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

Welcome!

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

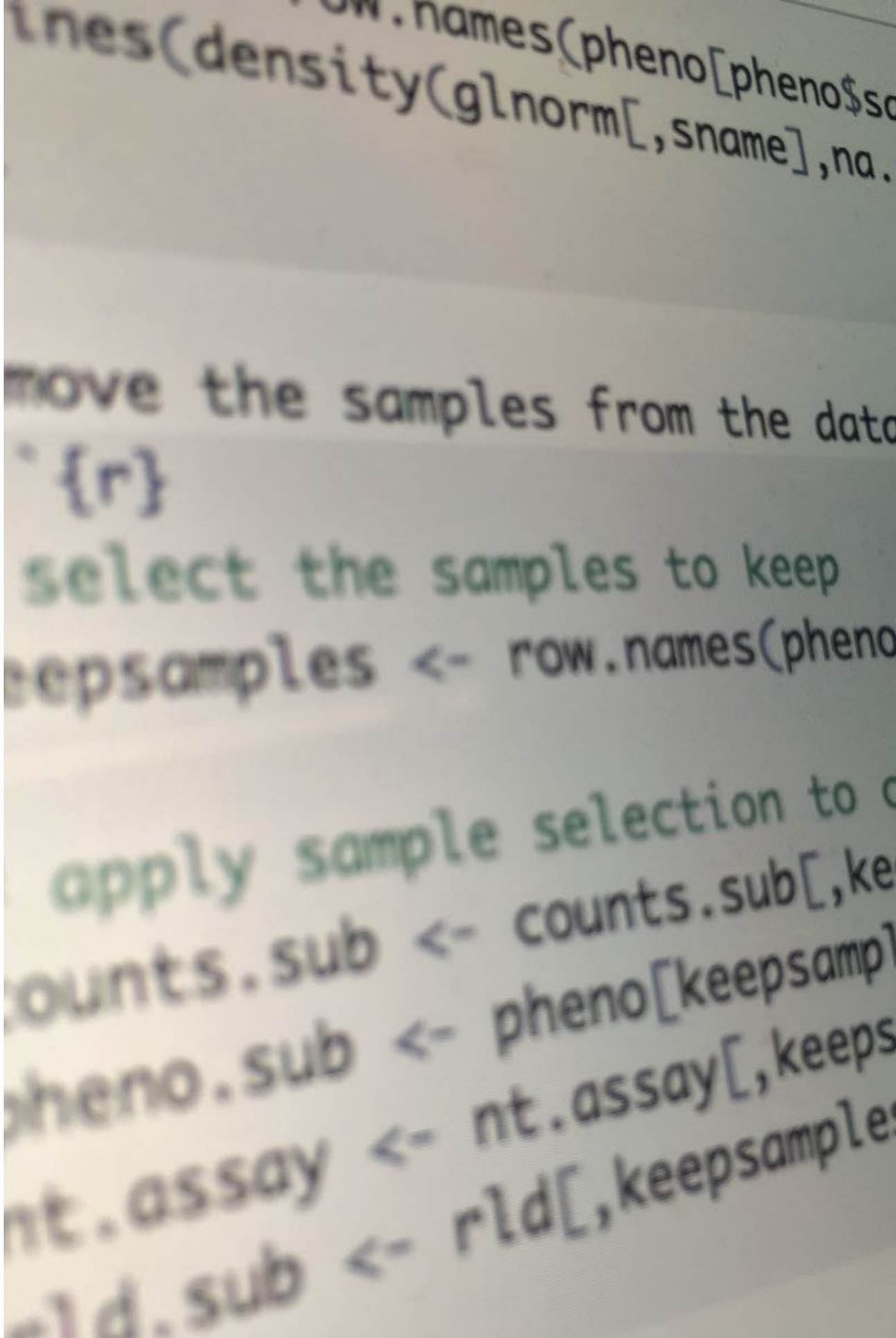
---

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: Modern R with tidyverse
- Final remarks (~17:00)



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

Bianca Kramer
subject specialist Biomedical Science
UU Library
b.m.r.kramer@uu.nl

Barbara Vreede
subject specialist Science
UU Library
b.m.i.vreede@uu.nl

Jacques Flores
data specialist @ RDM support
j.p.flores@uu.nl

Felix Weijdema
subject specialist Biomedical Science
UU Library
f.p.weijdema@uu.nl

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

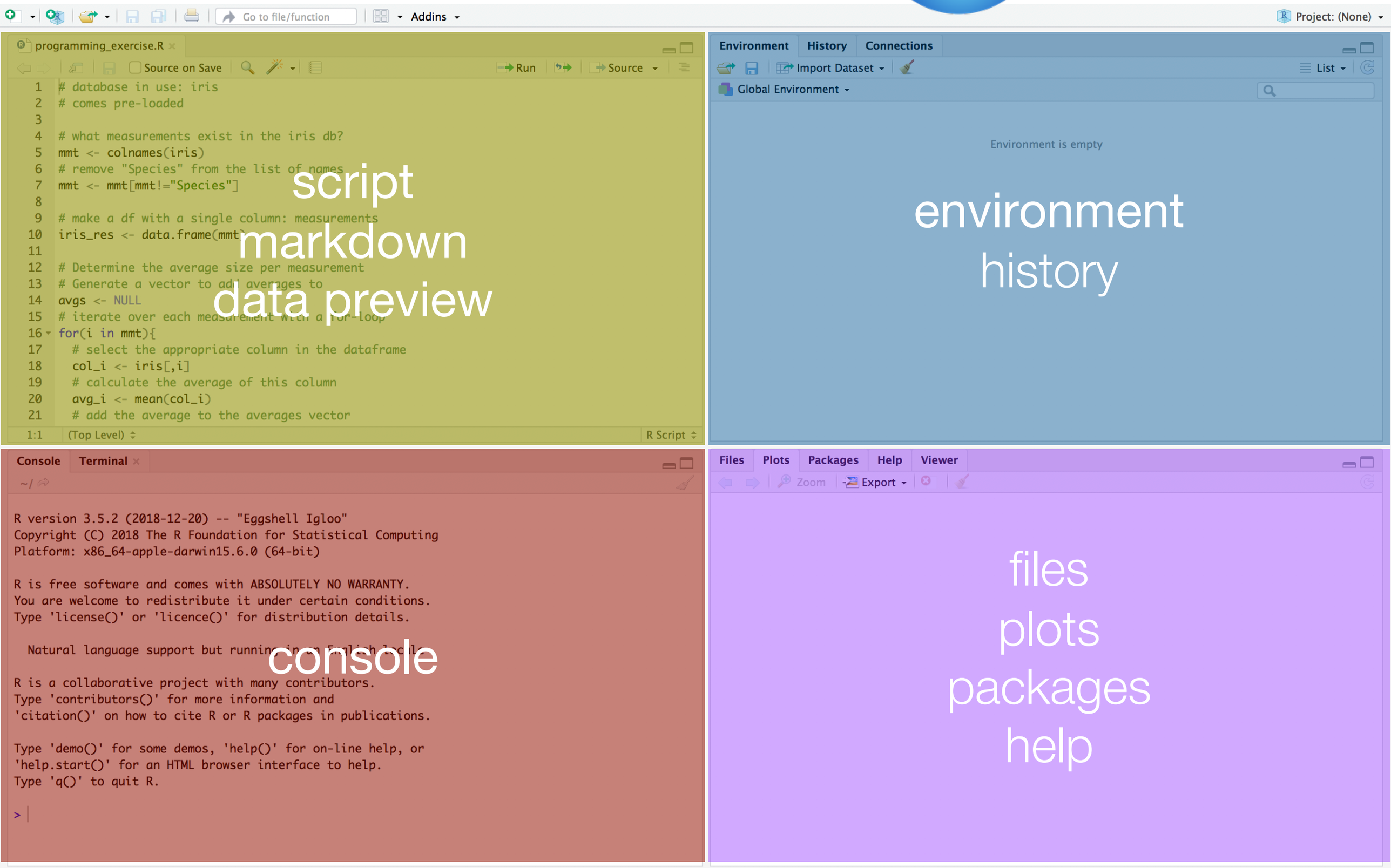
What is R?

