If you enjoy learning R with me please donate to my ride!

pelotonia.org/ezgi

All Training materials are provided "as is" and without warranty and RStudio disclaims any and all express and implied warranties including without limitation the implied warranties of title, fitness for a particular purpose, merchantability and noninfringement.

The Training Materials are licensed under the Creative Commons Attribution-Noncommercial 3.0 United States License. To view a copy of this license, visithttp://creativecommons.org/licenses/by-nc/3.0/us/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

© 2014 RStudio, Inc. All rights reserved. Follow @rstudioapp

Studio

The R language (2 of 2)

Retrieve and use information
in precise, efficient ways

Garrett Grolemund

Master Instructor, RStudio

August 2014

© 2014 RStudio, Inc. Follow @rstudioapp

1. Subsetting
2. R Packages
3. Logical tests
4. Missing values

© 2014 RStudio, Inc. All rights reserved.

Question

```
x <- c(0, 0, 0, 0, 1, 0 ,0)
y <- x
y
# 0 0 0 0 1 0 0
```

How can you save just the fifth element of `x` to `y`?

How can you change the fifth element of `x` to a `0`?

© 2014 RStudio, Inc. All rights reserved.

Subsetting

Your turn

```
vec <- c(6, 1, 3, 6, 10, 5)

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

With your neighbor, run the code on the following slide **IN YOUR HEADS**

© 2014 RStudio, Inc. All rights reserved.

vec

6	1	3	6	10	5

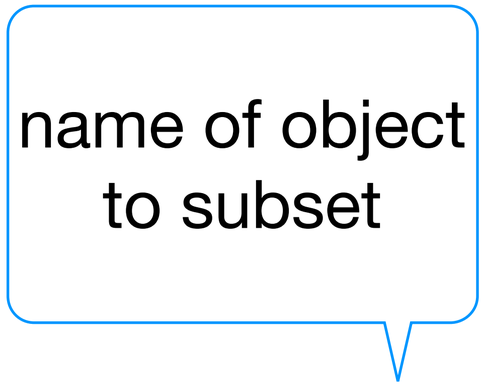
df

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

```
# Predict what the following code will do
# DON'T RUN IT!

vec[2]
vec[c(5, 6)]
vec[-c(5,6)]
vec[vec > 5]

df[c(2, 4), 3]
df[ , 1]
df[ , "instrument"]
df$instrument
```

© 2014 RStudio, Inc. All rights reserved.

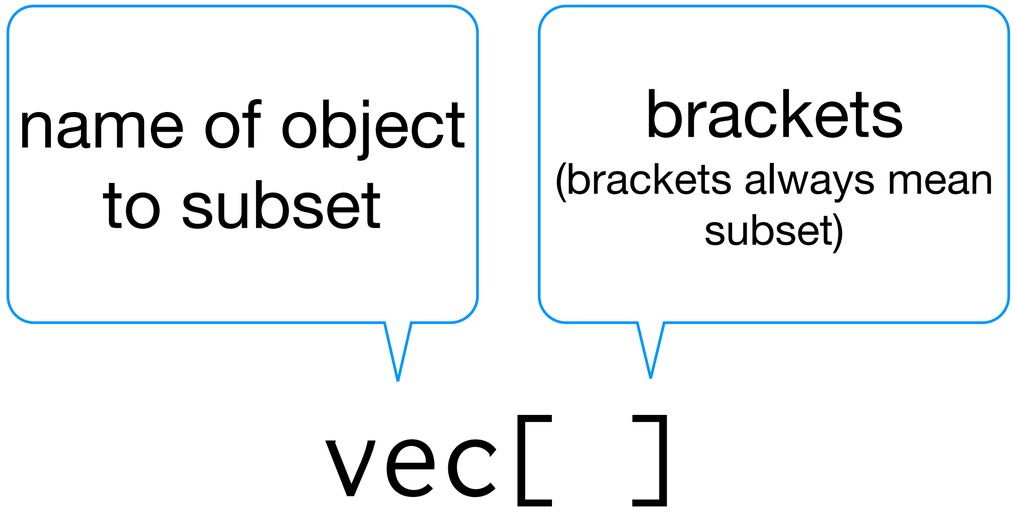
Subset notation



vec

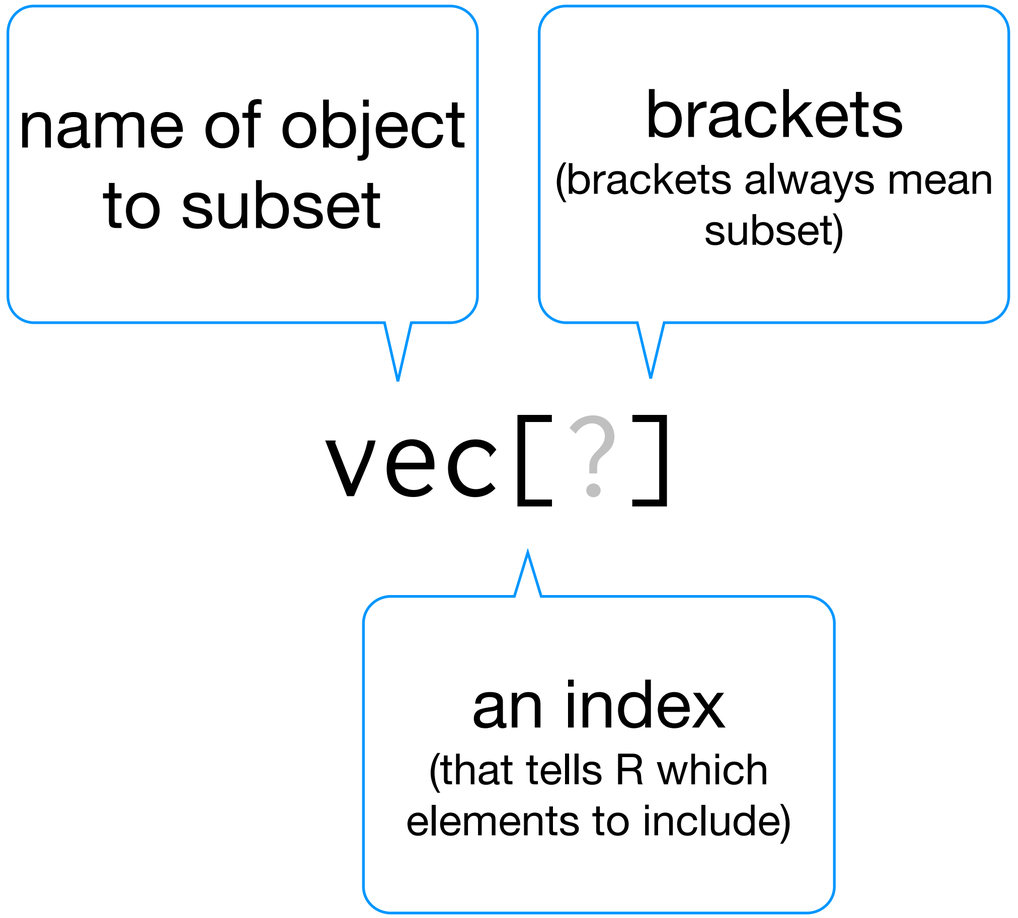
© 2014 RStudio, Inc. All rights reserved.

Subset notation



© 2014 RStudio, Inc. All rights reserved.

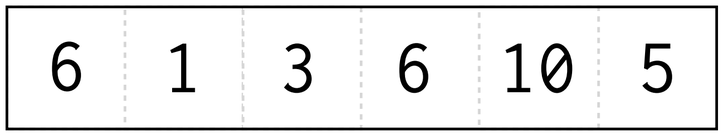
Subset notation



© 2014 RStudio, Inc. All rights reserved.

Each dimension needs its own index!

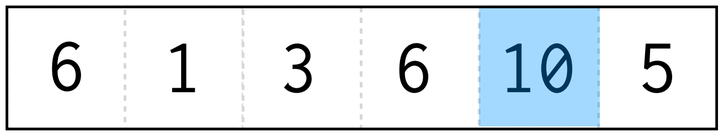
```
vec[?]
```



© 2014 RStudio, Inc. All rights reserved.

Each dimension needs its own index!

```
vec[?]
```



© 2014 RStudio, Inc. All rights reserved.

Each dimension needs its own index!

```
vec[?]
df[?,?]
```

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

© 2014 RStudio, Inc. All rights reserved.

Each dimension needs its own index!

```
vec[?]
df[?,?]
```

which **rows** to include

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

© 2014 RStudio, Inc. All rights reserved.

Each dimension needs its own index!

```
vec[?]
df[?,?]
```

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

which **rows** to include

which **columns** to include

© 2014 RStudio, Inc. All rights reserved.

Each dimension needs its own index!

```
vec[?]
df[?,?]
```

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

© 2014 RStudio, Inc. All rights reserved.

Each dimension needs its own index!

```
vec[?]
df[?,?]
```

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

But what should go in the indexes?

© 2014 RStudio, Inc. All rights reserved.

Four ways to subset

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

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[?,?]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[2,?]
```

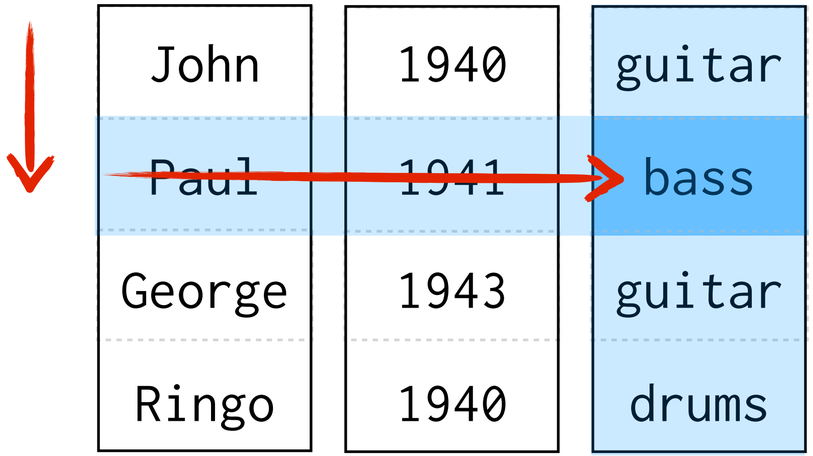
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[2,3]
```



© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[2,3]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[  ?  ,  ?  ]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[c(2,4),   ?   ]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[c(2,4),c(2,3)]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[c(2,4),c(2,3)]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[c(2,4),3]
```

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

?

© 2014 RStudio, Inc. All rights reserved.

Integers (positive)

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

```
df[c(2,4),3]
```

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

© 2014 RStudio, Inc. All rights reserved.

1. Colons are a useful way to create vectors

```
1:4
# 1 2 3 4
df[1:4, 1:2]
```

2. Repeating input repeats output

```
df[c(1,1,1,2,2), 1:3]
```

© 2014 RStudio, Inc. All rights reserved.

Integers (zero)

As an index, **zero will return nothing** from a dimension. This creates an empty object.

```
vec[0]
# numeric(0)

df[1:2, 0]
# data frame with 0 columns and 2 rows
```

© 2014 RStudio, Inc. All rights reserved.

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers in the **same** dimension

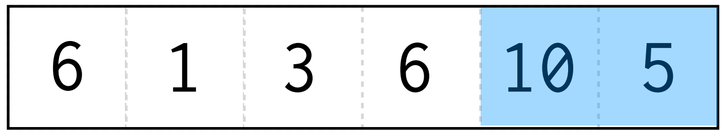
© 2014 RStudio, Inc. All rights reserved.

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers in the **same** dimension

```
vec[c(5,6)]
```



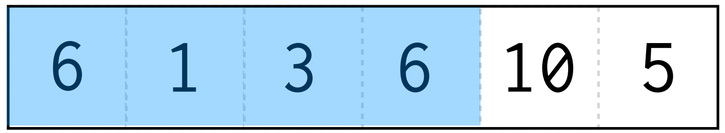
© 2014 RStudio, Inc. All rights reserved.

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers in the **same** dimension

```
vec[-c(5,6)]
```



© 2014 RStudio, Inc. All rights reserved.

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers in the **same** dimension

```
df[c(2:4), 2:3]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers in the **same** dimension

```
df[-c(2:4), 2:3]
```

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

© 2014 RStudio, Inc. All rights reserved.

Integers (negative)

Negative integers return **everything but** the elements at the specified locations.

You cannot use both negative and positive integers in the **same** dimension

```
df[-c(2:4),-(2:3)]
```

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

© 2014 RStudio, Inc. All rights reserved.

Your Turn

1. Fix these poorly written subset commands

```
vec(1:4)
vec[-1:4]
vec[3, 4, 5]
```

© 2014 RStudio, Inc. All rights reserved.

() for functions, [] for subsetting

```
vec[1:4]
# 6 1 3 6
```

Don’t mix positive and negative integers; distribute the negative sign (e.g., -1:4 = -1 0 1 2 3 4).

