

Introduction to R and RStudio

BINF 3121

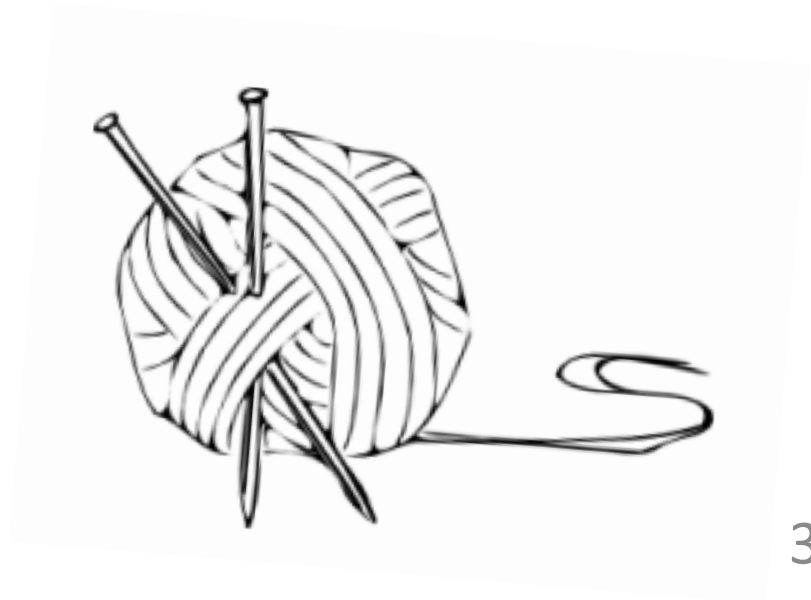
Why R?



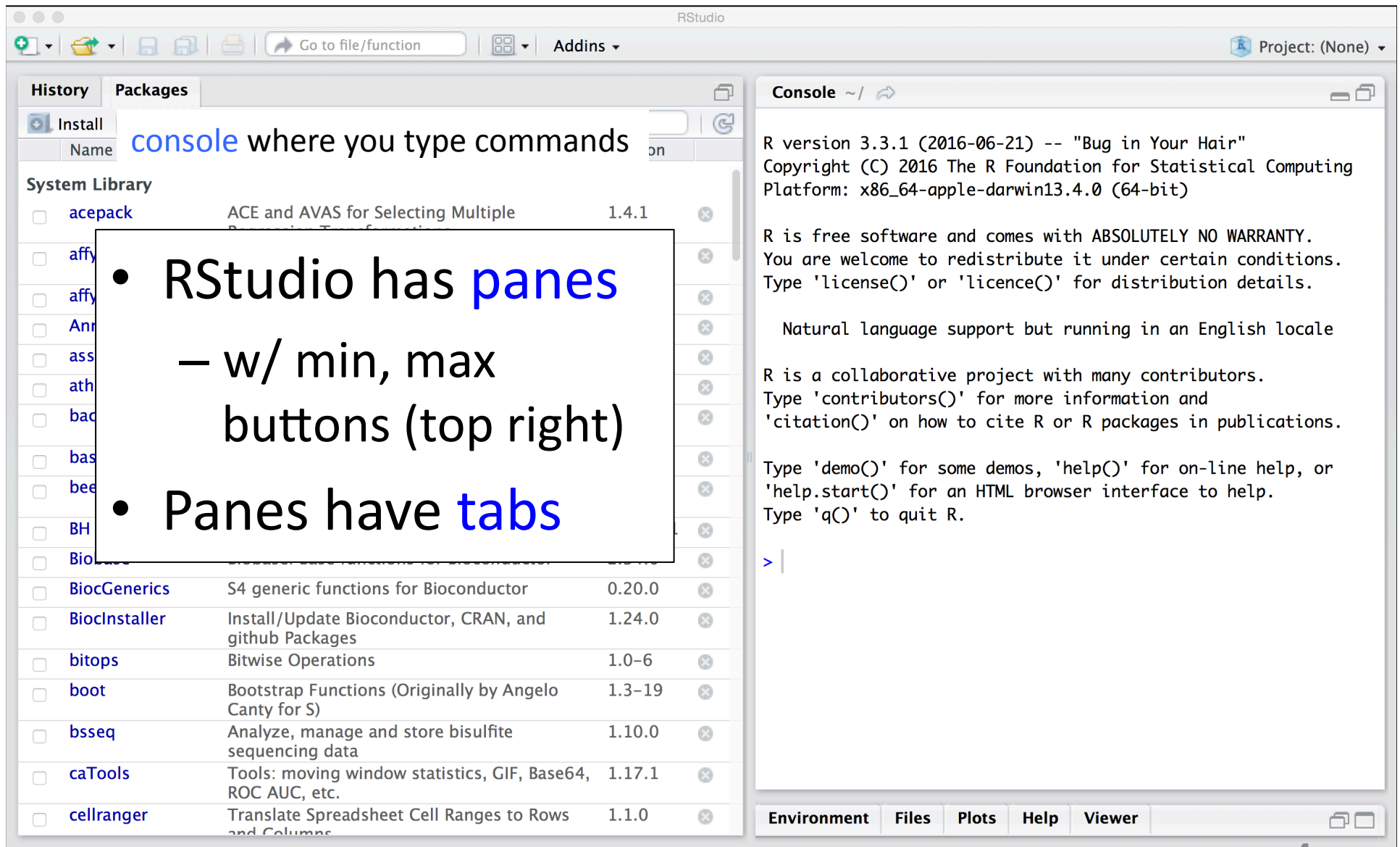
- Free & open source, popular
- Has a lot of support
 - Used in biology, finance, business analytics, statistics
 - Many tutorials available on-line
- Libraries available for biological data analysis through Bioconductor project
- Easy to use, free user interface - RStudio

