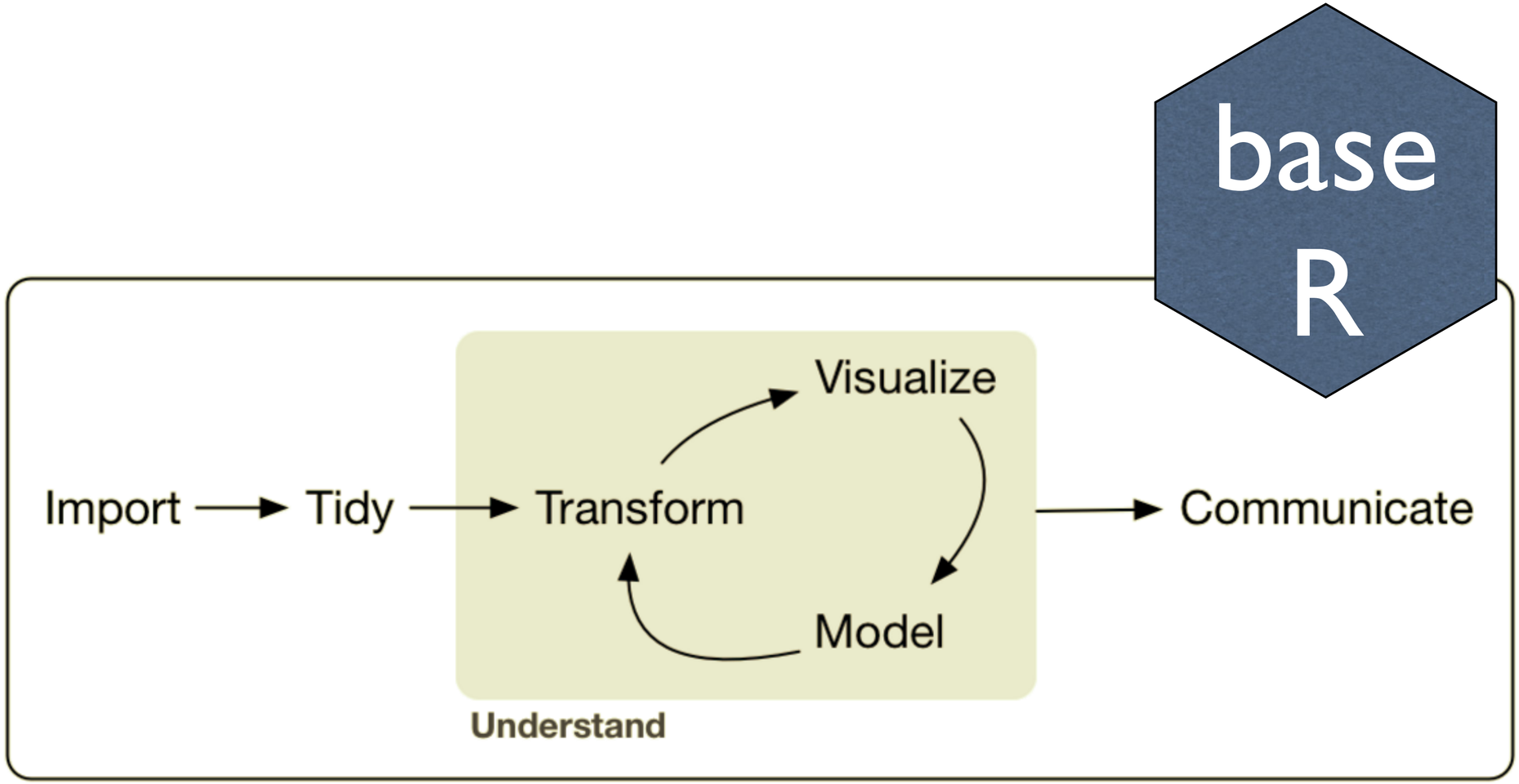
More on Data Types and Syntax — transitioning to programming in R

Data Types in R

1. Vectors
2. Matrices
3. Data types
4. Coercion



From *R for Data Science* by Hadley Wickham and Garrett Grolemund.

Your Turn 1

Look at the R object WorldPhones (by typing its name in your notebook or the Console and hitting enter).

What is inside of WorldPhones?

WorldPhones

	N.Amer	Europe	Asia	S.Amer	Oceania	Africa	Mid.Amer
1951	45939	21574	2876	1815	1646	89	555
1956	60423	29990	4708	2568	2366	1411	733
1957	64721	32510	5230	2695	2526	1546	773
1958	68484	35218	6662	2845	2691	1663	836
1959	71799	37598	6856	3000	2868	1769	911
1960	76036	40341	8220	3145	3054	1905	1008
1961	79831	43173	9053	3338	3224	2005	1076

You can save more than a single number in an object by creating a *vector*, *matrix*, or *array*.

Vectors

Your turn

How many dimensions does a vector have?

1 2 3 4 5 6

How many dimensions does a vector have?

vectors

Combine multiple elements into a one dimensional array.

Create with the c function (for "concatenate").

```
vec <- c(1, 2, 3, 10, 100)
vec
```

Your turn

What happens in your Environment when you run this code?

In your Notebook?

```
vec <- c(1, 2, 3, 10, 100)
vec
```

vectors

Combine multiple elements into a one dimensional array.

Create with the c function (for "concatenate").

```
vec <- c(1, 2, 3, 10, 100)
vec
# 1   2   3  10 100
```

Matrices

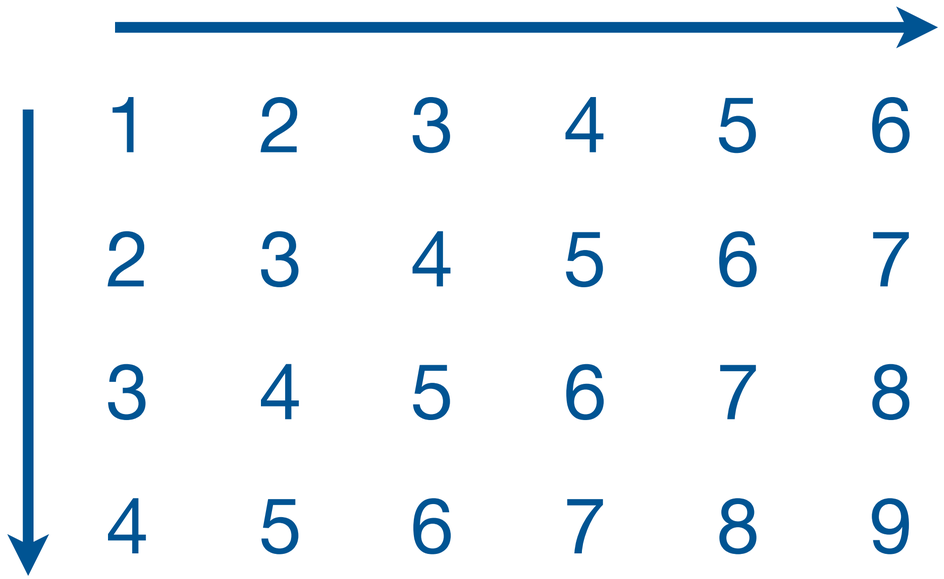
Uncommon, but good to know

Your turn

How many dimensions does a matrix have?

1	2	3	4	5	6
2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9

How many dimensions does a matrix have?

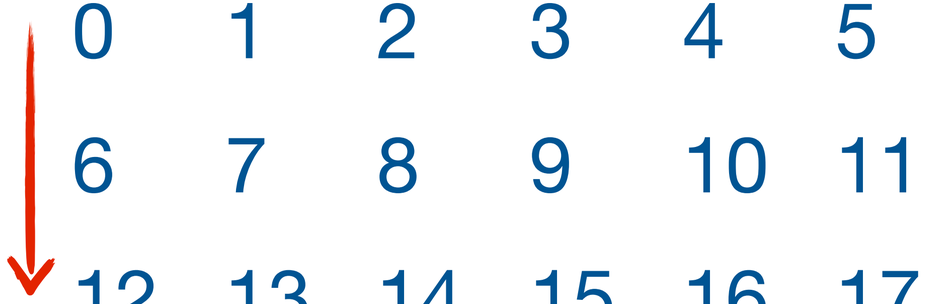


Your turn

The matrix below is named **M**.
What is the value of $\mathbf{M}_{34}$?

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

The matrix below is named **M**.
What is the value of $\mathbf{M}_{34}$?



The matrix below is named **M**.
What is the value of $\mathbf{M}_{34}$?

The matrix below is named **M**.
What is the value of $\mathbf{M}_{34}$?

matrices

multiple elements stored in a two dimensional array.

Create with the matrix function.

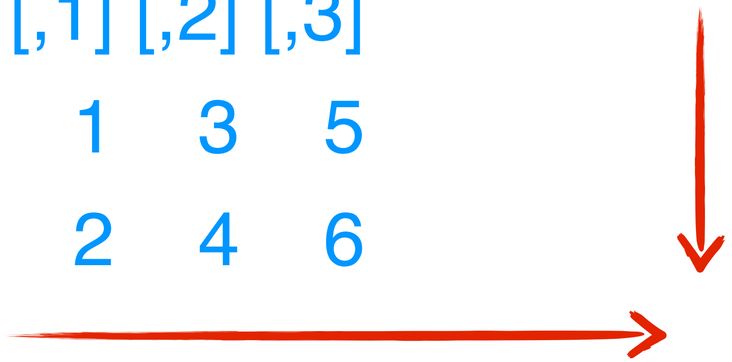
```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
mat
#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```

matrices

Combine multiple elements into a two dimensional array.

Create with the matrix function.

```
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
mat
#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```



vector of elements to go in the matrix

```
matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```

number of rows for matrix

```
matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6
```

```
matrix(c(1, 2, 3, 4, 5, 6), nrow = 3)
#      [,1] [,2]
# [1,]    1    4
# [2,]    2    5
# [3,]    3    6
```

R as a calculator (again)

Math: element-wise

vec + 4

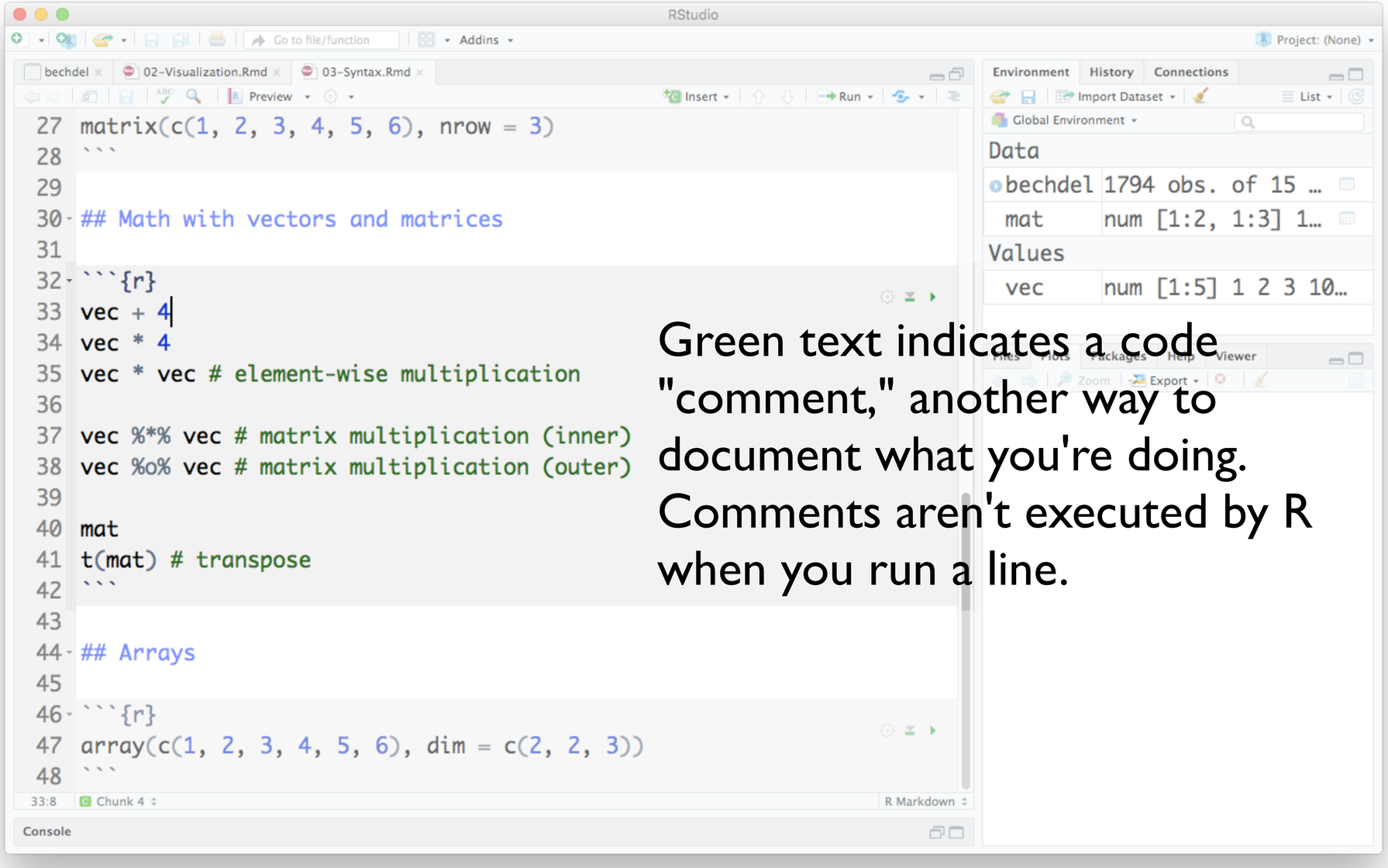
```
# 5   6   7  14 104
```

vec * 4

```
# 4   8  12  40 400
```

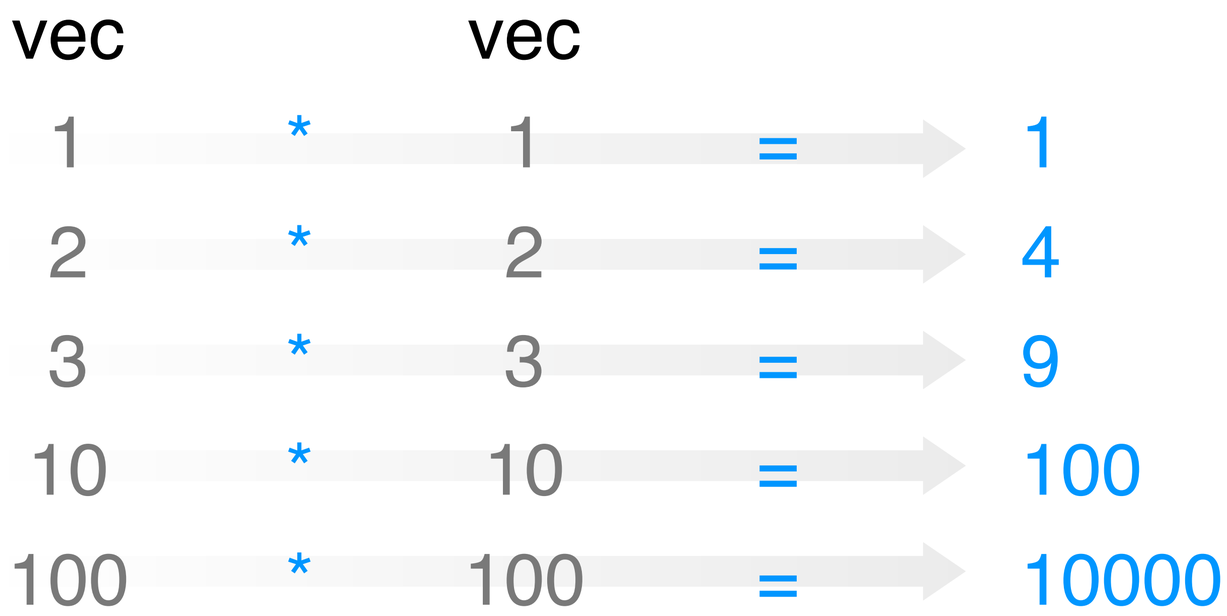
vec * vec

```
# 1     4     9   100 10000
```



Green text indicates a code "comment," another way to document what you're doing. Comments aren't executed by R when you run a line.

```
vec * vec
# 1    4    9   100 10000
```



Matrix multiplication

```
vec %*% vec # inner
#      [,1]
# [1,] 10114
```

```
vec %o% vec # outer
#      [,1] [,2] [,3] [,4]  [,5]
# [1,]    1    2    3   10   100
# [2,]    2    4    6   20   200
# [3,]    3    6    9   30   300
# [4,]   10   20   30  100  1000
# [5,]  100  200  300 1000 10000
```

```
mat
#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    2    4    6

t(mat)
#      [,1] [,2]
# [1,]    1    2
# [2,]    3    4
# [3,]    5    6
```

arrays

Combine multiple elements into an array that has three or more dimensions.

Create with the array function.

```
array(c(1, 2, 3, 4, 5, 6), dim = c(2, 2, 3))
```

arrays

Combine multiple elements into an array that has three or more dimensions.

Create with the array function.

```
array(c(1, 2, 3, 4, 5, 6), dim = c(2, 2, 3))
```

Another uncommon structure

Data types

Warm up

	A	B	C	D
1	date	president	democrat	unemploy
2	Mar 31, 1968	Lyndon Johnson	TRUE	2709
3	Apr 30, 1968	Lyndon Johnson	TRUE	2740
4	May 31, 1968	Lyndon Johnson	TRUE	2938
5	Jun 30, 1968	Lyndon Johnson	TRUE	2883
6	Jul 31, 1968	Lyndon Johnson	TRUE	2768
7	Aug 31, 1968	Lyndon Johnson	TRUE	2686
8	Sep 30, 1968	Lyndon Johnson	TRUE	2689
9	Oct 31, 1968	Lyndon Johnson	TRUE	2715
10	Nov 30, 1968	Lyndon Johnson	TRUE	2685
11	Dec 31, 1968	Lyndon Johnson	TRUE	2718
12	Jan 31, 1969	Richard Nixon	FALSE	2692
13	Feb 28, 1969	Richard Nixon	FALSE	2712
14	Mar 31, 1969	Richard Nixon	FALSE	2758
15	Apr 30, 1969	Richard Nixon	FALSE	2713
16	May 31, 1969	Richard Nixon	FALSE	2816
17	Jun 30, 1969	Richard Nixon	FALSE	286
18	Jul 31, 1969	Richard Nixon	FALSE	
19	Aug 31, 1969	Richard Nixon	FALSE	
20	Sep 30, 1969	Richard Nixon	FALSE	
21	Oct 31, 1969	Richard Nixon	FALSE	
22	Nov 30, 1969	Richard Nixon	FALSE	

What **types** of data appear in this spreadsheet?

data types

Like Excel, Numbers, etc., R can recognize different types of data.

We'll look at four basic types:

- numbers
- character strings (text)
- logical
- factor

numeric

Any number, no quotes.

Appropriate for math.

```
1 + 1

3000000

class(0.00001)
# "numeric"
```

character

Any symbols surrounded by quotes.

Appropriate for words, variable names, messages, any text.

```
"hello"

class("hello")
# "character"
```

```
"hello" + "world"
# Error

nchar("hello")
# 5

paste("hello", "world")
# "hello world"
```

Your turn

Which of these are numbers? **What are the others?** How can you tell?

1 "1" "one"

logical

TRUE or FALSE

R's form of binary data. Useful for logical tests.

```
3 < 4
# TRUE

class(TRUE)
# "logical"

class(T)
# "logical"
```

factor

R's form of categorical data. Saved as an integer with a set of labels (e.g. levels).

```
fac <- factor(c("a", "b", "c"))
fac
# a b c
# Levels: a b c

class(fac)
# factor
```

Use great caution with factors

Quiz

```
x <- c(1, 2, 3)
```

What is the difference between these?