- 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

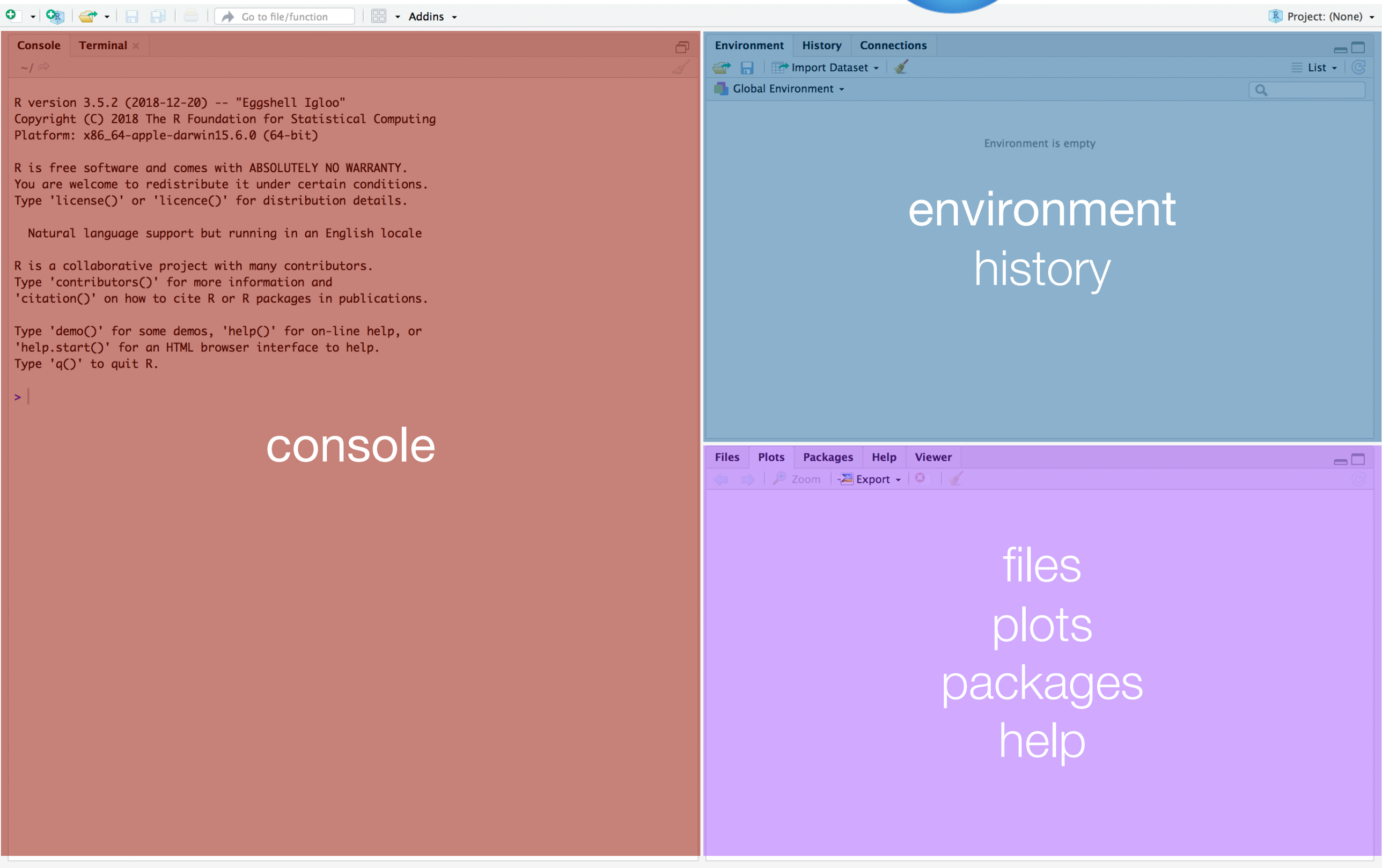
We will work in RStudio

- 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



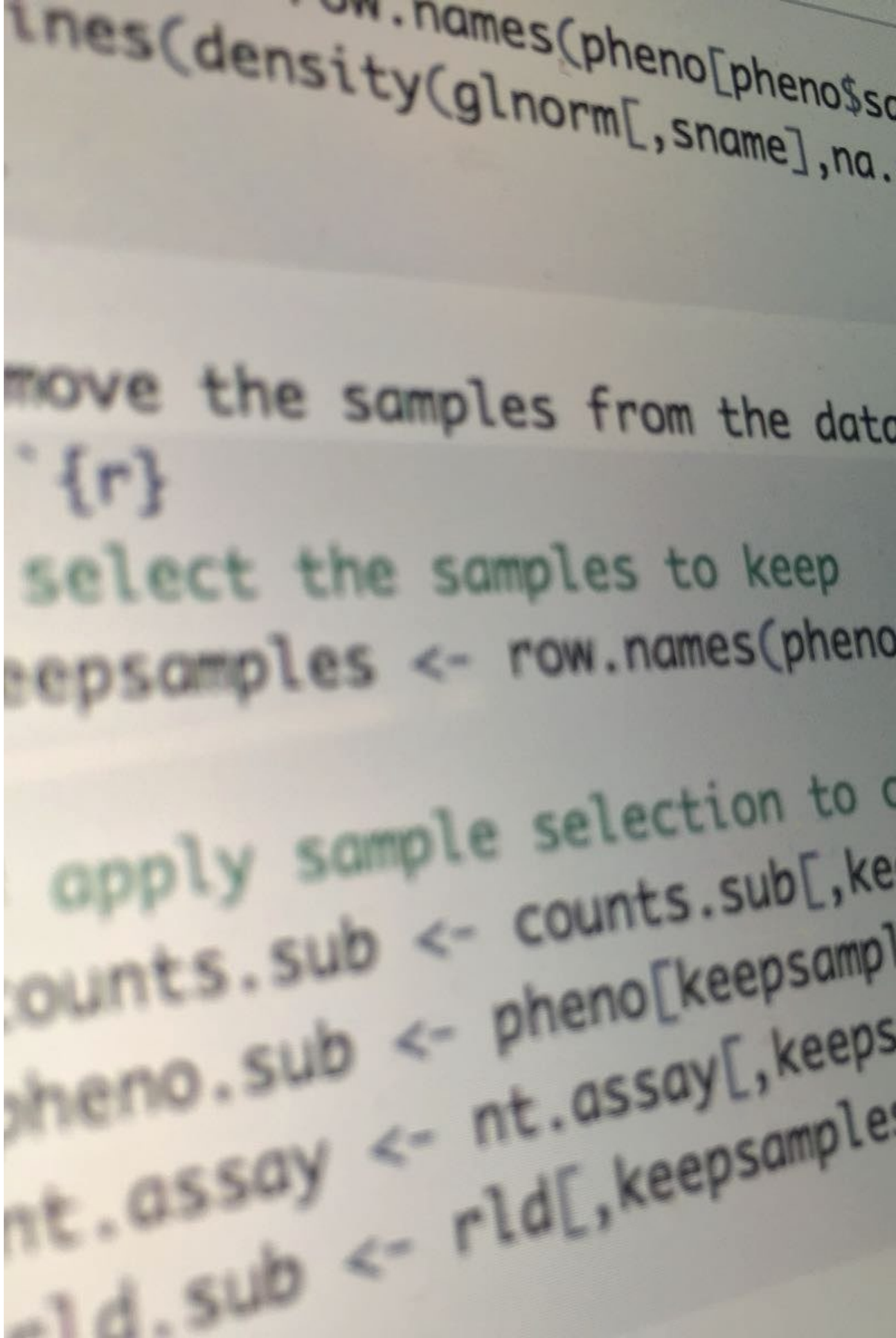
Have you downloaded the course material?

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
- Combining data
- First functions



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!
```

Variable assignment

```
>
```

Variable assignment

```
> x <- 6
```

Variable assignment

Base R
Cheat Sheet

```
> x <- 6

> x <- "hi!"

> 6 -> x
```

```
> x = 6

> 6 = x
Error in 6 = x : invalid (do_set) left-hand side to assignment
```

Variable Assignment

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