RStudio

- A very nice graphical user interface for R.
- It's **free**!
- Integrates well with **knitr**
 - tool for writing statistical reports w/ **R markdown**

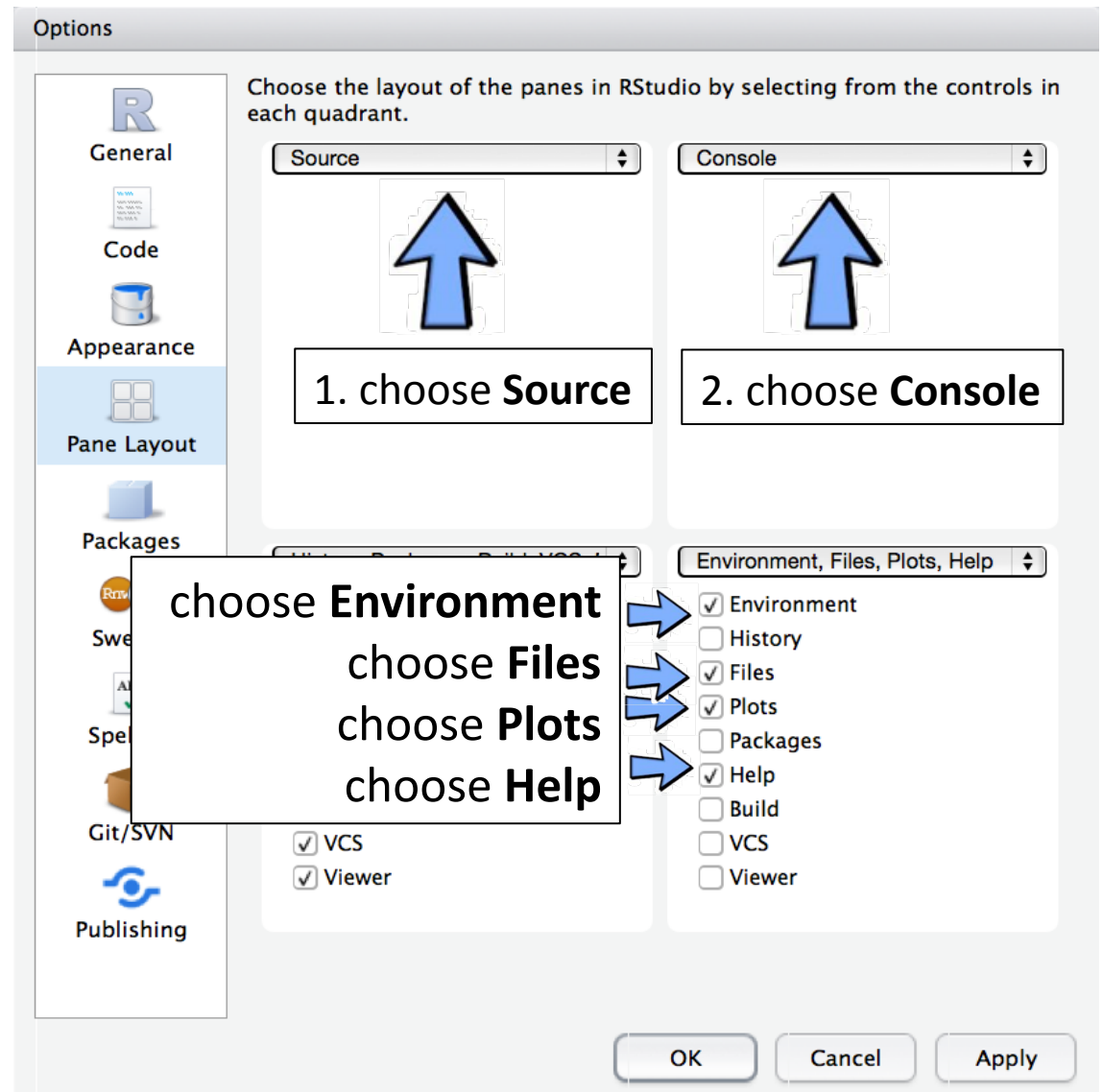


Start RStudio

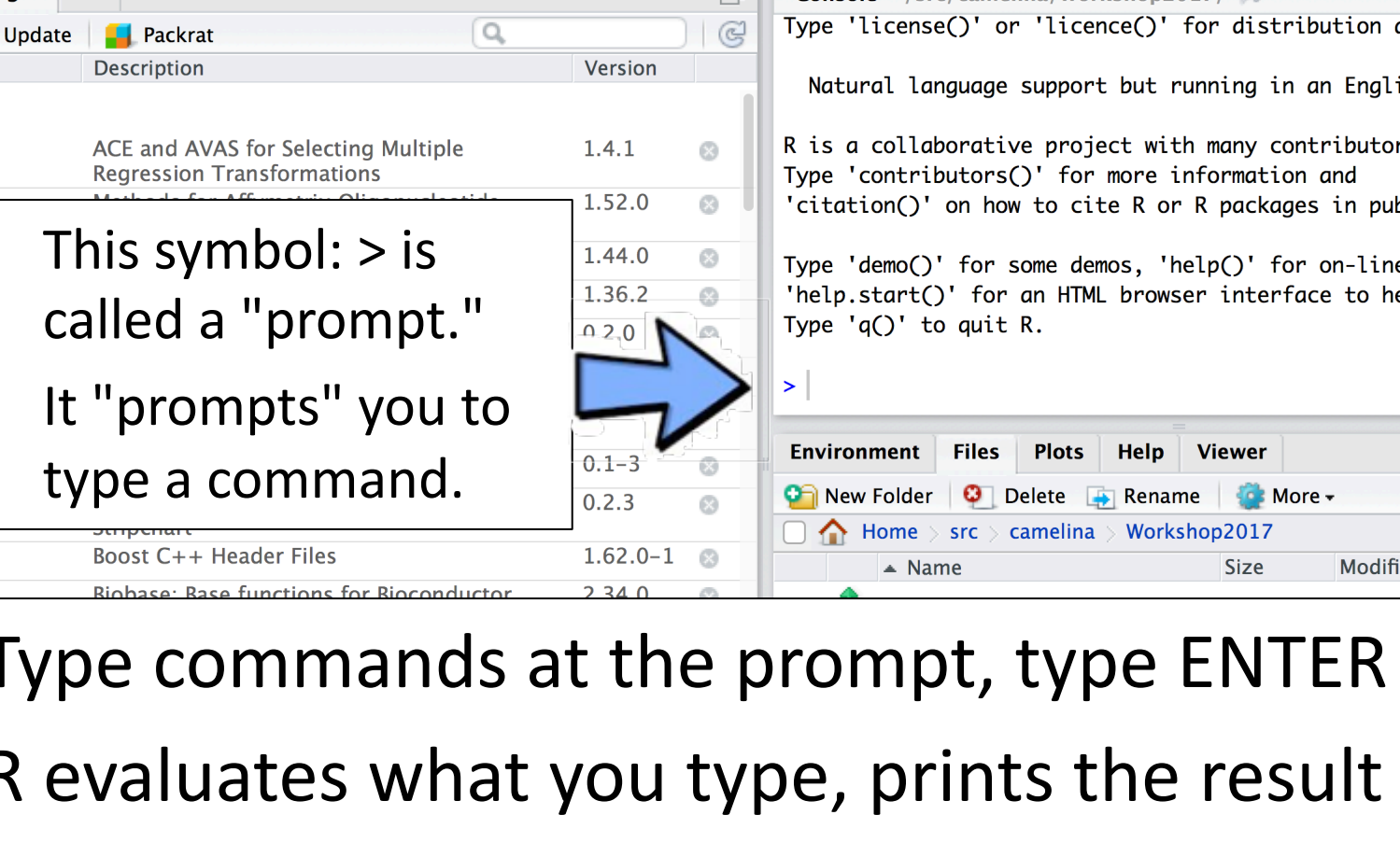


Customize RStudio panes

- Select **Tools > Global Options > Pane Layout**
- Configure:
 - Top left: **Source**
 - Top right: **Console**
 - Bottom right: **Environment, Files, Plots, Help**
 - Bottom left: everything else



Enter & run commands in **Console**



The screenshot shows the RStudio interface with the following components:

- Top Bar:** Shows the file path `~/src/camelina/Workshop2017 - master - RStudio` and the `Addins` menu.
- Left Pane:** Contains the **History**, **Packages**, and **Git** tabs. The **Packages** tab is active, showing a list of installed packages with columns for Name, Description, and Version. A blue arrow points from a text box to the prompt in the Console.
- Console:** Displays the R prompt `>` and the following text:

```
Type 'license()' or 'licence()' for distribution details.  
  
Natural language support but running in an English locale  
  
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.  
  
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.  
  
> |
```
- Environment Pane:** Shows the current environment with tabs for **Environment**, **Files**, **Plots**, **Help**, and **Viewer**. The **Environment** tab is active, showing a list of objects with columns for Name, Size, and Modified.

