Do you have a github account?
Give us a star! :)

Welcome!

Please download **all of** the course material:

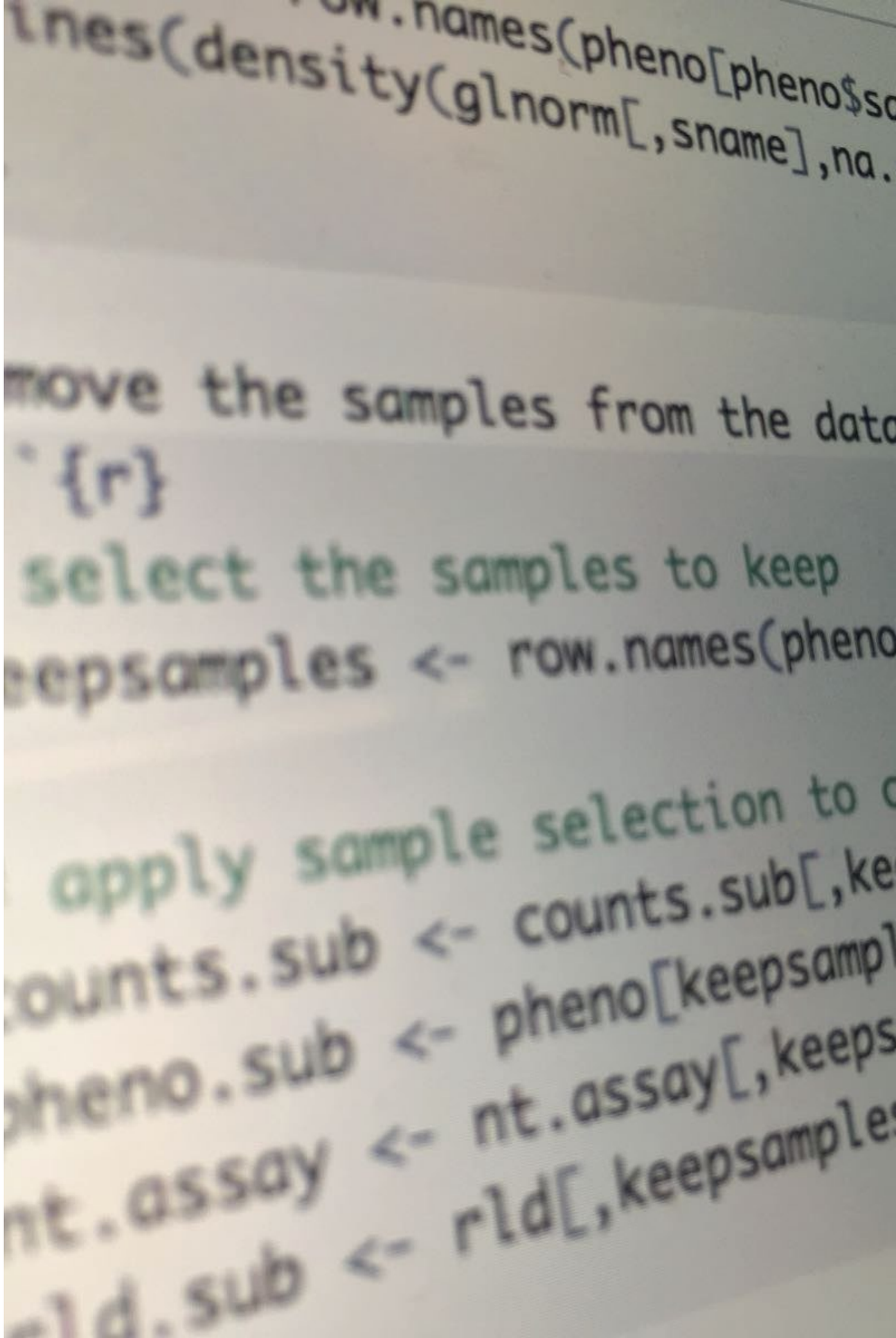
---

tinyurl.com/introRData > Clone or download > Download ZIP

and store it in a single, accessible folder on your computer. **Don’t forget to unzip!**

Introduction to R & data

- Part I: Basics of R
- Lunch (~12:15)
- Part II: data handling with tidyverse
- Final remarks (~17:00)



Quick intro: this is us! And who are you?

- What is your faculty/background? (Economics, Medicine, Biology…)
- What part of your education are you in? (Bachelor, PhD, prof…)
- What is your motivation for learning R?
- What is your experience with R?

What's with those sticky notes?

I need some assistance, please!

I have finished the exercise and am ready to move on!

No sticky note?

Introduction to R & data

Part 1: Basics of R

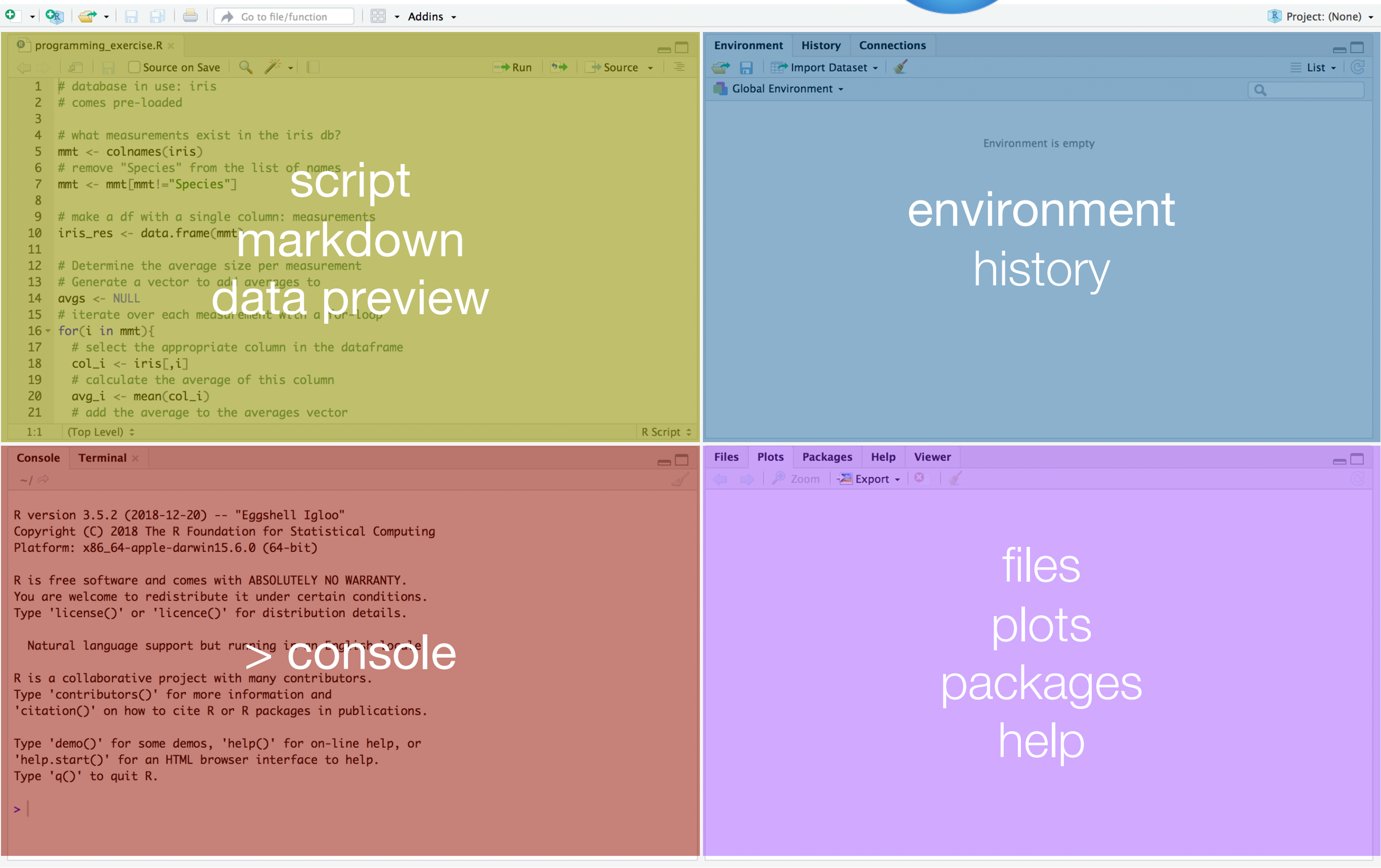
What is R?

- Widely used programming language for data analysis
- Based on statistical programming language **S** (1976)
- Developed by **R**oss Ihaka & **R**obert Gentleman (1995)
- Very active community, with many (often subject-specific) packages

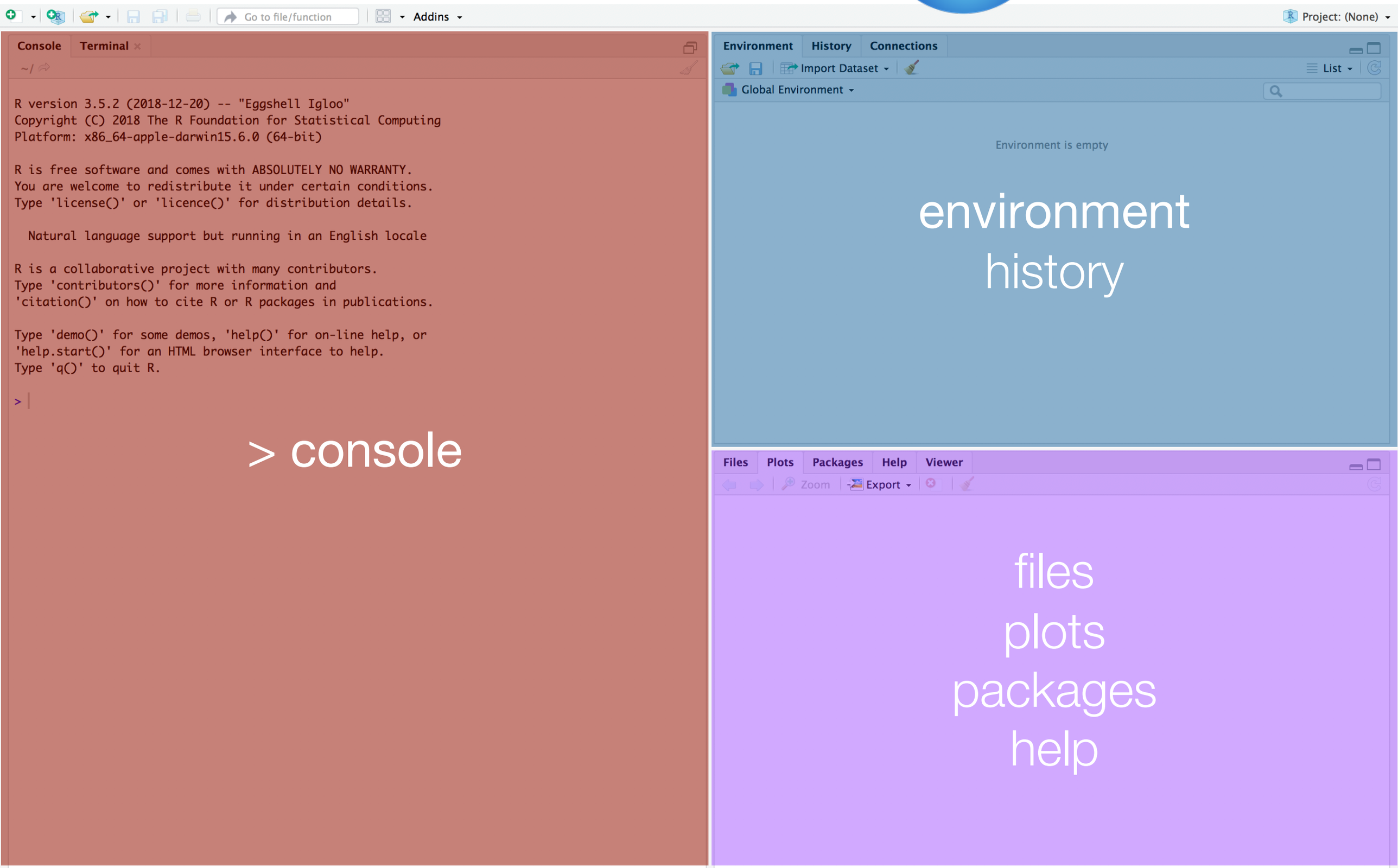
We will work in RStudio

- Integrated Development Environment for R
- Founded by JJ Allaire, available since 2010
- Bloody useful! Let’s take a look: please open RStudio!