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.

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`

```
> T
[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
```

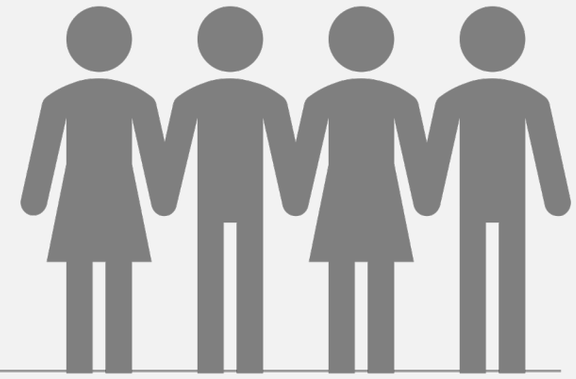
p * 2

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



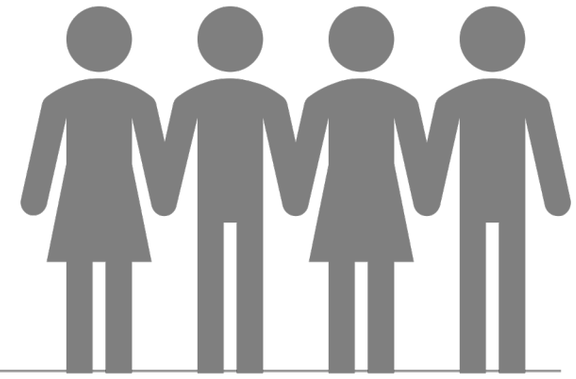
Meet Ann, Bob, Chloe, and Dan.

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
```

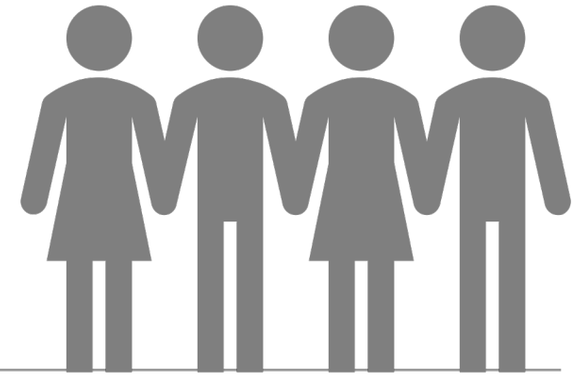
Vectors and factors

```
> 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")
```

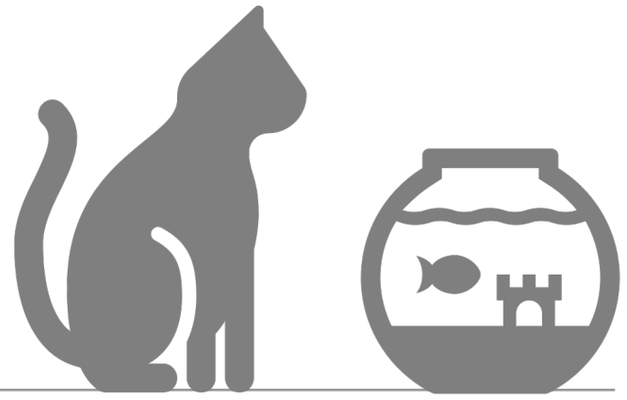
2. Create a data frame combining **`name`**, **`age`**, and **`country`**, and save it as **`df`**.