A blue arrow points from the text box to the prompt in the Console.

System Library

Name	Description	Version
acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
affy	Methods for Affymetrix Oligonucleotide Arrays	1.52.0
affyio	Interface to Affymetrix Oligonucleotide Arrays	1.44.0
Annotate	Annotation of Affymetrix Oligonucleotide Arrays	1.36.2
assertthat	Assertion that a value is what you expect	0.2.0
ath112	Annotation of Affymetrix Oligonucleotide Arrays	0.1-3
backports	Backports of R functions from newer versions of R	0.2.3
base64	Base64 encoding and decoding	0.1-3
beeswarm	Beeswarm plots for high density scatter plots	0.2.3
BH	Boost C++ Header Files	1.62.0-1
Biobase	Biobase: Base functions for Bioconductor	2.34.0

- This symbol: `>` is called a "prompt."
- It "prompts" you to type a command.

- Type commands at the prompt, type ENTER
- R evaluates what you type, prints the result
- Returns prompt

Practice: arithmetic expressions

> 5+6

[1] 11

> 5-6

[1] -1

> 2*5

[1] 10

> 2**4

[1] 16

- Add +
 - Subtract -
 - Multiply *
 - Raise to a power **
- Expressions return values as one-element **vectors**.
 - Also called **scalars**
 - [1] indicates that this is the first value in the vector
 - This is an **index** (an address)

R console tips

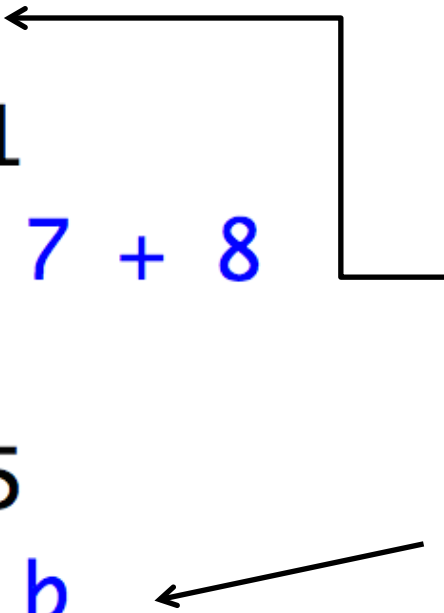
- Use UP arrow key to retrieve previous commands
 - Saves typing



- When you type a command, can type TAB to get a hint. (More on this later)
 - Makes writing commands much easier

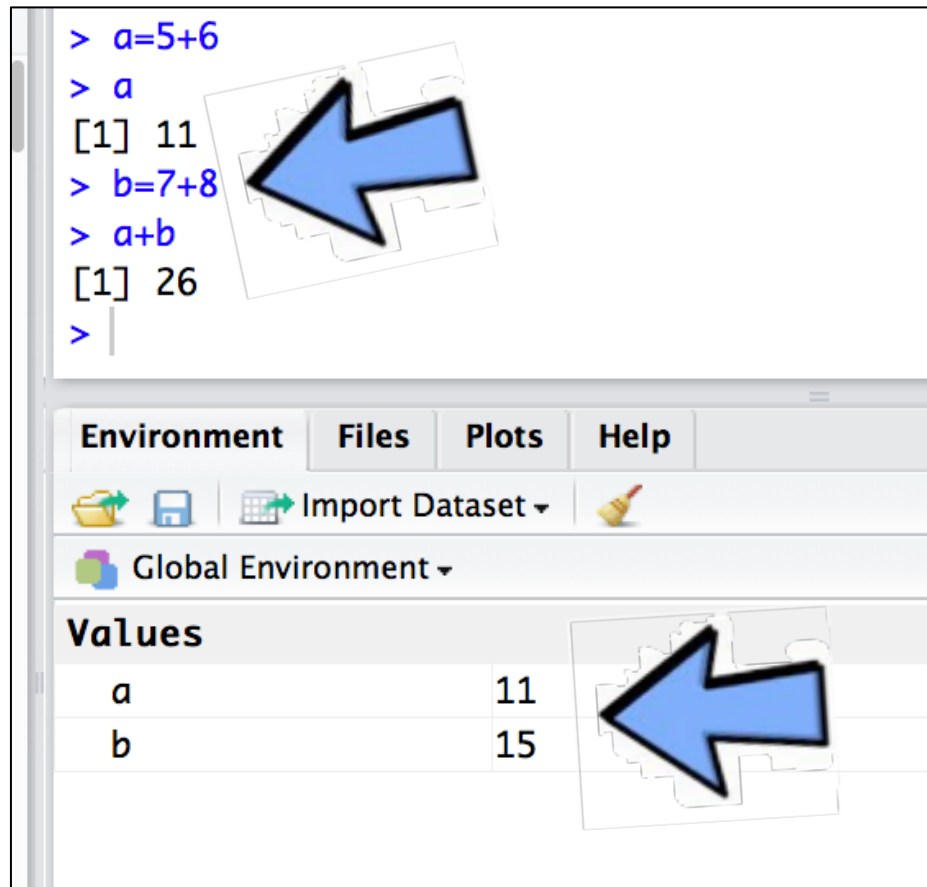
Practice: Save results to variables

```
> a = 5 + 6  
> a  
[1] 11  
> b = 7 + 8  
> b  
[1] 15  
> a + b  
[1] 26
```