```
x
```

```
"x"
```

Type	Examples
numeric	0, 1, -2, 3.1415, 0.0005
character	"Amelia", "Agree", "31"
logical	TRUE, FALSE
factor	a c c b Levels: a b c

Your turn 2

Make a vector that contains the number 1, the letter R, and the logical TRUE.

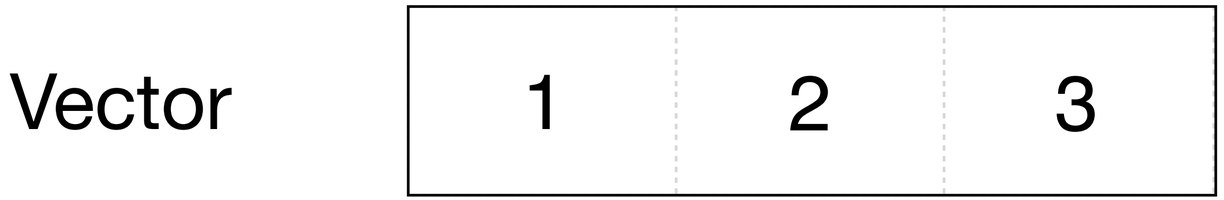
What class of data is the vector?

```
vec <- c(1, "R", TRUE)
class(vec)
# "character"

vec
# "1"    "R"    "TRUE"

# What is R doing?
```

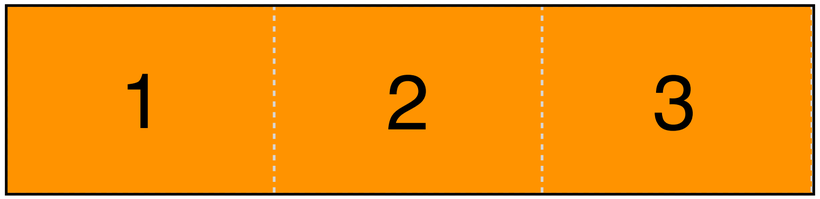
Your turn

Another way to see the class of an object is in the Environment pane. Does the Environment agree with what you found using class()?

Vector

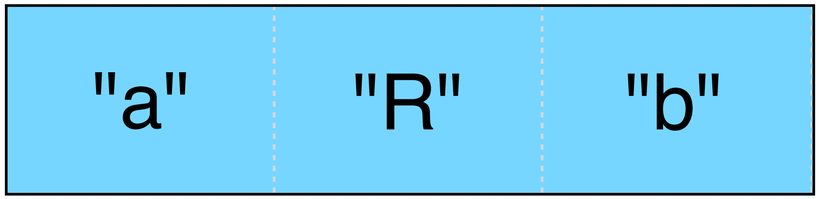

Vector
1
2
3

Vector



numeric

Vector

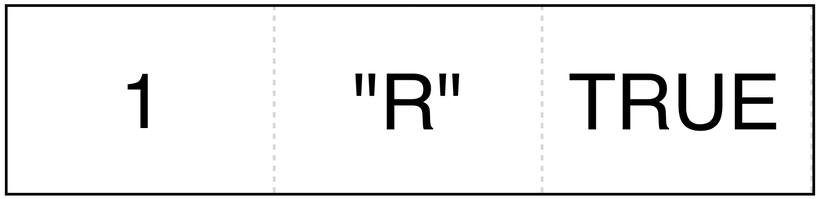


character

Vector

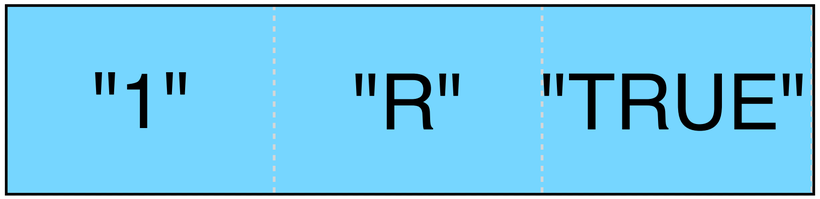
logical

Vector



?

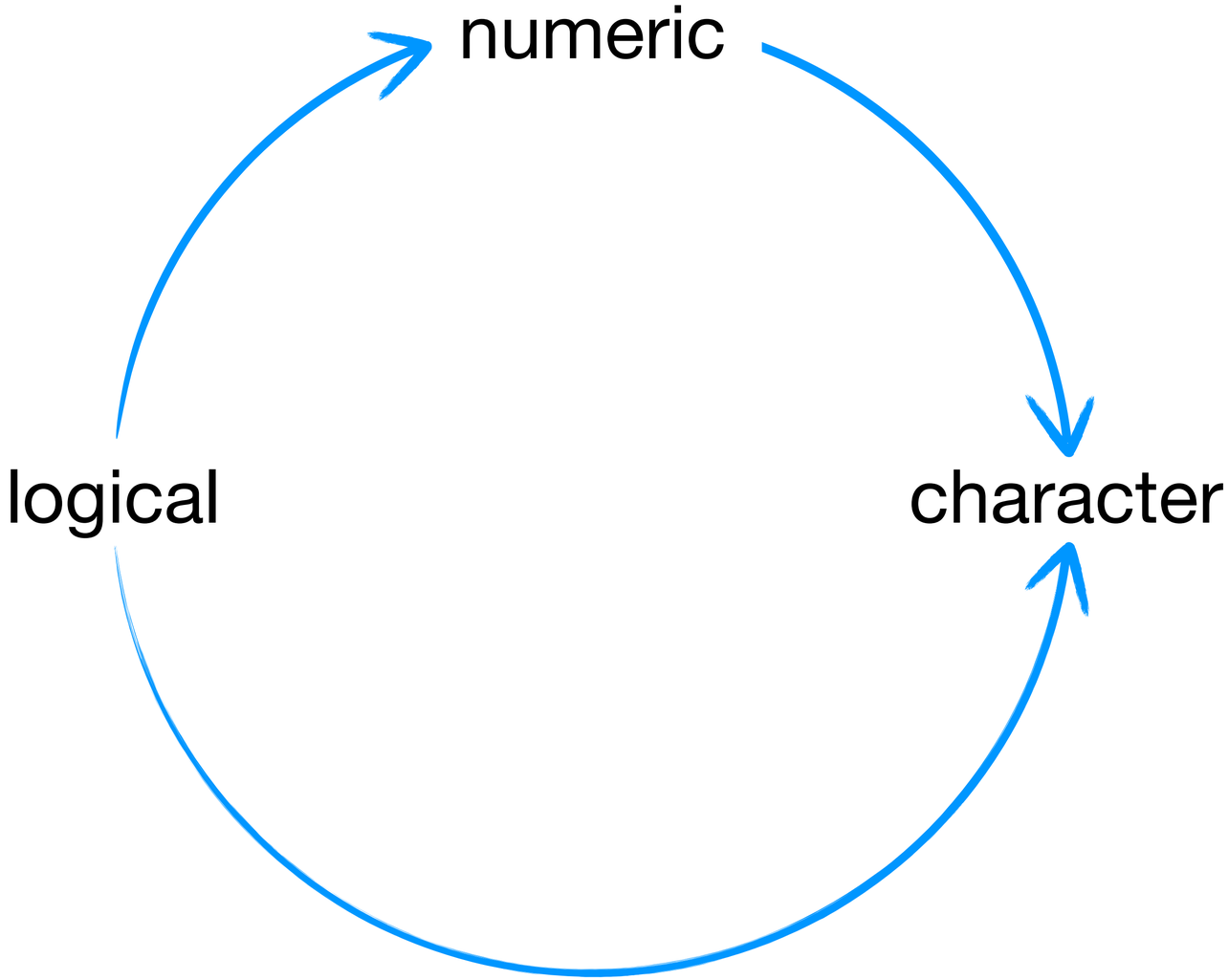
Vector



character

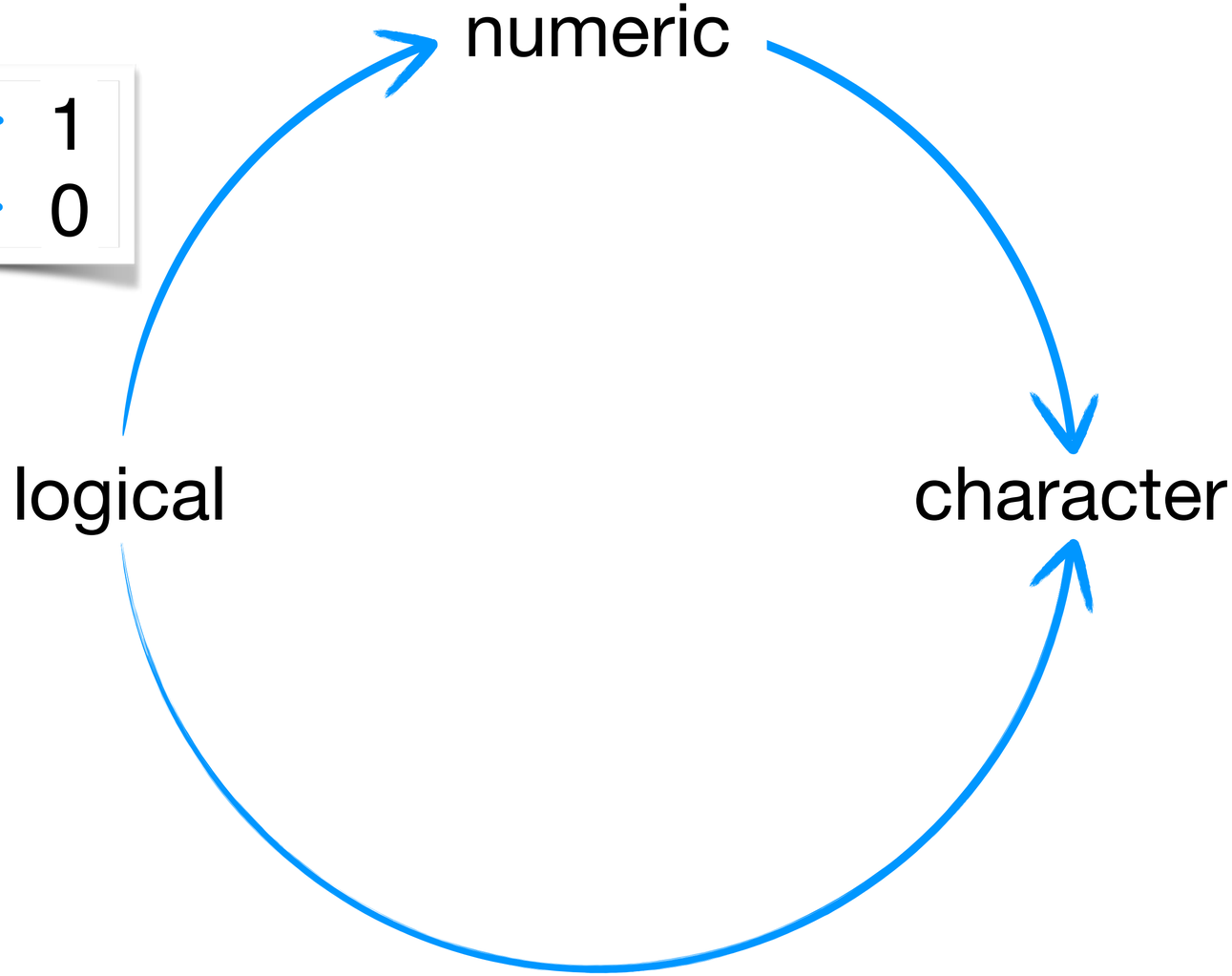
Coercion

coercion

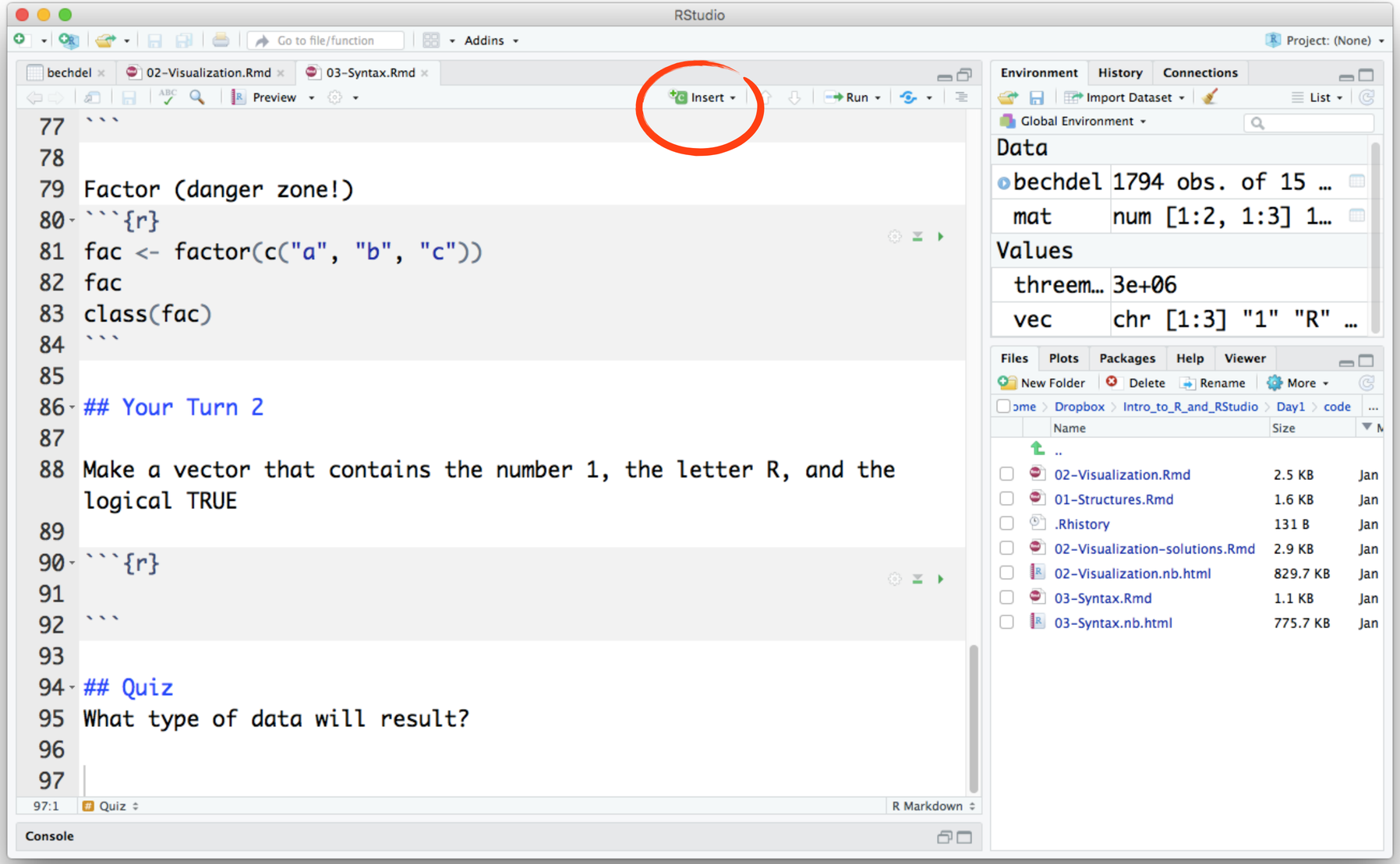


coercion

TRUE → 1
FALSE → 0

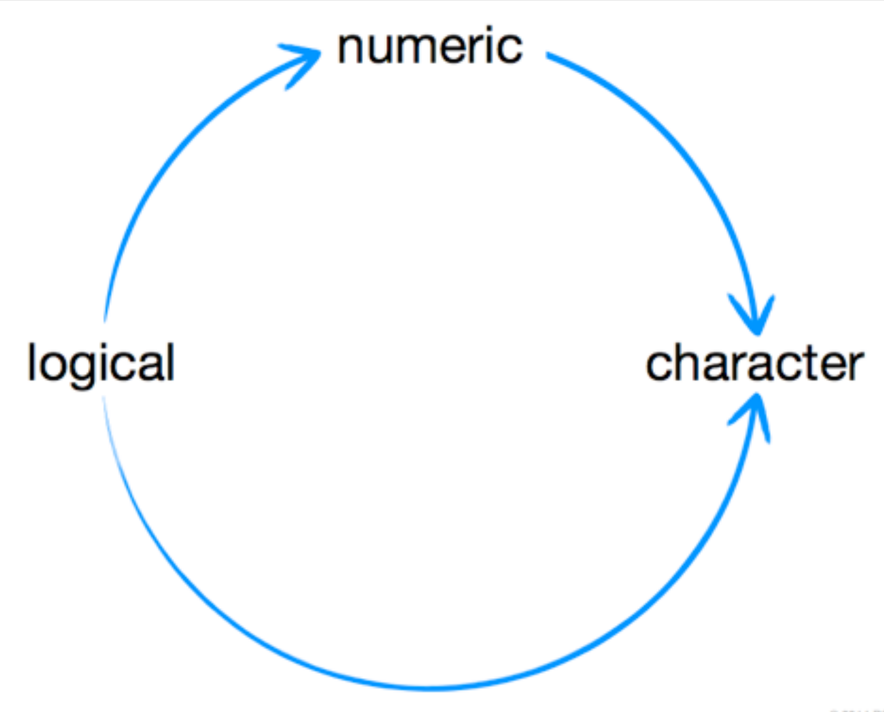


I'm going to give you a "quiz", and you might want to create your own chunk to try out some code. Use the Insert button to insert one



Quiz

What type of data will result?



c(5, "two")

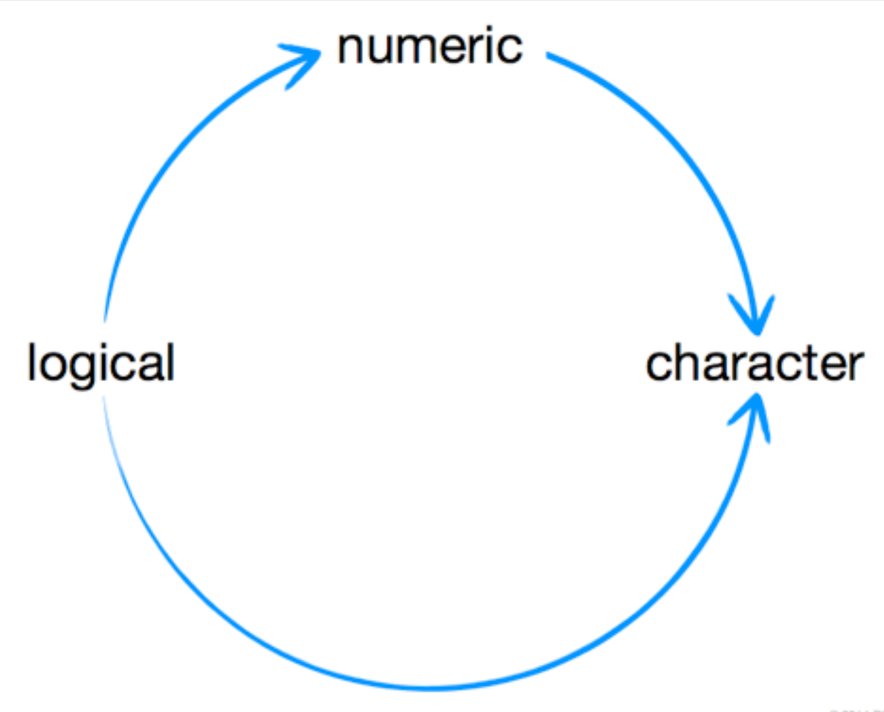
c(TRUE, "a")

c(1, "TRUE")

TRUE + 5

Quiz

What type of data will result?



c(5, "two")
character