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

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

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

Data without data: NA



```
> pet <- c("cat","none","fish",NA)
> pet
[1] "cat"  "none"  "fish" NA

> as.factor(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
```

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) *coming up!*

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.logical()
> 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

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 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.

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
```

```
> mylist[1]
[[1]]
[1] 4 5 6
```

select the first list element

```
> mylist[[1]]
[1] 4 5 6
```

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

```
> mylist[[1]][2]
[1] 5
> 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**

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  25     USA
```

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

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

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

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

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?

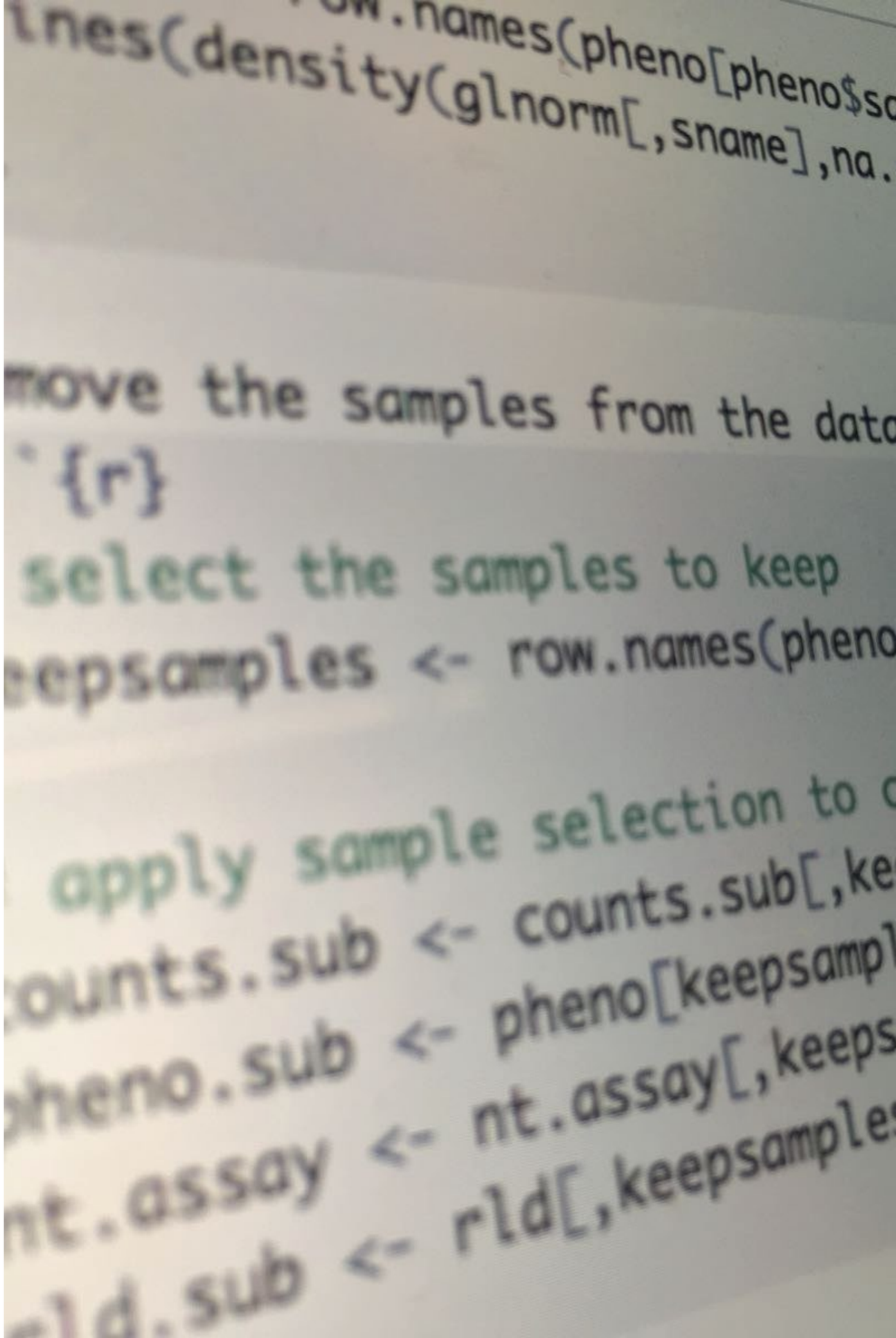
[ ] **Indexing** vectors, lists, dataframes…

( ) Passing **arguments** to functions

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

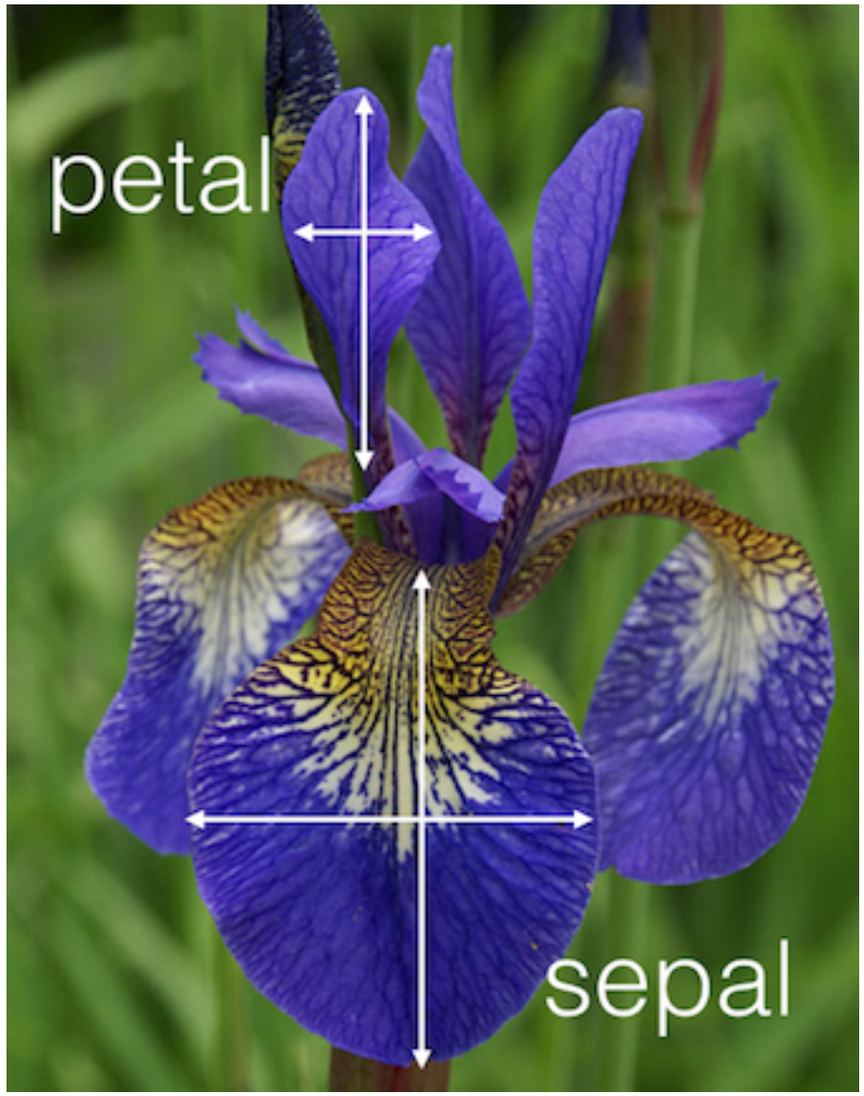
Programming

- 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()`

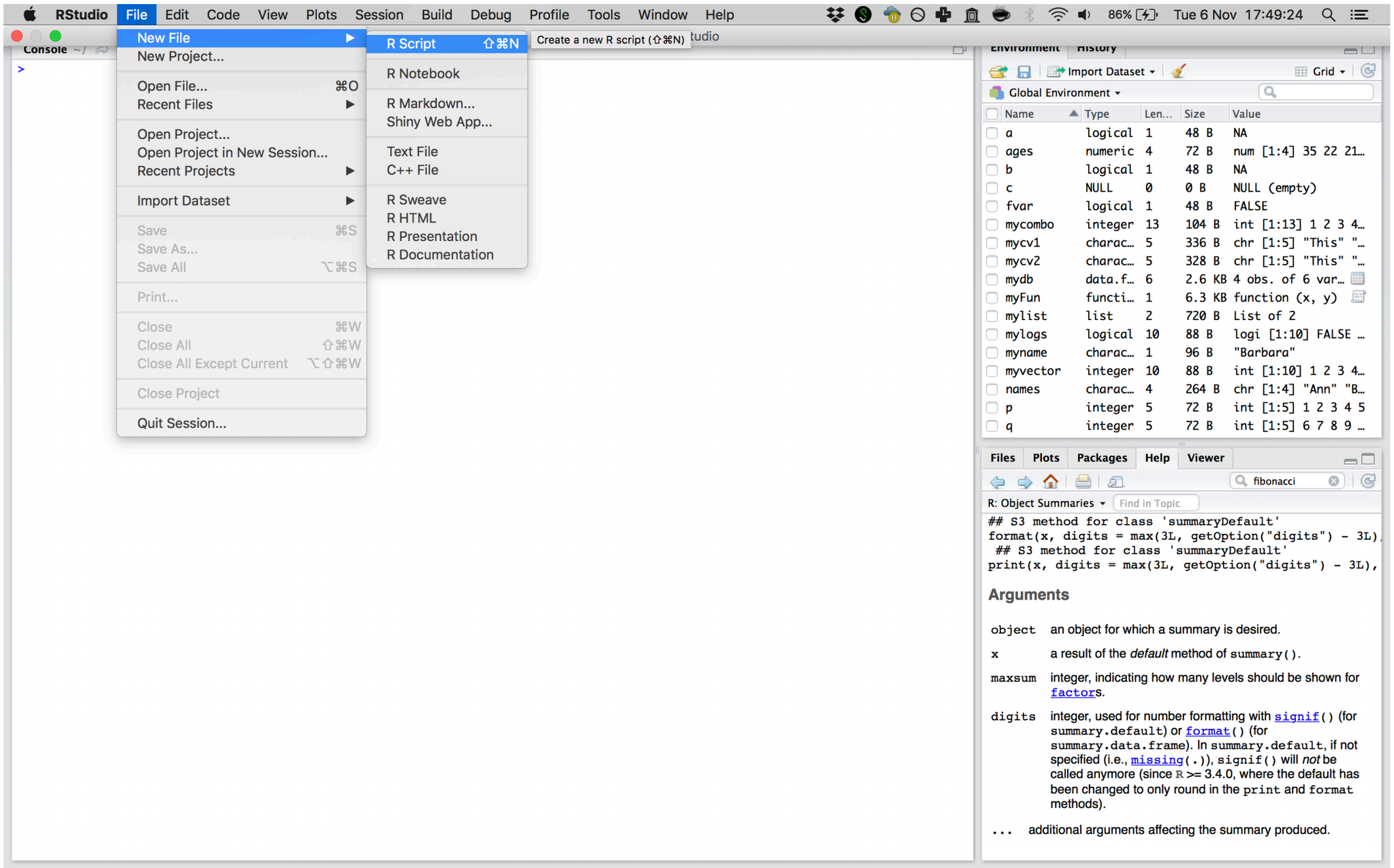