The Rstudio interface



The Rstudio interface



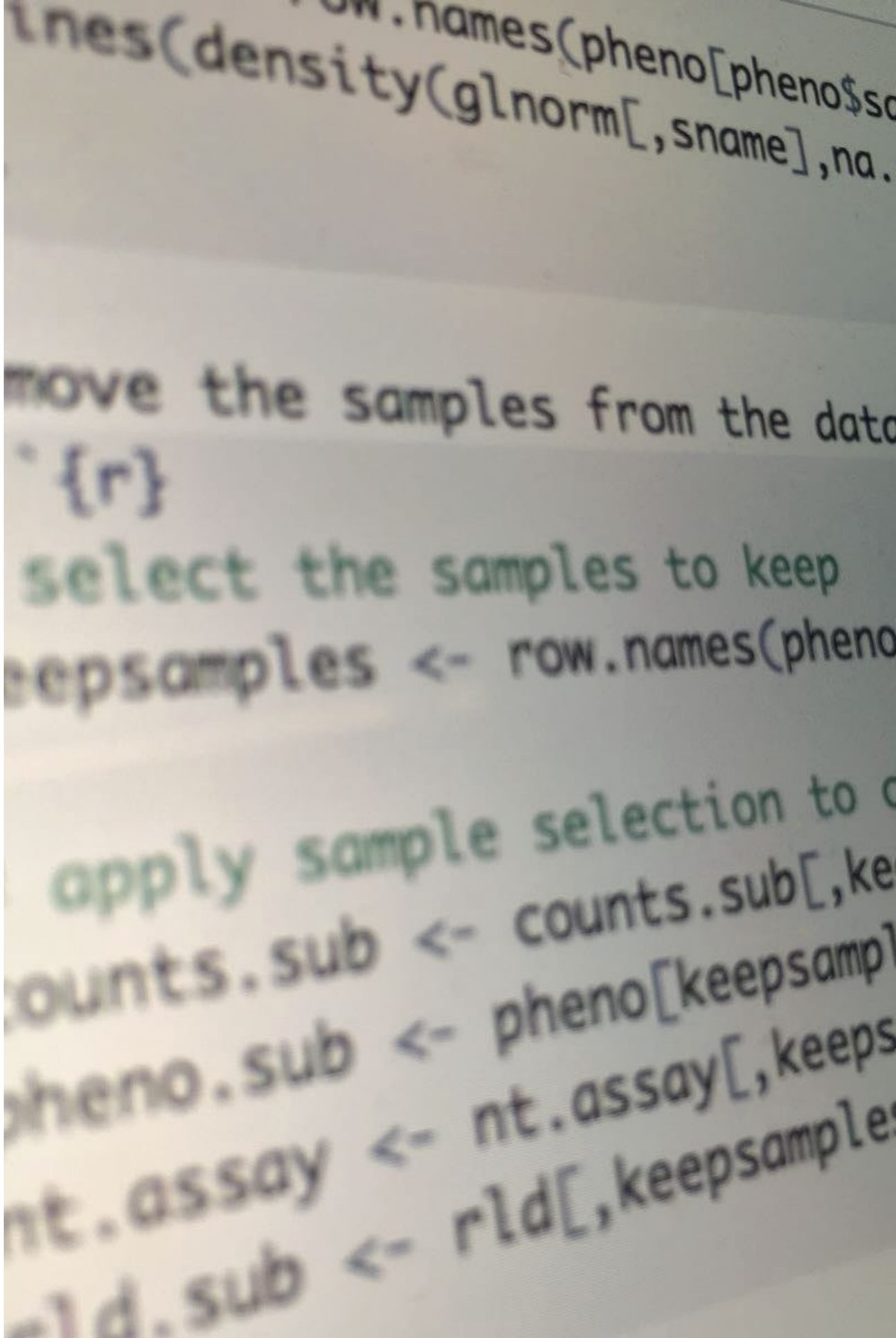
Have you downloaded the course material?

tinyurl.com/introRData > Clone or download > Download ZIP

Please store it in a single, accessible folder on your computer. **Don’t forget to unzip!**

R syntax & the console

- Data types in R
- Using functions
- Combining data
- Indexing data



A quick note on notes

- The console is for execution, not for storage
- Everything we do is on the slides!
- BUT: you can store, if you want, by copy-pasting to a text document
- Do you want to write notes in between?

```
# write your notes in the console like this
# using a #hash sign.
# pressing enter will do nothing.
# go ahead and try!
```

How to read our slides (console version)

you are writing in the console

```
> x <- 1
```

giving R information

console output

```
> x
[1] 1
```

asking R a question

index of output

Variable assignment

```
>
```

Variable assignment

```
> x <- 6
```

Variable assignment

```
> x <- 6

> x <- "hi!"

> 6 -> x

> x = 6

> 6 = x
Error in 6 = x : invalid (do_set)
```

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.
help.search('weighted mean')
Search the help files for a word or phrase.
help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.
class(iris)
Find the class an object belongs to.

Using Packages

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

c(2, 4, 6)	2 4 6	Join elements into a vector
2:6	2 3 4 5 6	An integer sequence
seq(2, 3, by=0.5)	2.0 2.5 3.0	A complex sequence
rep(1:2, times=3)	1 2 1 2 1 2	Repeat a vector
rep(1:2, each=3)	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x) Return x sorted.
rev(x) Return x reversed.
table(x) See counts of values.
unique(x) See unique values.

Selecting Vector Elements

By Position

x[4]	The fourth element.
x[-4]	All but the fourth.
x[2:4]	Elements two to four.
x[-(2:4)]	All elements except two to four.
x[c(1, 5)]	Elements one and five.

By Value

x[x == 10]	Elements which are equal to 10.
x[x < 0]	All elements less than zero.
x[x %in% c(1, 2, 5)]	Elements in the set 1, 2, 5.

Named Vectors

x['apple']	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

While Loop

```
while (condition){
   Do something
}
```

Example

```
while (i < 5){
   print(i)
   i <- i + 1
}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){
   Do something
   return(new_variable)
}
```

Example

```
square <- function(x){
   squared <- x*x
   return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
df <- read.table('file.txt')	write.table(df, 'file.txt')	Read and write a delimited text file.
df <- read.csv('file.csv')	write.csv(df, 'file.csv')	Read and write a comma separated value file. This is a special case of read.table/ write.table.
load('file.RData')	save(df, file = 'file.Rdata')	Read and write an R data file, a file type special for R.

Conditions

a == b	Are equal	a > b	Greater than	a >= b	Greater than or equal to	is.na(a)	Is missing
a != b	Not equal	a < b	Less than	a <= b	Less than or equal to	is.null(a)	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

as.logical	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
as.numeric	1, 0, 1	Integers or floating point numbers.
as.character	'1', '0', '1'	Character strings. Generally preferred to factors.
as.factor	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Maths Functions

log(x)	Natural log.	sum(x)	Sum.
exp(x)	Exponential.	mean(x)	Mean.
max(x)	Largest element.	median(x)	Median.
min(x)	Smallest element.	quantile(x)	Percentage quantiles.
round(x, n)	Round to n decimal places.	rank(x)	Rank of elements.
signif(x, n)	Round to n significant figures.	var(x)	The variance.
cor(x, y)	Correlation.	sd(x)	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

ls() List all variables in the environment.
rm(x) Remove x from the environment.
rm(list = ls()) Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

m <- matrix(x, nrow = 3, ncol = 3)
Create a matrix from x.

m[2,] - Select a row
m[, 1] - Select a column
m[2, 3] - Select an element

t(m) Transpose
m %*% n Matrix Multiplication
solve(m, n) Find x in: m * x = n

Lists

l <- list(x = 1:5, y = c('a', 'b'))
A list is a collection of elements which can be of different types.

l[[2]]	l[1]	l$x	l['y']
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** package.

Data Frames

df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

df$x **df[[2]]**

Understanding a data frame

View(df) See the full data frame.
head(df) See the first 6 rows.

Matrix subsetting

df[, 2] **df[2,]** **df[2, 2]**

nrow(df) Number of rows.
ncol(df) Number of columns.
dim(df) Number of columns and rows.

cbind - Bind columns.
rbind - Bind rows.

Strings

Also see the **stringr** package.

paste(x, y, sep = ' ')	Join multiple vectors together.
paste(x, collapse = ' ')	Join elements of a vector together.
grep(pattern, x)	Find regular expression matches in x.
gsub(pattern, replace, x)	Replace matches in x with a string.
toupper(x)	Convert to uppercase.
tolower(x)	Convert to lowercase.
nchar(x)	Number of characters in a string.

Factors

factor(x) Turn a vector into a factor. Can set the levels of the factor and the order.
cut(x, breaks = 4) Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

lm(y ~ x, data=df) Linear model.
glm(y ~ x, data=df) Generalised linear model.
summary Get more detailed information out a model.

t.test(x, y) Perform a t-test for difference between means.
pairwise.t.test Perform a t-test for paired data.

prop.test Test for a difference between proportions.
aov Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Poisson	rpois	dpois	ppois	qpois
Binomial	rbinom	dbinom	pbinom	qbinom
Uniform	runif	dunif	punif	qunif

Plotting

Also see the **ggplot2** package.

plot(x) Values of x in order.
plot(x, y) Values of x against y.
hist(x) Histogram of x.

Dates

See the **lubridate** package.

Maths Functions

```
> x * 3
[1] 18

> y <- x + 2

> log2(y)
[1] 3
```

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.
help.search('weighted mean')
Search the help files for a word or phrase.
help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.
class(iris)
Find the class an object belongs to.

Using Packages

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x) Return x sorted.
rev(x) Return x reversed.
table(x) See counts of values.
unique(x) See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){
   Do something
}
```

Example

```
for (i in 1:4){
   j <- i + 10
   print(j)
}
```

While Loop

```
while (condition){
   Do something
}
```

Example

```
while (i < 5){
   print(i)
   i <- i + 1
}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){
   Do something
   return(new_variable)
}
```