```
vec[-(1:4)]
# 10 5
```

Pass multiple values for the same dimension as a vector

```
vec[c(3, 4, 5)]
# 3  6  10
```

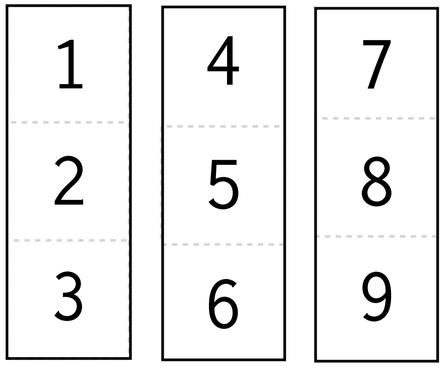
© 2014 RStudio, Inc. All rights reserved.

Your Turn

What is wrong with these subsetting commands? What will they do?

```
mat[2]

df[1]
```

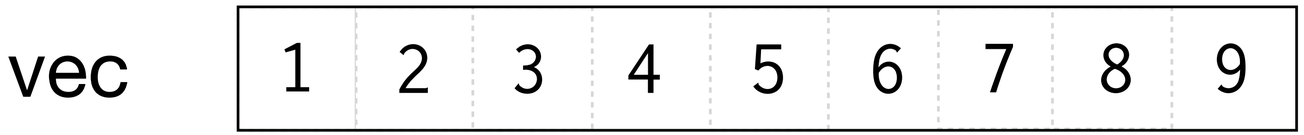
© 2014 RStudio, Inc. All rights reserved.



mat[2]

© 2014 RStudio, Inc. All rights reserved.

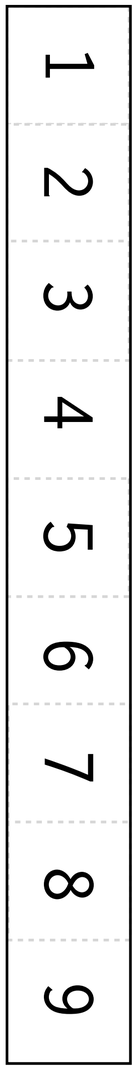
How R makes a matrix



© 2014 RStudio, Inc. All rights reserved.

How R makes a matrix

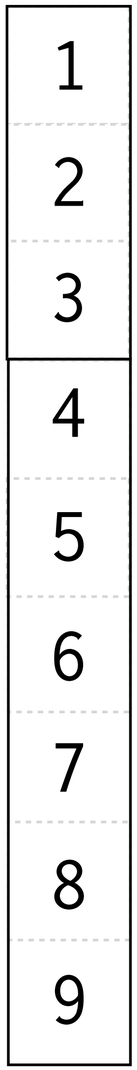
vec



© 2014 RStudio, Inc. All rights reserved.

How R makes a matrix

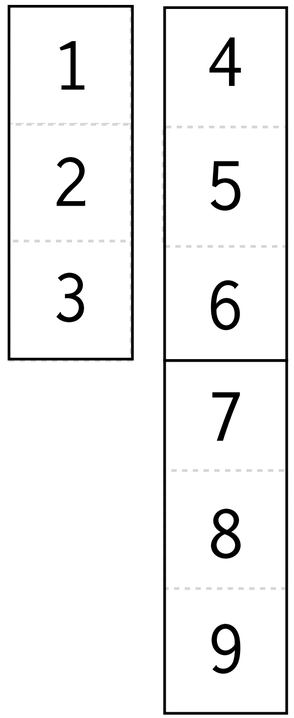
vec



© 2014 RStudio, Inc. All rights reserved.

How R makes a matrix

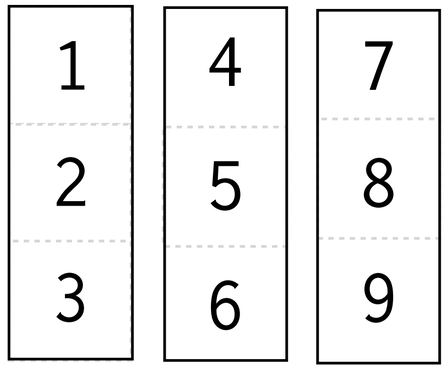
vec



© 2014 RStudio, Inc. All rights reserved.

How R makes a matrix

vec

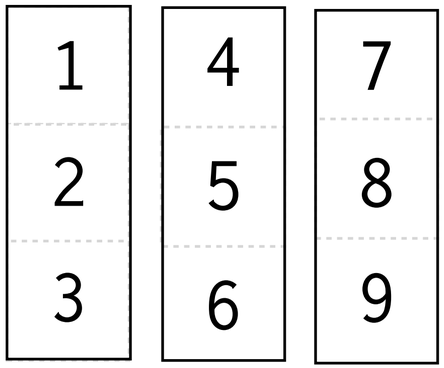


© 2014 RStudio, Inc. All rights reserved.

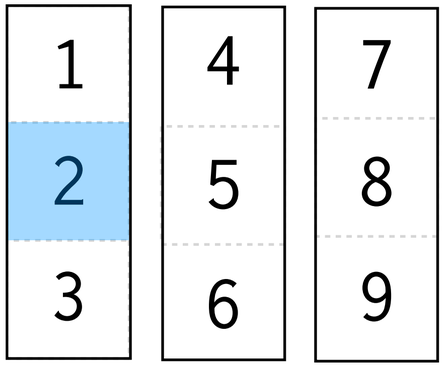
How R makes a matrix

~~vec~~

matrix

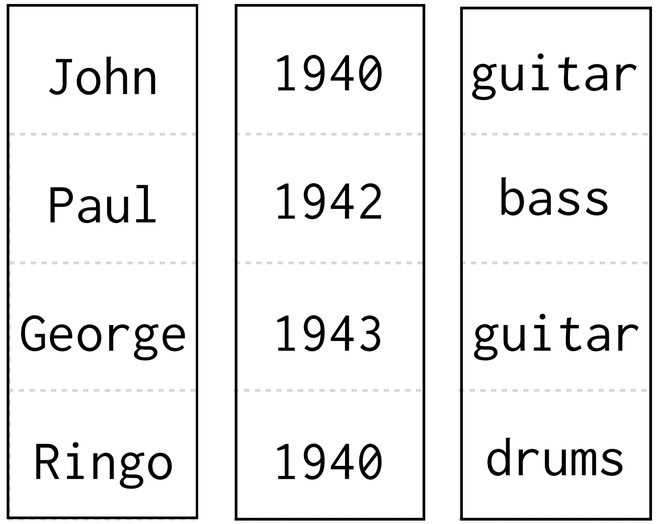


© 2014 RStudio, Inc. All rights reserved.



`mat[2]`

© 2014 RStudio, Inc. All rights reserved.

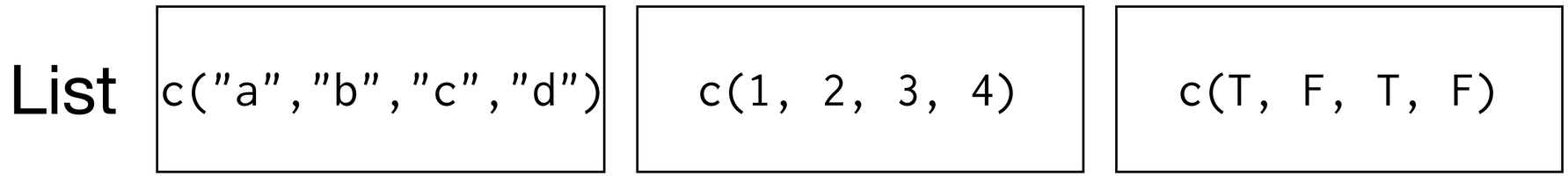


John	1940	guitar
Paul	1942	bass
George	1943	guitar
Ringo	1940	drums

df[2]

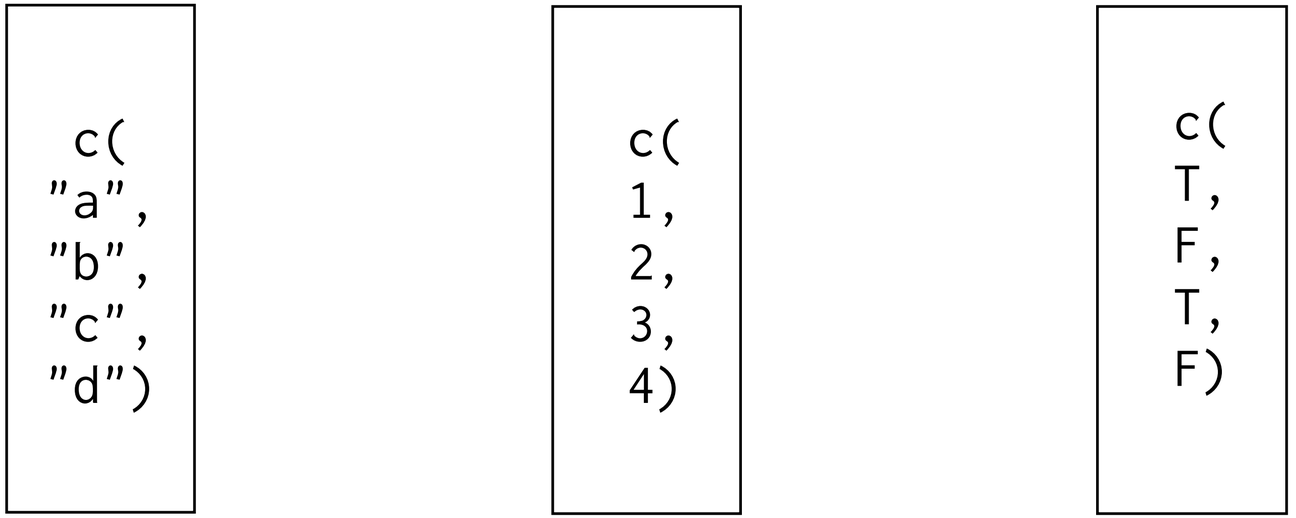
© 2014 RStudio, Inc. All rights reserved.

How R makes a data frame



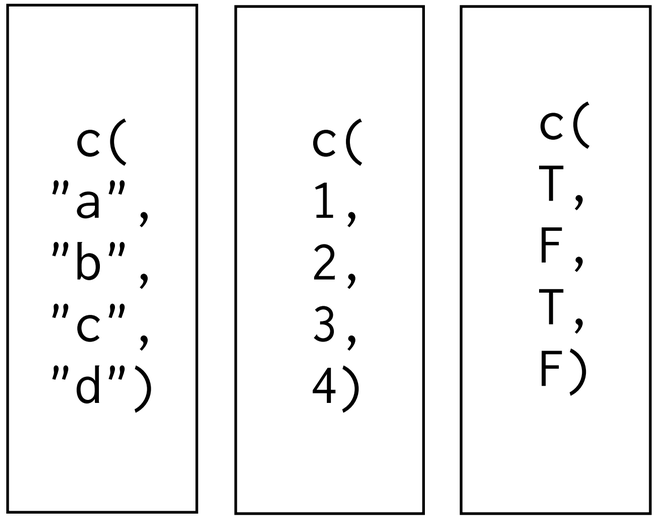
© 2014 RStudio, Inc. All rights reserved.

List



© 2014 RStudio, Inc. All rights reserved.

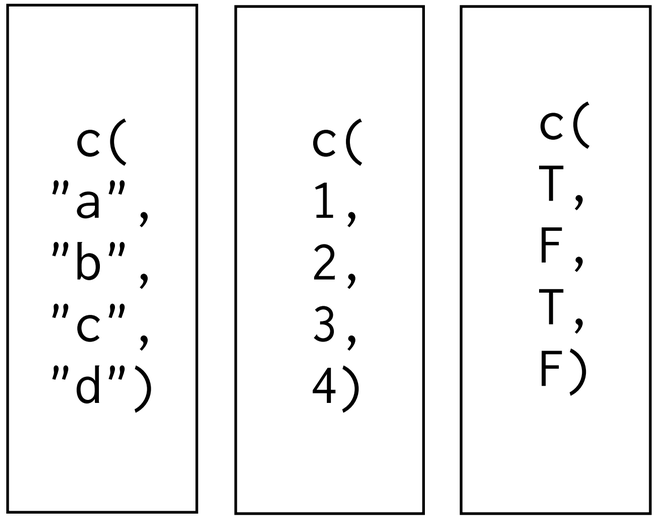
List



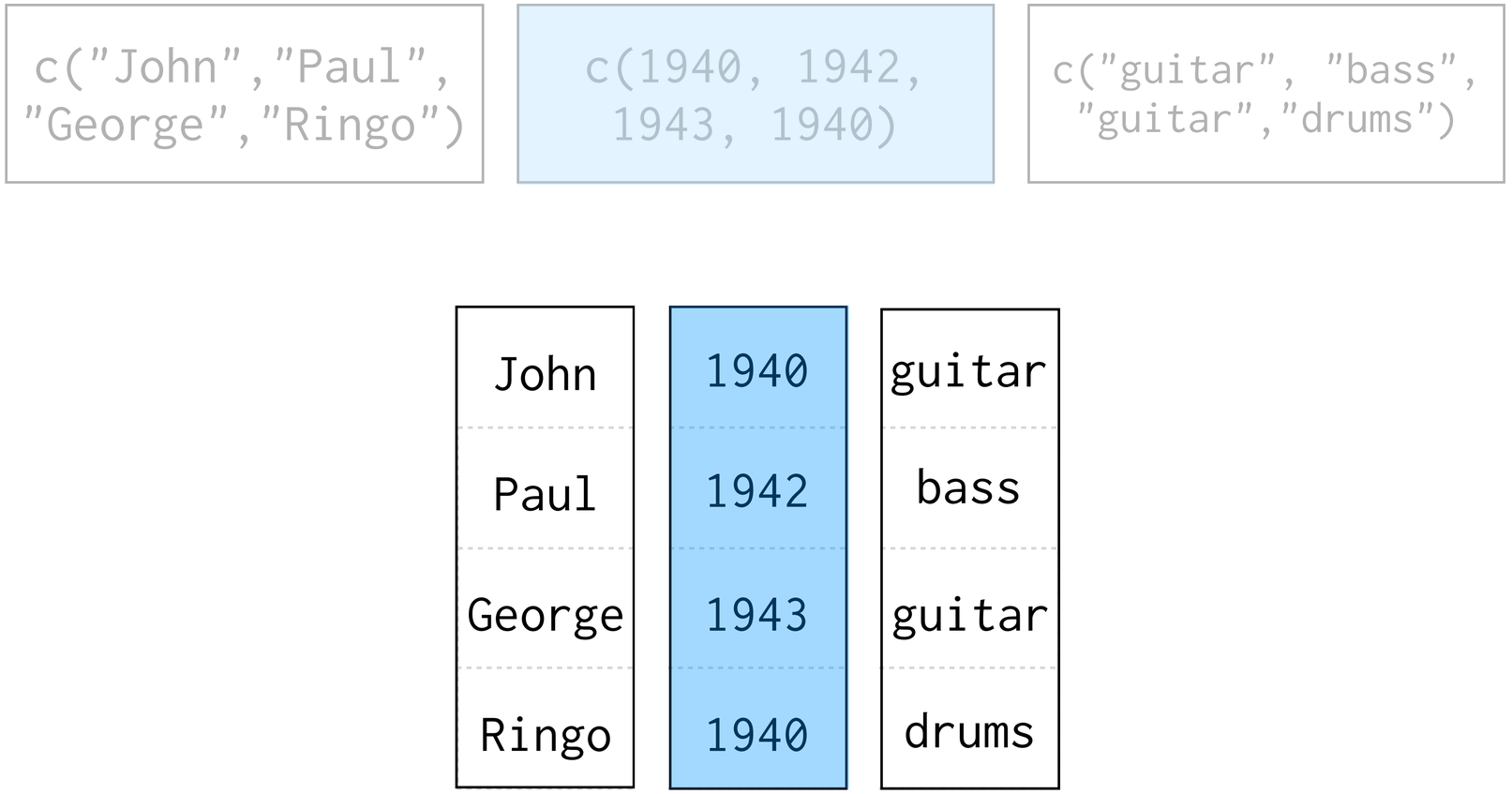
© 2014 RStudio, Inc. All rights reserved.

~~List~~

data frame



© 2014 RStudio, Inc. All rights reserved.



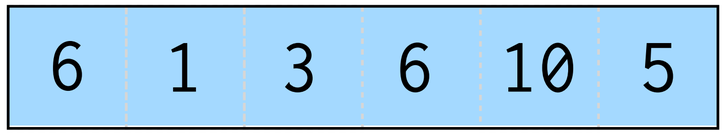
© 2014 RStudio, Inc. All rights reserved.

Blank spaces

Blank spaces return **everything**

(i.e., no subsetting occurs on that dimension)

