# Bio Stats II : Lab 1

Gavin Fay

01/27/2016

# Lab schedule

**1/27: Introduction to R and R Studio, working with data**
2/03: Intro to plotting, manipulating data
2/10: Probability, linear modeling, PCA
2/17: Programming practices, conditional statements
2/24: Creating functions, debugging
3/02: Permutation analysis
3/09: Advanced plotting

# Why R?

**Reproducible**

- command line interface encourages organization
- scripts allow others (and you!) to reproduce analyses from end-to-end

**Extensible**

- new methods delivered as developed
- continual expansion through new packages

**Open-source**

- all code can be examined by the user

**Free**

- available to large set of users (and therefore developers)

R is not the only solution out there.
The real goal is not to teach R, but concepts that all programming depends on.

Trevor Branch rule:

> *"Every analysis you do on a dataset will have to be redone 10–15 times before publication. Plan accordingly."*

# Recommended reading

An introduction to R (Venables et al.)
– `http://cran.r-project.org/doc/manuals/R-intro.pdf` -
Today's material: Chapters 1-6.

R reference card 2.0 (Baggott)
– `http://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf`
– Extremely useful handout: put on wall in view of your desk

There are many (many) R books out there. Good for reference.
e.g.
- The R Book (Crawley)
- Modern Applied Statistics with S (Venables and Ripley)
- Dalgaard (2002) Introductory Statistics with R.

# Installing R and R Studio

**R** (`http://r-project.org`)
- download appropriate version for your OS

**R Studio** (`http://rstudio.com`)
- a very good Integrated Development Environment (IDE) for R
provides:
- text editor
- syntax highlighting
- seamless code execution with R

You can use other text editors with R, but RStudio well organized.
(also looks same regardless of Operating System)

# Getting started

Enter instructions at R console command line prompt (>):
e.g. type

```
> 2 + 2
```

R acts as a calculator and returns (prints) the result.

```
> 2 + 2
[1] 4
```

The [1] indicates the first element of the result. It is not important here as our calculation involves scalars.

# Simple commands

```
> 3^2
[1] 9
> 2*(2+2)
[1] 8
> 2*2+2
[1] 6
> log(10)
[1] 2.302585
> exp(1)
[1] 2.718282
> x <- 3
> 2*x
[1] 6
```

The <- means 'assign'. i.e. 'assign a value of 3 to the variable x'. <- is preferable to using =