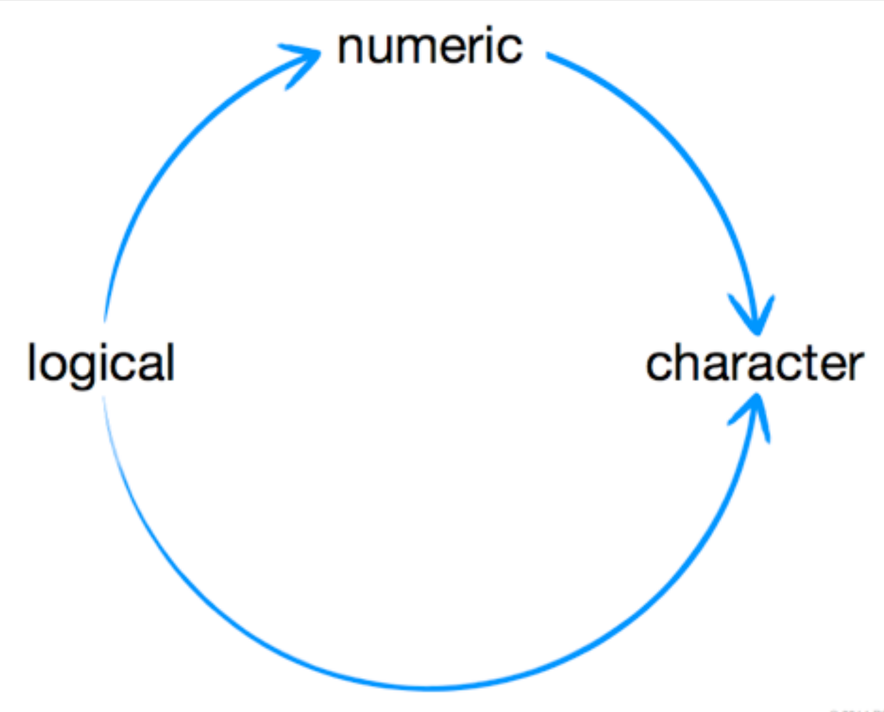
c(TRUE, "a")

c(1, "TRUE")

TRUE + 5

Quiz

What type of data will result?



c(5, "two")
character

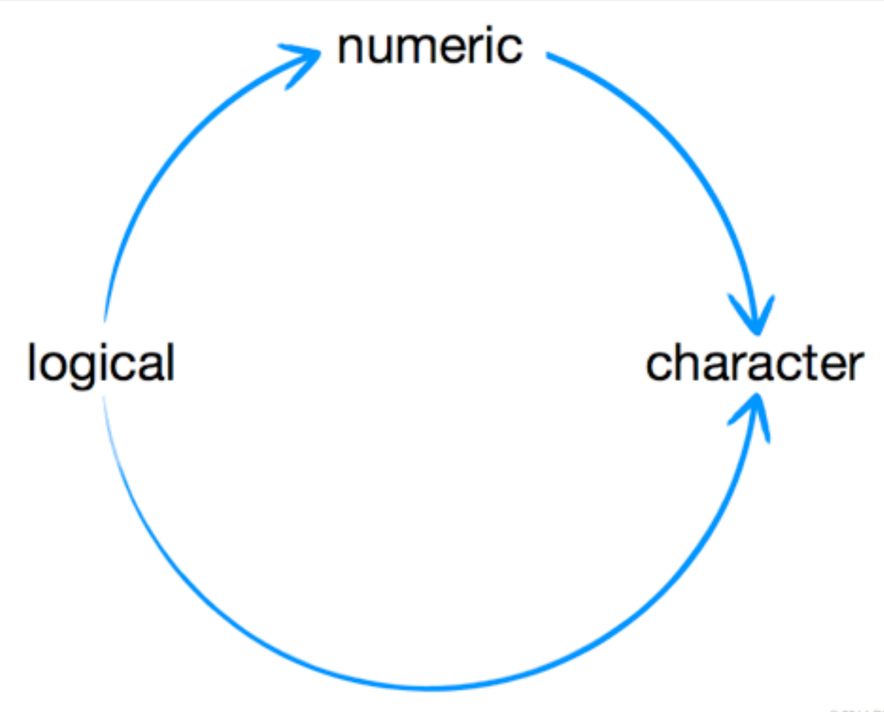
c(TRUE, "a")
character

c(1, "TRUE")

TRUE + 5

Quiz

What type of data will result?



c(5, "two")
character

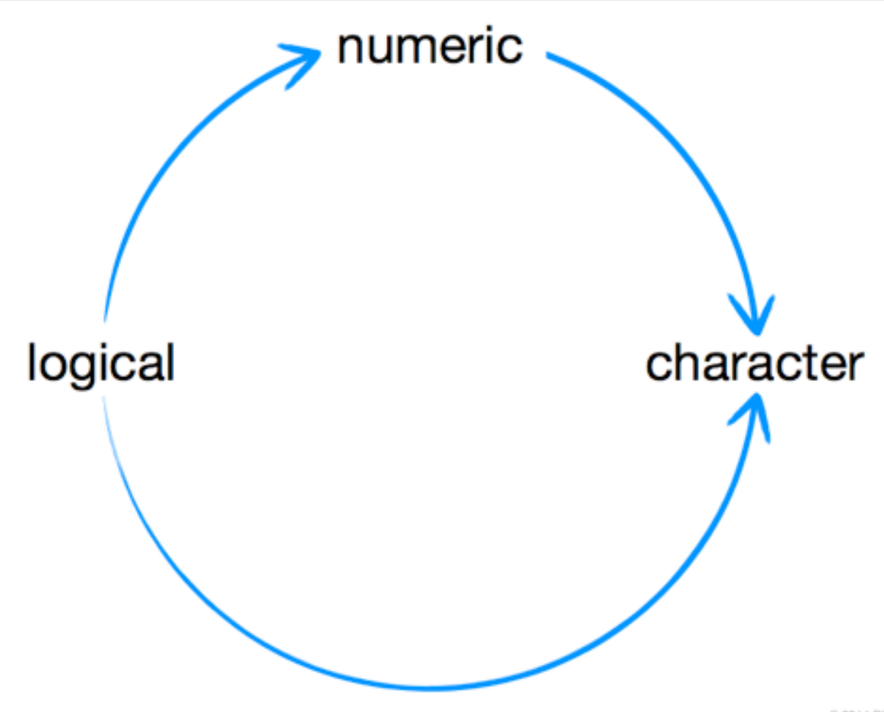
c(TRUE, "a")
character

c(1, "TRUE")
character

TRUE + 5

Quiz

What type of data will result?



c(5, "two")
character

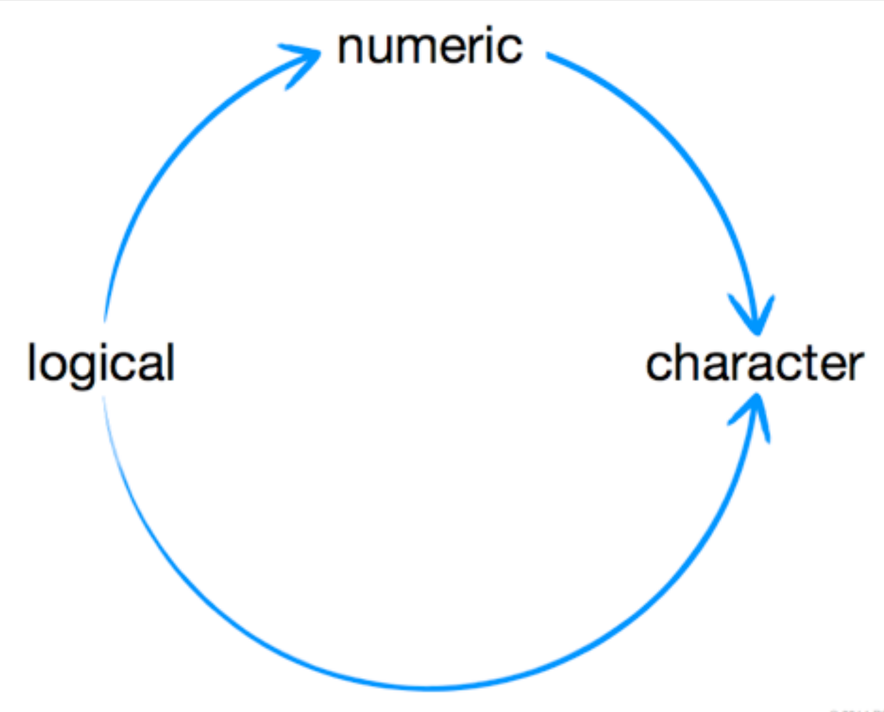
c(TRUE, "a")
character

c(1, "TRUE")
character

TRUE + 5

Quiz

What type of data will result?



c(5, "two")
character

c(TRUE, "a")
character

c(1, "TRUE")
character

TRUE + 5
numeric

manual coercion

function	coerces data to
as.numeric	numeric
as.character	character
as.logical	logical
as.factor	factor

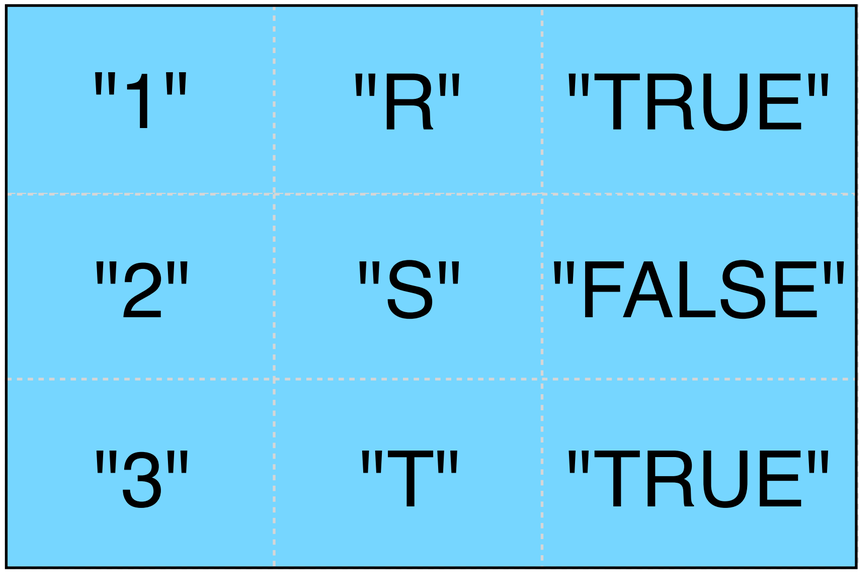
```
as.numeric("1")
as.character(TRUE)
```

Matrix

1	"R"	TRUE
2	"S"	FALSE
3	"T"	TRUE

?

Matrix



character

Matrix

"1"	"R"	"TRUE"
"2"	"S"	"FALSE"
"3"	"T"	"TRUE"

What if you want different data types in the same object?

Lists and data frames

lists and data frames

lists and *data frames* generalize vectors and matrices to allow multiple types of data

Lists

lists

A list is a one dimensional group of R objects.

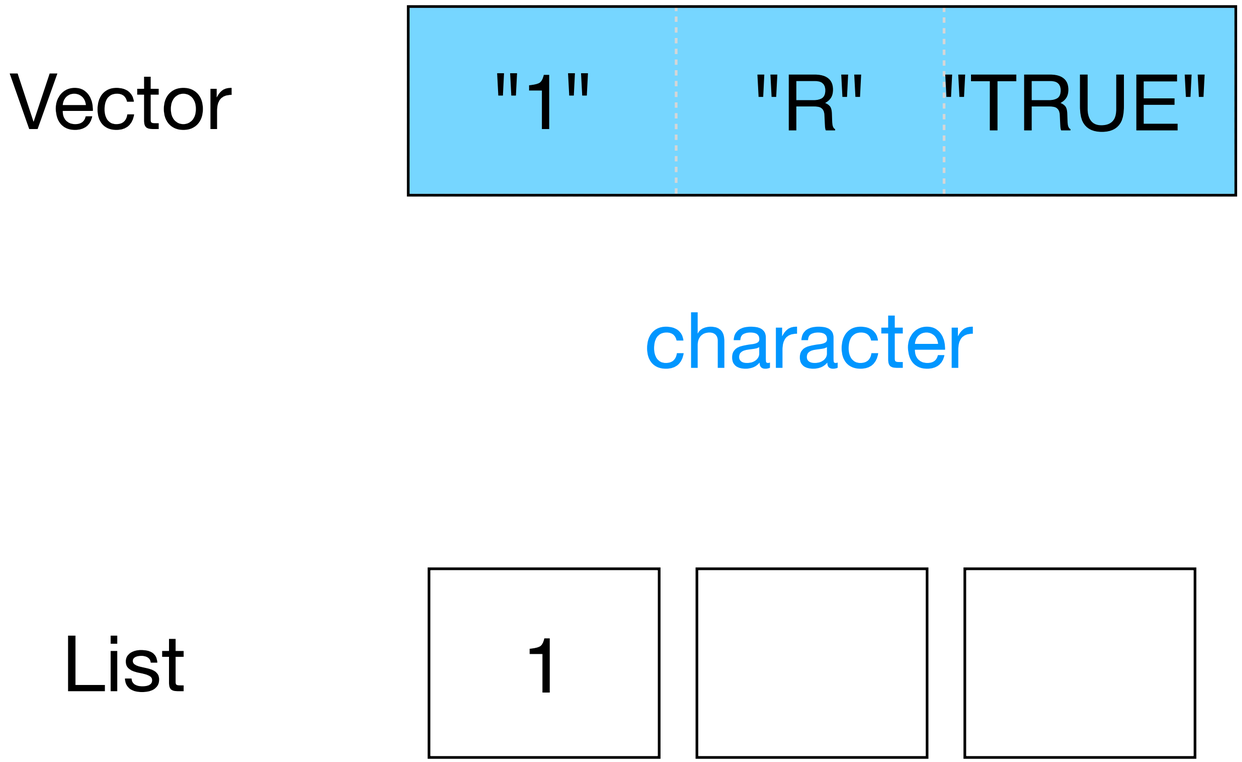
Create lists with list

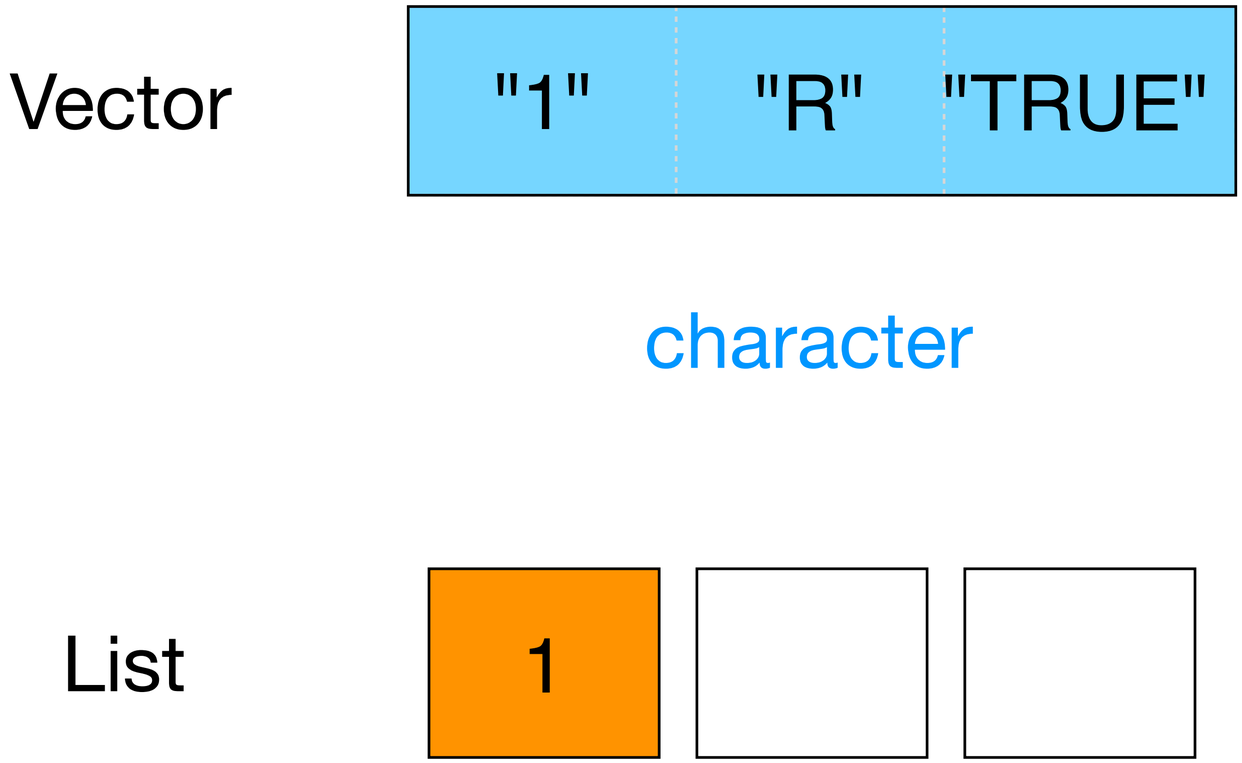
```
lst <- list(1, "R", TRUE)