```
vec[ ]
```



© 2014 RStudio, Inc. All rights reserved.

Blank spaces

Blank spaces return **everything**

(i.e., no subsetting occurs on that dimension)

```
df[1, ]
```

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

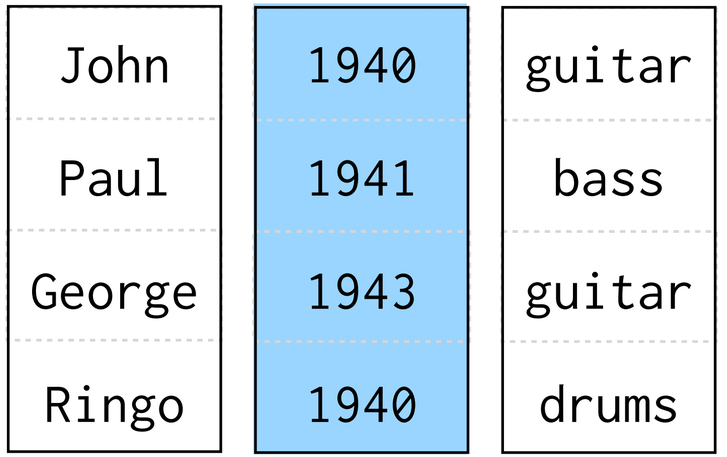
© 2014 RStudio, Inc. All rights reserved.

Blank spaces

Blank spaces return **everything**

(i.e., no subsetting occurs on that dimension)

```
df[ ,2]
```



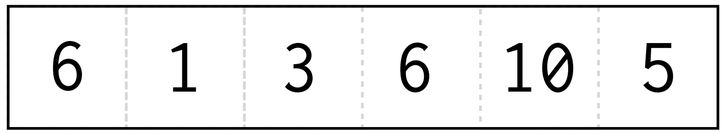
John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

© 2014 RStudio, Inc. All rights reserved.

Names

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

`vec[ ]`



© 2014 RStudio, Inc. All rights reserved.

Names

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

```
names(vec) <- c("a","b","c","d","e","f")
```

vec[ ]

a	b	c	d	e	f
6	1	3	6	10	5

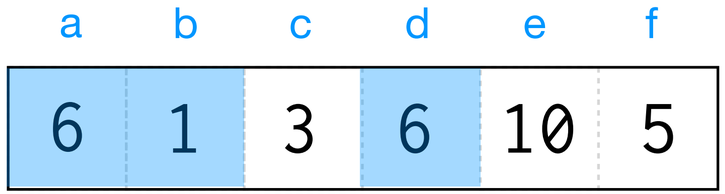
© 2014 RStudio, Inc. All rights reserved.

Names

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

```
names(vec) <- c("a","b","c","d","e","f")
```

```
vec[c("a","b","d")]
```



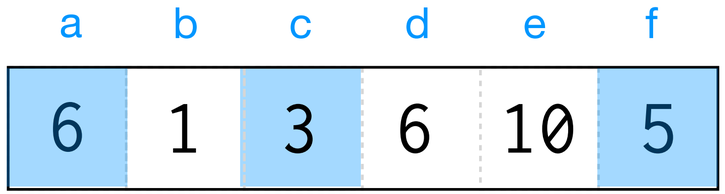
© 2014 RStudio, Inc. All rights reserved.

Names

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

```
names(vec) <- c("a","b","c","d","e","f")
```

```
vec[c("a","c","f")]
```

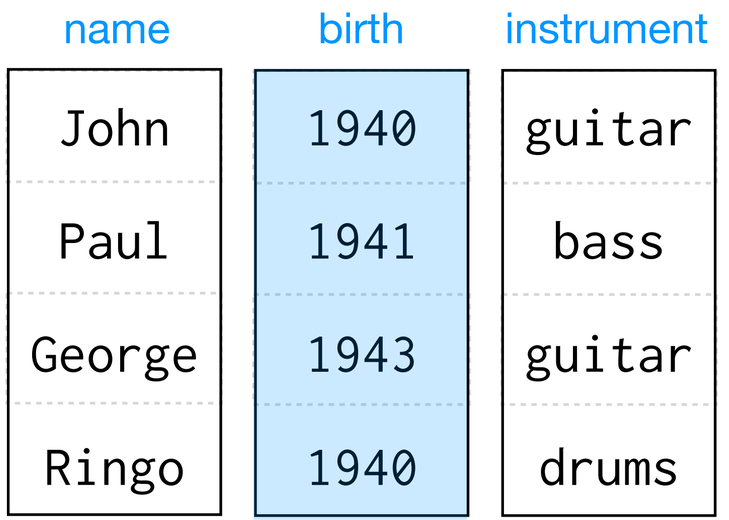


© 2014 RStudio, Inc. All rights reserved.

Names

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

```
df[ ,"birth"]
```



© 2014 RStudio, Inc. All rights reserved.

Names

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

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

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

© 2014 RStudio, Inc. All rights reserved.

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.

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

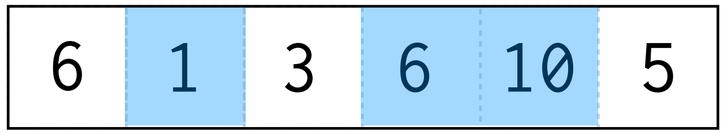
6	1	3	6	10	5

© 2014 RStudio, Inc. All rights reserved.

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.

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



© 2014 RStudio, Inc. All rights reserved.

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.

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

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

© 2014 RStudio, Inc. All rights reserved.

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.

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

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

© 2014 RStudio, Inc. All rights reserved.

Subset notation

	effect
integers	positive: returns specified elements
	0: returns nothing
	negative: returns everything but the specified elements
blank spaces	returns everything
names	returns elements or columns with the specified names
logicals	returns elements that correspond to TRUE

© 2014 RStudio, Inc. All rights reserved.

Your Turn

Write down as many ways to extract the name `"John"` from `df` as you can. Make sure each works. You have two minutes.

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

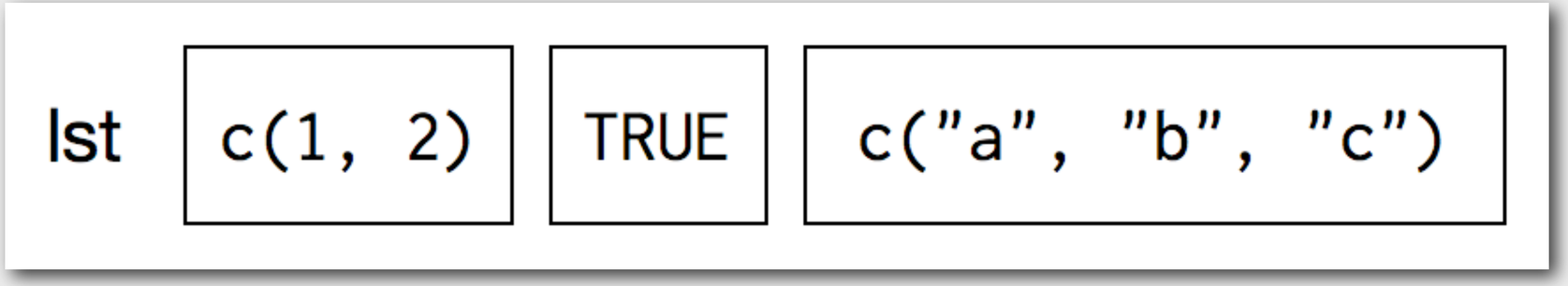
© 2014 RStudio, Inc. All rights reserved.

Answers

```
df[1, 1]
df[1, "name"]
df[1, -(2:3)]
df[1, c(TRUE, FALSE, FALSE)]
df[-(2:4), 1]
df[-(2:4), "name"]
df[-(2:4), -(2:3)]
df[-(2:4), c(TRUE, FALSE, FALSE)]
df[c(TRUE, FALSE, FALSE, FALSE), 1]
df[c(TRUE, FALSE, FALSE, FALSE), "name"]
df[c(TRUE, FALSE, FALSE, FALSE), -(2:3)]
df[c(T, F, F, F), c(T, F, F)]
```

© 2014 RStudio, Inc. All rights reserved.

Your Turn



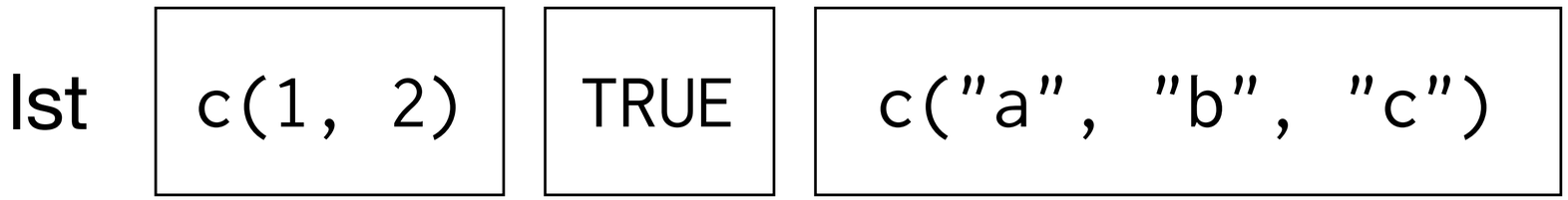

```
lst <- list(c(1, 2), TRUE, c("a", "b", "c"))
```

Can you extract the vector `c(1,2)` from `lst` and run `sum` on it? Note that sum calculates the sum of a vector:

```
sum(c(1,2,3)) # 6
```

© 2014 RStudio, Inc. All rights reserved.

Subsetting lists



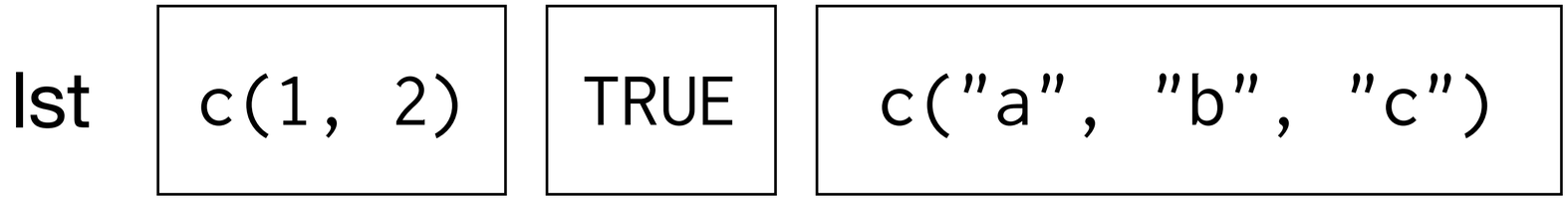

```
sum(lst[1]) # Error!