Example

```
square <- function(x){
   squared <- x*x
   return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions

`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	TRUE, FALSE, TRUE	Boolean values (TRUE or FALSE).
`as.numeric`	1, 0, 1	Integers or floating point numbers.
`as.character`	'1', '0', '1'	Character strings. Generally preferred to factors.
`as.factor`	'1', '0', '1', levels: '1', '0'	Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row
`m[ , 1]` - Select a column
`m[2, 3]` - Select an element

`t(m)` Transpose
`m %*% n` Matrix Multiplication
`solve(m, n)` Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Data Frames

Also see the **dplyr** package.

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x` `df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

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

`nrow(df)` Number of rows.
`ncol(df)` Number of columns.
`dim(df)` Number of columns and rows.

`cbind` - Bind columns.
`rbind` - Bind rows.

Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

factor(x) Turn a vector into a factor. Can set the levels of the factor and the order.
cut(x, breaks = 4) Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

lm(y ~ x, data=df) Linear model.
glm(y ~ x, data=df) Generalised linear model.
summary Get more detailed information out a model.

t.test(x, y) Perform a t-test for difference between means.
pairwise.t.test Perform a t-test for paired data.

prop.test Test for a difference between proportions.
aov Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** package.

`plot(x)` Values of x in order.
`plot(x, y)` Values of x against y.
`hist(x)` Histogram of x.

Dates

See the **lubridate** package.

Exercise

1. Do the following calculation in R:

$$\frac{1+5}{9}$$

2. Assign the result of the calculation to a variable.

3. **Bonus:** Round off the result to 1 decimal. *Tip: Use the **Maths Functions** section of your cheat sheet!*

Exercise

1. Do the following calculation in R:

$$\frac{1+5}{9}$$

```
> (1+5)/9
```

2. Assign the result of the calculation to a variable.

```
> MyCalc <- (1+5)/9
```

3. **Bonus:** Round off the result to 1 decimal. *Tip: Use the **Maths Functions** section of your cheat sheet!*

```
> round(MyCalc,1)
```

Another data type: logical

T	TRUE
F	FALSE

```
> TRUE
[1] TRUE

> F
[1] FALSE

> x==6
[1] TRUE

> 2>4
[1] FALSE
```

==	is equal to
!=	is not
>=	larger than or equal to
<	smaller than

Combining data: creating vectors

```
> c(1,2,3)
[1] 1  2  3
```

a **numeric** vector

```
> c("a","b","c")
[1] "a" "b" "c"
```

a **character** vector

```
> c(T,TRUE,F)
[1] TRUE  TRUE FALSE
```

a **logical** vector

Vector functions

```
> p <- 1:5
> p
[1] 1 2 3 4 5

> mean(p)
[1] 3

> p * 2
[1] 2 4 6 8 10

> q <- 5:1
> q
[1] 5 4 3 2 1

> p * q
[1] 5 8 9 8 5
```

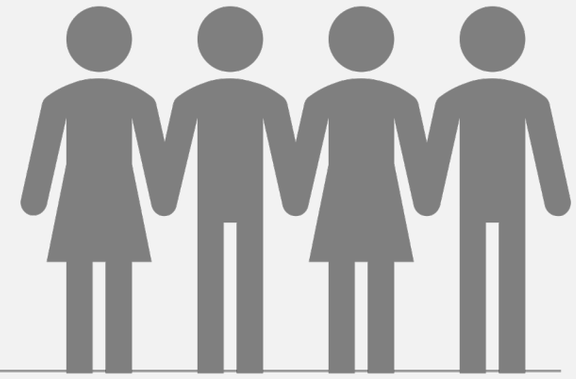
p * 2

p	2	
1	2	2
2	2	4
3	2	6
4	2	8
5	2	10

p * q

p	q	
1	5	5
2	4	8
3	3	9
4	2	8
5	1	5

Exercise: create vectors



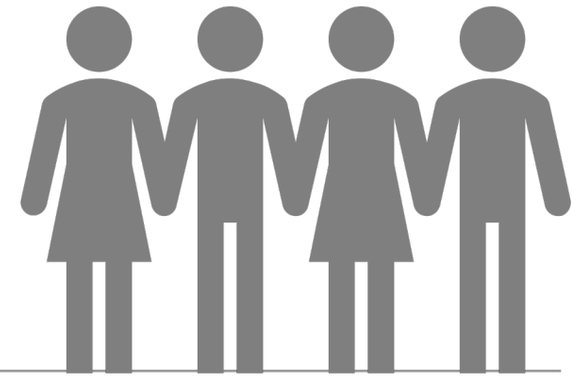
Meet Ann, Bob, Chloe, and Dan.

1. Make a character vector with their names, using the function `c()`. Save the vector as `name`.

2. How old are Ann, Bob, Chloe, and Dan? Design a numeric vector with their respective ages. Save it as `age`.

3. **Bonus:** What is their average age? Use a function in R to calculate this.

Exercise: create vectors



Meet Ann, Bob, Chloe, and Dan.

1. Make a character vector with their names, using the function **`c()`**. Save the vector as **`name`**.

```
> name <- c("Ann", "Bob", "Chloe", "Dan")
```

2. How old are Ann, Bob, Chloe, and Dan? Design a numeric vector with their respective ages. Save it as **`age`**.

```
> age <- c(35,22,50,51)
```

3. **Bonus:** What is their average age? Use a function in R to calculate this.

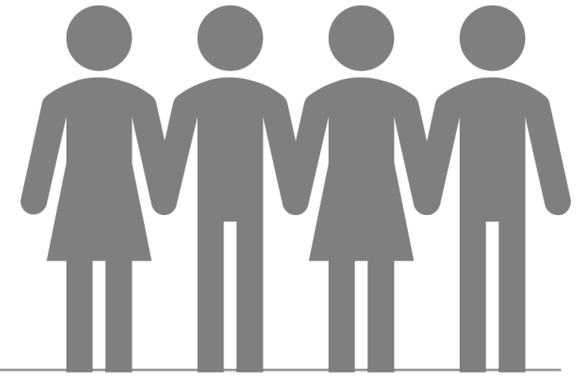
```
> mean(age)
[1] 39.5
```

Vectors and factors

```
> country <- c("UK","USA","USA","UK")
> country
[1] "UK"  "USA" "USA" "UK"

> as.factor(country)
[1] UK  USA USA UK
Levels: UK USA
```

A **factor** is defined by its **levels**
(most useful for a category)

Combining data



```
> name
[1] "Ann"   "Bob"   "Chloe" "Dan"
> age
[1] 35 22 50 51

> c(name,age)
[1] "Ann"   "Bob"   "Chloe" "Dan"   "35"    "22"    "50"    "51"
```

```
> list(name,age)
[[1]]
[1] "Ann"   "Bob"   "Chloe" "Dan"

[[2]]
[1] 35 22 50 51

> data.frame(name,age)
   name  age
1   Ann   35
2   Bob   22
3 Chloe   50
4   Dan   51
```

Vectors, lists, and data frames

	how many dimensions?	function
vector	1	`c()`
list	any number	`list()`
data frame	2	`data.frame()`

Exercise: combining data

1. Create a vector **`country`** containing four countries (use at least one duplicate!).

2. Create a data frame combining **`name`**, **`age`**, and **`country`**, and save it as **`df`**.

3. **Bonus:** create a list combining **`name`**, **`age`**, and **`country`**.

Exercise: combining data

1. Create a vector **`country`** containing four countries (use at least one duplicate!).

```
> country <- c("UK","USA","USA","UK")
```

2. Create a data frame combining **`name`**, **`age`**, and **`country`**, and save it as **`df`**.

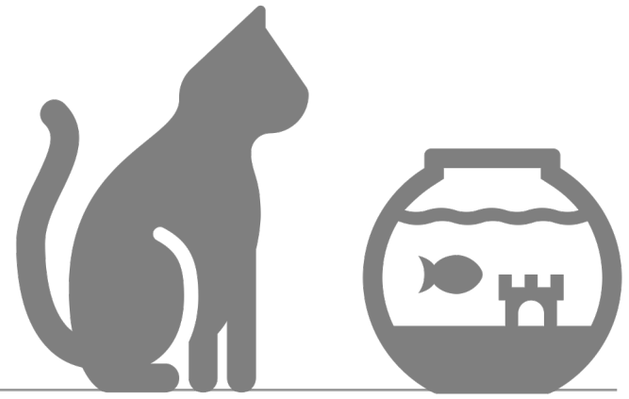
```
> df <- data.frame(name,age,country)
```

3. **Bonus:** create a list combining **`name`**, **`age`**, and **`country`**.

```
> listcombo <- list(name,age,country)
```

adding info to an existing data frame

Data without data: NA



```
> df$pet <- c("cat","none","fish",NA)
> df
 name age country  pet
1   Ann  35      UK  cat
2   Bob  22     USA none
3 Chloe  50     USA fish
4   Dan  51      UK <NA>

> as.factor(df$pet)
[1] cat none   fish   <NA>
Levels: cat fish none
```

NA = **N**ot **A**vailable

name	age	country	pet
Ann	35	UK	cat
Bob	22	USA	**none**
Chloe	50	USA	fish
Dan	51	UK	**NA**

In other words:

We **know** that Bob has **no pets**.

We **do not know** if Dan has pets.

Exercise: predict the answer

Predict the results. Does the (real) answer make sense to you?