class(lst)

# "list"
```

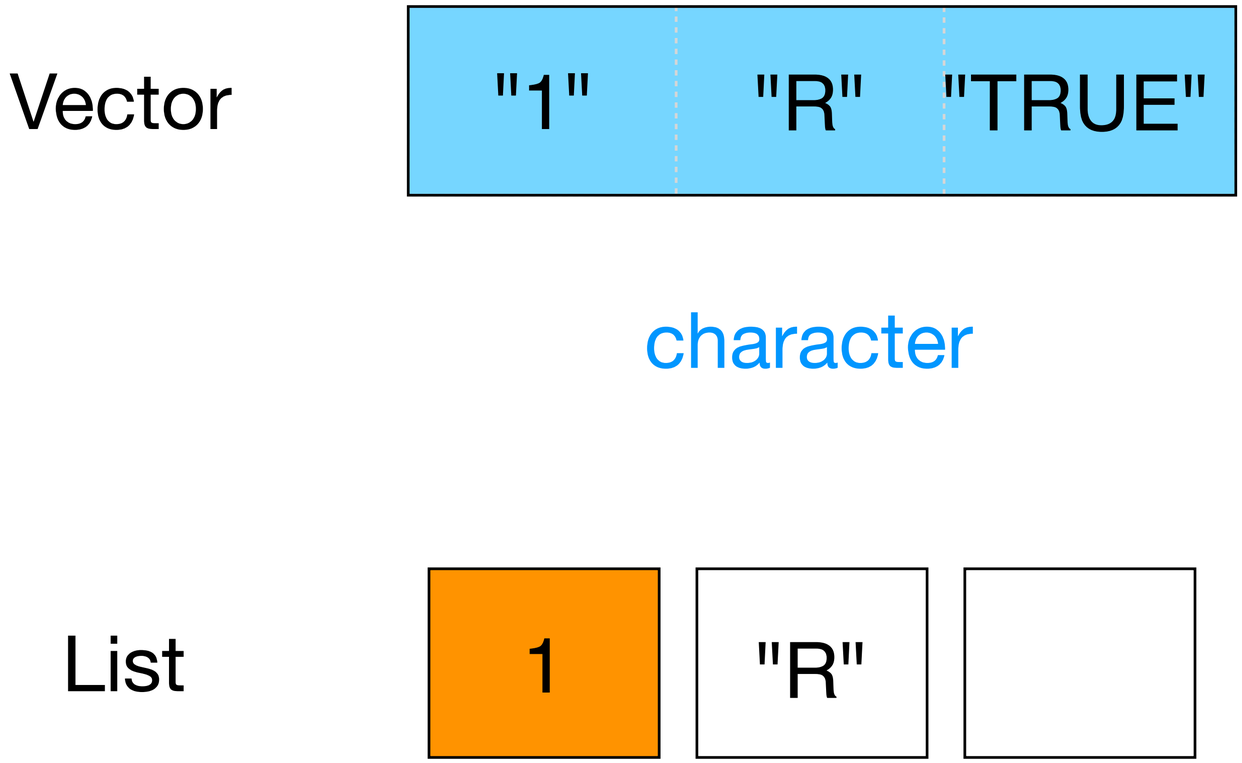
Vector

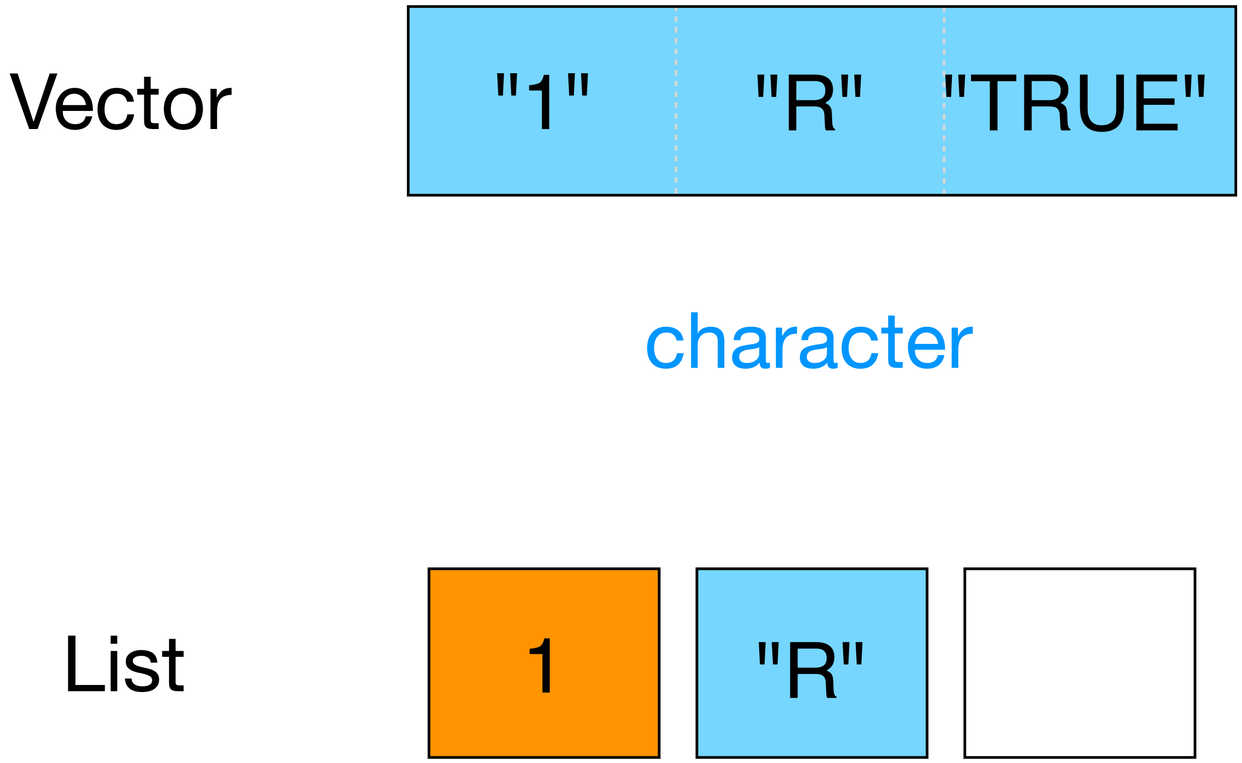
character

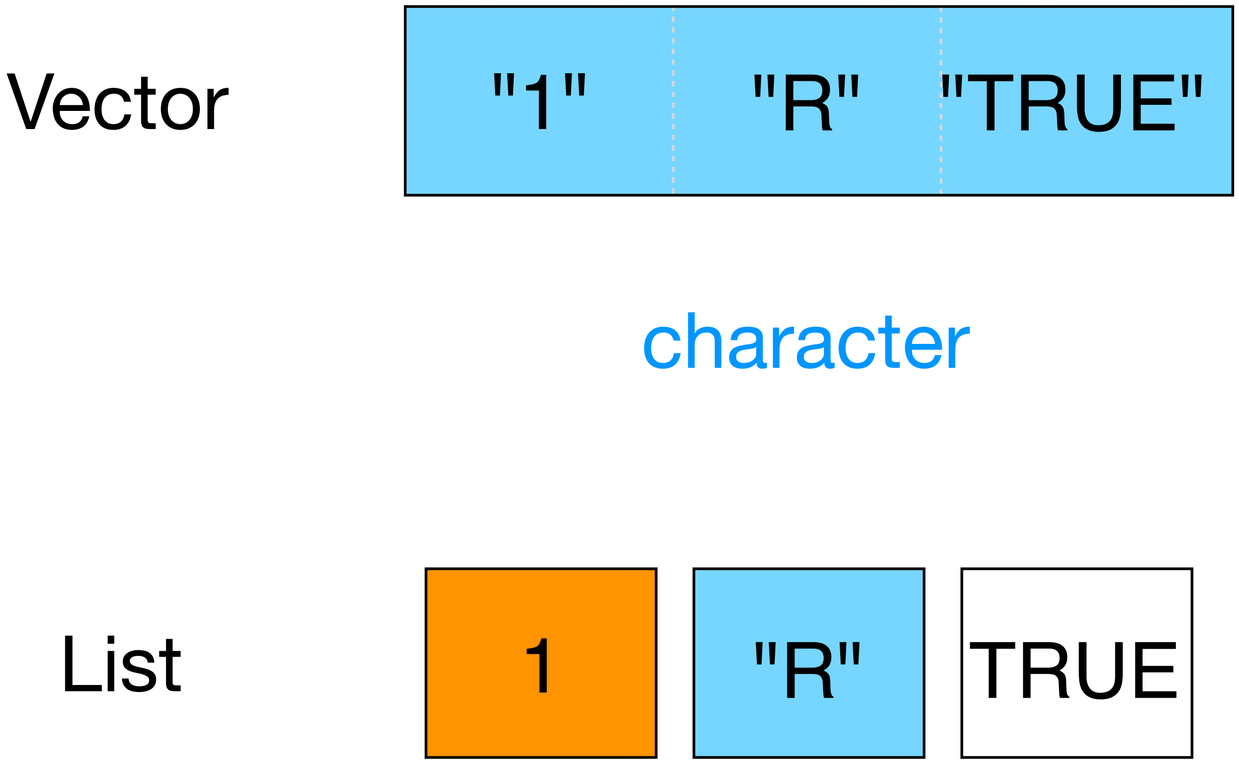
List

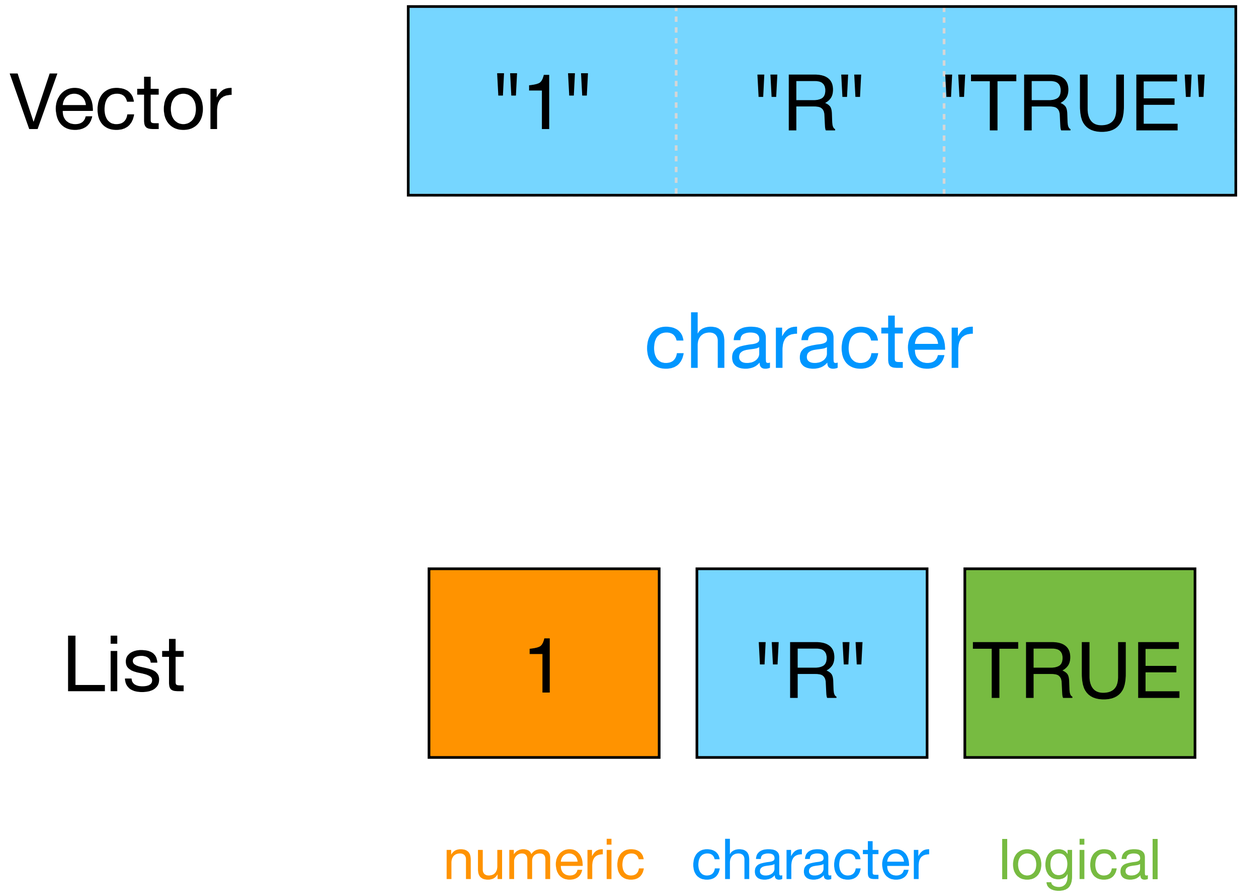

Vector
"1"
"R"
"TRUE"
character
List
1


Vector
"1"
"R"
"TRUE"
character
List
1
numeric


Vector
"1"
"R"
"TRUE"
character
List
1
"R"
numeric

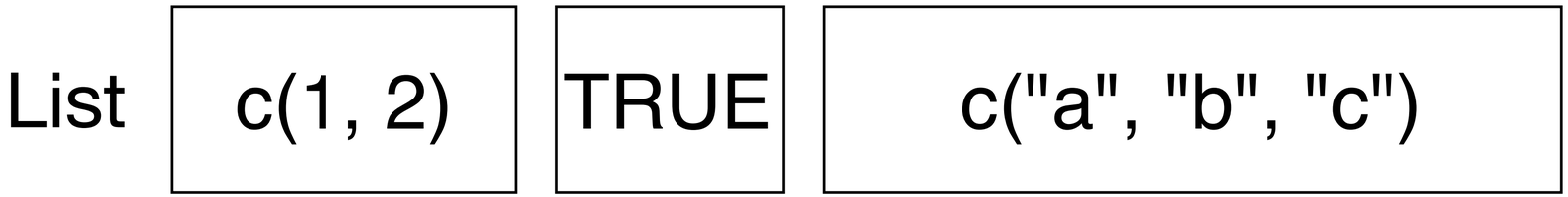

Vector
"1"
"R"
"TRUE"
character
List
1
"R"
numeric
character


Vector
"1"
"R"
"TRUE"
character
List
1
"R"
TRUE
numeric
character

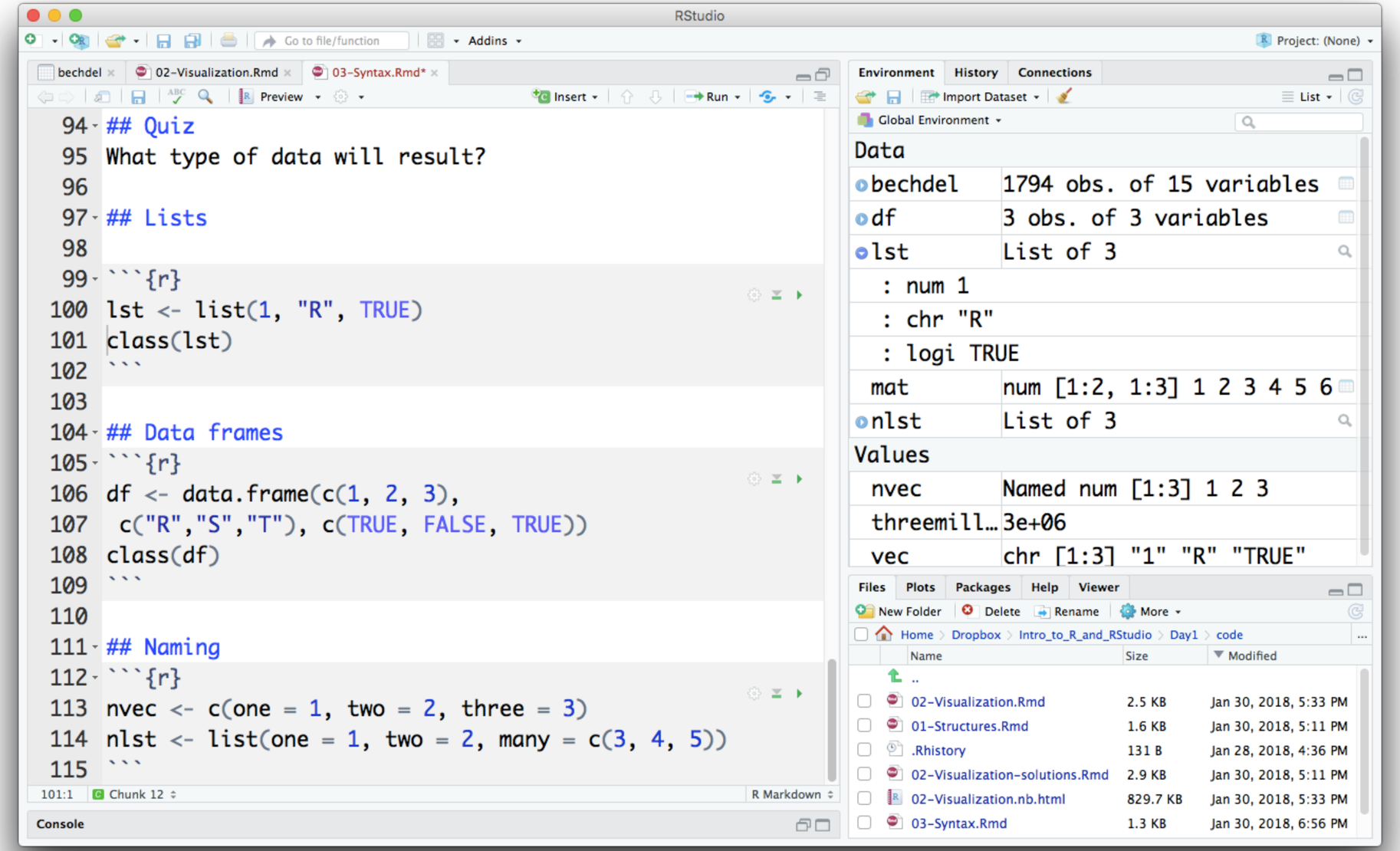

Vector
"1"
"R"
"TRUE"
character
List
1
"R"
TRUE
numeric
character
logical

The elements of a list can be anything. Even vectors or other lists.

```
list(c(1, 2), TRUE, c("a", "b", "c"))
```



List viewer in RStudio



Data frames

data frame

A data frame is a two dimensional group of R objects.

Each column in a data frame can be a different type