# What is the difference?
lst[c(1,2)]
lst[1]
lst[[1]]
```

© 2014 RStudio, Inc. All rights reserved.

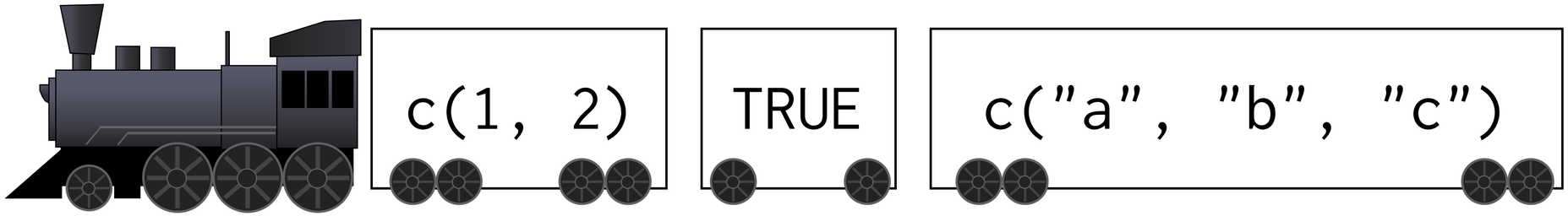
If list `x` is a train carrying objects, then `x[[5]]` is the object in car 5; `x[4:6]` is a train of cars 4-6.

http://twitter.com/#!/RLangTip/status/118339256388304896

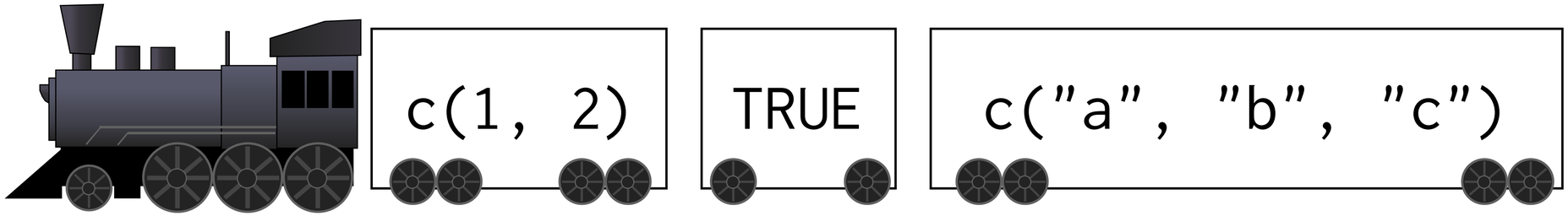
© 2014 RStudio, Inc. All rights reserved.



© 2014 RStudio, Inc. All rights reserved.

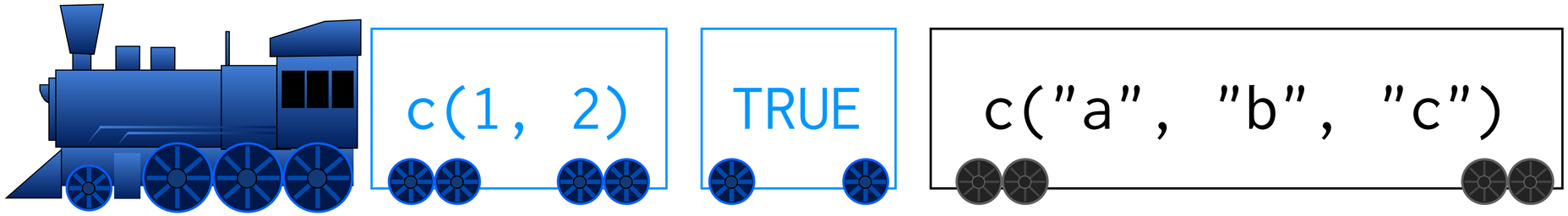


© 2014 RStudio, Inc. All rights reserved.

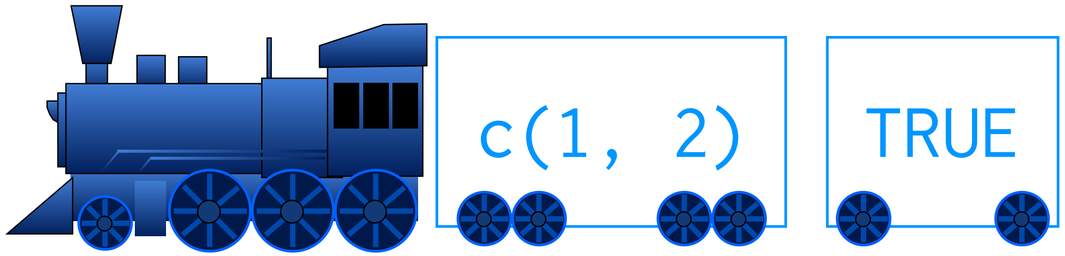



```
lst[c(1,2)]
```

© 2014 RStudio, Inc. All rights reserved.

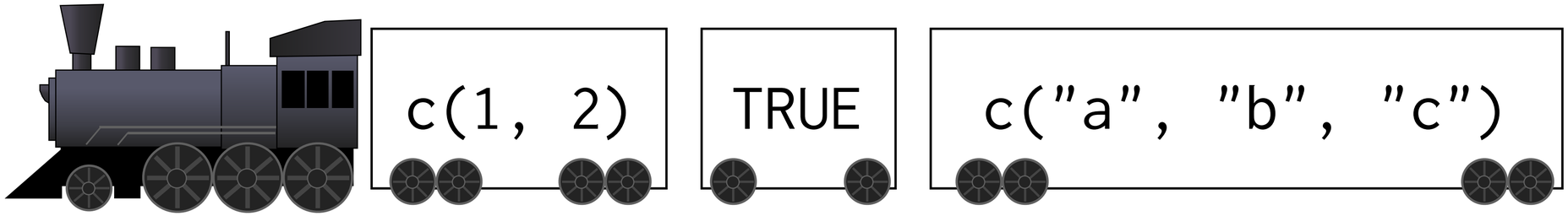



```
lst[c(1,2)]
```



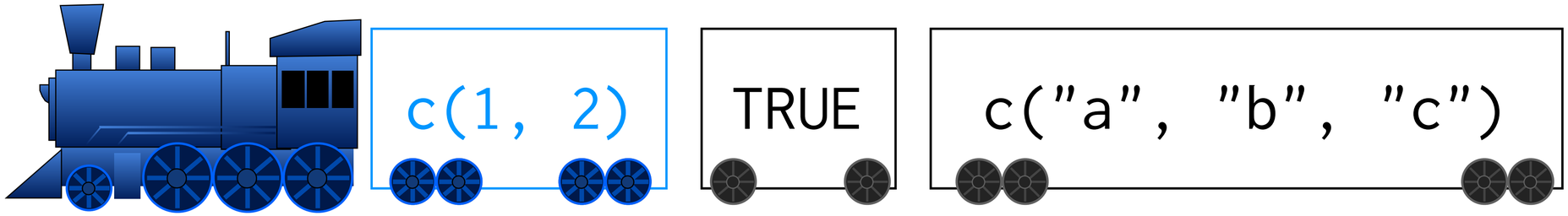

```
# [[1]]
# [1] 1 2
#
# [[2]]
# [1] TRUE
```

© 2014 RStudio, Inc. All rights reserved.

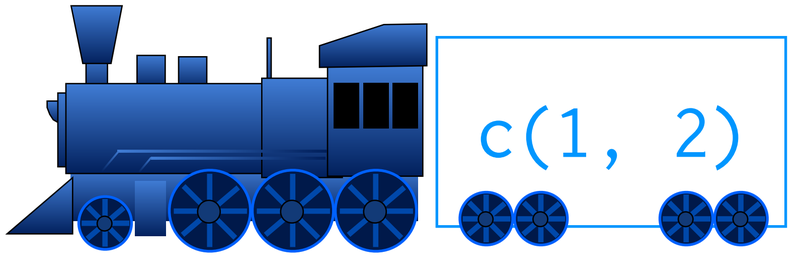



```
lst[1]
```

© 2014 RStudio, Inc. All rights reserved.

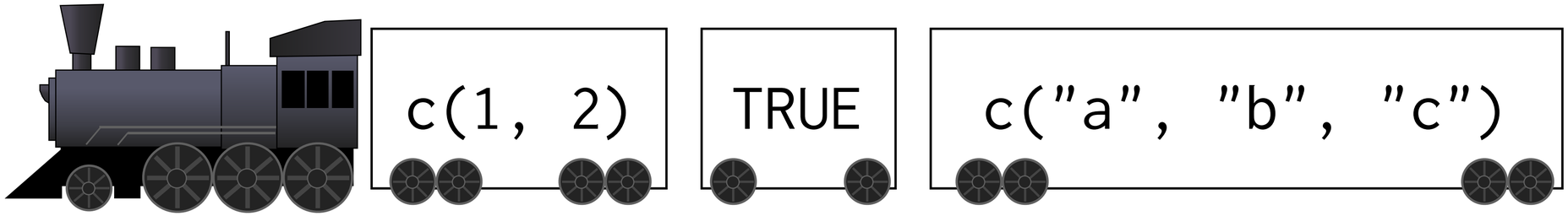



```
lst[1]
```



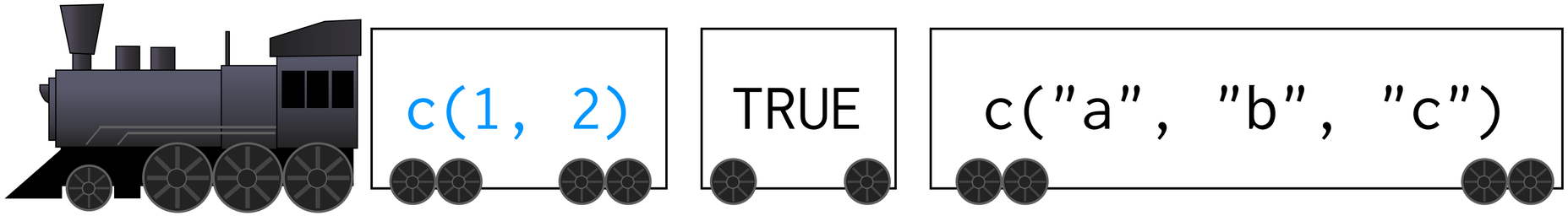

```
# [[1]]
# [1] 1 2
```

© 2014 RStudio, Inc. All rights reserved.




```
lst[[1]]
```

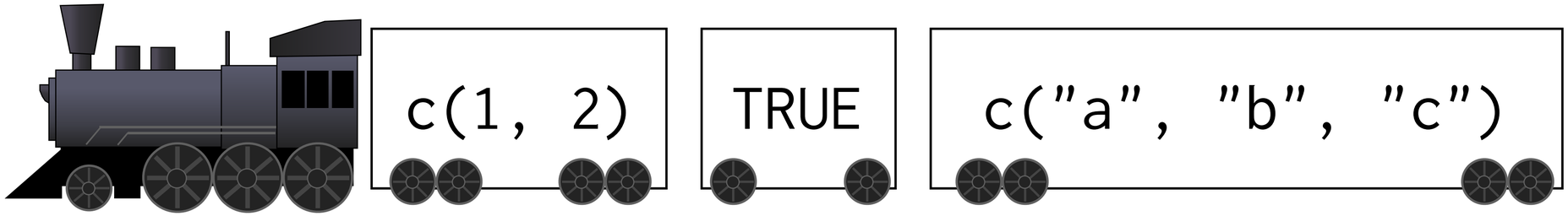
© 2014 RStudio, Inc. All rights reserved.