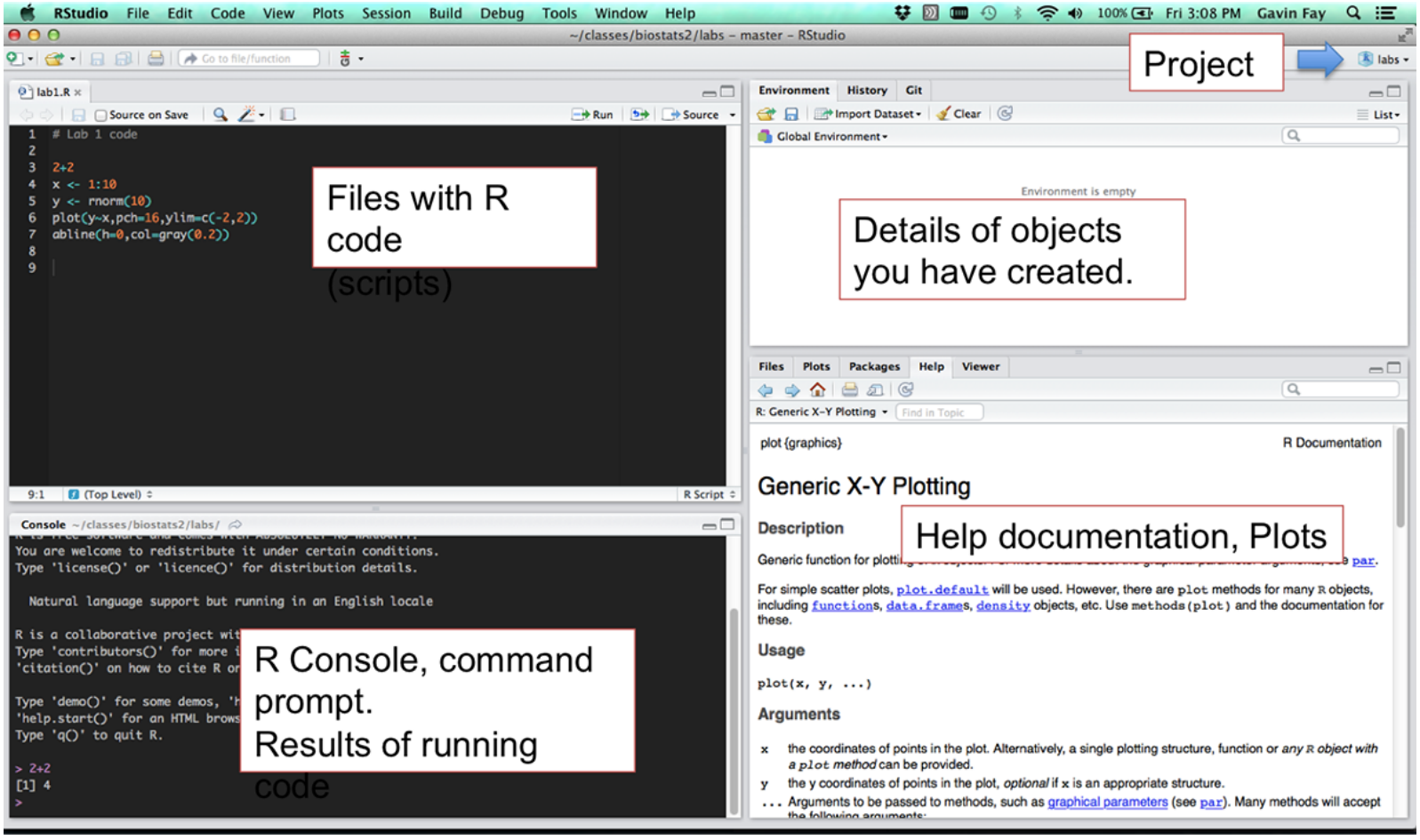
# Scripts and RStudio

- Typing commands into the console can get tedious.
- Scripts are text files containing lines of code.
- Provide a complete record of analyses.
- Code can be run (executed) from these files repeatedly.
- Scripts can be created in a text editor and copied into the R console.

Or. . .

- RStudio integrates scripts, R console, and output in a user-friendly development environment.
- To run code in RStudio, select code and type
  `Ctrl+Enter` (Windows)
  `Command+Enter` (Mac)
  The code will run in the R console.

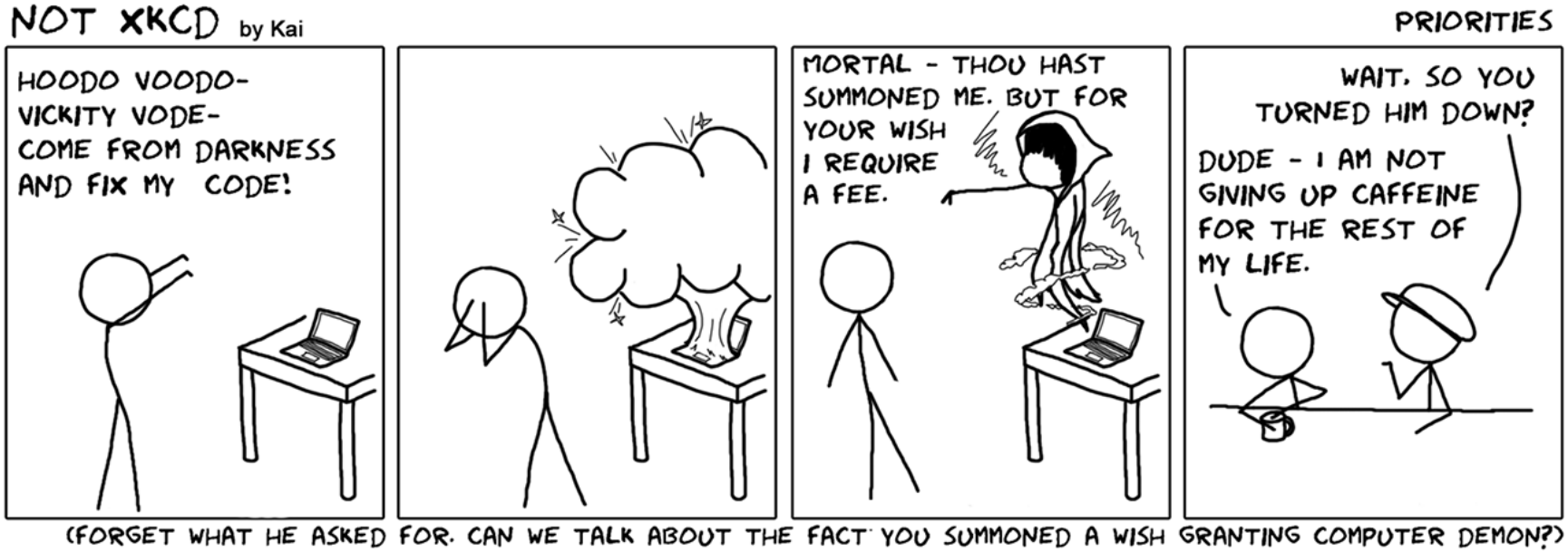
# RStudio


Project
Files with R code (scripts)
Details of objects you have created.
Help documentation, Plots
R Console, command prompt. Results of running code

# Getting Help in R

R is a programming language, there is a learning curve.