```
df <- data.frame(c(1, 2, 3),
 c("R","S","T"), c(TRUE, FALSE, TRUE))
class(df)
# "data.frame"
```

Your turn

We've already seen a data frame today. What was it called? What kinds of data were in it?

Matrix

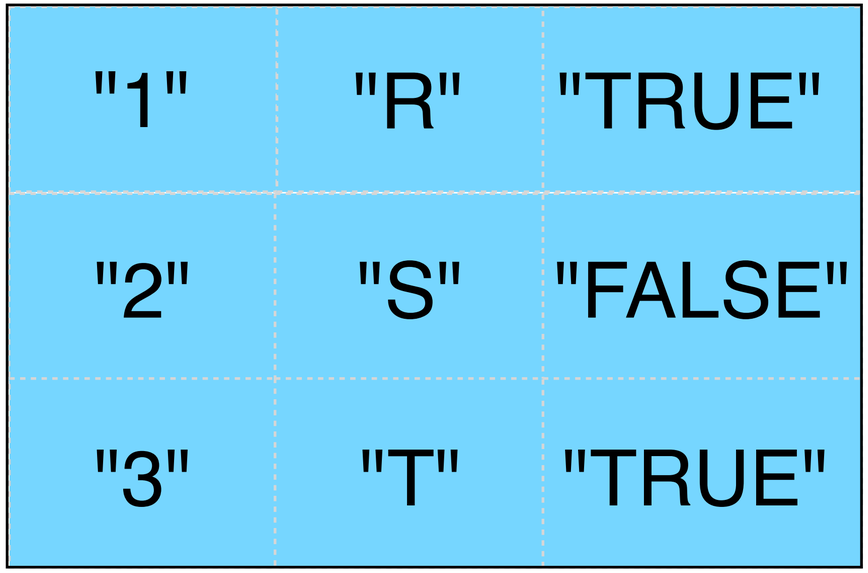
"1"	"R"	"TRUE"
"2"	"S"	"FALSE"
"3"	"T"	"TRUE"

character

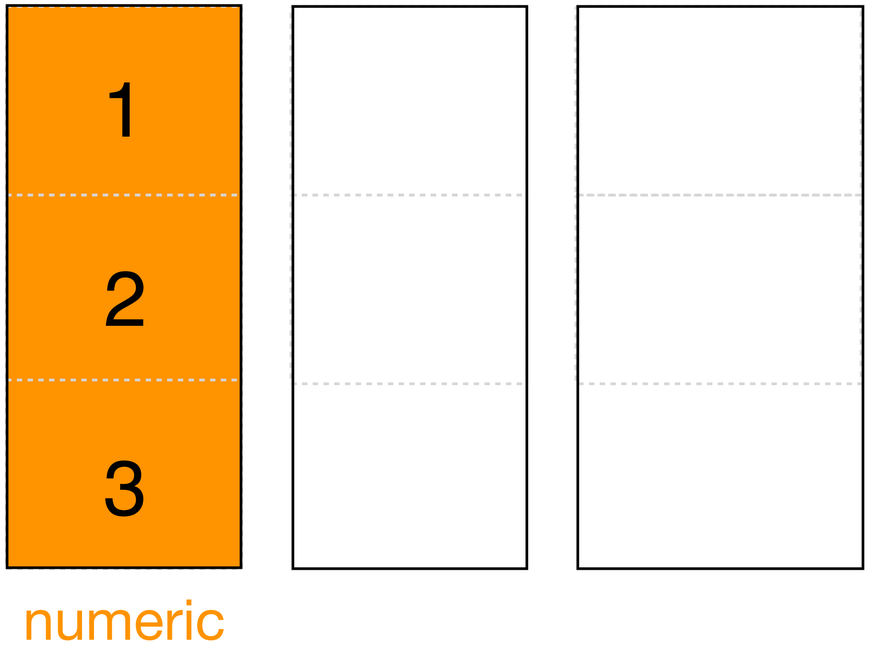
data frame

1		
2		
3		

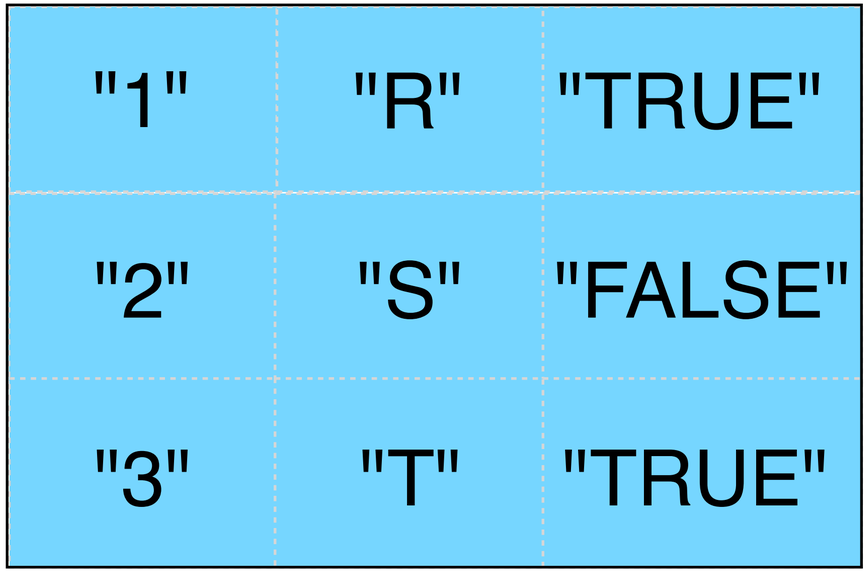
Matrix



data frame

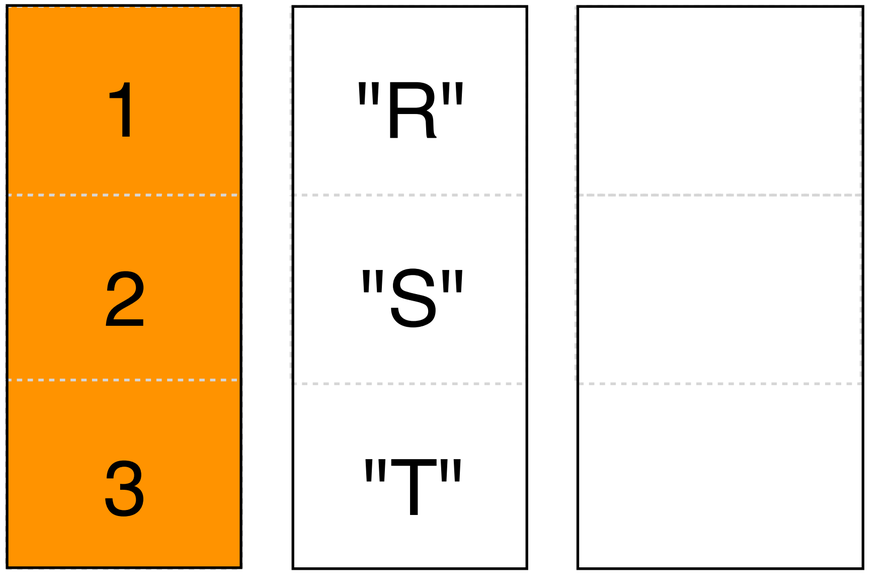


Matrix



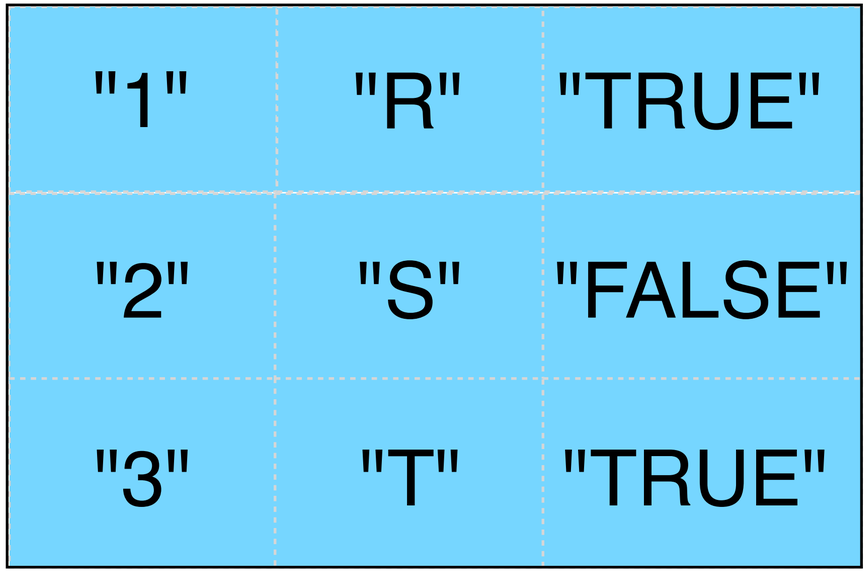
character

data frame

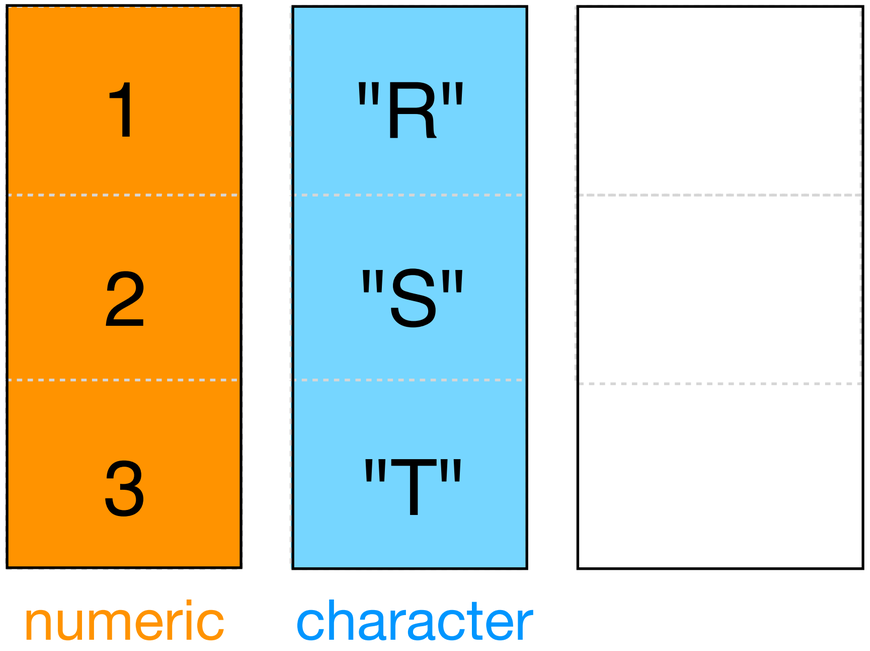


numeric

Matrix



data frame



Matrix

"1"	"R"	"TRUE"
"2"	"S"	"FALSE"
"3"	"T"	"TRUE"

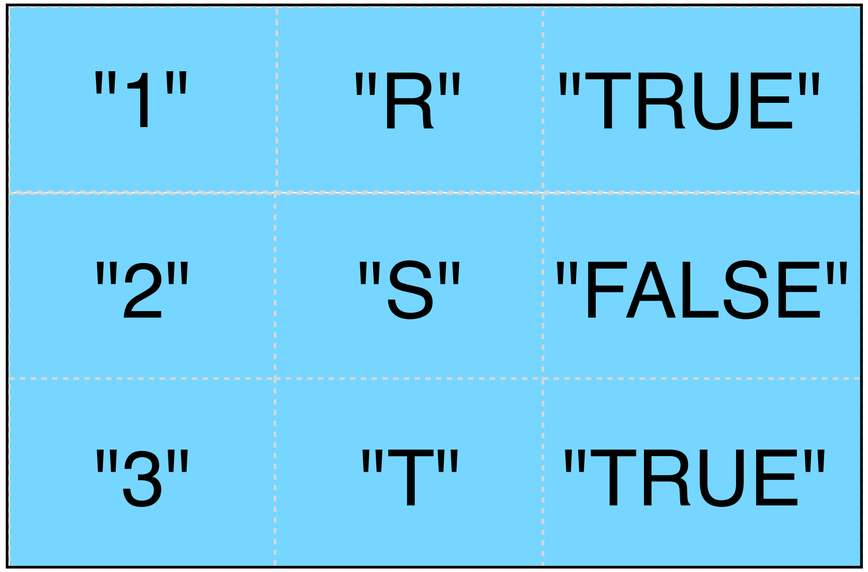
character

data frame

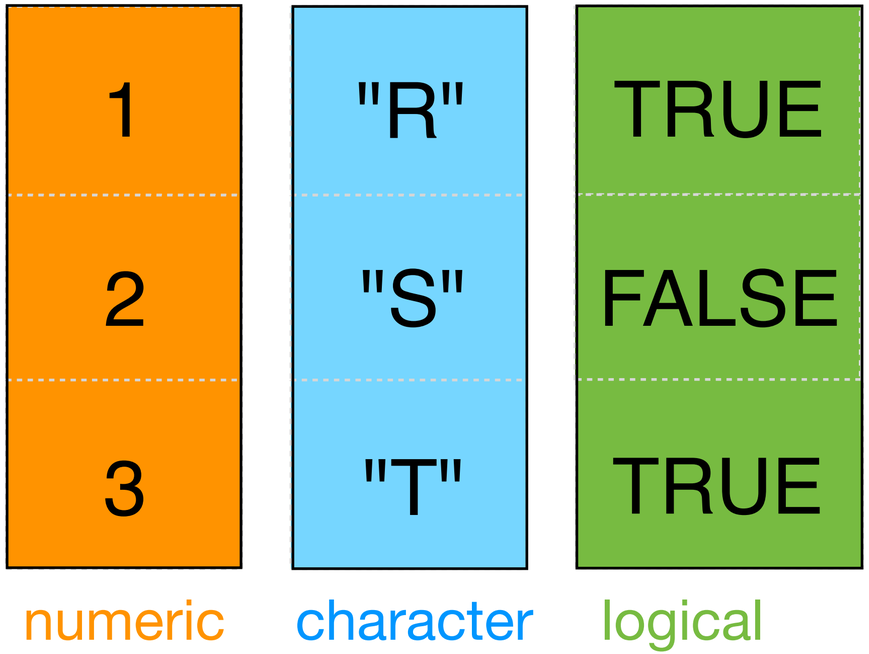
1	"R"	TRUE
2	"S"	FALSE
3	"T"	TRUE

numeric character

Matrix



data frame



names

You can name the elements of a vector, list, or data frame when you create them.

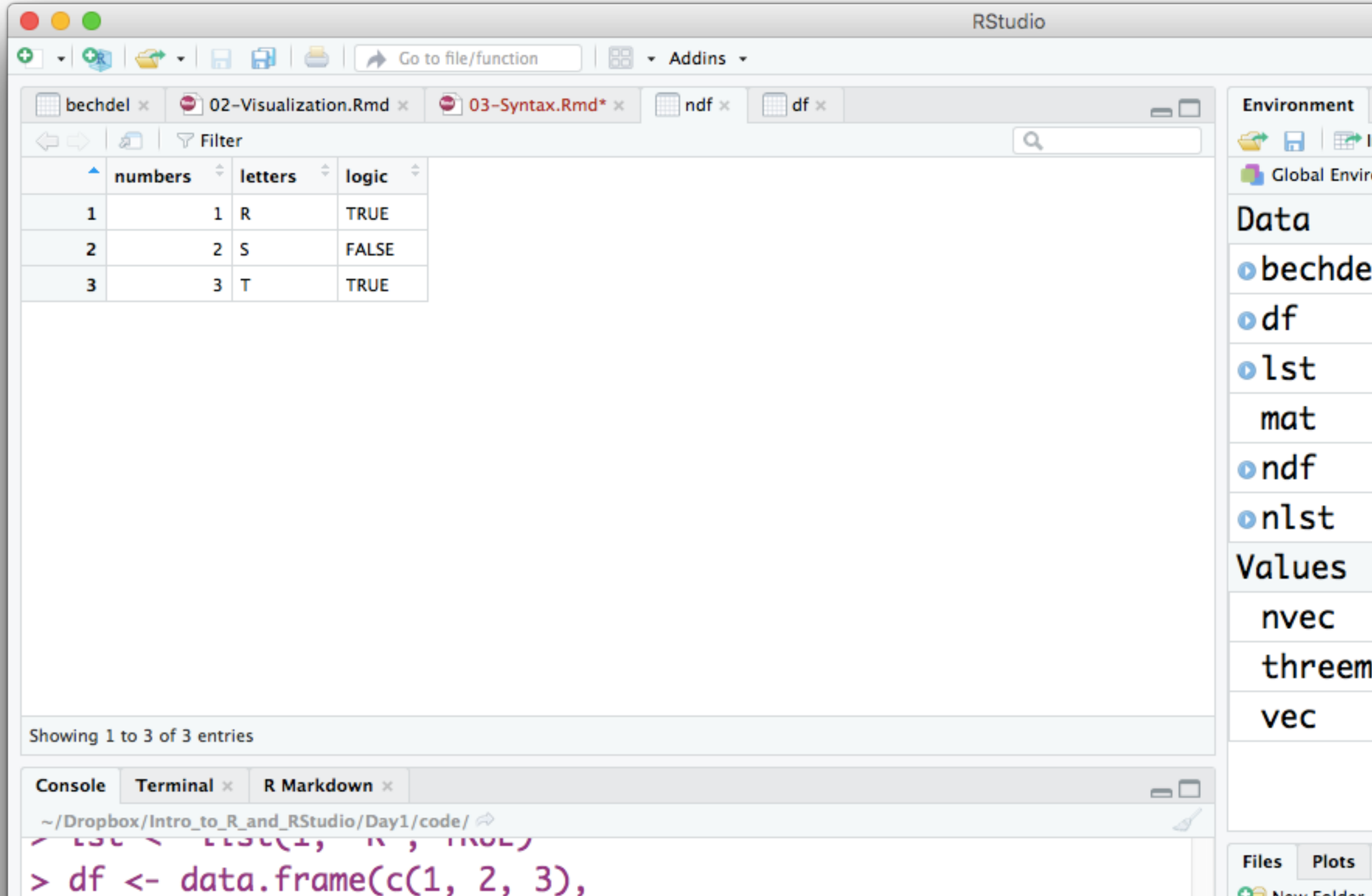
```
nvec <- c(one = 1, two = 2, three = 3)

nvec
# one  two three
#   1    2     3
```

```
nlst <- list(one = 1, two = 2,
             many = c(3, 4, 5))

nlst
# $one
# [1] 1
#
# $two
# [1] 2
#
# $many
# [1] 3 4 5
```

```
ndf <- data.frame(numbers = c(1, 2, 3),
                  letters = c("R","S","T"),
                  logic = c(TRUE, FALSE, TRUE))

ndf
#   numbers letters logic
# 1       1       R  TRUE
# 2       2       S FALSE
# 3       3       T  TRUE
```

Your turn

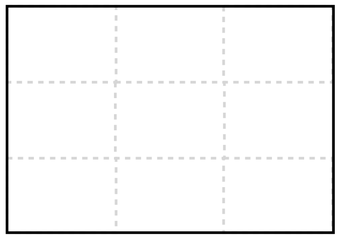
Use the RStudio data preview to compare `df` and `ndf`

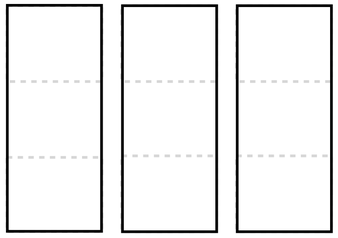

RStudio
Go to file/function
Addins
bechdel
02-Visualization.Rmd
03-Syntax.Rmd*
ndf
df
Filter
numbers
letters
logic
1 1 R TRUE
2 2 S FALSE
3 3 T TRUE
Showing 1 to 3 of 3 entries
Environment
Global Envir
Data
bechde
df
lst
mat
ndf
nlst
Values
nvec
threem
vec
Console
Terminal
R Markdown
~/Dropbox/Intro_to_R_and_RStudio/Day1/code/
> df <- data.frame(c(1, 2, 3),
Files
Plots

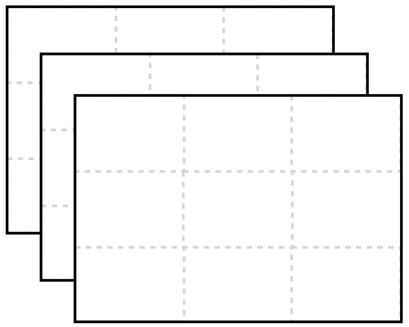
You can also see the names with names

```
names(ndf)
# [1] "numbers" "letters" "logic"

names(nvec)
# [1] "one"   "two"   "three"
```

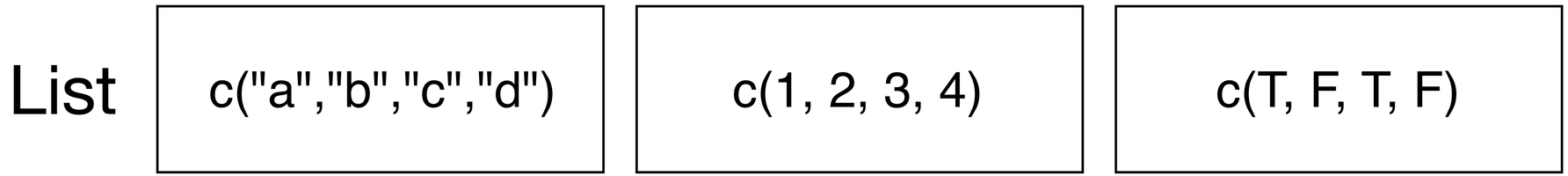
	single type	multiple types
1D	Vector	List
2D	Matrix	Data frame
nD	Array	



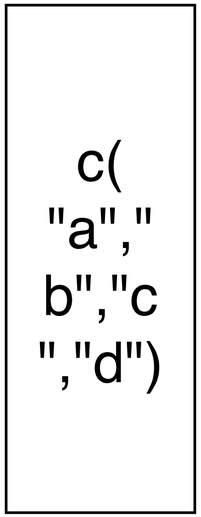


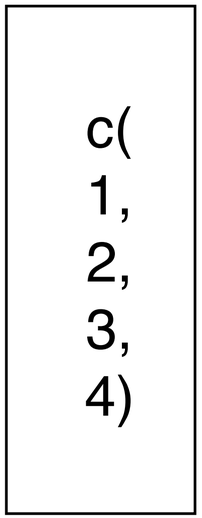


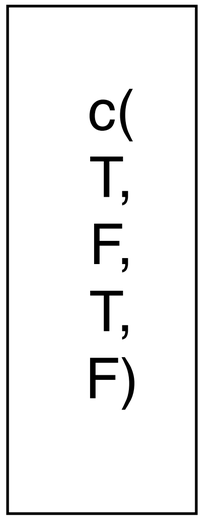
How R makes a data frame



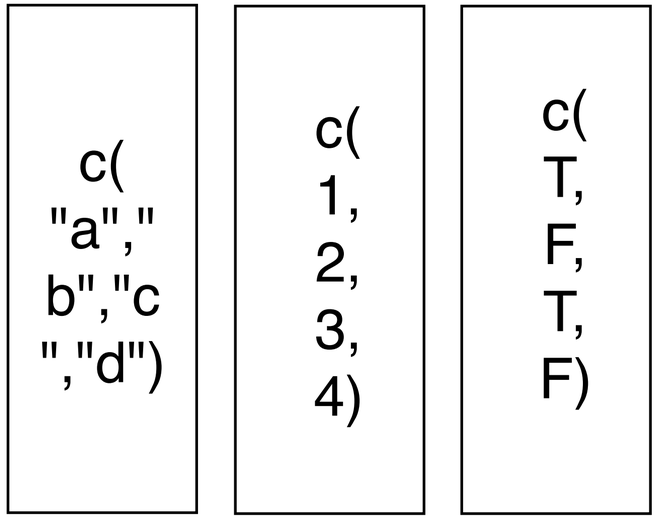
List





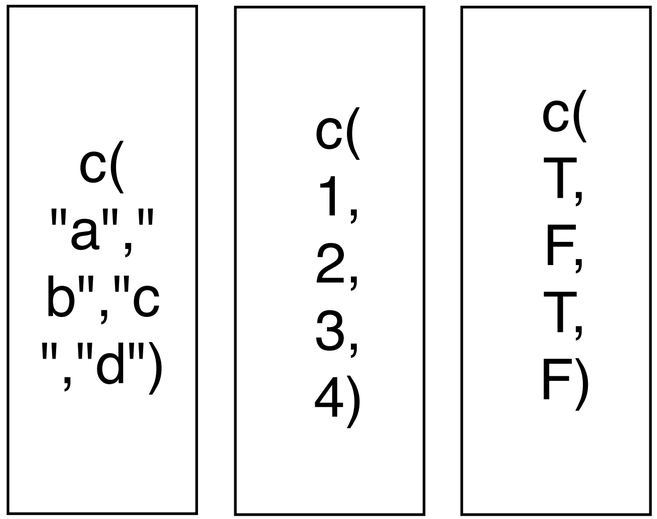


List



~~List~~

data frame



helper functions for data structures

	create	change to	check	get names	get dimensions
vector	c, vector	as.vector	is.vector	names	length
matrix	matrix	as.matrix	is.matrix	rownames, colnames	dim, nrow, ncol
array	array	as.array	is.array	dimnames	dim
list	list	as.list	is.list	names	length
data frame	data.frame	as.data.frame	is.data.frame	names	dim, nrow, ncol

Syntax

Syntax is the set of rules that govern what code works and doesn't work in a programming language. Most programming languages offer one standardized syntax, but R allows package developers to specify their own syntax. As a result, there is a large variety of (equally valid) R syntaxes.

R Syntax Comparison : : **CHEAT SHEET**

Dollar sign syntax

`goal(data$x, data$y)`

SUMMARY STATISTICS:

one continuous variable:
```
mean(mtcars$mpg)
```

one categorical variable:
```
table(mtcars$cyl)
```

two categorical variables:
```
table(mtcars$cyl, mtcars$am)
```

one continuous, one categorical:
```
mean(mtcars$mpg[mtcars$cyl==4])
mean(mtcars$mpg[mtcars$cyl==6])
mean(mtcars$mpg[mtcars$cyl==8])
```

PLOTTING:

one continuous variable:
```
hist(mtcars$disp)