```
lst[[1]]
```

```
c(1, 2)
```

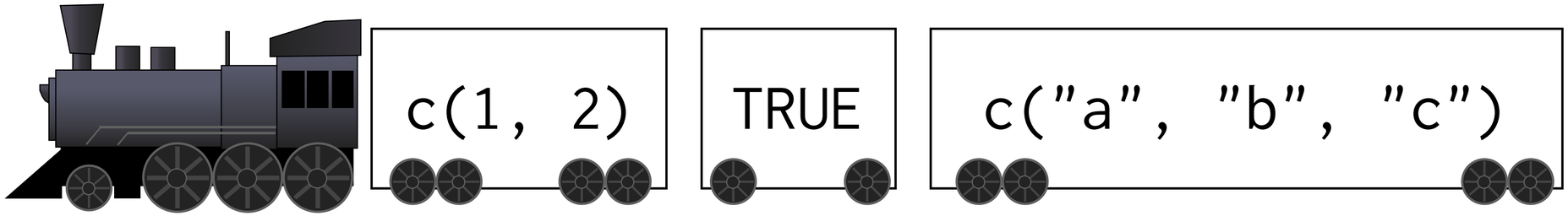
© 2014 RStudio, Inc. All rights reserved.




```
lst[[1]][2]
```

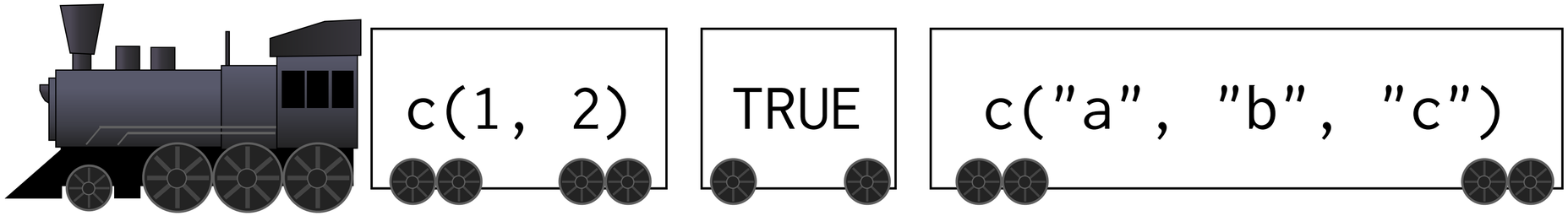
What will this return?

© 2014 RStudio, Inc. All rights reserved.




```
lst[[1]][2]
```

© 2014 RStudio, Inc. All rights reserved.

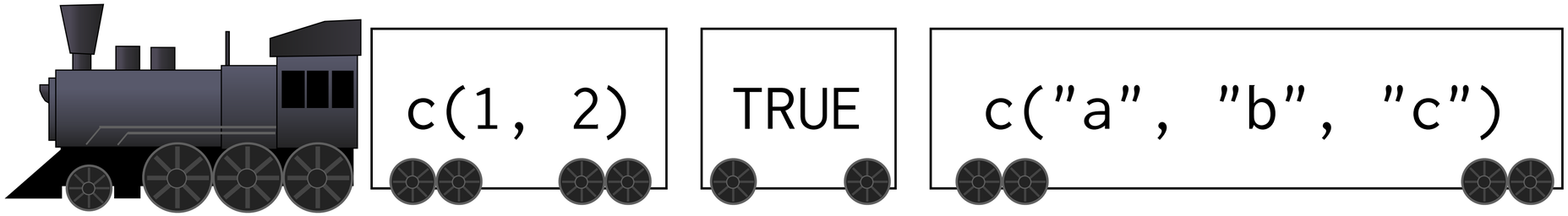



```
lst[[1]][2]
```



```
c(1, 2)[2]
```

© 2014 RStudio, Inc. All rights reserved.




```
lst[[1]][2]
```



```
c(1, 2)[2]
```



```
2
```

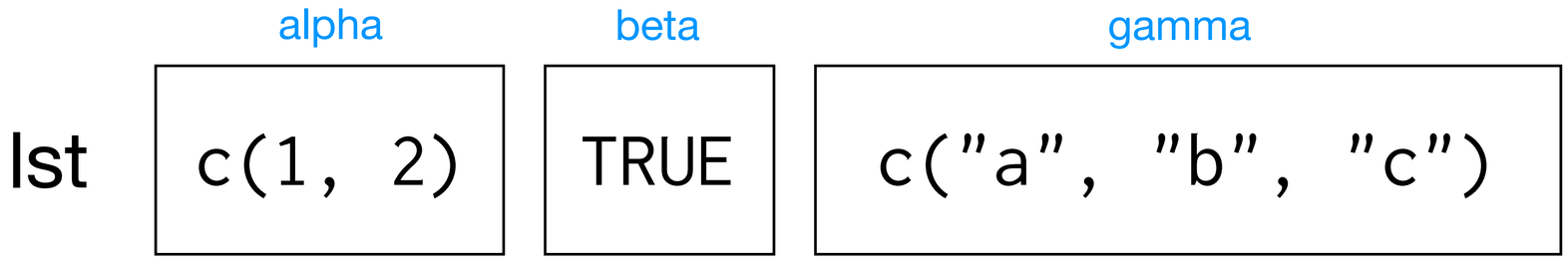
© 2014 RStudio, Inc. All rights reserved.

$

The most common syntax for subsetting lists and data frames

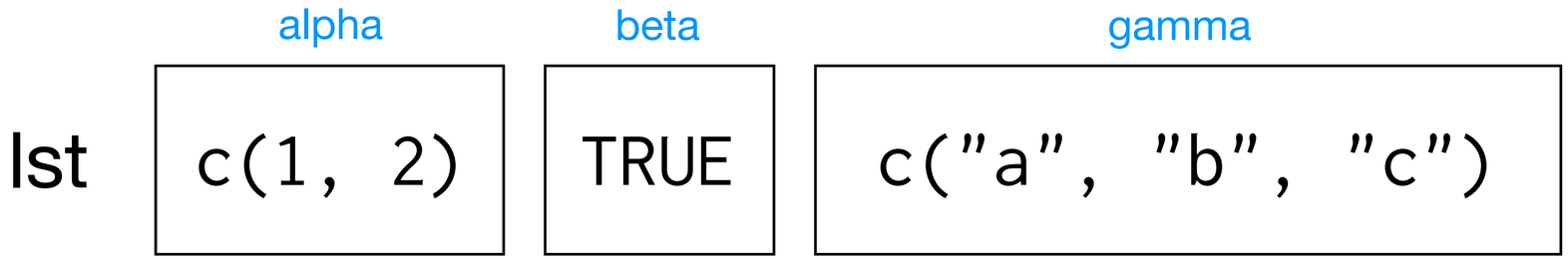
```
names(lst) <- c("alpha", "beta", "gamma")
```

© 2014 RStudio, Inc. All rights reserved.




```
lst$alpha
```

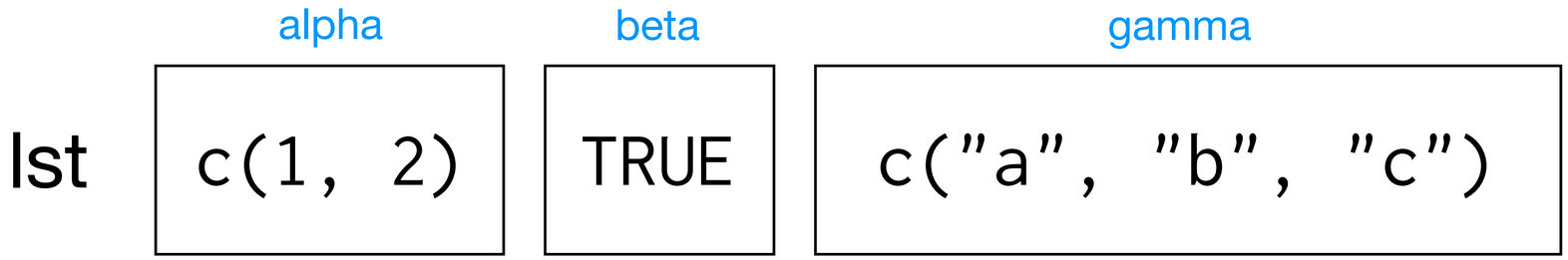
© 2014 RStudio, Inc. All rights reserved.




```
lst$alpha
```

name of list

© 2014 RStudio, Inc. All rights reserved.

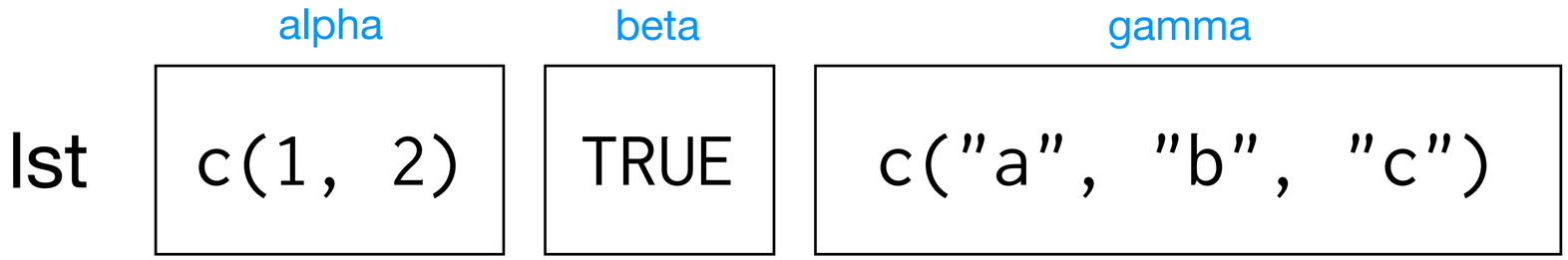



```
lst$alpha
```

name of list

$

© 2014 RStudio, Inc. All rights reserved.



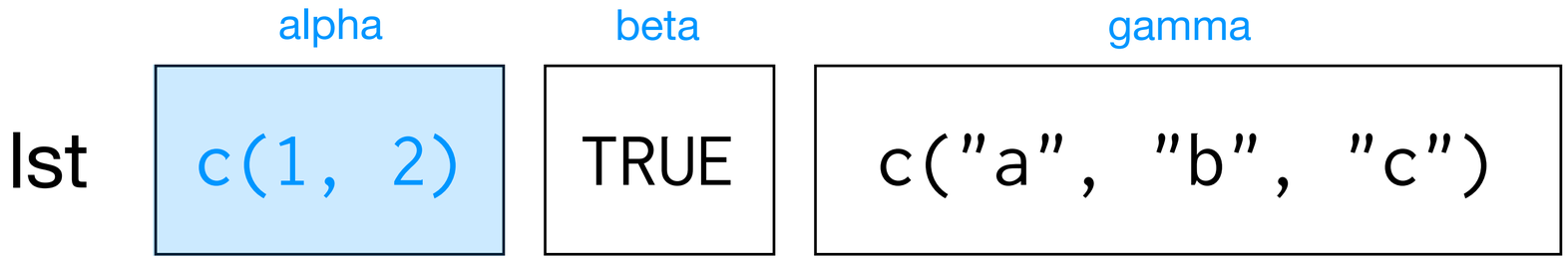

```
lst$alpha
```

name of list

$

name of element
(no quotes)

© 2014 RStudio, Inc. All rights reserved.



`c(1, 2)`

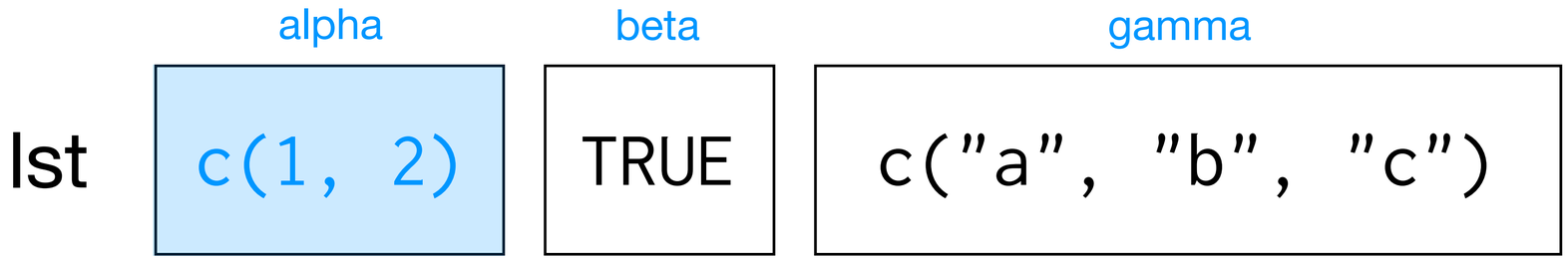
```
lst$alpha
```

name of list

$

name of element
(no quotes)

© 2014 RStudio, Inc. All rights reserved.



`c(1, 2)`

```
lst$alpha
```

Same as `lst[["alpha"]]`

© 2014 RStudio, Inc. All rights reserved.

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

df$birth

© 2014 RStudio, Inc. All rights reserved.

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

df$birth

name of data frame

© 2014 RStudio, Inc. All rights reserved.

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

df$birth

name of data frame

$

© 2014 RStudio, Inc. All rights reserved.

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

df$birth

name of data frame

$

name of column
(no quotes)

© 2014 RStudio, Inc. All rights reserved.

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