Can you figure out what these functions do? What do they teach you about the kind of data in `iris`?

Console and Scripts

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

Ready to script? [demo!]

RStudio > File > New File > R Script



Console vs script on our slides

writing in the console

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

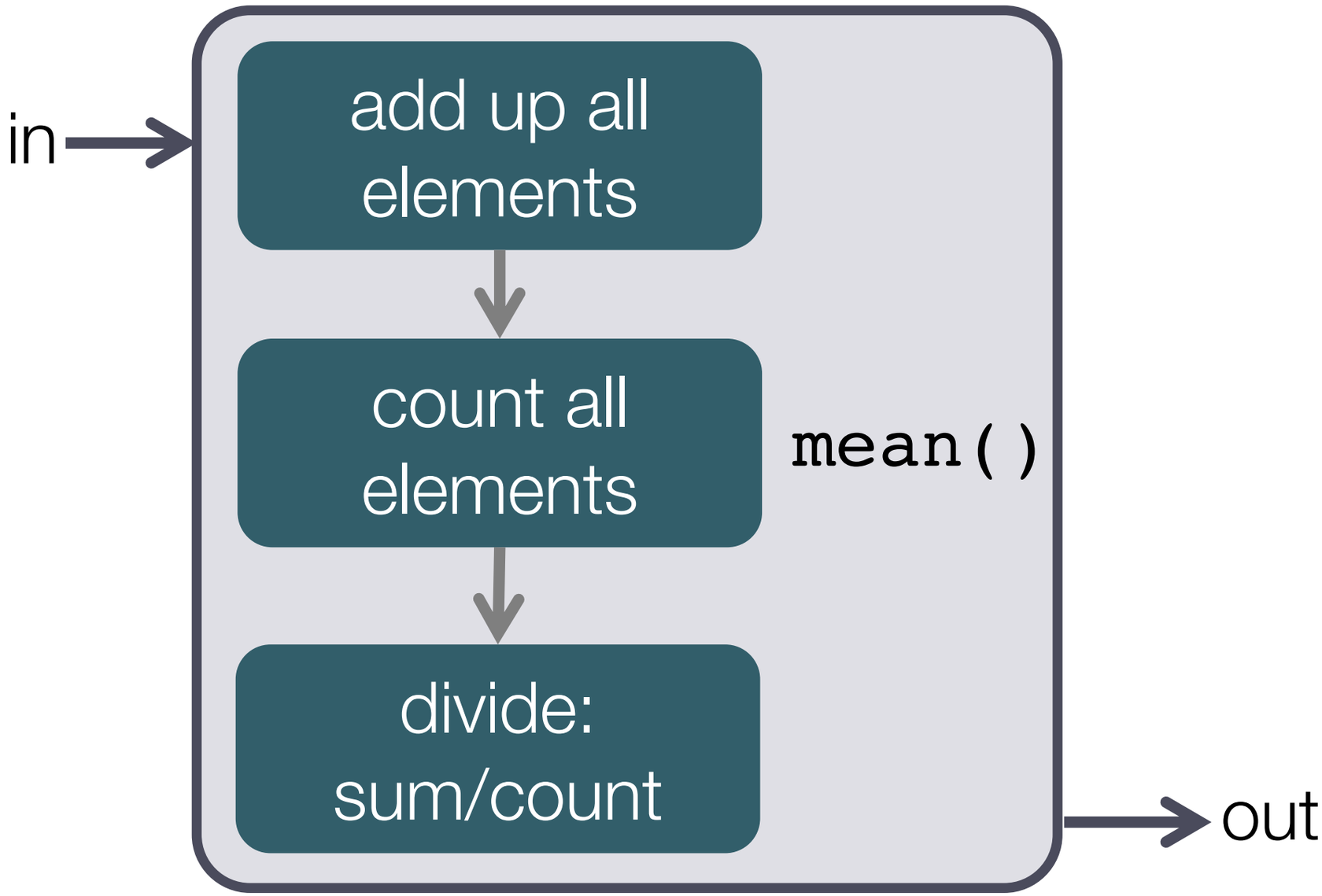
console output

 # indicates comment
computer stops reading
(but you should not!)

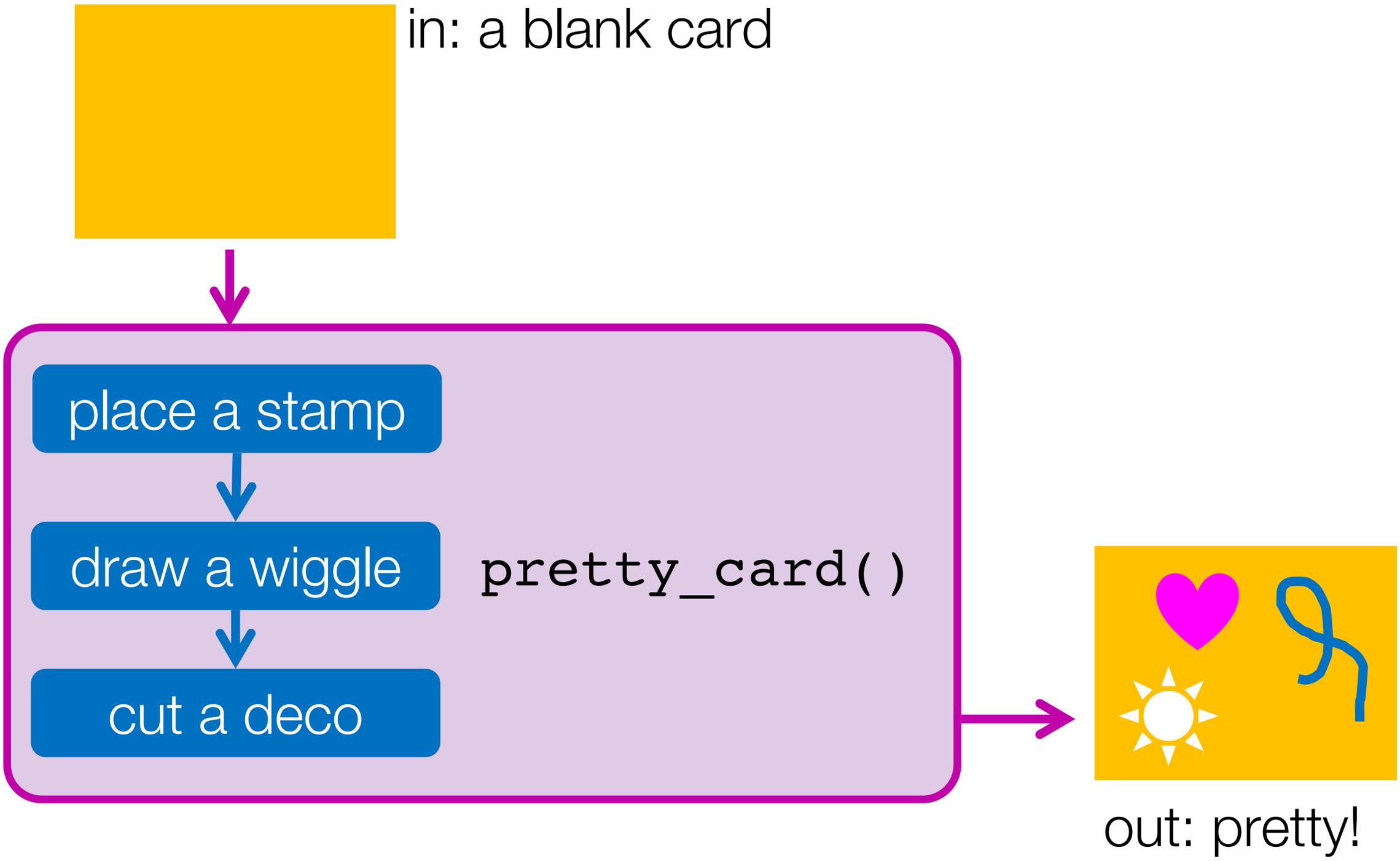
```
# Assigning the value 1 to x
x <- 1
```

Programming: functions

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

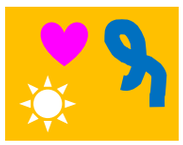


Live exercise: the assembly line of a function



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
```