boxplot(mtcars$disp)
```

one categorical variable:
```
barplot(table(mtcars$cyl))
```

two continuous variables:
```
plot(mtcars$disp, mtcars$mpg)
```

two categorical variables:
```
mosaicplot(table(mtcars$am, mtcars$cyl))
```

one continuous, one categorical:
```
histogram(mtcars$disp[mtcars$cyl==4])
histogram(mtcars$disp[mtcars$cyl==6])
histogram(mtcars$disp[mtcars$cyl==8])

boxplot(mtcars$disp[mtcars$cyl==4])
boxplot(mtcars$disp[mtcars$cyl==6])
boxplot(mtcars$disp[mtcars$cyl==8])
```

WRANGLING:

subsetting:
```
mtcars[mtcars$mpg>30, ]
```

making a new variable:
```
mtcars$efficient[mtcars$mpg>30] <- TRUE
mtcars$efficient[mtcars$mpg<30] <- FALSE
```

Formula syntax

`goal(y~x|z, data=data, group=w)`

SUMMARY STATISTICS:

one continuous variable:
```
mosaic::mean(~mpg, data=mtcars)
```

one categorical variable:
```
mosaic::tally(~cyl, data=mtcars)
```

two categorical variables:
```
mosaic::tally(cyl~am, data=mtcars)
```

one continuous, one categorical:
```
mosaic::mean(mpg~cyl, data=mtcars)
```

tilde

PLOTTING:

one continuous variable:
```
lattice::histogram(~disp, data=mtcars)

lattice::bwplot(~disp, data=mtcars)
```

one categorical variable:
```
mosaic::bargraph(~cyl, data=mtcars)
```

two continuous variables:
```
lattice::xyplot(mpg~disp, data=mtcars)
```

two categorical variables:
```
mosaic::bargraph(~am, data=mtcars, group=cyl)
```

one continuous, one categorical:
```
lattice::histogram(~disp|cyl, data=mtcars)

lattice::bwplot(cyl~disp, data=mtcars)
```

The variety of R syntaxes give you many ways to "say" the same thing

read **across** the cheatsheet to see how different syntaxes approach the same problem

Tidyverse syntax

`data %>% goal(x)`

SUMMARY STATISTICS:

one continuous variable:
```
mtcars %>% dplyr::summarize(mean(mpg))
```

one categorical variable:
```
mtcars %>% dplyr::group_by(cyl) %>%
dplyr::summarize(n())
```

the pipe

two categorical variables:
```
mtcars %>% dplyr::group_by(cyl, am) %>%
dplyr::summarize(n())
```

one continuous, one categorical:
```
mtcars %>%  dplyr::group_by(cyl) %>%
   dplyr::summarize(mean(mpg))
```

PLOTTING:

one continuous variable:
```
ggplot2::qplot(x=mpg, data=mtcars, geom = "histogram")

ggplot2::qplot(y=disp, x=1, data=mtcars, geom="boxplot")
```

one categorical variable:
```
ggplot2::qplot(x=cyl, data=mtcars, geom="bar")
```

two continuous variables:
```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

two categorical variables:
```
ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") +
       facet_grid(.~am)
```

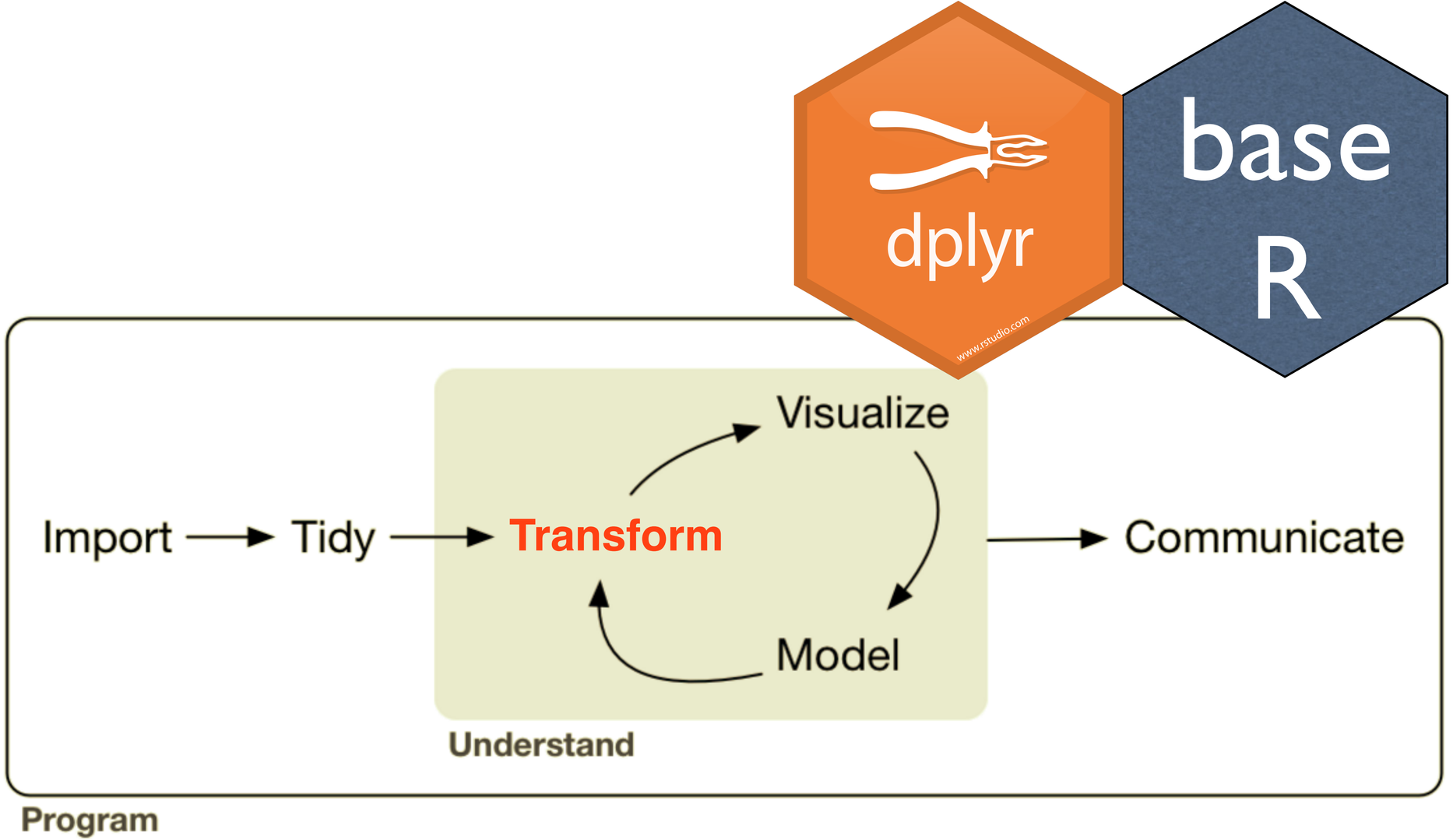
one continuous, one categorical:
```
ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") +
       facet_grid(.~cyl)

ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars,
              geom="boxplot")
```

WRANGLING:

subsetting:
```
mtcars %>% dplyr::filter(mpg>30)
```

making a new variable:
```
mtcars <- mtcars %>%
dplyr::mutate(efficient = if_else(mpg>30,  TRUE, FALSE))
```

Subsetting



From *R for Data Science* by Hadley Wickham and Garrett Grolemund.

Toy data

```
beatles <- data.frame(
  name = c("John", "Paul", "George", "Ringo"),
  birth = c(1940, 1942, 1943, 1940),
  instrument = c("guitar", "bass", "guitar", "drums")
)
```

First— the tidyverse way: dplyr

dplyr methods for isolating data

select() - extract **variables**

filter() - extract **cases**

arrange() - reorder **cases**

select()

Extract columns by name.

```
select(.data, …)
```

data frame to transform

name(s) of columns to extract
(or a select helper function)

select()

Extract columns by name.

```
select(beatles, name, birth)
```

name	birth	instrument
John	1940	guitar
Paul	1942	base
George	1943	guitar
Ringo	1940	drums

name	birth
John	1940
Paul	1942
George	1943
Ringo	1940

Your Turn

Alter the code to select just the **instrument** column:

```
select(beatles, name, birth)
```

select() helpers

: - Select range of columns

```
select(storms, storm:pressure)
```

- - Select every column but

```
select(storms, -c(storm, pressure))
```

starts_with() - Select columns that start with…

```
select(storms, starts_with("w"))
```

ends_with() - Select columns that end with…

```
select(storms, ends_with("e"))
```

select() helpers

contains() - Select columns whose names contain...

```
select(storms, contains("d"))
```

matches() - Select columns whose names match regular expression

```
select(storms, matches("^.{4}$"))
```

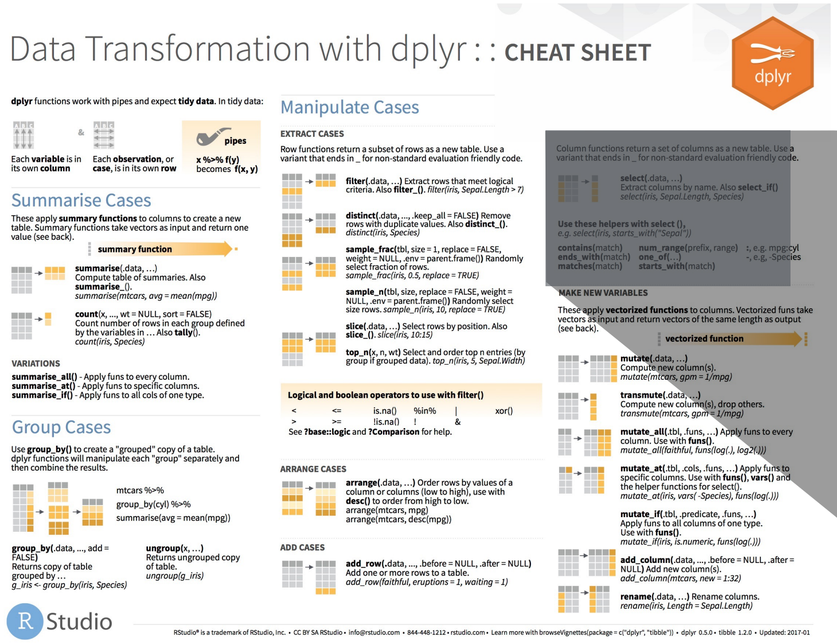
one_of() - Select columns whose names are one of a set

```
select(storms, one_of(c("storm", "storms", "Storm"))
```

num_range() - Select columns named in prefix, number style

```
select(storms, num_range("x", 1:5))
```

select() helpers



Data Transformation with dplyr : : **CHEAT SHEET**

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**. Each **observation**, or **case**, is in its own **row**.

pipes: **x %>% f(y)** becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …**)** Compute table of summaries. Also **summarise_()**.
summarise(mtcars, avg = mean(mpg))

count(x, …, wt = NULL, sort = FALSE**)** Count number of rows in each group defined by the variables in … Also **tally()**.
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE**)** Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …**)** Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

filter(.data, …**)** Extract rows that meet logical criteria. Also **filter_()**. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values. Also **distinct_()**. *distinct(iris, Species)*

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …**)** Select rows by position. Also **slice_()**. *slice(iris, 10:15)*

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL**)** Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, …**)** Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	-, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …**)** Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …**)** Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …**)** Apply funs to every column. Use with **funs()**.
mutate_all(faithful, funs(log(.), log2(.)))

mutate_at(.tbl, .cols, .funs, …**)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

mutate_if(.tbl, .predicate, .funs, …**)** Apply funs to all columns of one type. Use with **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL**)** Add new column(s).
add_column(mtcars, new = 1:32)

rename(.data, …**)** Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

select(.data, …**)**
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	:, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	-, e.g, -Species
matches(match)	**starts_with**(match)	

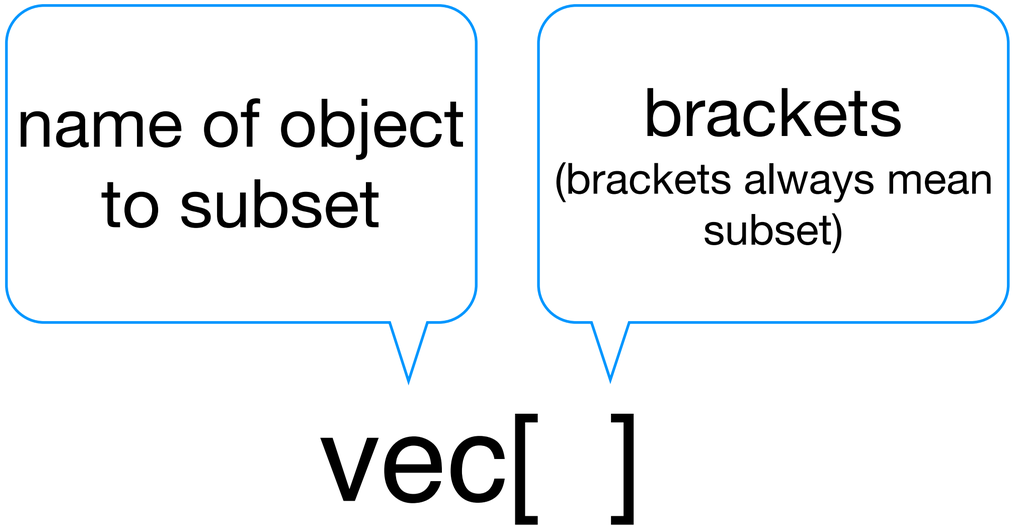
Now, the base R way: brackets and dollar signs

Base R bracket subset notation

in base R, you use the same syntax to

extract **variables**

extract **cases**



Subset notation

name of object to subset

vec

Subset notation

name of object to subset

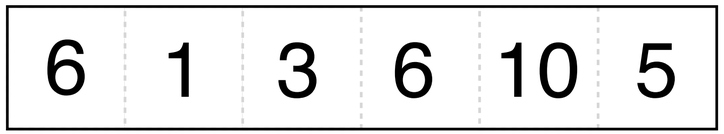
brackets
(brackets always mean subset)

vec[?]

an index
(that tells R which elements to include)

Each dimension needs its own index!

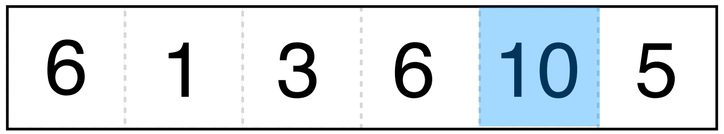
vec[?]



base R

Each dimension needs its own index!

vec[?]



base R

Each dimension needs its own index!

vec[?]

beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Each dimension needs its own index!

vec[?]

beatles[?,?]

which **rows** to include

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Each dimension needs its own index!

vec[?]

beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

which **rows** to include

which **columns** to include

Each dimension needs its own index!

vec[?]

beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

which **rows** to include

separate dimensions with a **comma**

which **columns** to include

base R

Each dimension needs its own index!

vec[?]

beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

What goes in the indexes?