```
c(1940, 1941, 1943, 1940)
```

```
df$birth
```

name of data frame

$

name of column
(no quotes)

© 2014 RStudio, Inc. All rights reserved.

R Packages

R Packages

A collection of code and functions written for the R language.

Usually focuses on a specific task or problem.

Most of the useful R applications appear in packages.

© 2014 RStudio, Inc. All rights reserved.

Start RStudio

© 2014 RStudio, Inc. All rights reserved.

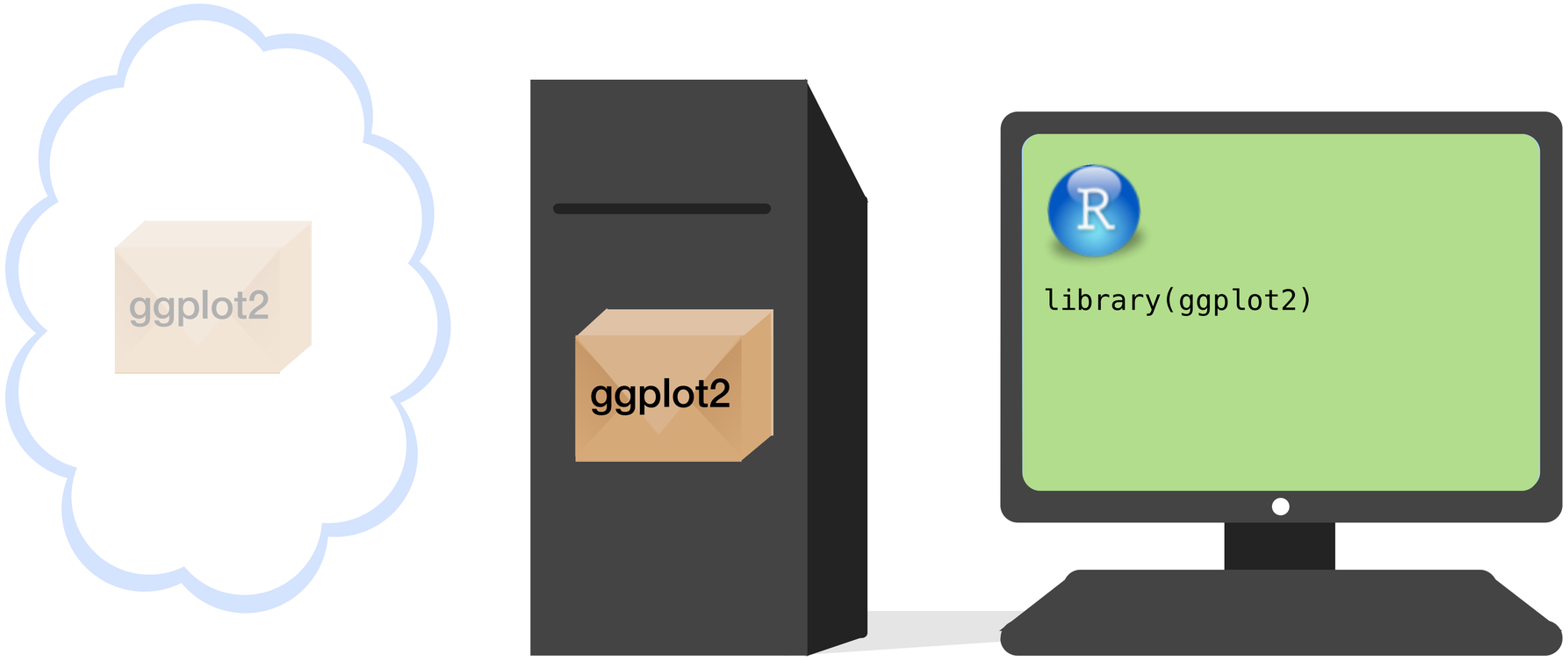
Install your package with `install.packages("ggplot2")`

(ONE TIME)

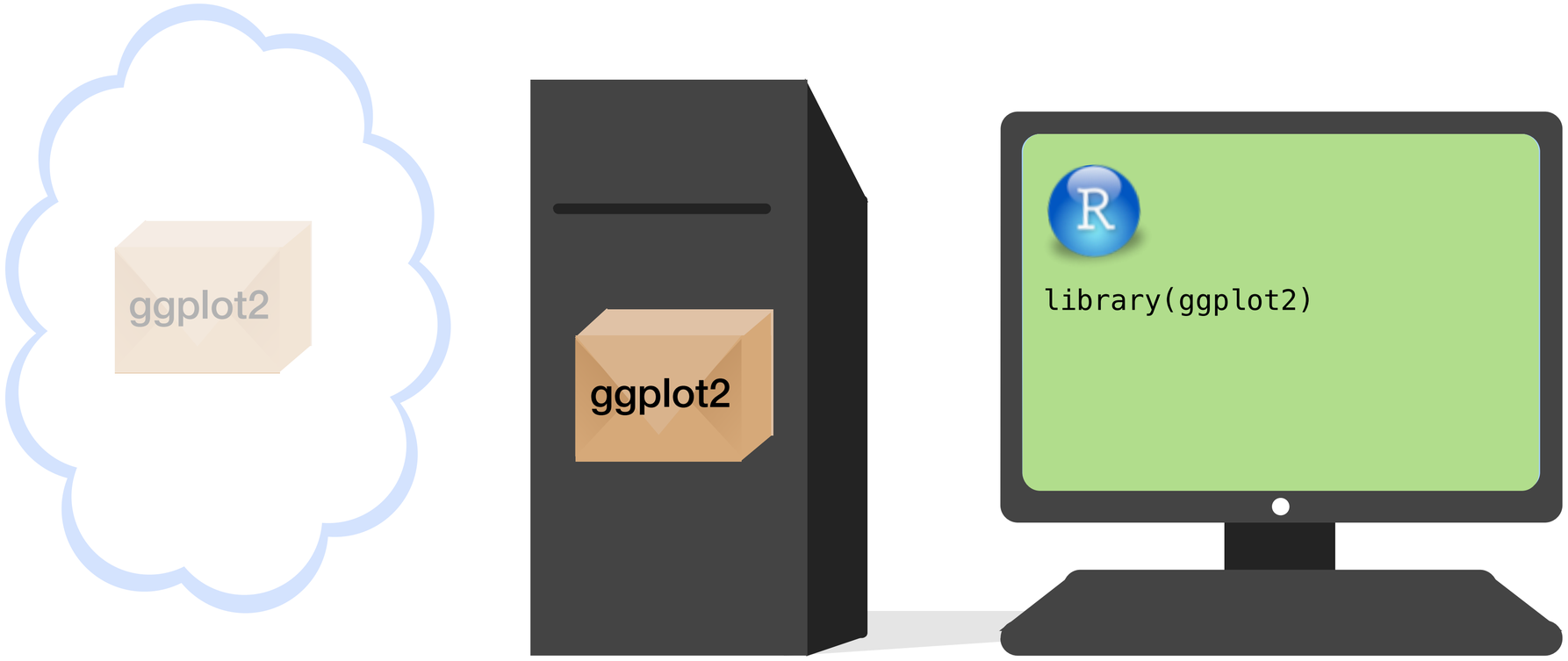
© 2014 RStudio, Inc. All rights reserved.

Load your package with `library(ggplot2)`

(EVERY TIME)



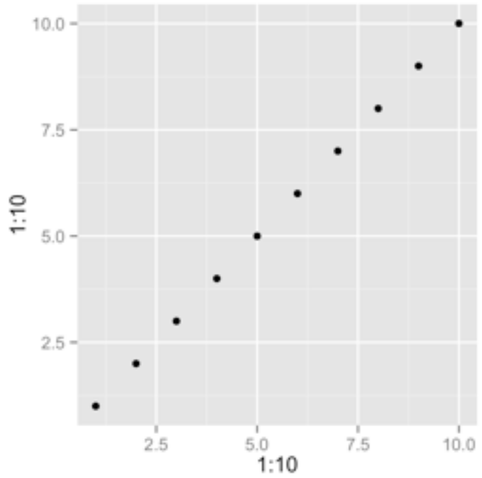
© 2014 RStudio, Inc. All rights reserved.



© 2014 RStudio, Inc. All rights reserved.

```
qplot(1:10, 1:10)
## Error: could not find function "qplot"
```

```
library(ggplot2)
qplot(1:10, 1:10)
##
```



You cannot use a function in a package until you load the package

© 2014 RStudio, Inc. All rights reserved.

Package summary

1. Download the package with `install.packages("name")`
   - You only have to do this once
   - You should be connected to the internet
2. Load the package with `library("name")`
   - You have to do this each time you start an R session.

There are over 5100 R packages

© 2014 RStudio, Inc. All rights reserved.

Your Turn

We're going to use the ggplot2, maps, RColorBrewer, and scales packages today.

Load them with

```
library("ggplot2")
library("maps")
library("RColorBrewer")
```

Note: If you have not yet installed them, you'll need to run `install.packages(c("ggplot2", “maps", "RColorBrewer"))` first.

© 2014 RStudio, Inc. All rights reserved.

Diamonds

Diamonds data

- ~**54,000** round diamonds from http://www.diamondse.info/
- comes in the ggplot2 package
- Carat, colour, clarity, cut
- Total depth, table, depth, width, height
- Price

© 2014 RStudio, Inc. All rights reserved.

Your turn

`diamonds` is huge!

Use subsetting to look at just the first six rows of `diamonds`

Challenge: use subsetting to look at just the **last** six rows

© 2014 RStudio, Inc. All rights reserved.

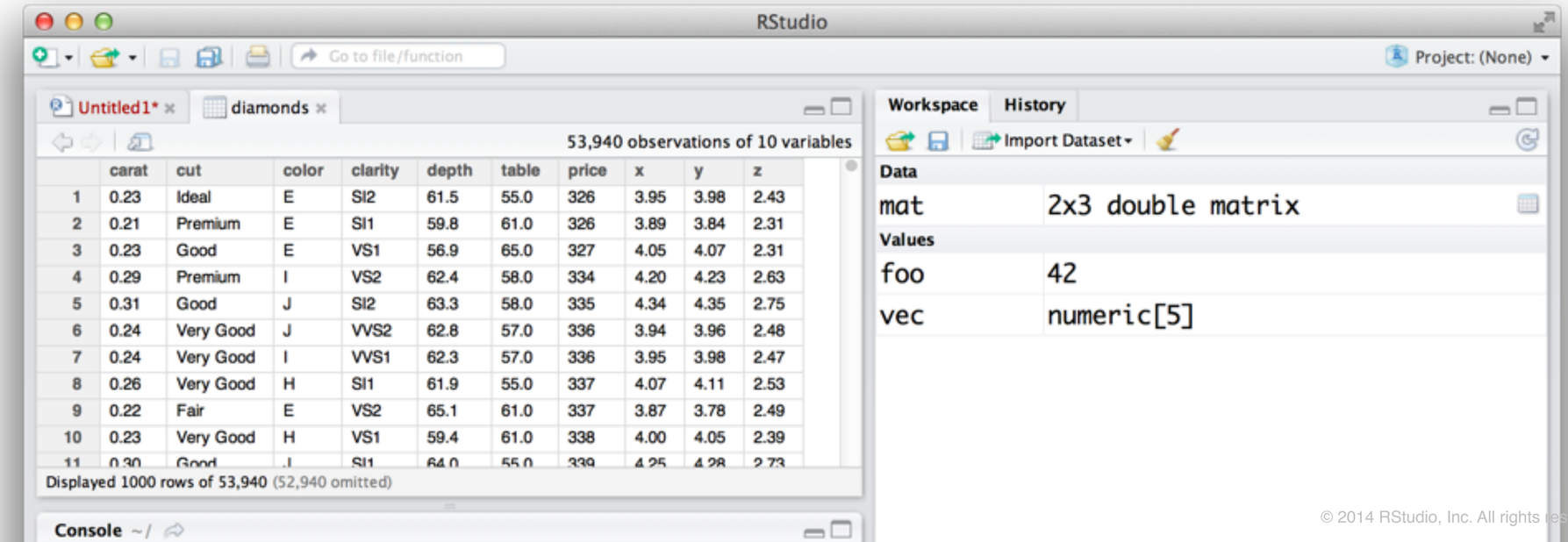
```
diamonds[1:6, ]
nrow(diamonds)
# 53940
diamonds[53935:53940, ]

# Same as
head(diamonds)
tail(diamonds)
```

© 2014 RStudio, Inc. All rights reserved.

View

The View function can also help you examine a data set, it opens a spreadsheet like data viewer.