- Use '=' to assign result to a variable
 - Nothing returned; just creates a variable
- Type variable name to see what's in it
- Use variables in expressions

Environment tab shows variables defined thus far



- Variables refer to **objects**
- Most of what you do in R involves interacting with objects

R has many functions

- R has functions for
 - plotting
 - statistical testing
 - predictive modeling
- Functions accept inputs called **arguments**

argument



```
> sqrt(4)
[1] 2
> sqrt(a)
[1] 3.316625
> |
```

How to use a function in 4 steps

1. Type function **name**
2. Type " (" open parenthesis
 - ★ RStudio types closing parenthesis for you
3. Type **argument**
 - if more than one argument, insert "," (comma) between them
4. Type **ENTER**

```
> sqrt(4)
[1] 2
> sqrt(a)
[1] 3.316625
> |
```

sqrt calculates
square root

Practice: R has ordered arguments and named arguments

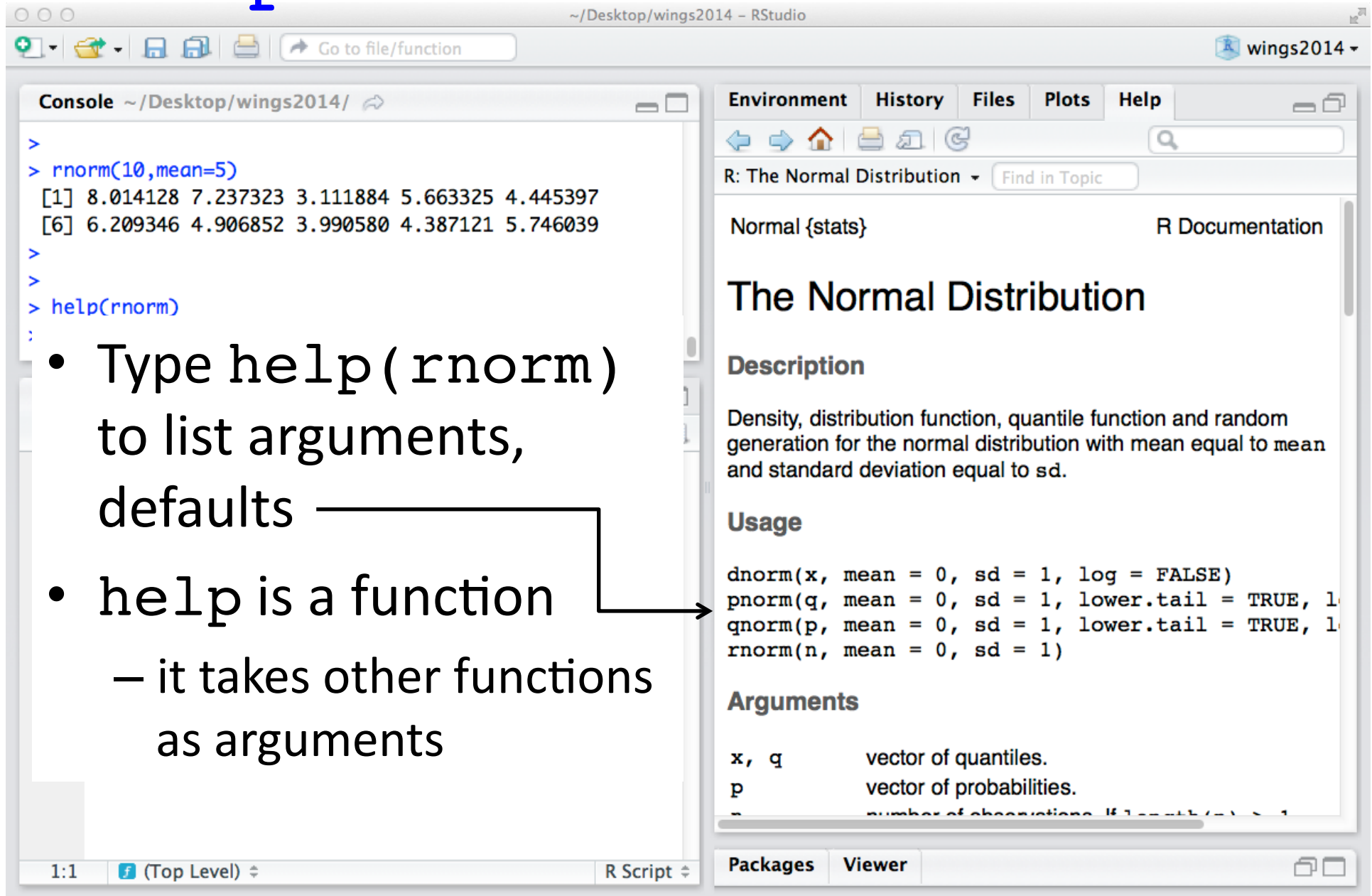
`rnorm(10 , mean=5 , sd=2)`



```
> rnorm(10,mean=5,sd=10)
[1]  5.992955  1.585141  8.044804 19.693630
[5] 22.016677 20.514699 10.653253 16.236811
[9] -10.182795  2.034047
> rnorm(10,sd=10,mean=5)
[1] -9.520978 -3.374612  8.046060 10.372663
[5]  5.750143 13.205880 11.925961 -15.266223
[9]  6.809343  3.808163
> rnorm(10,sd=10)
[1]  7.0928288 -0.5725398 -0.3306876 -0.5715225
[5]  8.3853718 -11.2119116 -2.8124066 -6.2559263
[9]  7.3724010 -15.7621108
```