```
> 5 == 5
```

```
> 5 == NA
```

```
> NA == NA
```

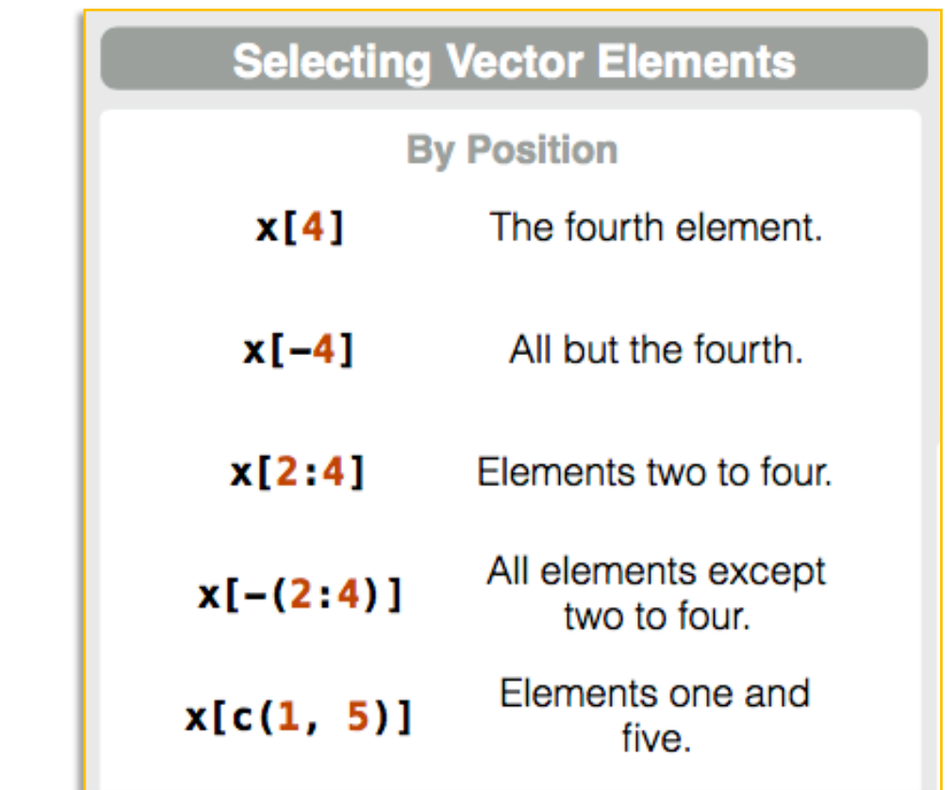
Exercise: predict the answer

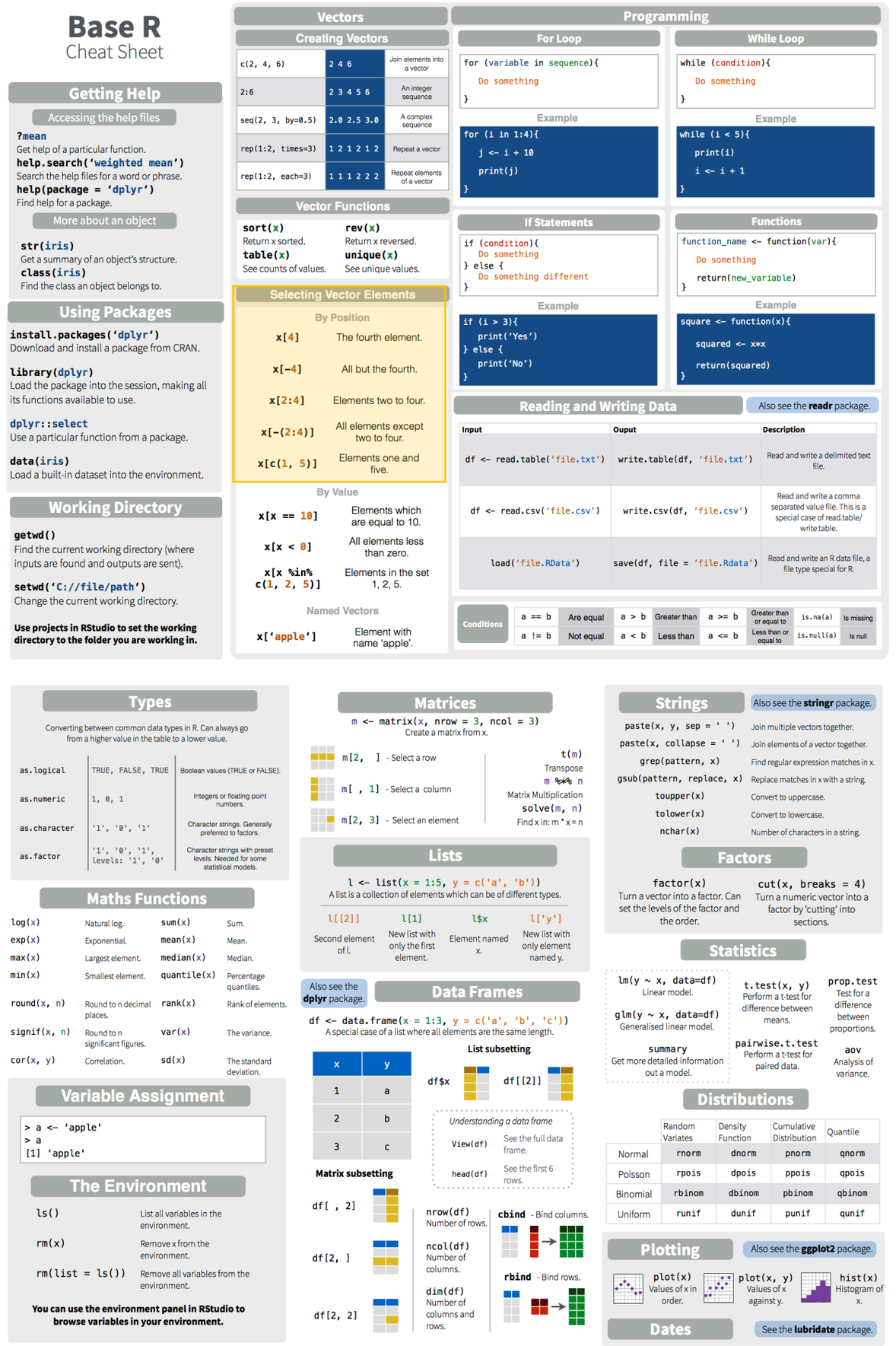
Predict the results. Does the (real) answer make sense to you?

```
> 5 == 5
[1] TRUE

> 5 == NA
[1] NA

> NA == NA
[1] NA

> is.na(NA)
[1] TRUE
```

Selecting vector elements by position

```
> name
[1] "Ann"   "Bob"   "Chloe" "Dan"

> name[2]
[1] "Bob"

> name[1:3]
[1] "Ann"   "Bob"   "Chloe"
```





Selecting vector elements by value

```
> age
[1] 35 22 50 51

> age[age>40]
[1] 50 51
```

age	age>40	result
35	FALSE	
22	FALSE	
50	TRUE	50
51	TRUE	51

```
> age>40
[1] FALSE FALSE TRUE TRUE

> name[name %in% c("Chloe","Ann")]
[1] "Ann"   "Chloe"

> name[NA]
[1] NA NA NA NA
```

Selecting Vector Elements

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Exercise

1. Return only the first number in your vector **`age`**.

2. Return the 2nd and 4th name in your vector **`name`**.

3. Return only ages under 30 from your vector **`age`**.

Exercise

1. Return only the first number in your vector **`age`**.

```
> age[1]
[1] 35
```

2. Return the 2nd and 4th name in your vector **`name`**.

```
> name[c(2,4)]
[1] "Bob" "Dan"
```

3. Return only ages under 30 from your vector **`age`**.

```
> age[age<30]
[1] 22
```

Indexing lists

```
> mylist <- list(name,age)

> mylist
[[1]]
[1] "Ann"   "Bob"   "Chloe" "Dan"

[[2]]
[1] 35 22 50 51
```

select the first list element

```
> mylist[1]
[[1]]
[1] "Ann"   "Bob"   "Chloe" "Dan"
```

select the **content** of the first list element

```
> mylist[[1]]
[1] "Ann"   "Bob"   "Chloe" "Dan"
```

Indexing lists (continued)

```
> mylist[[1]][2]
[1] "Bob”

> mylist[1][2]
[[1]]
NULL
```

NA != NULL

`NA`	Information is **Not Available**
`NULL`	Information **does not exist**
`"None"` or `0`	Data entry specifying **content of 0**

It's like determining how much water is in a glass:
None, the glass is empty.
NA, we don't know what's in there, is it even water?
NULL, there is no glass. Asking the question is futile.

Exercise: predict the answer

```
> is.na(NA)
[1] TRUE

> is.null(NULL)
[1] TRUE
```

Predict the results. Does the (real) answer make sense to you?

```
> is.null(NA)

> is.na(NULL)
```

Exercise: predict the answer

```
> is.na(NA)
[1] TRUE

> is.null(NULL)
[1] TRUE
```

Predict the results. Does the (real) answer make sense to you?

```
> is.null(NA)
[1] FALSE
> is.na(NULL)
[1] logical(0)
```

Indexing a data frame

Matrix subsetting

`df[ , 2]`

`df[2, ]`

row

column

`df[2, 2]`

name	age	country
Ann	35	UK
Bob	22	USA
Chloe	50	USA
Dan	51	UK

Indexing a data frame

```
> df[,2]
[1] 35 22 50 51
> df[,"age"]
[1] 35 22 50 51
> df$age
[1] 35 22 50 51
```

```
> df[2,]
  name age country
2  Bob  22     USA
> df[df$name=="Bob",]
 name age country
2  Bob  22     USA
```

```
> df[df$name=="Bob","age"]
[1] 22
```

df[,2]
df[,"age"]
df$age

df[2,]
df[df$name=="Bob",]

name	age	country
Ann	35	UK
Bob	22	USA
Chloe	50	USA
Dan	51	UK

Remove variables

Before we start with the exercise, please **remove the vectors** that were the basis of your data frame from your environment, like this:

```
> rm(name,age,country)
```

(We do this so they cannot confuse you during the following exercise!)

Exercise

1. From your dataframe df, return the entries for everyone living in a country of your choice.

2. Return **only the names** of everyone in your data frame **`df` under 40**.
   (Hint: what information should you use for row indexing?
   What information should you use for column indexing?)

3. **Bonus:** can you use vector indexing on a column to achieve the same result?

Exercise

1. From your dataframe df, return the entries for everyone living in a country of your choice.

```
> df[df$country=="USA",]
   name  age   country
2   Bob   22       USA
3 Chloe   50       USA
```

2. Return **only the names** of everyone in your data frame **`df` under 40**.
(Hint: what information should you use for row indexing?
What information should you use for column indexing?)

```
> df[df$age<40,"name"]
> df[df$age<40,1]
[1] Ann Bob
Levels: Ann Bob Chloe Dan
```

indexing the dataframe **`df`**

3. **Bonus:** can you use vector indexing on a column to achieve the same result?

```
> df$name[df$age<40]
```

indexing the vector **`df$name`**

Which bracket does what?

[ ] **Indexing** vectors, lists, dataframes…

( ) Passing **arguments** to functions

{ } **Defining content** of loops, functions, etc.

Let’s breathe and recap!

What data types have you encountered so far?

`logical`

`numeric`

`character`

How can data be absent?

`NA` (not available)

`NULL` (non-existent)

And what data collections have you encountered?

`vector` (one dimension)

`factor` (one dimension, level-based)

`data frame` (two dimensions)

`list` (++ dimensions)

Functions

What functions have you encountered so far?

```
> c()
> as.factor()
> data.frame()
> is.na()
> mean()
...
```

And do you still know what they mean? And how to use them? **No?**

```
> ?mean()
> help.search("standard deviation")
```

(or use the Help window to the right of your console)

Help!

table {base} ← function & package names

R Documentation

Cross Tabulation and Table Creation

Description

`table` uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

Usage

how would you use the function?