```
View(diamonds) # notice: Capital V
```



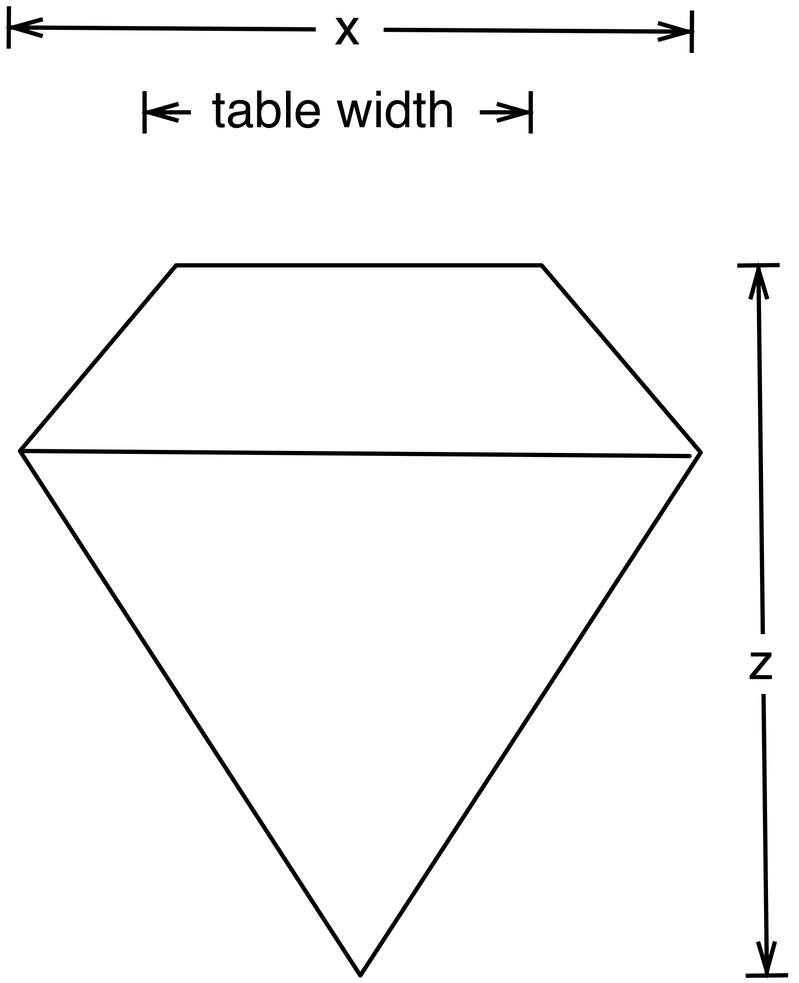
© 2014 RStudio, Inc. All rights reserved.

Help pages

You can open the help page for any R object (including functions) by typing ? followed by the object's name

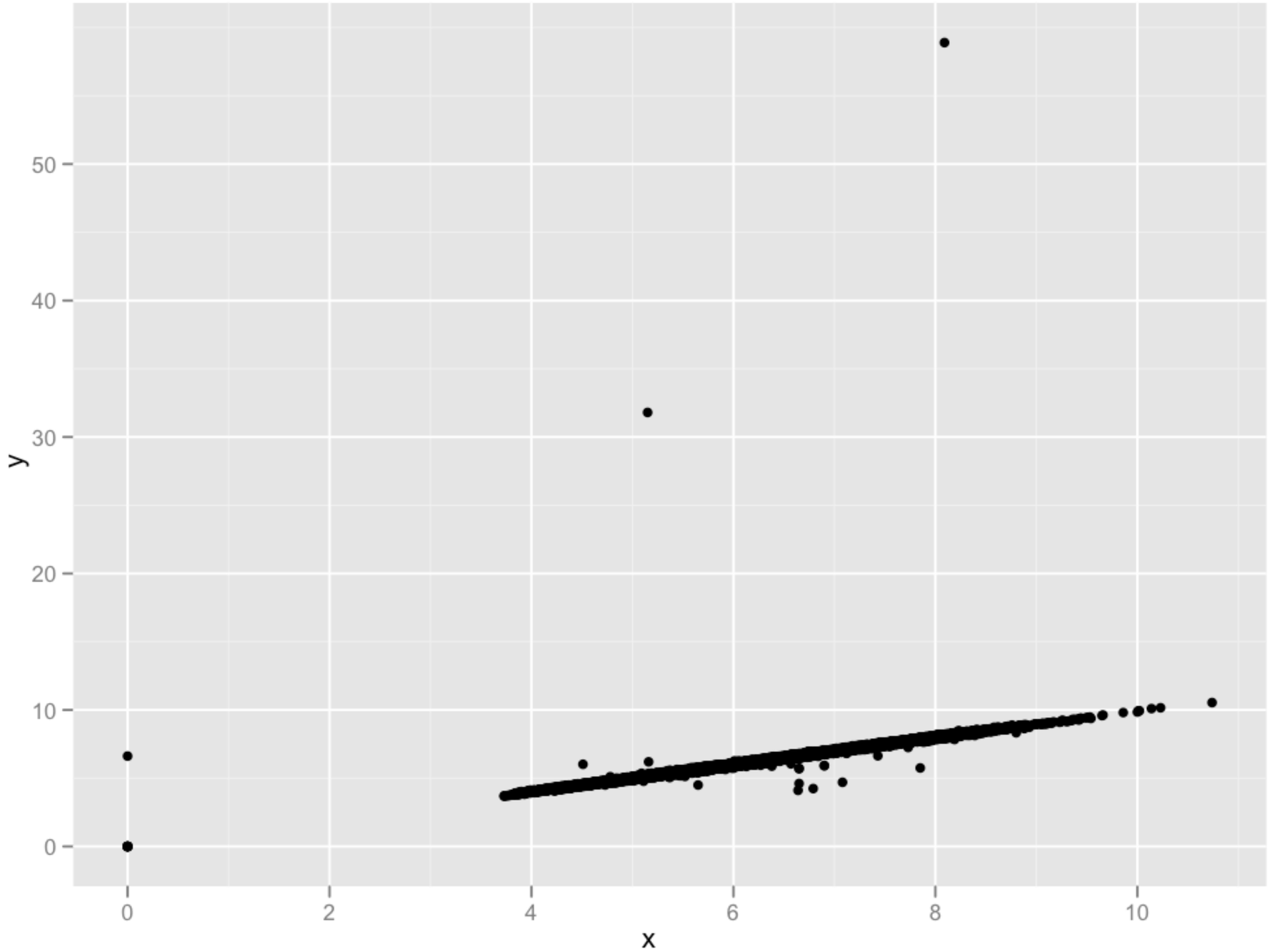
```
?diamonds
```

© 2014 RStudio, Inc. All rights reserved.



depth = z / diameter
table = table width / x * 100
x, y, z in mm

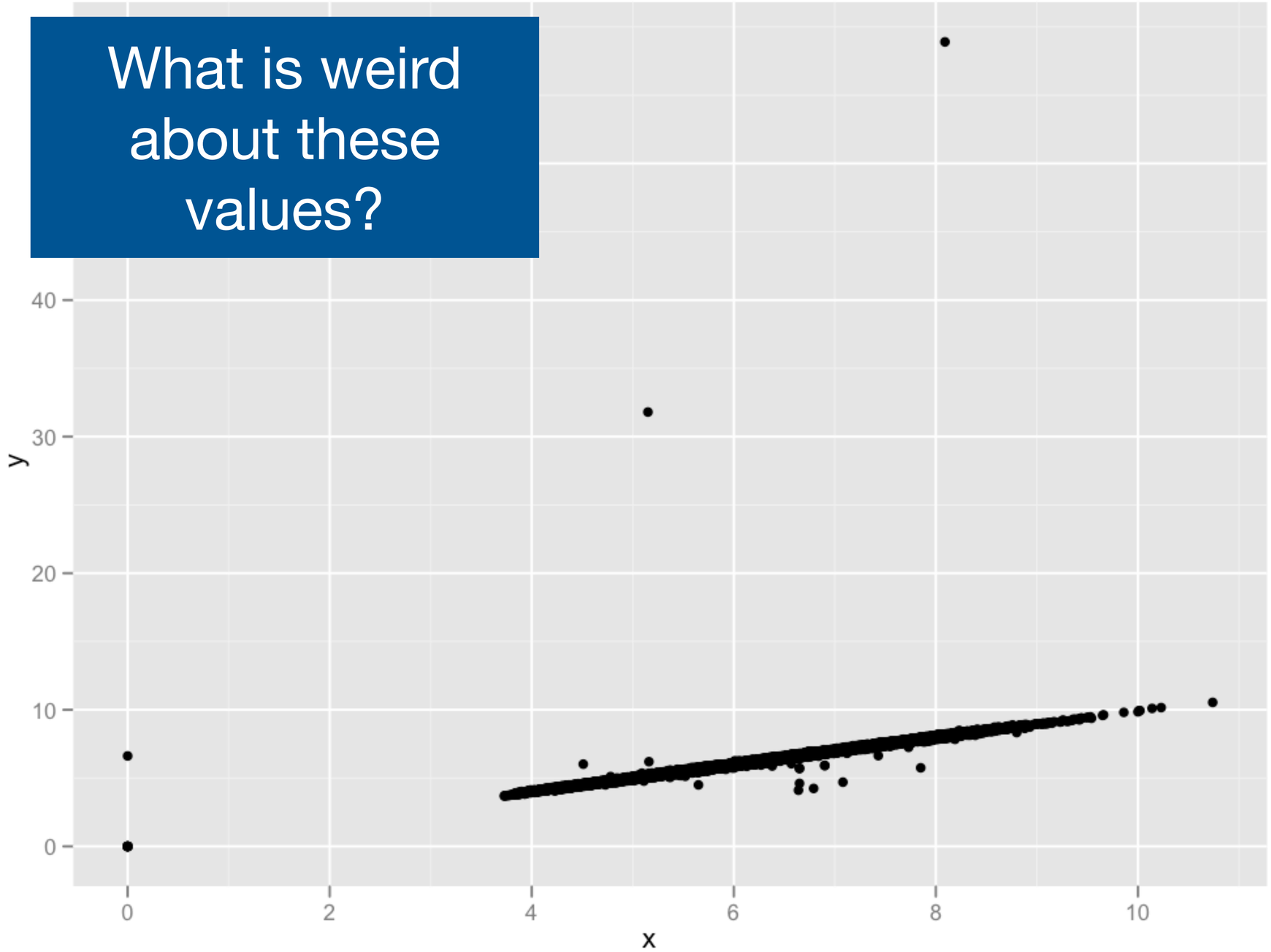
© 2014 RStudio, Inc. All rights reserved.




```
qplot(x, y, data = diamonds)
```

© 2014 RStudio, Inc. All rights reserved.

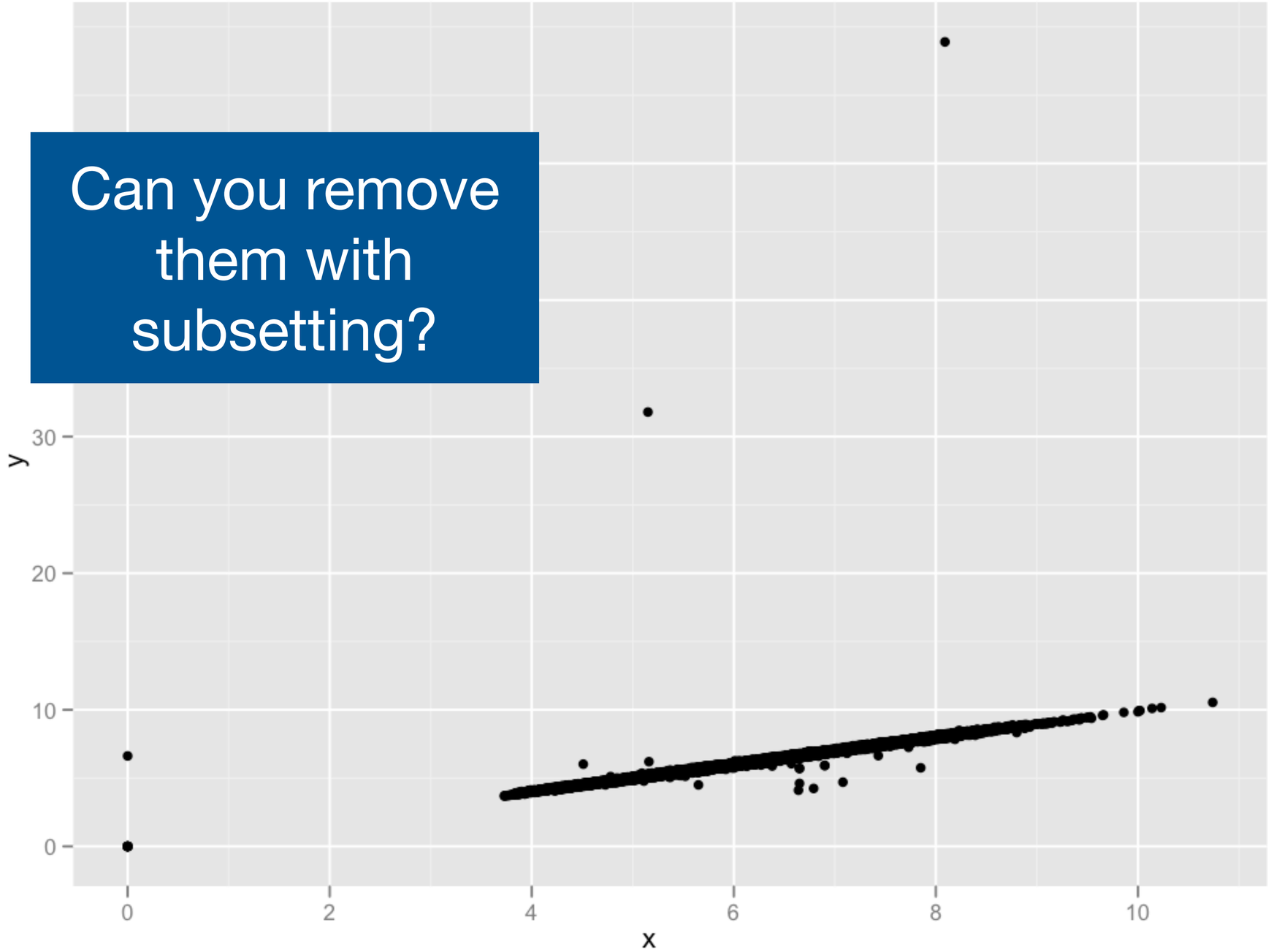
What is weird about these values?