Fortunately, there are lots of resources:




**Don't** summon a wish-granting computer demon.

# Getting Help in R

R is a programming language, there is a learning curve.

Fortunately, there are lots of resources:
- help files
- online search results (Seriously, how good is Google?)
- books
- colleagues

```
> ?(mean)
> help("mean")
```

The above both get help for the function `mean`.
Use `help.search("function.name")` to search across packages.

`str(object.name)` shows the structure of an object.

# R Help files (`?mean`)

Common format:

- Description (what the function does)
- Usage (how to use it)
- Arguments (what the function needs, options)
- Value (what does the function return)
- See Also (related functions)
- Examples (sample code showing how the function works)

Read function documentation and explore behavior by running examples!

# Lab exercise 1/4

(Instructions also in lab1_exercise.pdf)

Open a new R script. Save it. (name it lastname_lab1.R or something similar)
At the top of the script, add comments with your name and lab 1. (comments are text preceded by a "#")
Work in pairs or individually.
Submit your R script to Gavin before lab next week.

Write code that evaluates the following when run.

$$7 + 5(4 + 3)$$

$$e^{-5(0.2+0.15)}$$

$$\frac{\sqrt{1 + 2(3 + 2)}}{\ln(3^2 + 2)}$$

# Objects

Common types of objects
- Numbers
- Characters (i.e. text or strings)
- Tables
- Vectors and matrices
- Plots
- Statistical output
- Functions

Objects in R are global

Viewing objects: In RStudio see top-right Workspace tab
More generally:

```
>   print(myobject)
>   myobject
```

`ls()` lists all objects in the workspace.
Use `rm()` to remove an object.

# Data types (modes)

Describe how objects are stored in computer memory.

In R you do **not** need to specify the data type.

Common data types:

- Numeric (integer, floating point numbers or doubles)
- Logical (Boolean, true or false)
- Characters (text or string data)

Types are not always obvious in R, but can be important to know.

# Data types II

```
> myobject <- log(10)
> mode(myobject)
[1] "numeric"
> is.numeric(myobject)
[1] TRUE
> typeof(myobject)
[1] "double"
> newobject <- as.integer(myobject)
> typeof(newobject)
[1] "integer"
> is.character(myobject)
[1] FALSE
> typeof("hello world")
[1] "character"
```

# Vectors

```
> weights <- c(2.3, 5.4, 7.5, 9)
> print(weights)
[1] 2.3 5.4 7.5 9.0
> years <- 2007:2016
> print(years)
 [1] 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016
> years <- seq(from = 2000, to = 2016, by = 2)
> print(years)
[1] 2000 2002 2004 2006 2008 2010 2012 2014 2016
> x <- rep(3, times = 10)
> print(x)
 [1] 3 3 3 3 3 3 3 3 3 3
> rep(1:3, times = 3)
[1] 1 2 3 1 2 3 1 2 3
> rep(1:3, length = 10)
 [1] 1 2 3 1 2 3 1 2 3 1
```

# More on Vectors

Vectors are ordered and can be referred to by element(s) using [ ]

```
> (years <- 2007:2016)
 [1] 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016
> years[3]
[1] 2009
> years[5:6]
[1] 2011 2012
> which(years==2010)
[1] 4
> years[-c(2,4)] # A negative index excludes elements
[1] 2007 2009 2011 2012 2013 2014 2015 2016
```

Vector operations are element-wise

```
> (x <- 1:5)
[1] 1 2 3 4 5
> 2*x
[1]  2  4  6  8 10
```

# Lab exercise 2/4

(Instructions also in lab1_exercise.pdf)

Create vectors using `seq()`, `rep()`, and mathematical operators.
Only use `c()` when absolutely necessary.
*hint* Remember you can get help on a function by typing
`?functionname`

- Positive integers from 1 to 99
- Odd integers between 1 and 99
- The numbers 1,1,1, 2,2,2, 3,3,3
- The numbers -5,-4,-3,-5,-4,-3,-5,-4,-3
- The fractions 1, 1/2, 1/3, 1/4, . . . , 1/10
- The cubes 1, 8, 27, 64, 125, 216