```
table(...,
      exclude = if (useNA == "no") c(NA, NaN),
      useNA = c("no", "ifany", "always"),
      dnn = list.names(...), deparse.level = 1)
```

wait what? These are extra arguments. If you see `argument = "default"` then the setting is already specified, so you won't have to.

functions so related they share a help page

```
as.table(x, ...)
is.table(x)

## S3 method for class 'table'
as.data.frame(x, row.names = NULL, ...,
```

Help? Scroll down!

Examples

```
require(stats) # for rpois and xtabs
## Simple frequency distribution
table(rpois(100, 5))
## Check the design:
with(warpbreaks, table(wool, tension))
table(state.division, state.region)
```

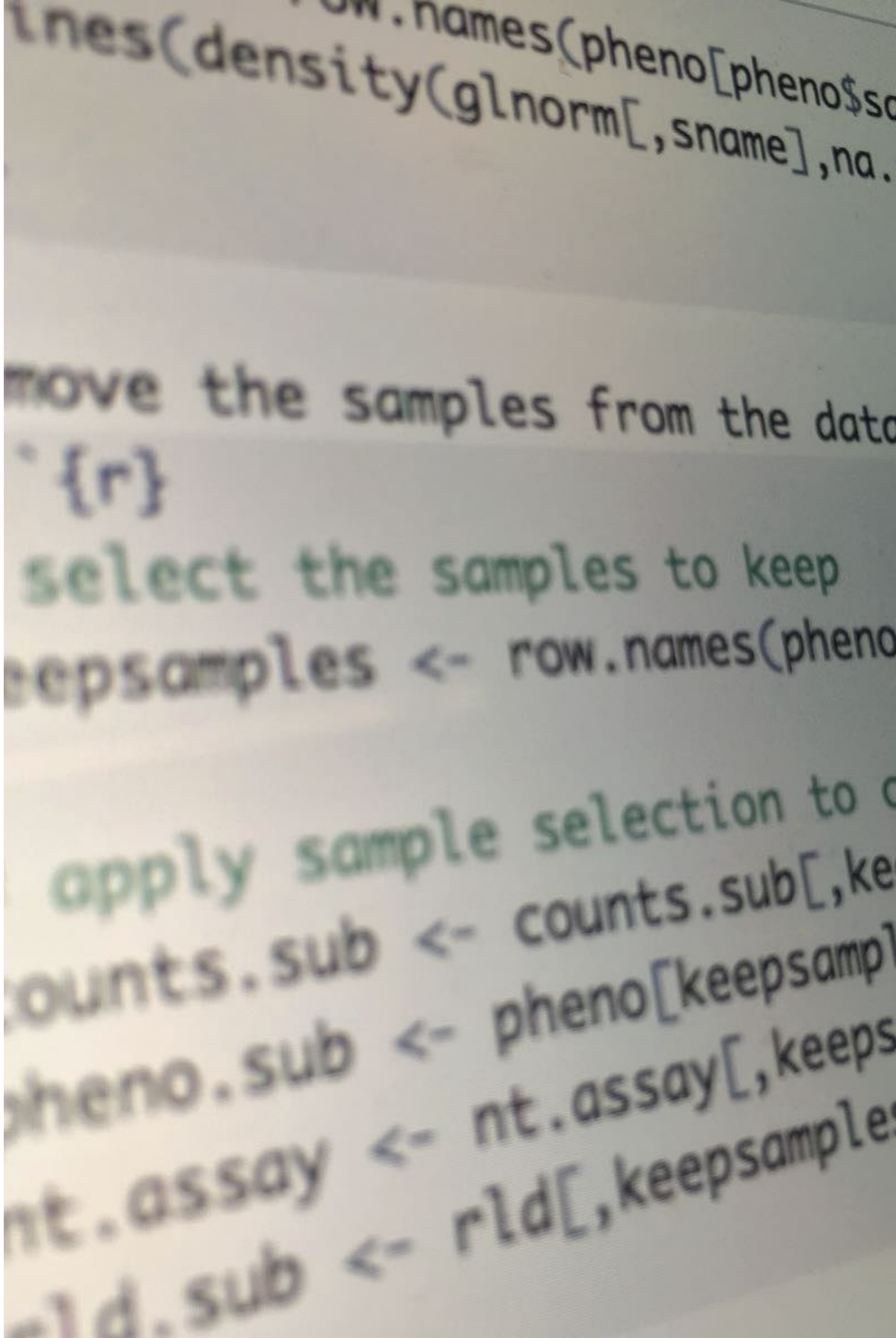
```
# simple two-way contingency table
with(airquality, table(cut(Temp, quantile(Temp)), Month))

a <- letters[1:3]
table(a, sample(a))                         # dnn is c("a", "")
table(a, sample(a), deparse.level = 0) # dnn is c("", "")
table(a, sample(a), deparse.level = 2) # dnn is c("a", "sample(a)")

## xtabs() <-> as.data.frame.table() :
UCBAdmissions ## already a contingency table
DF <- as.data.frame(UCBAdmissions)
class(tab <- xtabs(Freq ~ ., DF)) # xtabs & table
## tab *is* "the same" as the original table:
all(tab == UCBAdmissions)
```

Take a break!

"Fresh Cup" by mikecpeck

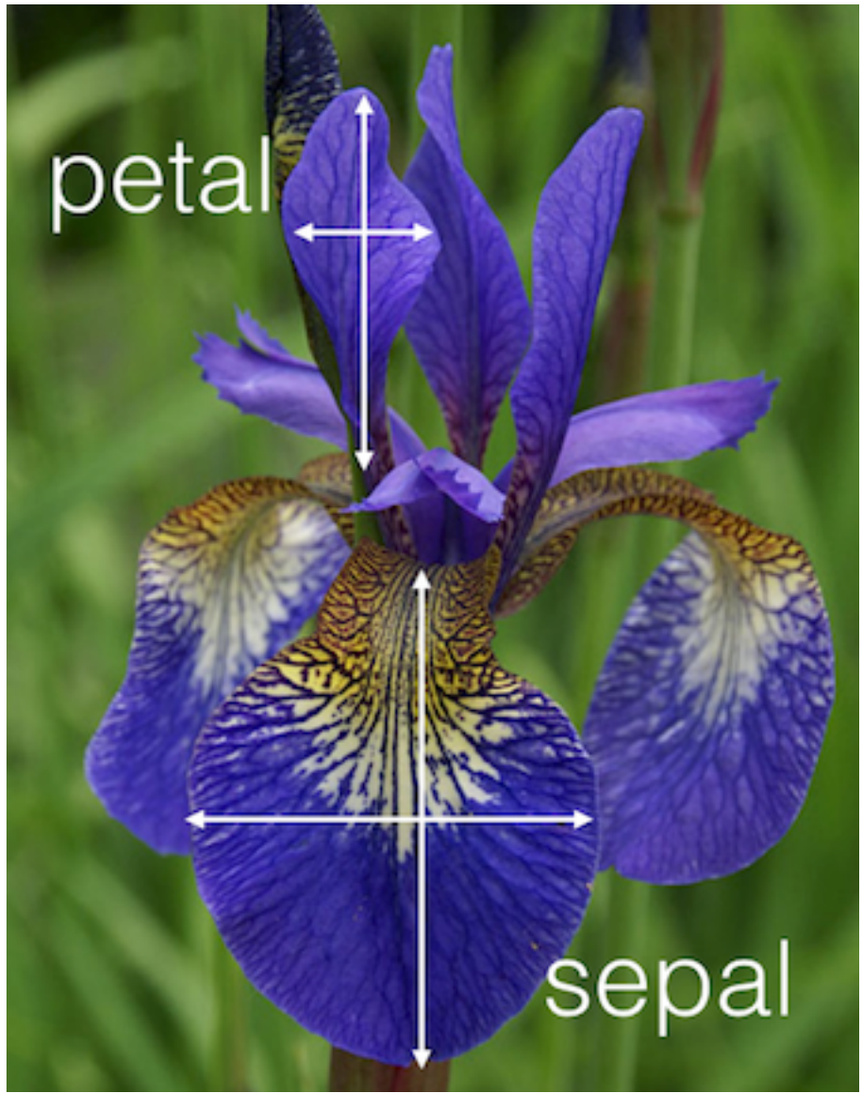
Programming

- Writing your first script
- Programming with loops and functions
- Handling and processing a dataset



Introducing a sample dataset

- Dataset: ‘iris’
- Standard dataset in R, measurements on 3 species of iris flowers



Introducing a sample dataset

- Dataset: ‘iris’

- Standard dataset in R, measurements on 3 species of iris flowers

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa     :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Exercise

Explore the `iris` dataframe, using some of the following functions:

- `head()`
- `tail()`
- `names()`
- `summary()`
- `dim()`
- `str()`

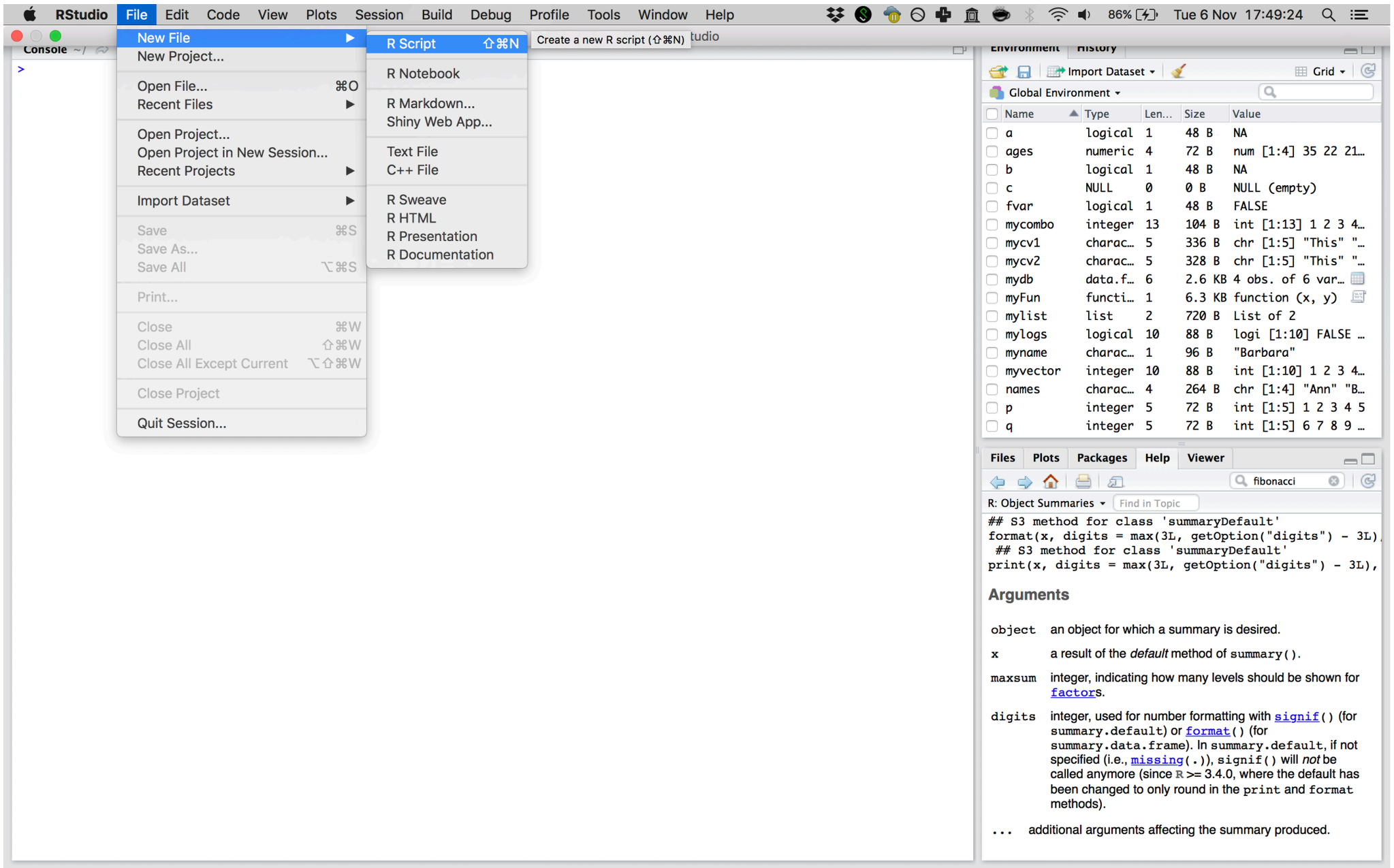
Can you figure out what these functions do? What do they teach you about the kind of data in `iris`?

Console and Scripts

Console	Code execution	-
Script	Code	Extension: .R (example_script.R)
Rmarkdown	Code + Narrative	Extension: .Rmd (example_report.Rmd)

Ready to script?

RStudio > File > New File > R Script



Executing an R script

- Place your cursor in the line of code you want to execute
- Press  or `ctrl + enter`
- When running multiple lines: select all lines, then press ‘run’ or ‘ctrl+enter’
- Try it out! Enter in your script:

```
head(iris)
dim(iris)
```

Then execute these lines!

Console vs script on our slides

tell-tale **console** sign, immediate communication with R

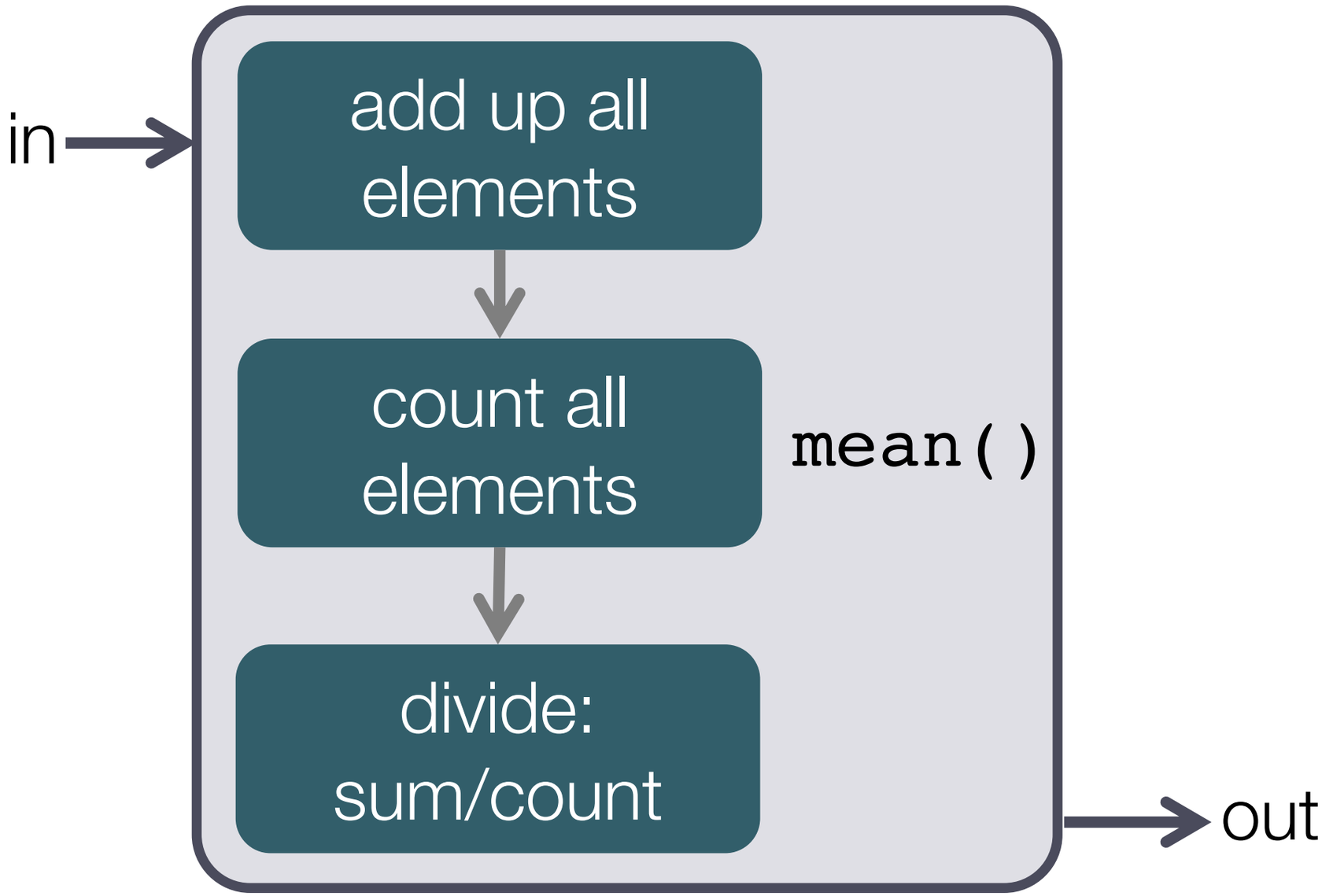
```
> x <- 1
> x
[1] 1
```

comment in a **script**: computer stops reading (<u>but you should not!</u>)

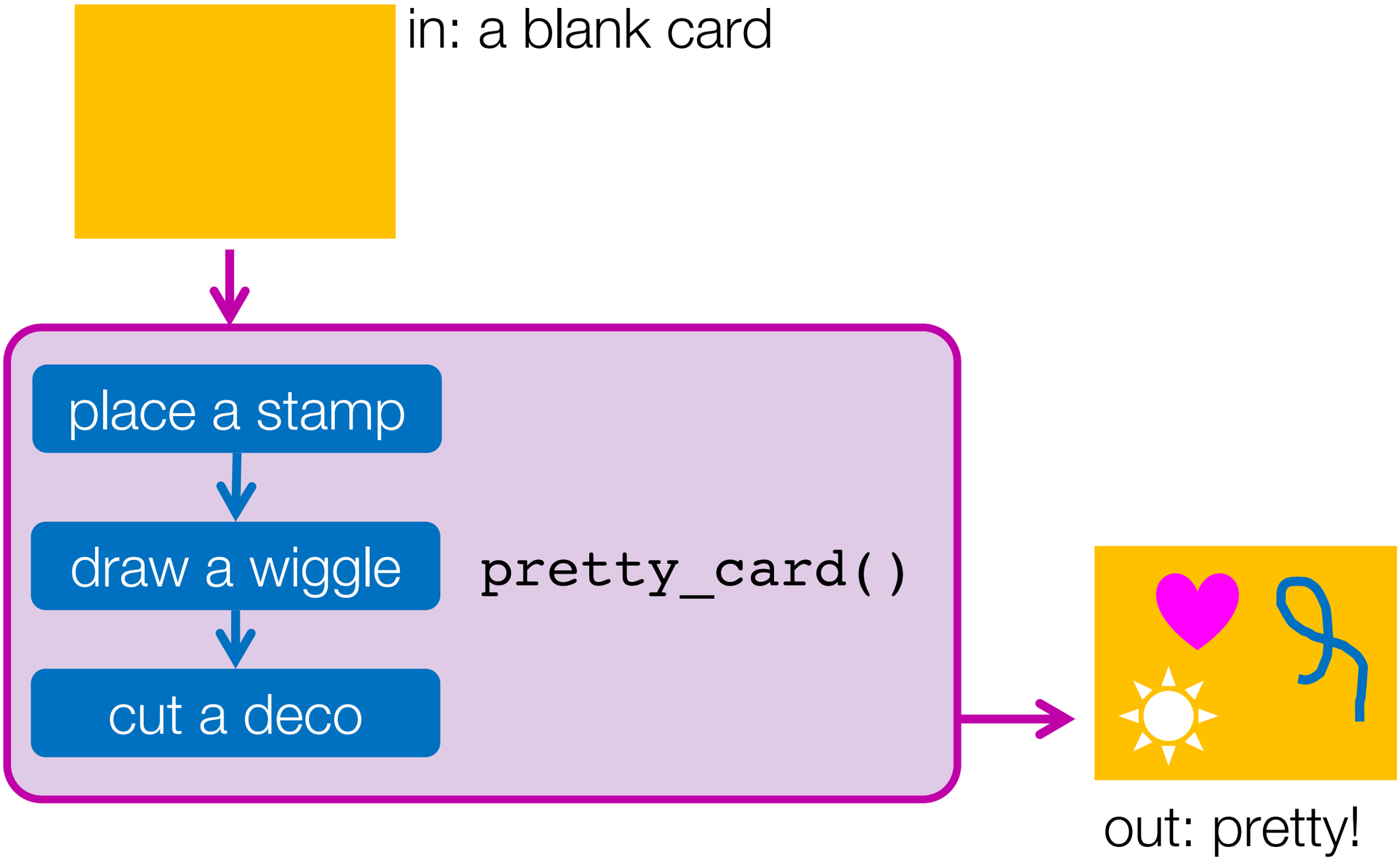
```
# Assigning the value 1 to x
x <- 1
y <- 5
```

Programming: functions

- Multiple instructions that form a cohesive unit
- Should be able to be repeated on different inputs

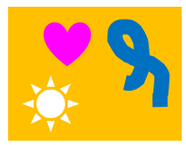


Live exercise: the assembly line of a function



How would this look in R?

```
pretty_card <- function(x){
  x <- place_stamp(x)
  x <- draw_wiggle(x)
  x <- cut_deco(x)
  return(x)
}
```

```
> pretty_card(  )
[1] 
```

Programming: functions

```
myFun <- function(x,y){
   z <- x*y
   return(z)
}
```

```
> myFun(2,4)
[1] 8
```

Step 1: run the function. This saves the function to memory under the *name* you assigned it.

Step 2: use the function. You can now execute the function by calling its *name.*

Exercise

1. Write a function that takes a vector as input, and returns the mean of this vector (you can use the existing function **`mean()`** inside your function).

```
apply_calc <- function(...){
    ...
    return(...)
}
```

2. Add further options to your function:
   a. for example, the standard deviation (**`sd()`**), the minimum (**`min()`**), and the maximum (**`max()`**) of your input vector.
   b. Put all of these calculations in a vector using the function **`c()`**, and return this result vector.

```
apply_calc <- function(...){
    ...


    allres <- c(...)
    return(...)
}
```

Exercise

1. Write a function that takes a vector as input, and returns the mean of this vector (you can use the existing function **`mean()`** inside your function).

```
apply_calc <- function(x){
    m <- mean(x)
    return(m)
}
```

2. Add further options to your function:
   a. for example, the standard deviation (**`sd()`**), the minimum (**`min()`**), and the maximum (**`max()`**) of your input vector.
   b. Put all of these calculations in a vector using the function **`c()`**, and return this result vector.