Named
argument
order can vary

help shows how to use a function



The screenshot shows the RStudio interface with the following components:

- Console:** Displays the execution of `rnorm(10, mean=5)` resulting in 10 random values, and the command `help(rnorm)`.
- Environment/History/Files/Plots/Help:** The 'Help' tab is active, showing the documentation for 'The Normal Distribution'.
- Help Panel:** Contains the title 'The Normal Distribution', a 'Description' section explaining the density, distribution, quantile, and random generation functions, a 'Usage' section listing `dnorm`, `pnorm`, `qnorm`, and `rnorm` with their default arguments, and an 'Arguments' section defining `x`, `q`, `p`, and `n`.

- Type `help(rnorm)` to list arguments, defaults
- `help` is a function — it takes other functions as arguments

Practice: Use `c` to create new vectors

- Make a new **character** vector and save it to a variable

```
char_vec = c("How", "are", "you?")
```

- Make a new **numeric** vector and save it to a variable

```
num_vec = c(1, 23, 5)
```

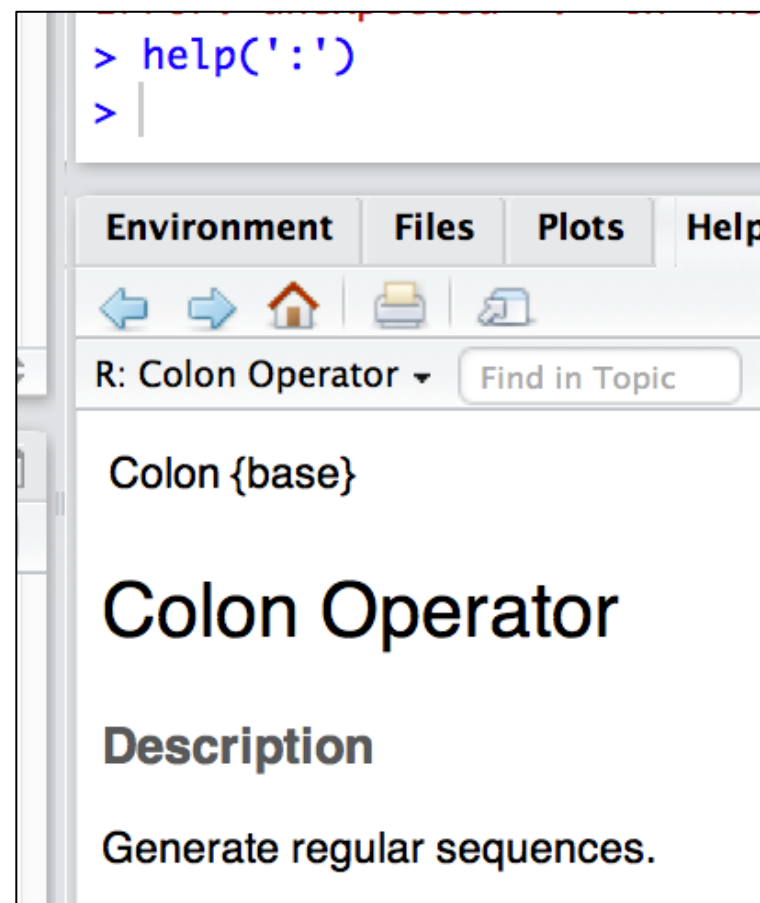
- Make a new **logical** vector and save it to a variable

```
logi_vec = c(TRUE, FALSE, TRUE)
```