Four ways to subset

1. Integers
2. Blank spaces
3. Names
4. Logical vectors (TRUE and FALSE)

Integers (positive)

Positive integers behave just like *ij* notation in linear algebra

beatles[?,?]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

Positive integers behave just like *ij* notation in linear algebra

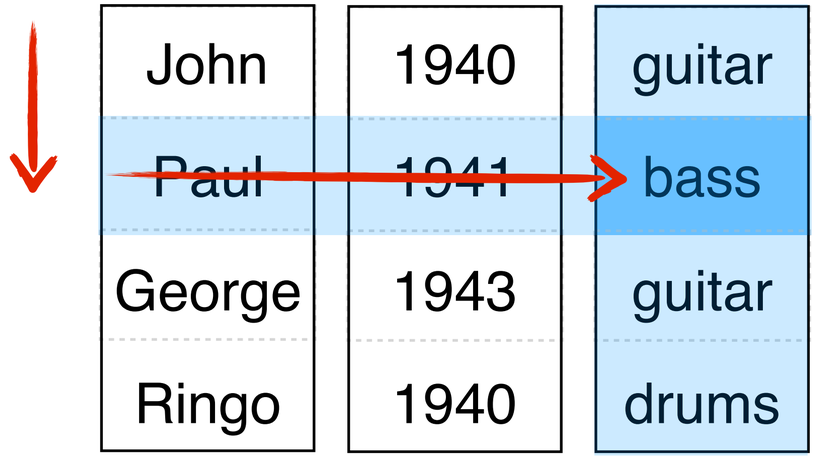
```
beatles[2,?]
```

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Integers (positive)

Positive integers behave just like *ij* notation in linear algebra

beatles[2,3]



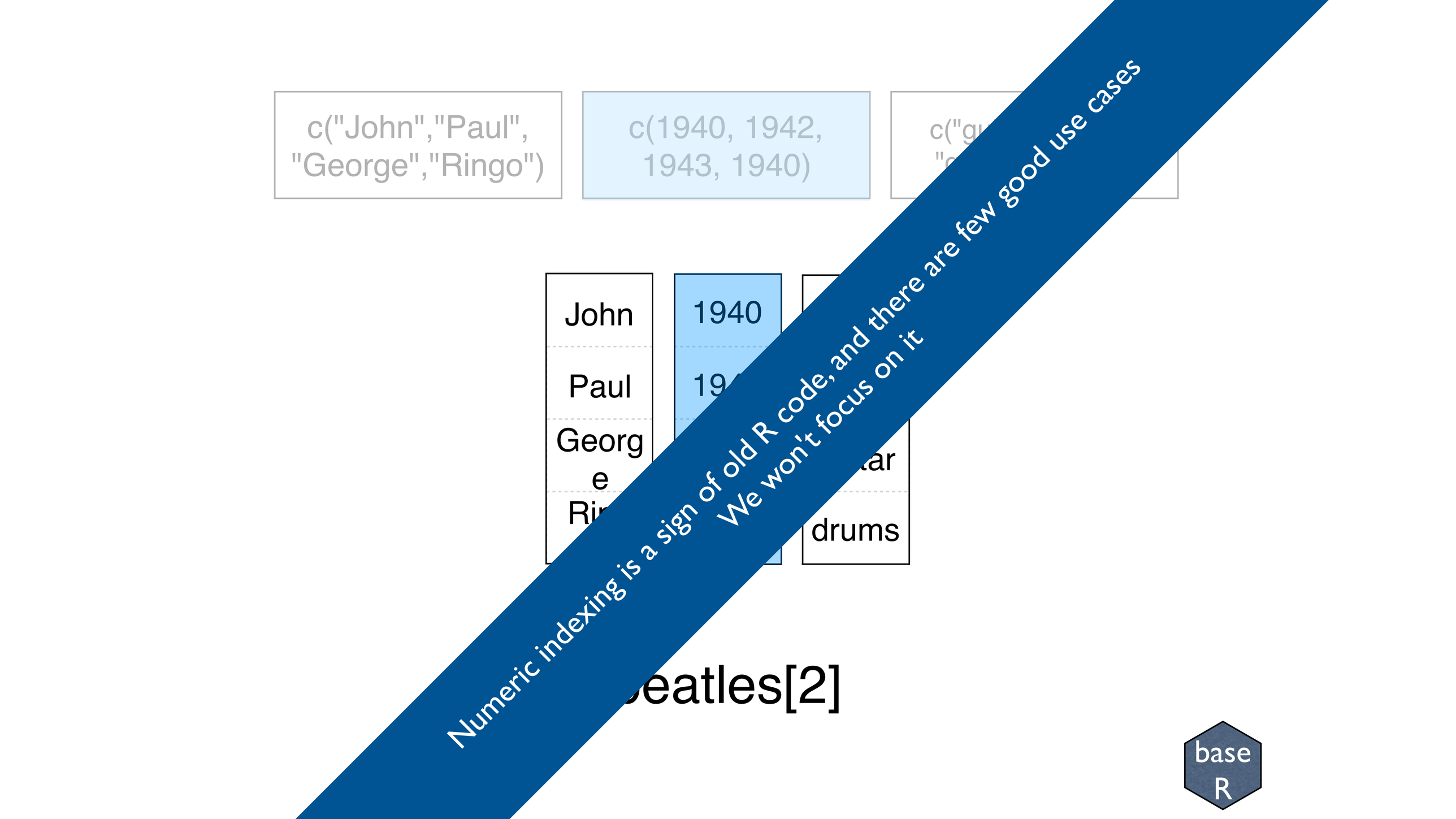
base R

Integers (positive)

Positive integers behave just like *ij* notation in linear algebra

beatles[2,3]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums


c("John","Paul",
"George","Ringo")
c(1940, 1942,
1943, 1940)
John
1940
Paul
Georg
e
drums
Numeric indexing is a sign of old R code, and there are few good use cases
We won't focus on it
eatles[2]
base
R

Names

If your object has names, you can ask for elements or columns back by name.

```
beatles[ ,"birth"]
```

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Names

If your object has names, you can ask for elements or columns back by name.

```
beatles[ ,c("name","birth")]
```

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Your Turn

Modify the code below to select just the `instrument` column

```
beatles[ ,"birth"]
```

$

The most common syntax for subsetting lists and data frames

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

beatles$birth

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

beatles$birth

name of
data frame

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

beatles$birth

name of data frame

$

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

beatles$birth

name of data frame

$

name of column
(no quotes)

base R

name	birth	instrument
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

c(1940, 1941, 1943, 1940)

beatles$birth

name of data frame

$

name of column
(no quotes)

Your Turn

Modify the code below to select just the `instrument` column

```
beatles$birth
```

Logical comparisons

Logical comparisons

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Logical comparisons

What will these return?

```
1 < 3
1 > 3
c(1, 2, 3, 4, 5) > 3
```

%in%

```
# What does this do?
1 %in% c(1, 2, 3, 4)
1 %in% c(2, 3, 4)
c(3,4,5,6) %in% c(2, 3, 4)
```

%in%

%in% tests whether the object on the left is a member of the group on the right.

```
1 %in% c(1, 2, 3, 4)
# TRUE
1 %in% c(2, 3, 4)
# FALSE
c(3,4,5,6) %in% c(2, 3, 4)
# TRUE TRUE FALSE FALSE
```

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3		
x < 3		
x <= 3		
x == 3		
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3		
x <= 3		
x == 3		
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3		
x == 3		
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3		
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3	`c(F, F, T, F, F)`	equal to
x != 3		
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3	`c(F, F, T, F, F)`	equal to
x != 3	`c(T, T, F, T, T)`	not equal to
x = 3		

Your turn

x <- c(1, 2, 3, 4, 5)

Operator	Result	Comparison
x > 3	`c(F, F, F, T, T)`	greater than
x >= 3	`c(F, F, T, T, T)`	greater than or equal to
x < 3	`c(T, T, F, F, F)`	less than
x <= 3	`c(T, T, T, F, F)`	less than or equal to
x == 3	`c(F, F, T, F, F)`	equal to
x != 3	`c(T, T, F, T, T)`	not equal to
x = 3		same as <-

Boolean operators

Boolean operators

?base::Logic

`a & b`	and
`a | b`	or
`xor(a,b)`	exactly or
`!a`	not

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$$x > 2 \ \& \ x < 9$$

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$x > 2$ & $x < 9$

TRUE &

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$$x > 2 \;\&\; x < 9$$

TRUE & TRUE

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$x > 2 \ \& \ x < 9$

TRUE & TRUE

TRUE

&

Are both condition 1 **and** condition 2 true?

expression	outcome
TRUE & TRUE	TRUE
TRUE & FALSE	FALSE
FALSE & TRUE	FALSE
FALSE & FALSE	FALSE

|

Is either condition 1 **or** condition 2 true?

expression	outcome
TRUE | TRUE	TRUE
TRUE | FALSE	TRUE
FALSE | TRUE	TRUE
FALSE | FALSE	FALSE

xor

Is either condition 1 **or** condition 2 true, **but not both**?

expression	outcome
xor(TRUE, TRUE)	FALSE
xor(TRUE, FALSE)	TRUE
xor(FALSE, TRUE)	TRUE
xor(FALSE, FALSE)	FALSE

!

Negation

expression	outcome
!(TRUE)	FALSE
!(FALSE)	TRUE

Boolean operators

Boolean operators

?base::Logic

`a & b`	and
`a | b`	or
`xor(a,b)`	exactly or
`!a`	not

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$$x > 2 \ \& \ x < 9$$

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$x > 2$ & $x < 9$

TRUE &

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$$x > 2 \;\&\; x < 9$$

TRUE & TRUE

Boolean operators

You can combine logical tests with &, |, xor, !, any, and all

$x > 2 \ \& \ x < 9$

TRUE & TRUE

TRUE

&

Are both condition 1 **and** condition 2 true?

expression	outcome
TRUE & TRUE	TRUE
TRUE & FALSE	FALSE
FALSE & TRUE	FALSE
FALSE & FALSE	FALSE

|

Is either condition 1 **or** condition 2 true?

expression	outcome
TRUE | TRUE	TRUE
TRUE | FALSE	TRUE
FALSE | TRUE	TRUE
FALSE | FALSE	FALSE

xor

Is either condition 1 **or** condition 2 true, **but not both**?

expression	outcome
xor(TRUE, TRUE)	FALSE
xor(TRUE, FALSE)	TRUE
xor(FALSE, TRUE)	TRUE
xor(FALSE, FALSE)	FALSE

!

Negation

expression	outcome
!(TRUE)	FALSE
!(FALSE)	TRUE

filter()

Extract rows that meet *every* logical criteria.

```
filter(beatles, birth==1940 & instrument == "guitar")
```

name	birth	instrument
John	1940	guitar
Paul	1942	base
George	1943	guitar
Ringo	1940	drums

→

John	1940	guitar

Your Turn

Modify the code below to filter out the rows for which birth is 1943 or instrument is drums

```
filter(beatles, birth==1940 & instrument == "guitar")
```

Base R

Logical

You can subset with a logical vector of the same length as the dimension you are subsetting. Each element that corresponds to a TRUE will be returned.

```
beatles[c(FALSE,TRUE,TRUE,FALSE), ]
```

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Logical

You can subset with a logical vector of the same length as the dimension you are subsetting. Each element that corresponds to a TRUE will be returned.

beatles[**c(FALSE,TRUE,TRUE,FALSE)**,]

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Logical

You can provide a statement that **evaluates** to a logical to get something similar to a `dplyr` `filter()` statement.

```
beatles[beatles$birth == 1940, ]
beatles[c(TRUE, FALSE, FALSE, TRUE), ]
```

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

Your Turn

Modify the code below to filter out rows where birth is 1943 or instrument is drums

```
beatles[beatles$birth == 1940, ]
```

More lists

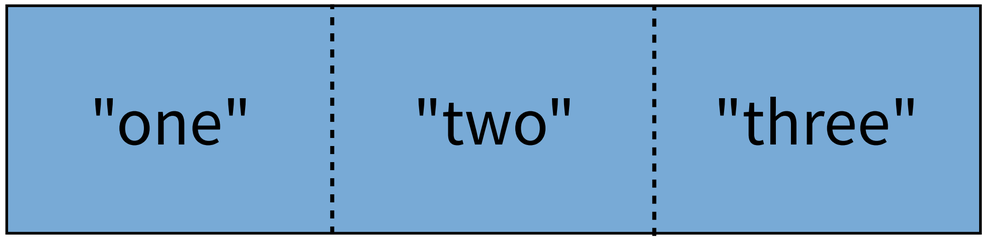
Quiz

What is the difference between an atomic vector and a list?

Atomic Vector

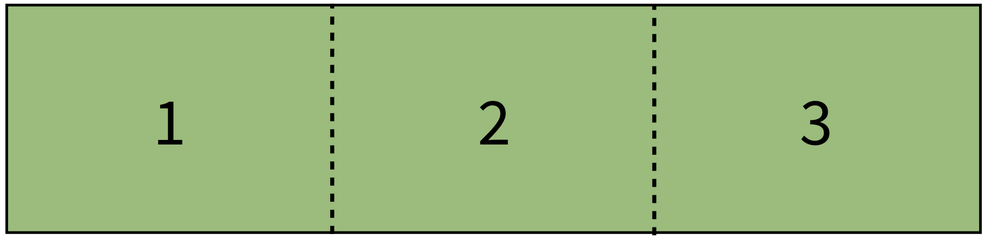
type

Atomic Vector



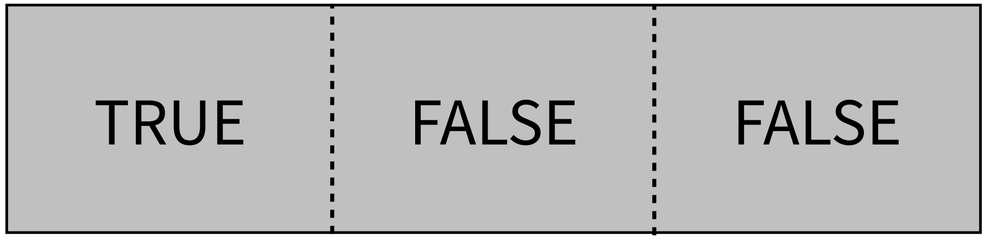
character

Atomic Vector



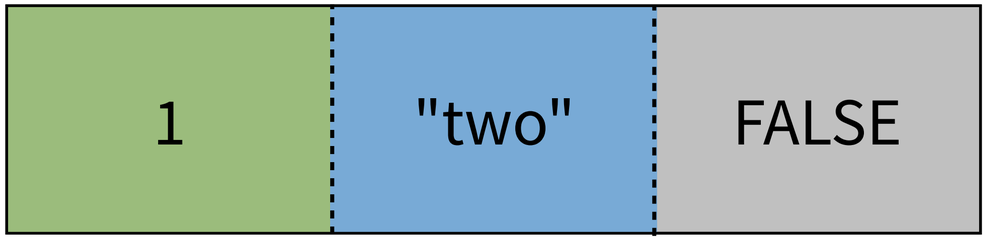
double

Atomic Vector



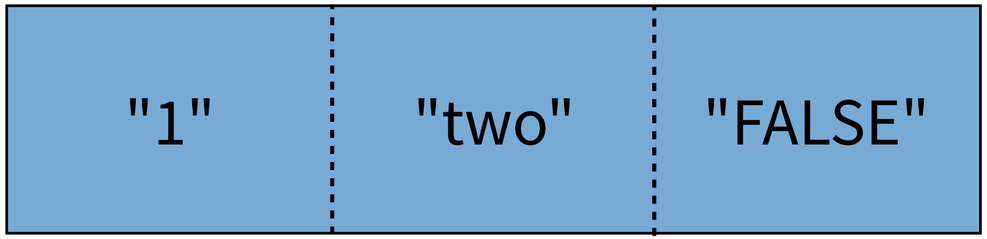
logical

Atomic Vector



?