```
qplot(x, y, data = diamonds)
```

© 2014 RStudio, Inc. All rights reserved.

Can you remove them with subsetting?




```
qplot(x, y, data = diamonds)
```

© 2014 RStudio, Inc. All rights reserved.

Logical tests

Logical comparisons

What will these return?

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

© 2014 RStudio, Inc. All rights reserved.

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		

© 2014 RStudio, Inc. All rights reserved.

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		

© 2014 RStudio, Inc. All rights reserved.

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		

© 2014 RStudio, Inc. All rights reserved.

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		

© 2014 RStudio, Inc. All rights reserved.

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		

© 2014 RStudio, Inc. All rights reserved.

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		

© 2014 RStudio, Inc. All rights reserved.

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 <-

© 2014 RStudio, Inc. All rights reserved.

%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)
```

© 2014 RStudio, Inc. All rights reserved.

%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
```

© 2014 RStudio, Inc. All rights reserved.

Boolean operators

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

```
x > 2 & x < 9
```

© 2014 RStudio, Inc. All rights reserved.

Boolean operators

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

```
x > 2 & x < 9
```



```
TRUE  &
```

© 2014 RStudio, Inc. All rights reserved.

Boolean operators

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

```
x > 2 & x < 9
```

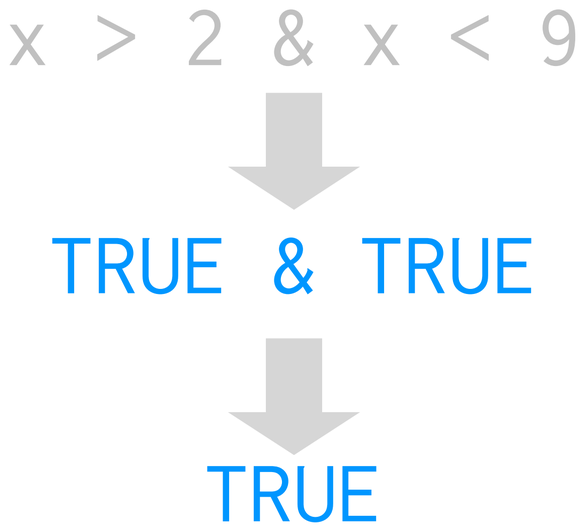


```
TRUE & TRUE
```

© 2014 RStudio, Inc. All rights reserved.

Boolean operators

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



© 2014 RStudio, Inc. All rights reserved.

&

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

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

© 2014 RStudio, Inc. All rights reserved.

|

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

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

© 2014 RStudio, Inc. All rights reserved.

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

© 2014 RStudio, Inc. All rights reserved.

!

Negation

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

© 2014 RStudio, Inc. All rights reserved.

any

Is **any** condition TRUE?

expression	outcome
any(c(TRUE, FALSE, FALSE))	TRUE
any(c(FALSE, FALSE, FALSE))	FALSE

© 2014 RStudio, Inc. All rights reserved.

all

Is **every** condition TRUE?

expression	outcome
all(c(TRUE, TRUE, TRUE))	TRUE
all(c(TRUE, FALSE, TRUE))	FALSE

© 2014 RStudio, Inc. All rights reserved.

Logical operators

operator	tests
x > y	is x greater than y?
x >= y	is x greater than or equal to y?
x < y	is x less than y?
x <= y	is x less than or equal to y?
x == y	is x equal to y?
x != y	is x not equal to y?
x %in% c(y, z)	is x in the set c(y, z)?

© 2014 RStudio, Inc. All rights reserved.

Boolean operators

operator	tests
a & b	both a and b are TRUE
a | b	at least one of a and b is TRUE (or)
xor(a, b)	a is TRUE or b is TRUE, but not both
!(a)	not a (TRUE goes to FALSE, FALSE goes to TRUE)
any(a, b, c)	at least one of a, b , or c is TRUE
all(a, b, c)	each of a, b, and c is TRUE

© 2014 RStudio, Inc. All rights reserved.

```
w <- c(-1, 0, 1)
x <- c(5, 15)
y <- "February"
z <- c("Monday", "Tuesday", "Friday")
```

© 2014 RStudio, Inc. All rights reserved.

Your turn

Turn these sentences into logical tests in R

Is `w` positive?

Is `x` greater than 10 and less than 20?

Is object `y` the word February?

Is every value in `z` a day of the week?

© 2014 RStudio, Inc. All rights reserved.

```
# Answers
w > 0
10 < x & x < 20
y == "February"
all(z %in% c("Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday", "Sunday"))
```

© 2014 RStudio, Inc. All rights reserved.

Common mistakes

```
x > 10 & < 20
y = "February"
all(z == "Monday" | "Tuesday" | "Wednesday"...)
```

© 2014 RStudio, Inc. All rights reserved.

```
x > 10 & < 20
```

© 2014 RStudio, Inc. All rights reserved.

```
x > 10 & < 20
```



```
TRUE &
```

© 2014 RStudio, Inc. All rights reserved.

```
x > 10 & < 20
```



```
TRUE & error!
```

© 2014 RStudio, Inc. All rights reserved.

```
x > 10 & < 20
```



```
TRUE & error!
```



```
error!
```

© 2014 RStudio, Inc. All rights reserved.

Logical subsetting

Logical subsetting

Combining logical tests with subsetting is a very powerful technique!

```
x_zeroes <- diamonds$x == 0
# FALSE FALSE FALSE FALSE FALSE FALSE ...

# What will this return?
diamonds[x_zeroes, ]
```

© 2014 RStudio, Inc. All rights reserved.

Saving results

```
# Prints to screen
diamonds[diamonds$x > 10, ]

# Saves to new data frame
big <- diamonds[diamonds$x > 10, ]

# Overwrites existing data frame.  Dangerous!
diamonds <- diamonds[diamonds$x < 10,]
```

© 2014 RStudio, Inc. All rights reserved.

```
diamonds <- diamonds[1, 1]
diamonds

# Uh oh!

rm(diamonds)
str(diamonds)

# Phew!
```

© 2014 RStudio, Inc. All rights reserved.

Missing values

Data errors

Typically removing the entire row because of one error is overkill. Better to selectively replace problem values with missing values.

In R, missing values are indicated by `NA`

© 2014 RStudio, Inc. All rights reserved.

Expression	Guess	Actual
`5 + NA`		
`mean(c(5, NA))`		
`NA < 3`		
`NA == 3`		
`NA == NA`		

© 2014 RStudio, Inc. All rights reserved.

Expression	Guess	Actual
`5 + NA`		`NA`
`mean(c(5, NA))`		
`NA < 3`		
`NA == 3`		
`NA == NA`		

© 2014 RStudio, Inc. All rights reserved.

Expression	Guess	Actual
`5 + NA`		NA
`mean(c(5, NA))`		NA
`NA < 3`		
`NA == 3`		
`NA == NA`		

© 2014 RStudio, Inc. All rights reserved.

Expression	Guess	Actual
`5 + NA`		NA
`mean(c(5, NA))`		NA
`NA < 3`		NA
`NA == 3`		
`NA == NA`		

© 2014 RStudio, Inc. All rights reserved.

Expression	Guess	Actual
`5 + NA`		NA
`mean(c(5, NA))`		NA
`NA < 3`		NA
`NA == 3`		NA
`NA == NA`		

© 2014 RStudio, Inc. All rights reserved.

Expression	Guess	Actual
`5 + NA`		NA
`mean(c(5, NA))`		NA
`NA < 3`		NA
`NA == 3`		NA
`NA == NA`		NA

© 2014 RStudio, Inc. All rights reserved.

NA Behavior

Missing values propagate

Use `is.na()` to check for missing values

```
a <- c(1, NA)
a == NA
# NA NA
is.na(a)
# FALSE TRUE
```

Many functions (e.g. `sum` and `mean`) have `na.rm` argument to remove missing values prior to computation.

© 2014 RStudio, Inc. All rights reserved.

na.rm

Many functions (e.g. `sum` and `mean`) have `na.rm` argument to remove missing values prior to computation.

```
b <- c(1, 2, 3, 4 NA)

sum(b)
# NA

sum(b, na.rm = TRUE)
# 10
```

© 2014 RStudio, Inc. All rights reserved.

Assignment

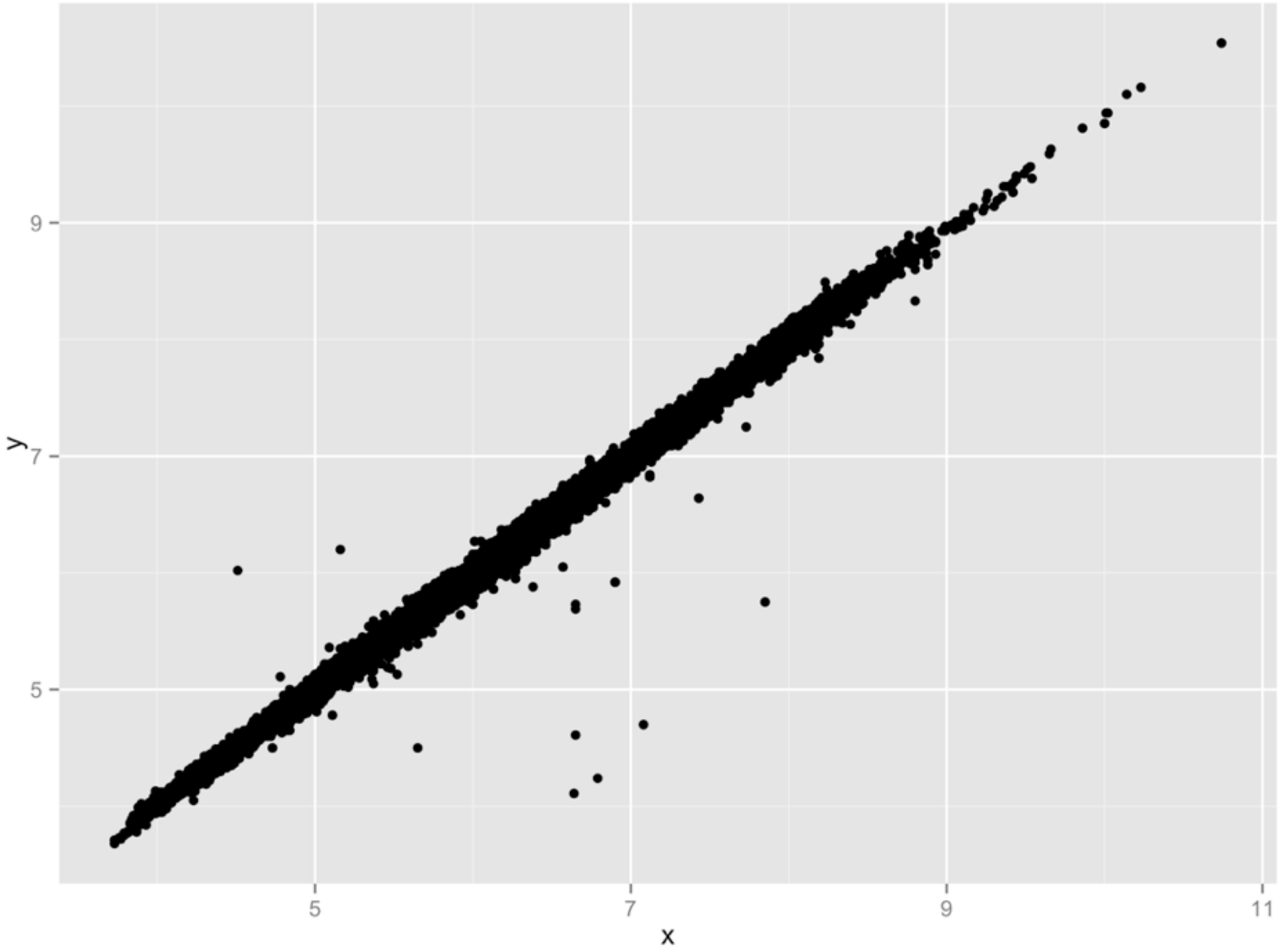
You can use subset notation with <- to change individual values within an object

```
summary(diamonds$x)

diamonds$x[diamonds$x == 0]
diamonds$x[diamonds$x == 0] <- NA

summary(diamonds$x)
```

© 2014 RStudio, Inc. All rights reserved.

```
summary(diamonds$y)

diamonds$y[diamonds$y == 0] <- NA

y_big <- diamonds$y > 20
diamonds$y[y_big] <- NA

summary(diamonds$y)
```

© 2014 RStudio, Inc. All rights reserved.




```
qplot(x, y, data = diamonds)
```

© 2014 RStudio, Inc. All rights reserved.