Practice: Use `:` as a shortcut to make new numeric vectors

- Using colon operator:
 - `num_vec = 1:10`
 - `another_vec = 0:50`
 - `backward = 10:1`

Useful shortcut when subscripting - you'll see it a lot today.



Practice: Use `==` to test for equality & make new logical vectors

- Make two character vectors

```
a = c("I", "like", "coffee.")
```

```
b = c("I", "like", "tea.")
```

- Compare with `==`:

```
> a == b
```

```
[1] TRUE TRUE FALSE
```

Practice: Use square brackets `[]`
(subscripting) to get elements from vectors

- Make a vector:

```
a = 10:1
```

- Get the 5th element:

```
a[5]
```

- Get the 2nd to the 5th elements:

```
a[2:5]
```

- Get elements in any order; ***repeats are OK***

```
a[c(10, 8, 1, 1)]
```

The thing inside the
square brackets is
called an **index**.



a[?]

Practice: Use a logical vector as an index to retrieve elements from a vector

- Make a numeric vector:

```
> b = 1:5
```

```
> b
```

```
[1] 1 2 3 4 5
```

- Make a logical vector

```
> v = b < 3
```

```
> v
```

```
[1] TRUE TRUE FALSE FALSE FALSE
```

- Use it to get elements of b that are < 3

```
> b[v]
```

```
[1] 1 2
```

Vectors can have **character** indices

- Use names to give each element a name

```
> a = 1:3
```

```
> names(a) = c("Mary", "loves", "John")
```

```
> a
```

```
  Mary loves   John
    1      2      3
```

- Character vectors retrieve elements **by name**

```
> a[c("John", "loves", "Mary")]
```

```
John loves   Mary
    3      2      1
```

Matrices are two-dimension vectors - they have rows & columns

- Use **matrix** to make a matrix from a vector

```
> a = 1:15
```

```
> a_matrix = matrix(a,nrow=3)
```

```
> a_matrix
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	4	7	10	13
[2,]	2	5	8	11	14
[3,]	3	6	9	12	15

Use square brackets with comma to retrieve data from a matrix

- Get elements from where the specified rows & columns **overlap**

– `a_matrix [rows, columns]`

Can be single value or a vector

```
> a_matrix[1:2, 3:4]
```

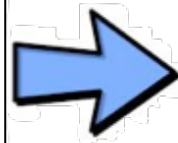
	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	4	7	10	13
[2,]	2	5	8	11	14
[3,]	3	6	9	12	15

	[,1]	[,2]
[1,]	7	10
[2,]	8	11

Empty *rows* position means "all rows"

```
> a_matrix[,3:4]
      [,1] [,2] [,3] [,4] [,5]
[1,]      1      4      7     10     13
[2,]      2      5      8     11     14
[3,]      3      6      9     12     15
```

The returned
matrix has new
indices




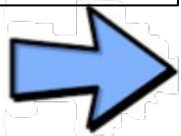
```
> a_matrix[,3:4]
      [,1] [,2]
[1,]      7     10
[2,]      8     11
[3,]      9     12
```

Empty *columns* position means all columns

```
> a_matrix[c(1,3),]
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	4	7	10	13
[2,]	2	5	8	11	14
[3,]	3	6	9	12	15

same
elements,
new
indices



	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	4	7	10	13
[2,]	3	6	9	12	15

Matrices can have **character** indices

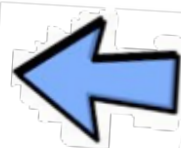
```
> colnames(a_matrix)=c("T1","T2","C1","C2","C3")
> rownames(a_matrix)=c("G1","G2","G3")
> a_matrix
```

	T1	T2	C1	C2	C3
G1	1	4	7	10	13
G2	2	5	8	11	14
G3	3	6	9	12	15

Tip: use **column** and **row names** to make your code more readable!

```
> a_matrix[c("G1","G3"),c("C1","C3")]
```