```
apply_calc <- function(x){
    m <- mean(x)
    s <- sd(x)
    mi <- min(x)
    ma <- max(x)
    allres <- c(m,s,mi,ma)
    return(allres)
}
```

Applying `apply_calc`

```
apply_calc <- function(x){
    m <- mean(x)
    return(m)
}
```

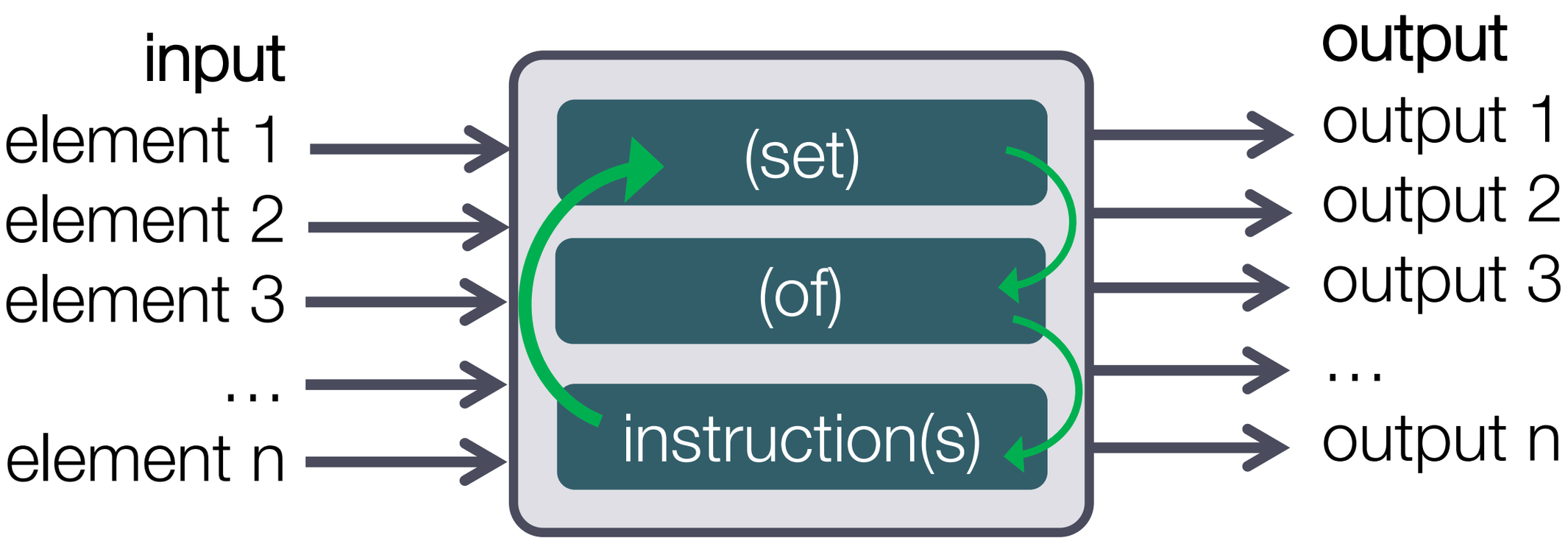
```
> apply_calc(iris$Sepal.Length)
[1] 5.8433333
```

```
apply_calc <- function(x){
    m <- mean(x)
    s <- sd(x)
    mi <- min(x)
    ma <- max(x)
    allres <- c(m,s,mi,ma)
    return(allres)
}
```

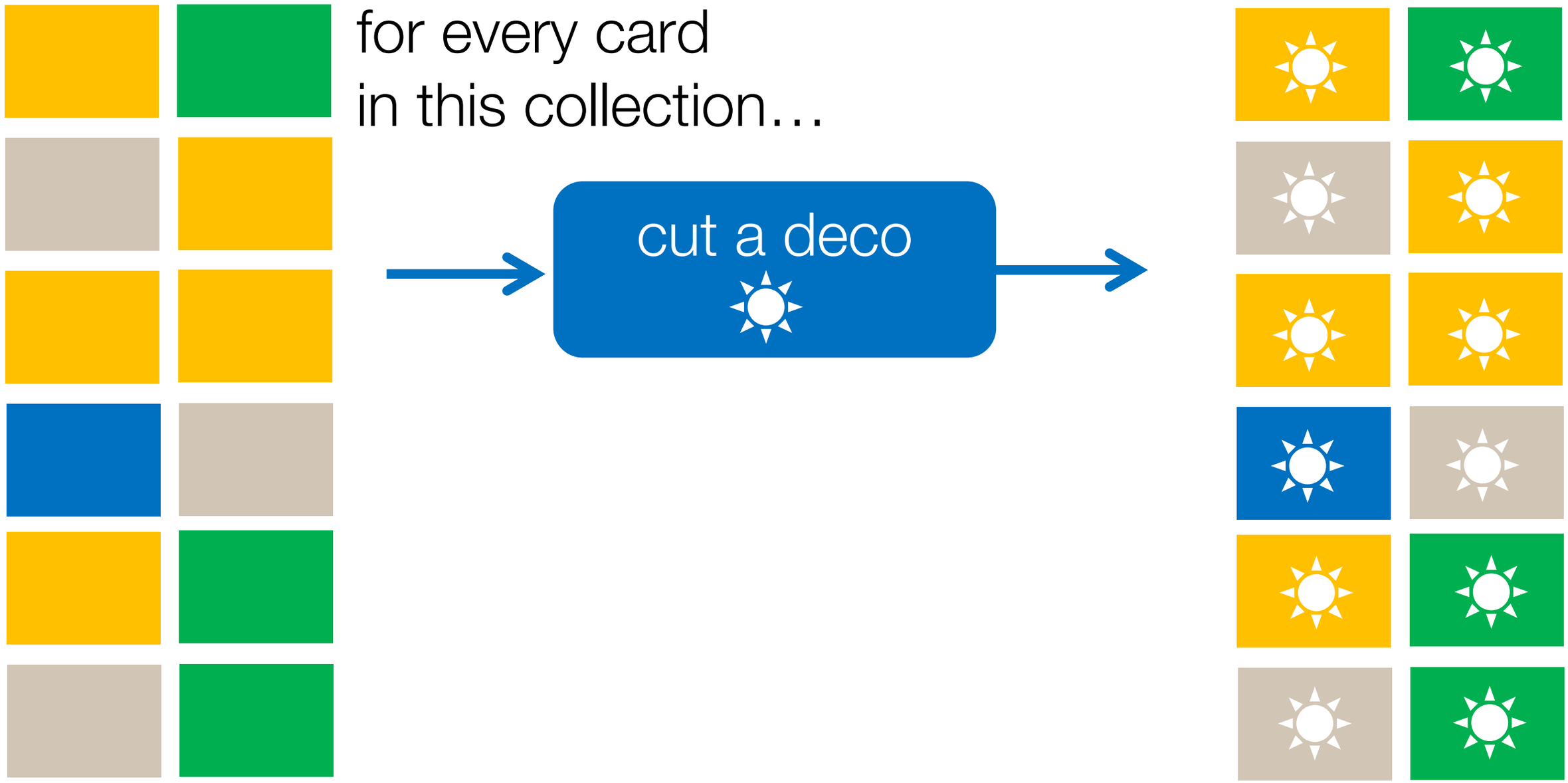
```
> apply_calc(iris$Sepal.Length)
[1] 5.8433333 0.8280661 4.3000000 7.9000000
```

Programming: for-loops

- Instruction(s) that need(s) to be applied multiple times
- Input is an iterable object (it consists of multiple similar elements)



Live exercise: the repeated action in a for-loop



How would this look in R?

```
for( ■ in ■■■■■■ ){
  cut_deco(■)
}
```

```
[1] ☀
[1] ☀
[1] ☀
[1] ☀
[1] ☀
[1] ☀
```

```
for(i in 1:6){
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
```

Exercise

1. Make a vector with all the column names in `iris`.
   Hint: use the function `colnames()`.

```
iriscols <- ...
```

2. Make a for-loop that iterates over all the column names in `iris`, and prints these column names.

```
for(...){
    ...
}
```

3. Elaborate on this for-loop: select the corresponding column in `iris`, and print the mean. *Hint: yes,* ***you should get a warning****! Do you understand why?*

```
for(...){
    ...


}
```

Exercise

1. Make a vector with all the column names in **`iris`**.
   Hint: use the function **`colnames()`**.

```
iriscols <- colnames(iris)
```

2. Make a for-loop that iterates over all the column names in **`iris`**, and prints these column names.

```
for(i in iriscols){
    print(i)
}
```

3. Elaborate on this for-loop: select the corresponding column in **`iris`**, and print the mean. *Hint: yes,* ***you should get a warning****! Do you understand why?*

```
for(i in iriscols){
    column <- iris[,i]
    stat <- mean(column)
    print(stat)
}
```

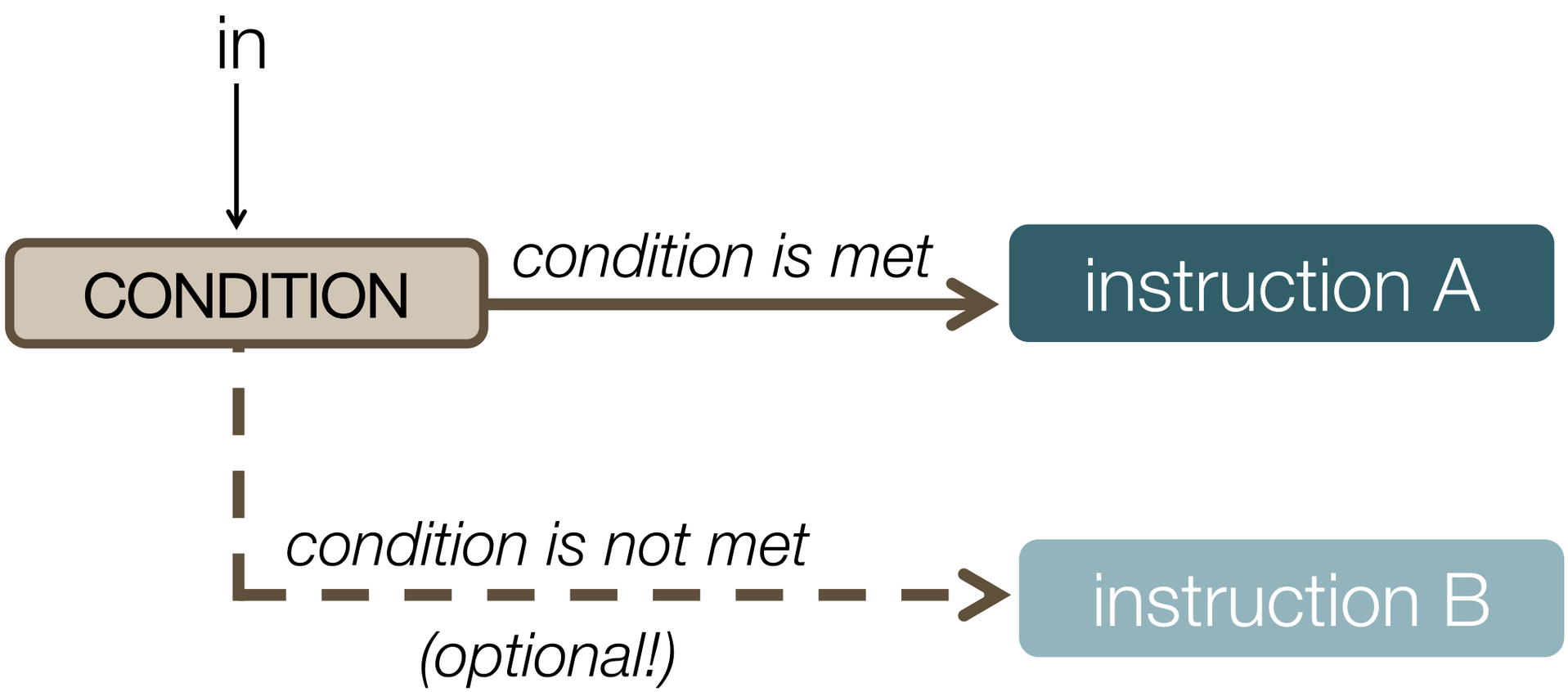
What's wrong? And how do we fix it?