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`

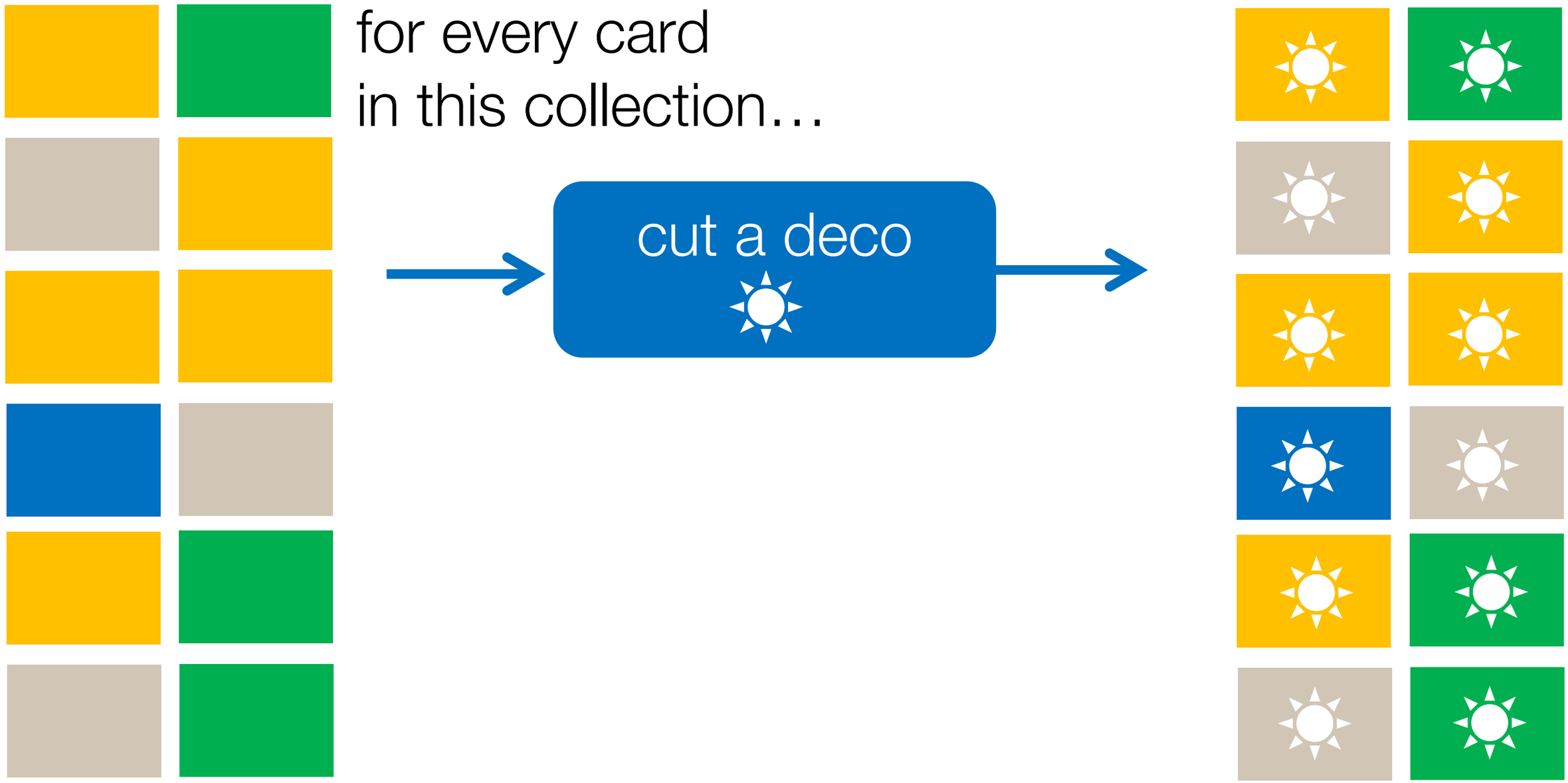
```
apply_calc <- function(x){
    m <- mean(x)
    return(m)
}

> 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
```