	C1	C3
G1	7	13
G3	9	15

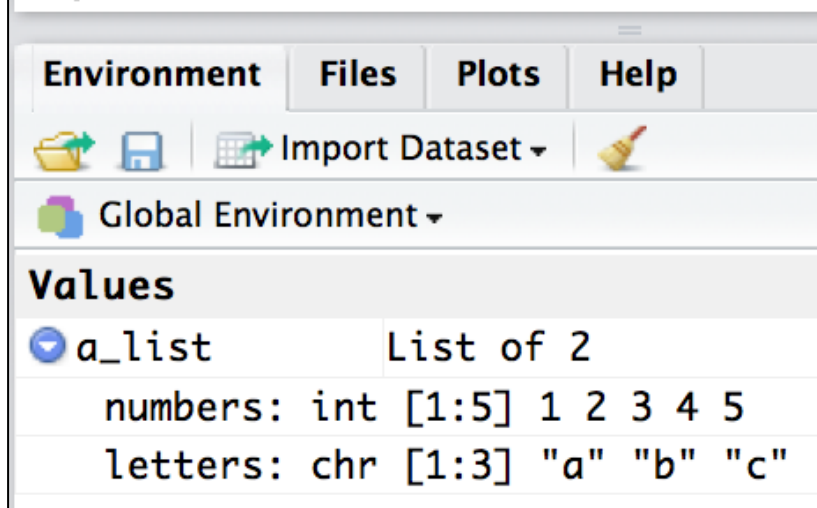


Observe how the new matrix uses original row & column names.

A **list** is a general-purpose container for other objects

- The **list** command makes a new list
- **New**: Use dollar operator **\$** to assign or retrieve items from a list **by name**
- Use the **Environment** tab or **names** command to find out what a list contains

```
> a_list = list()
> a_list$numbers = 1:5
> a_list$letters = c("a","b","c")
> a_list$letters
[1] "a" "b" "c"
> a_list$numbers
[1] 1 2 3 4 5
> names(a_list)
[1] "numbers" "letters"
> |
```



The screenshot shows the RStudio Environment pane. At the top are tabs for 'Environment', 'Files', 'Plots', and 'Help'. Below these are icons for file operations and a button labeled 'Import Dataset'. The main area shows the 'Global Environment' with a section titled 'Values'. Under 'Values', the object 'a_list' is listed as a 'List of 2'. It contains two elements: 'numbers' of type 'int' with values '1 2 3 4 5', and 'letters' of type 'chr' with values '"a" "b" "c"'.

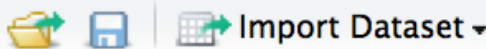
Values	
a_list	List of 2
numbers: int [1:5]	1 2 3 4 5
letters: chr [1:3]	"a" "b" "c"

A **data frame** is a list with rows & columns

- It's like a matrix
 - Rows are same length
 - Columns are same length
- The **data.frame** command makes data frames
- Use dollar operator **\$** to retrieve columns as vectors or assign new values

```
df$numbers=4:6
df$numbers[1]=2
df$numbers[1:2]=c(3,1)
```
- Use the **Environment** tab or **names** command to find out what columns a data frame contains

```
> letters = c("a","b","c")
> numbers = 1:3
> df = data.frame(letters,numbers)
> df
  letters numbers
1      a        1
2      b        2
3      c        3
> df$numbers
[1] 1 2 3
> names(df)
[1] "letters" "numbers"
> |
```

Environment	Files	Plots	Help
			
Global Environment			
Data			
df	3 obs. of 2 variables		
Values			
letters	chr [1:3]	"a" "b" "c"	
numbers	int [1:3]	1 2 3	

Use `[]` to get data from a data frame - just like with a matrix

- Uses same syntax as for a matrix
 - Subscripting accepts
 - single values
 - vectors
 - Assign names to **rows** using `rownames`
 - Tip: use logical vectors to extract rows that satisfy a condition:
 - Example - retrieve **rows** where numbers column > 1
- ```
df[df$numbers>1,]
```

```
> df[1,2]
[1] 1
> df[1,1:2]
 letters numbers
1 a 1
> df[,1:2]
 letters numbers
1 a 1
2 b 2
3 c 3
> rownames(df)=c("1st","2nd","3rd")
> df
 letters numbers
1st a 1
2nd b 2
3rd c 3
> |
```

# Data frame and list ideas you'll see again

- Lists can contain ***anything***
  - vectors, matrices, data frames, other lists
  - Bioconductor makes heavy use of list-based objects to store results from statistical testing
    - `DGEList`, `DGELRT`
- Data frames mainly used to store and manage data imported into R from files
  - use `read.delim` (or similar) commands to read data from files into data frames