```
for(i in iriscols){
  # select the appropriate column
  column <- iris[,i]
  # calculate the mean
  stats <- mean(column)
  # print the mean
  print(stats)
}
```

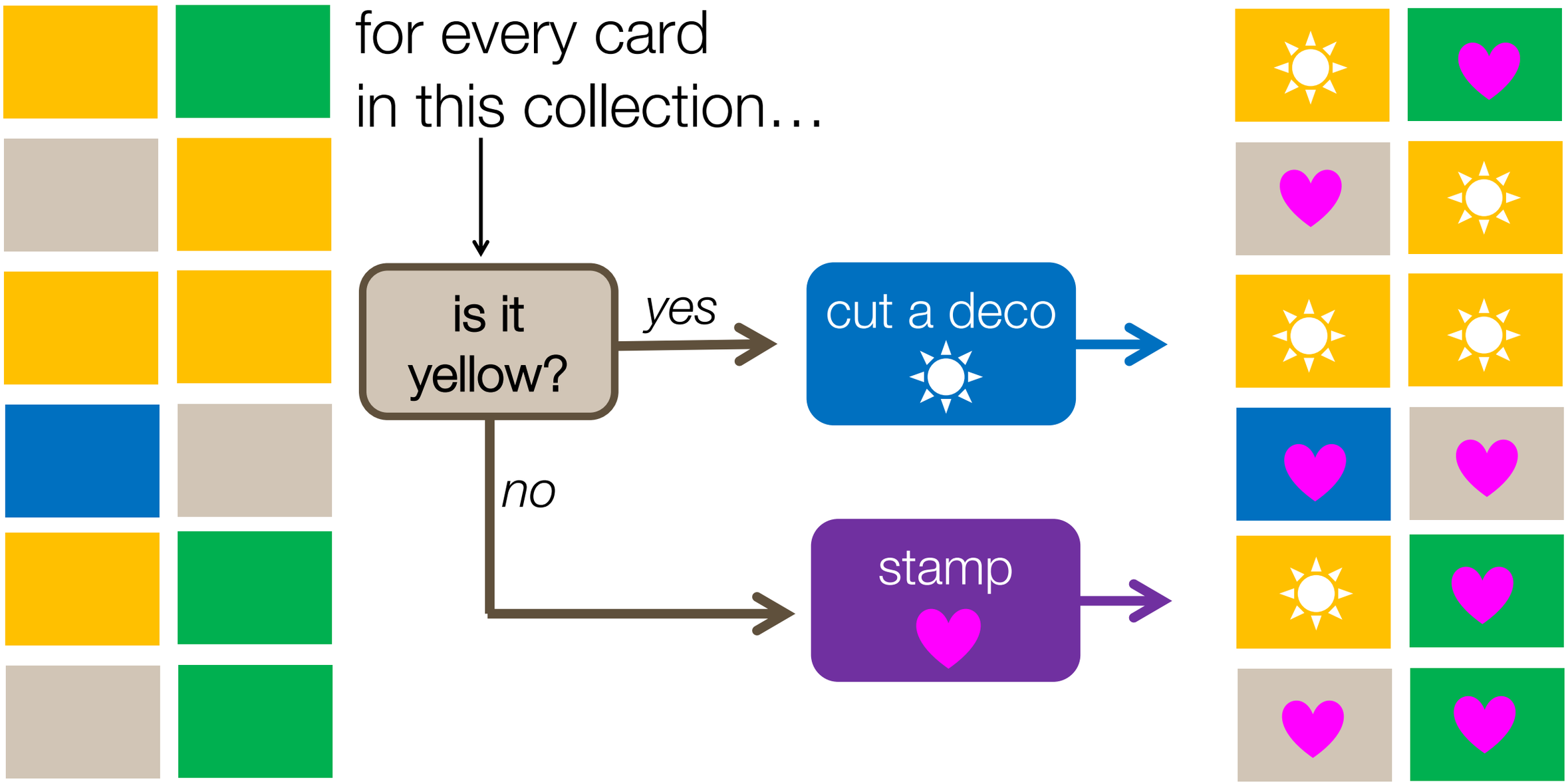
```
[1] 5.843333
[1] 3.057333
[1] 3.758
[1] 1.199333
[1] NA
Warning message:
In mean.default(c) : argument is not numeric or logical:
returning NA
```

Programming: if-statement

- Test to conditionally apply an instruction
- Possibility for an alternative instruction (if the test is negative)



Live exercise: the selectivity of an if-statement



PS: note how this is still a for-loop? That's not a requirement!

How would this look in R?

```
for( ■ in ■■■■■■ ){
  if( ■ == ■ ){
    cut_deco( ■ )
  } else{
    stamp( ■ )
  }
}
```

```
[1] ♥
[1] ☀
[1] ☀
[1] ♥
[1] ☀
[1] ♥
```

```
for(i in 1:6){
  if(i > 3){
    print("Large!")
  } else{
    print("small...")
  }
}
```

```
[1] "small..."
[1] "small..."
[1] "small..."
[1] "Large!"
[1] "Large!"
[1] "Large!"
```

Programming: if-statement

```
d <- 5

if(is.na(d)){
  print("My data is missing!")
} else if(is.null(d)){
  print("My data does not even exist...")
} else{
  print("I have data!")
}
```

```
[1] "I have data!"
```

Exercise: choose one!

1. Copy-paste the for-loop you made in the previous exercise. Inside this for-loop, add an if-statement, so that **`mean()`** is only performed on numeric vectors.
   Hint: check the function **`is.numeric()`**.

```
for(i in iriscols){
    column <- iris[,i]
    ...
}
```

2. Are you feeling comfortable with the material?
   a. Open the file ‘programming_exercise.R’
   b. Read through the code, see if you understand it *(it is mostly the previous two exercises, but with some small tweaks!)*
   c. Scroll down to line 32 for the exercise.
   d. Read the bonus material — you can check your work with the code there.

Exercise: choose one!

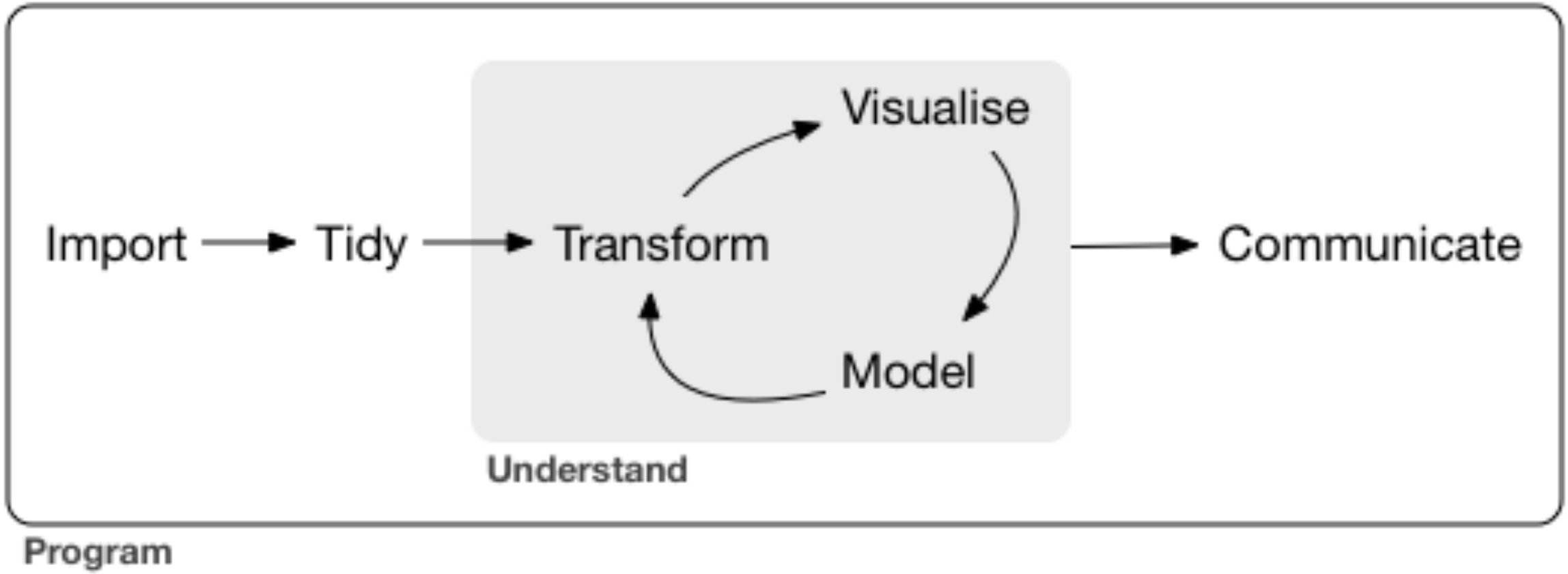
1. Copy-paste the for-loop you made in the previous exercise. Inside this for-loop, add an if-statement, so that **`mean()`** is only performed on numeric vectors.
   Hint: check the function **`is.numeric()`**.

```
for(i in iriscols){
    column <- iris[,i]
    if(is.numeric(column)){
        stat <- mean(column)
        print(stat)
    }
}
```

2. Are you feeling comfortable with the material?
   a. Open the file 'programming_exercise.R'
   b. Read through the code, see if you understand it *(it is mostly the previous two exercises, but with some small tweaks!)*
   c. Scroll down to line 32 for the exercise.
   d. Read the bonus material — you can check your work with the code there.

Data science workflow: scripting is crucial

- Scripting combines commands to a comprehensive set of instructions.
- A script is code that can be **saved, reused, shared, published**!
- In short: a crucial step towards reproducible data analysis.



a single script for a single purpose!

Starting the script: write a header

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data", at Utrecht University
```

Load packages and dependencies

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data", at Utrecht University

# Load required packages
library(dplyr)
library(tidyr)
library(ggplot2)
```

Custom functions

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data", at Utrecht University

# Load required packages
library(dplyr)
library(tidyr)
library(ggplot2)

# Functions
myFun <- function(var){
  var <- var*2*pi
  return(var)
}
```

Starting your script: header, packages, functions

```
## Date: 7 November 2018
## Author: S. Tudent
## This script was written as part of the R course
## "Introduction to R & Data", at Utrecht University
```

```
# Load required packages
library(dplyr)
library(tidyr)
library(ggplot2)
```

```
# Functions
myFun <- function(var){
  var <- var*2*pi
  return(var)
}
```

To check before lunch: please load tidyverse

```
> library(tidyverse)
── Attaching packages ─────────────────────────────────── tidyverse 1.2.1 ──
   ggplot2 3.1.0       purrr   0.3.0
   tibble  2.0.1       dplyr   0.7.8
   tidyr   0.8.2       stringr 1.4.0
   readr   1.3.1       forcats 0.3.0
── Conflicts ────────────────────────────────────── tidyverse_conflicts() ──
dplyr::filter() masks stats::filter()
dplyr::lag()    masks stats::lag()
```

Enjoy your lunch!