Atomic Vector

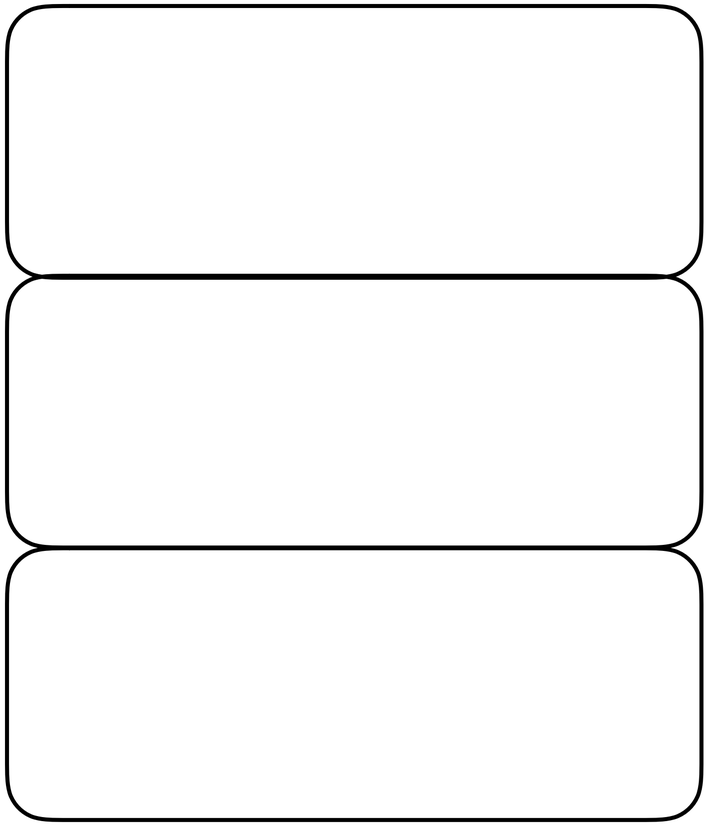


character

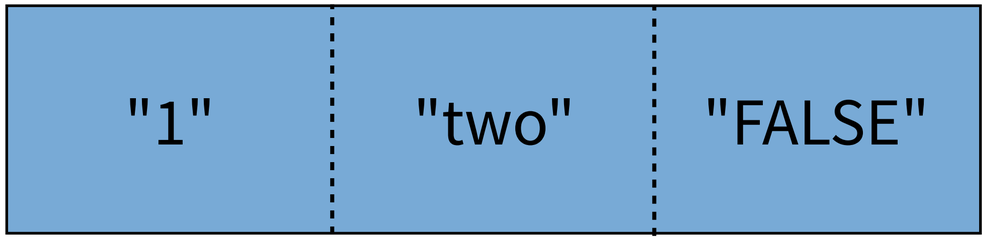
Atomic Vector

type

List

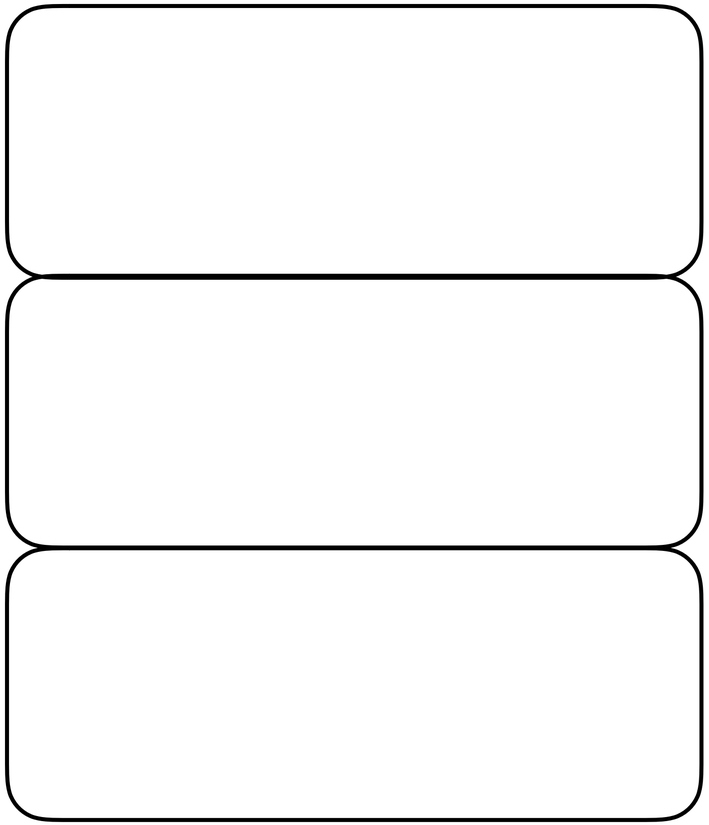


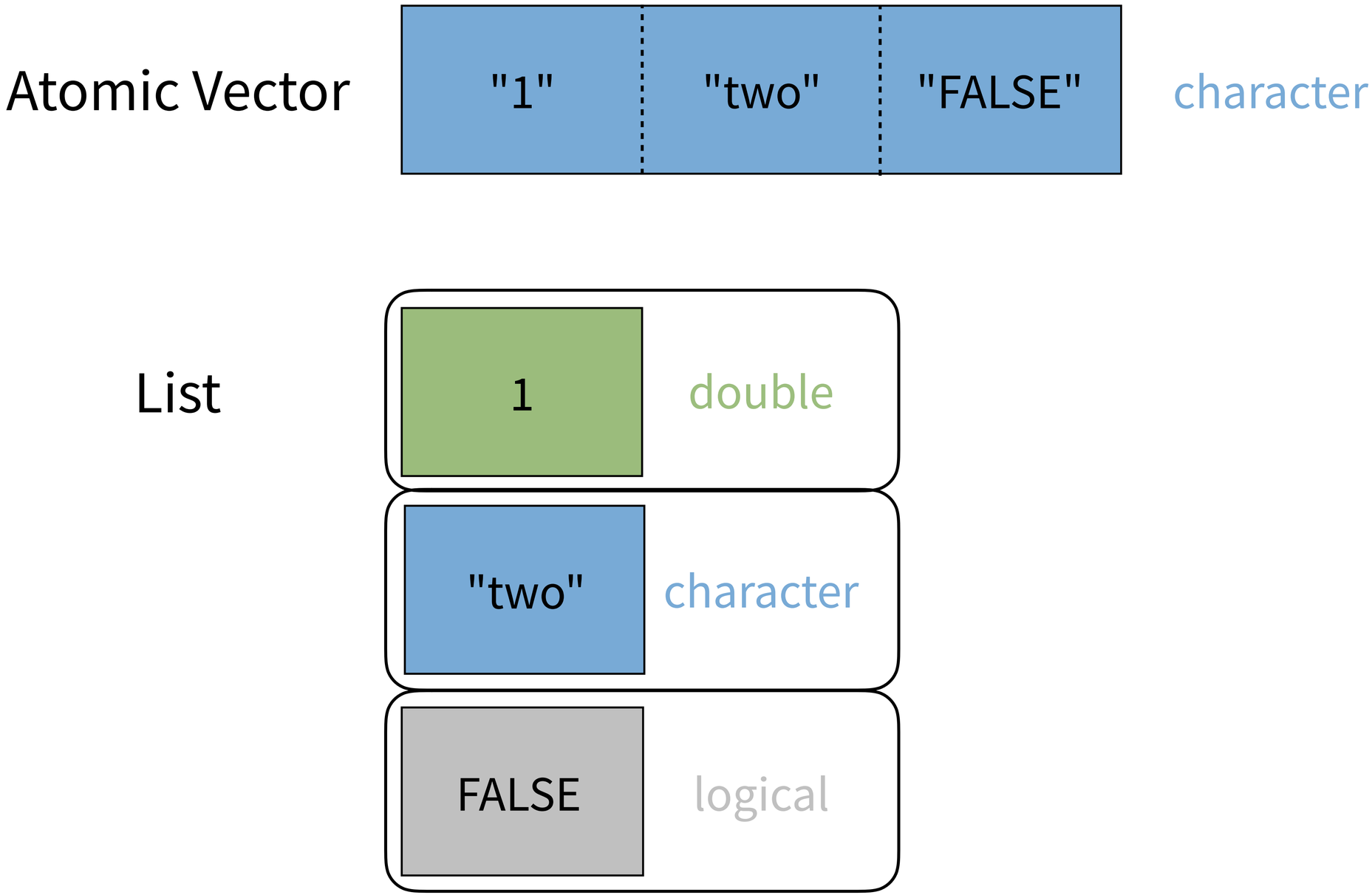
Atomic Vector

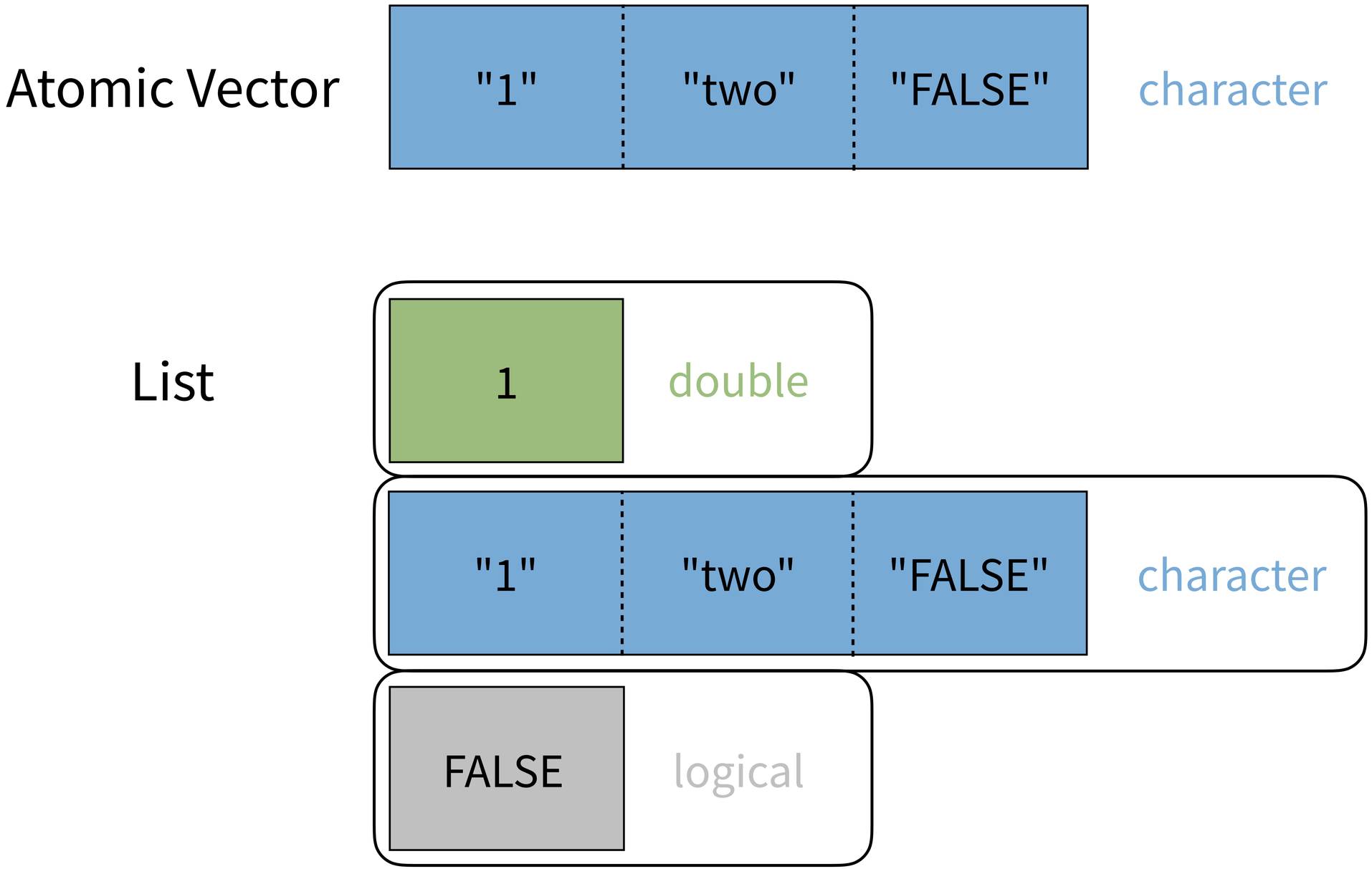


character

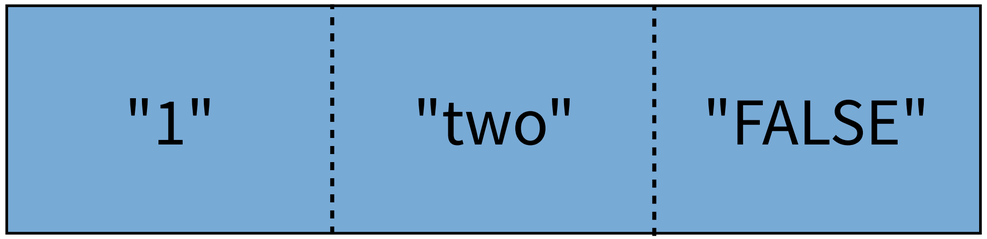
List



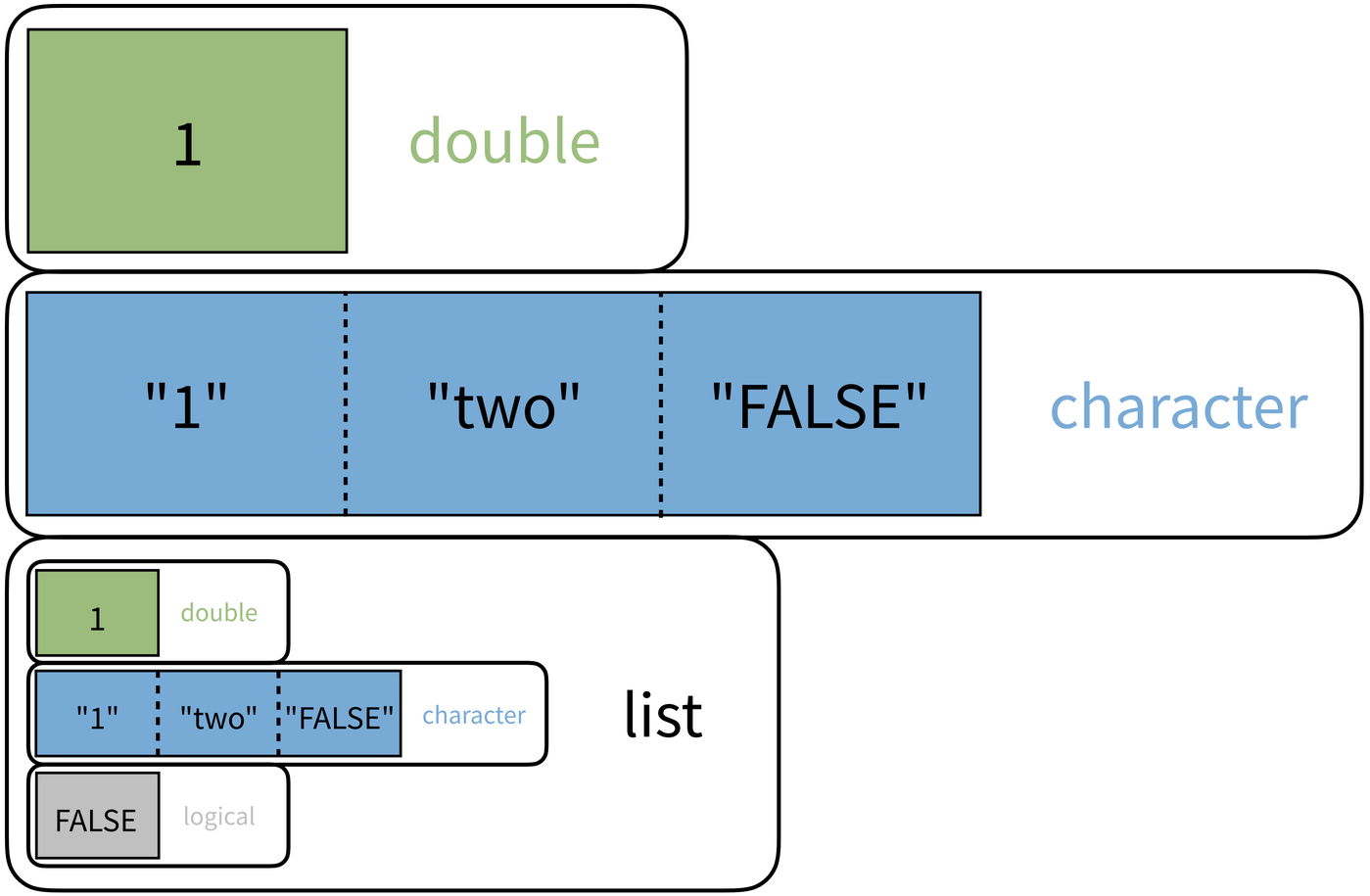

Atomic Vector
"1"
"two"
"FALSE"
character
List
1
double
"two"
character
FALSE
logical


Atomic Vector
"1"
"two"
"FALSE"
character
List
1
double
"1"
"two"
"FALSE"
character
FALSE
logical

Atomic Vector



List



Your Turn 1

Here is a list:

```
a_list <- list(num = c(8, 9),
               log = TRUE,
               cha = c("a", "b", "c"))
```

Here are two subsetting commands. Do they return the same values? Run the code chunks to confirm

```
a_list["num"]
```

```
a_list[["num"]]
```

```
a_list["num"]
```

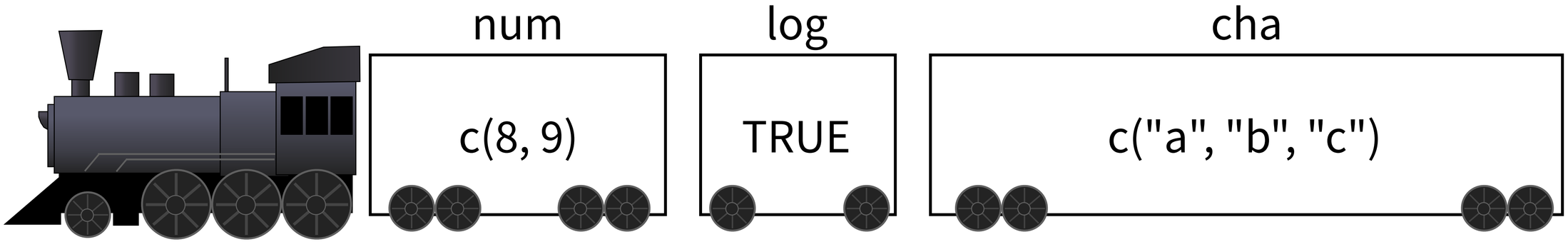
```
$num
[1] 8 9
```

A list
(with one element named num that contains an atomic vector)

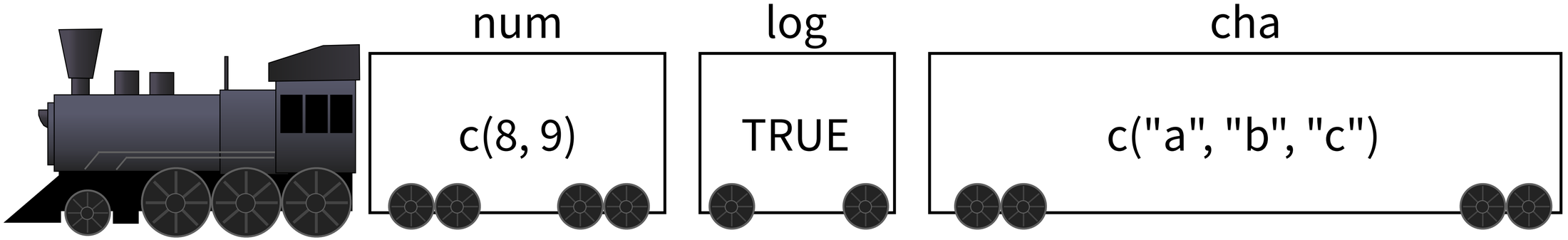
```
a_list[["num"]]
```

```
[1] 8 9
```

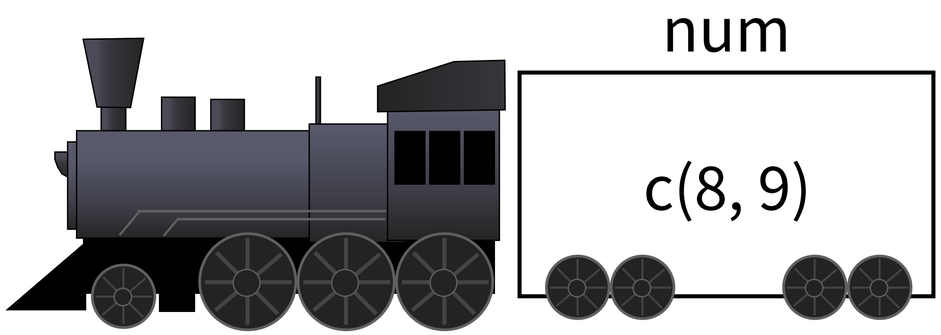
An atomic vector

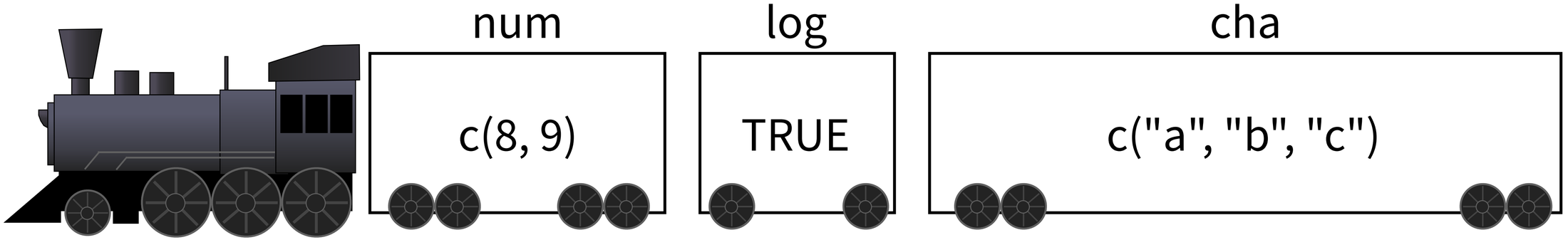



```
a_list <- list(num = c(8,9), log = TRUE, cha = c("a", "b", "c"))
```

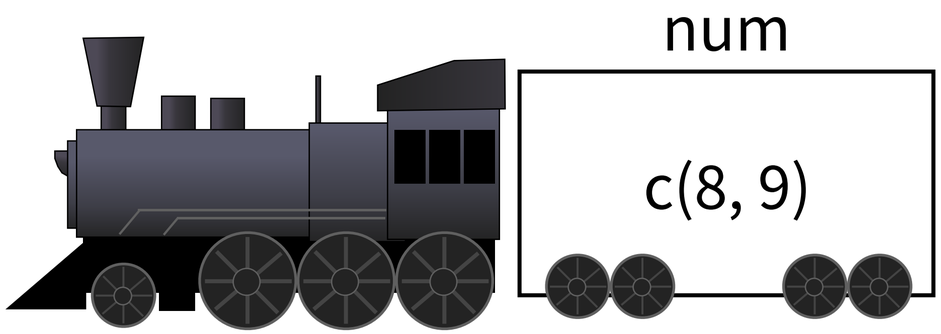



```
a_list["num"]
```



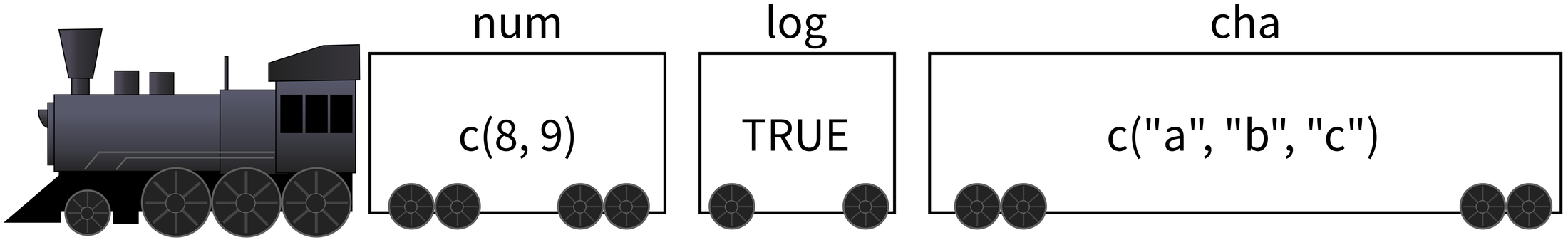


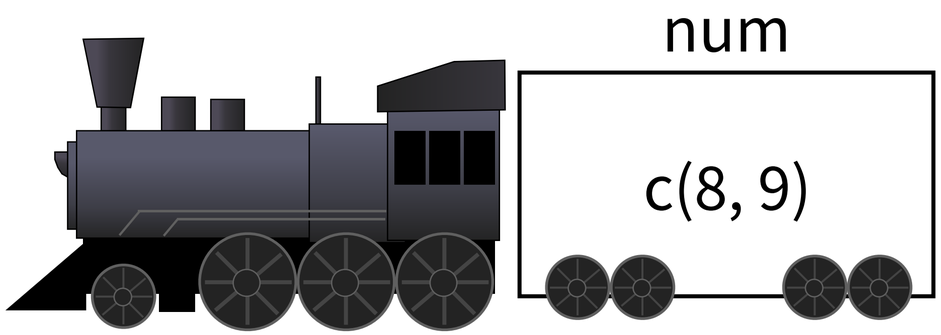

```
a_list["num"]
```




```
a_list[["num"]]
```

c(8, 9)






```
a_list["num"]
```

```
a_list[["num"]]
```

c(8, 9)

```
a_list$num
```

c(8, 9)