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?*

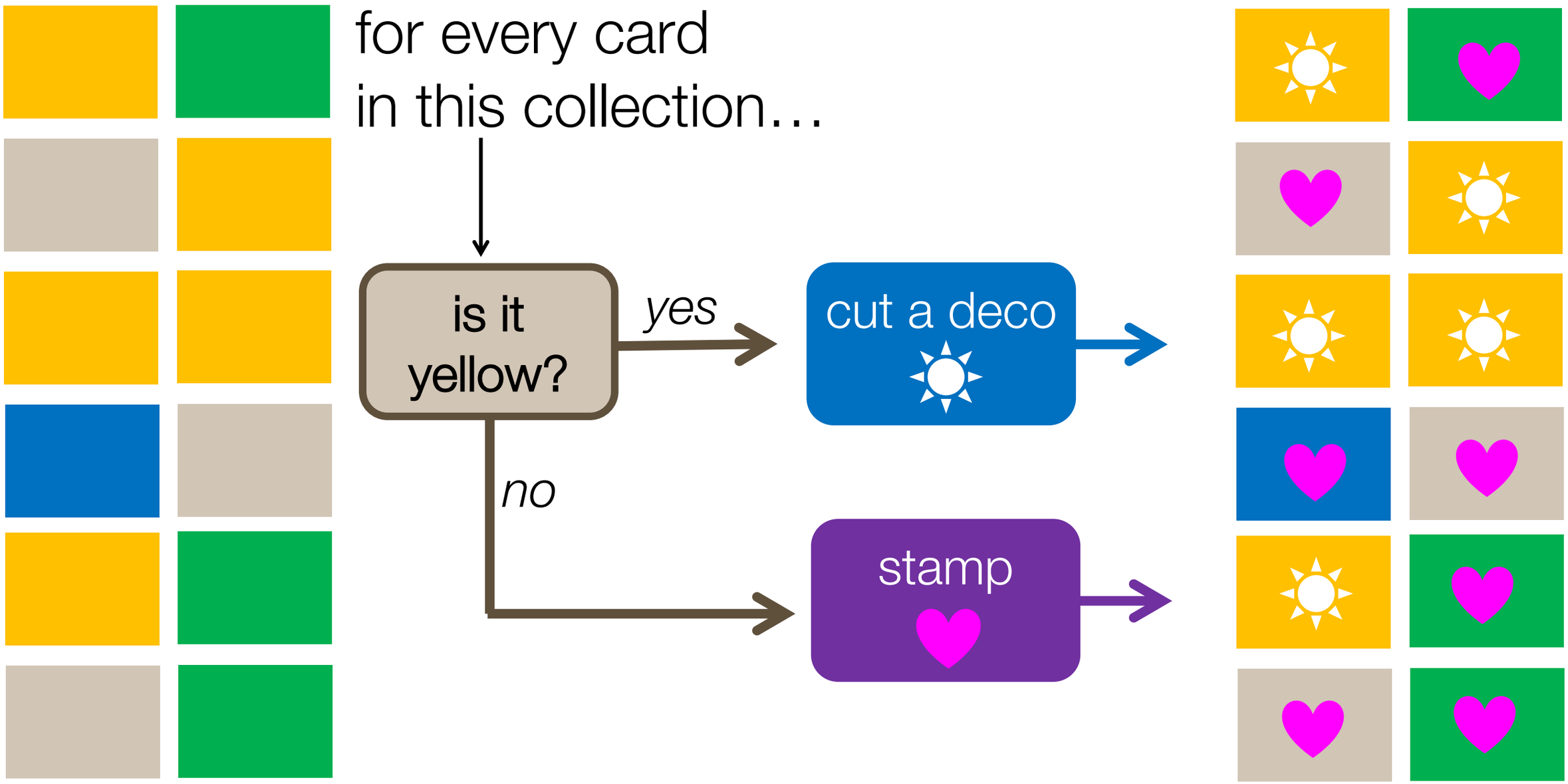
```
for(i in iriscols){
    c <- iris[,i]
    stat <- mean(c)
    print(stat)
}
```

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

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

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){
    c <- 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!

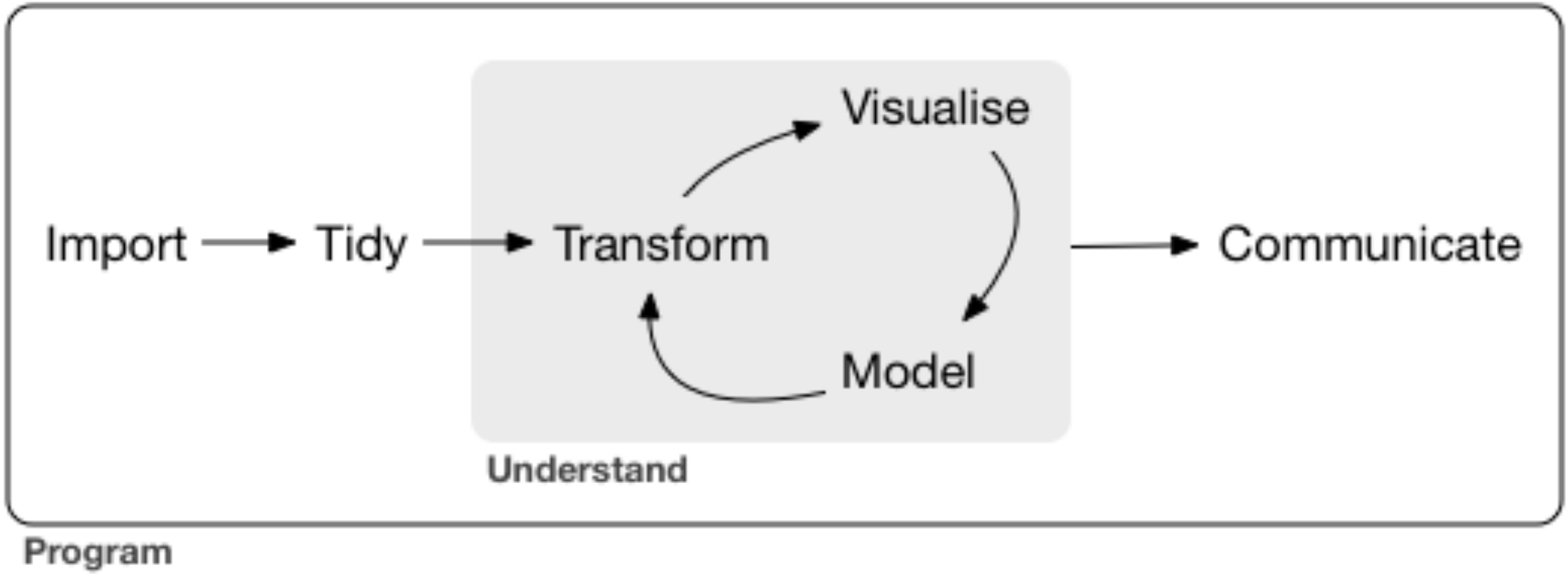
1. Copy-paste the for-loop you made in the previous exercise. Inside this for-loop, add an if-statement, so that **`mean()`** is only performed on numeric vectors.
   Hint: check the function **`is.numeric()`**.

```
for(i in iriscols){
    c <- iris[,i]
    if(is.numeric(c)){
        stat <- mean(c)
        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: Barbara Vreede
## 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: Barbara Vreede
## 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: Barbara Vreede
## 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: Barbara Vreede
## 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!