# Useful functions

```
> x <- c(5,3,2,6,3,9,1,18)
> length(x)  # length of vector x
[1] 8
> sort(unique(x))  # sorted vector of unique values in x
[1]  1  2  3  5  6  9 18
> min(x)     # minimum value in x
[1] 1
> max(x)     # maximum value in x
[1] 18
> mean(x)    # mean of x
[1] 5.875
> median(x)  # median of x
[1] 4
> sd(x)      # standard deviation of x
[1] 5.514591
> range(x)   # range of values in x
[1]  1 18
> range(x)[2]  # 2nd element of values returned by range()
[1] 18
> quantile(x) # optional argument 'probs' can be handy
   0%   25%   50%   75%  100%
 1.00  2.75  4.00  6.75 18.00
```

# Boolean logic operators

| Operator | R Code |
|---|---|
| AND | & (&&) |
| OR | \| (\|\|) |
| NOT | ! |
| less than | < |
| greater than | > |
| less than or equal | <= |
| greater than or equal | >= |
| equals | == |
| NOT equal | != |

&& and || are used when asking IF statements.
These only use a single value, not a vector.

# Boolean examples

```
> x <- 7
> x == 7
[1] TRUE
> x < 10
[1] TRUE
> x < -3
[1] FALSE
> x > 0 & x <= 12
[1] TRUE
> x >= 10 | x < 0
[1] FALSE
```

```
> y <- c(4,8)
> y > 5   #returns a logical vector
[1] FALSE  TRUE
> y[y>5]  #returns elements of y that meet condition
[1] 8
> which(y>5) #index of y that meets condition
[1] 2
> any(y>5)
[1] TRUE
> all(y>5)
[1] FALSE
```

# Lab exercise 3/4

Complete the following using the vector `y`:

$$y < -c(3, 2, 15, -1, 22, 1, 9, 17, 5)$$

- Display the first and last values.
- Find the last value for a vector of any length.
- Display the values that are greater than the mean of `y`.
- Display the positions (indices) of the values greater than the mean.
- Are all the values positive?
- Are any of the values equal to the mean?
- Are any of the values equal to the median?

# Other types of objects

**matrices** (more generally, arrays)
- multi-dimensional generalizations of vectors.
- are vectors that can be indexed by two or more indices.

**factors**
- compact ways to handle categorical data.

**lists**
- general form of vector, elements need not be the same type.
- elements often themselves vectors or lists.
- convenient way to return results of statistical computations.

**dataframes**
- matrix-like structures, columns can be of different types.
- often 'data matrices' with one row per observational unit but with (possibly) both numerical and categorical variables.
- experiments are often best described by data frames: treatments are categorical but the response is numeric.

**functions**
- are themselves objects in R which can be stored in the project's workspace.
- provide a simple and convenient way to extend R.

# Dataframes

There are lots of data set examples in R.

e.g. record times for 35 Scottish hill races

```
> library(MASS)
> head(hills,n=3) # shows first few lines. Also tail()
              dist climb   time
Greenmantle    2.5   650 16.083
Carnethy       6.0  2500 48.350
Craig Dunain   6.0   900 33.650
> names(hills)  # get the names of the data frame
[1] "dist"  "climb" "time"
```

Creating dataframes

```
> fish <- c("cod","haddock","dogfish","pollock")
> length <- c(34,23,75,18)
> age <- c(6,3,17,2)
> fish.data <- data.frame(fish=fish,length=length,age=age)
> head(fish.data)
     fish length age
1     cod     34   6
2 haddock     23   3
3 dogfish     75  17
```

# Extracting information from data frames

Use the $ to extract vectors from a data frame

```
> hills$dist
 [1]  2.5  6.0  6.0  7.5  8.0  8.0 16.0  6.0  5.0  6.0 28.0  5.0
[15]  4.5 10.0 14.0  3.0  4.5  5.5  3.0  3.5  6.0  2.0  3.0  4.0
[29]  6.5  5.0 10.0  6.0 18.0  4.5 20.0
```

You can also specify the row index, column index, or both
`object[row,column]`

```
> # extract the element in row 1, column 2
> hills[1,2]
[1] 650
> hills$climb[1]
[1] 650
> # extract the first row
> hills[1,]
             dist climb   time
Greenmantle   2.5   650 16.083
```

```
> # extract all of column 2
> hills[,2]    # also hills[,"climb"]
 [1]  650 2500  900  800 3070 2866 7500  800  800  650 2100 2000
[15] 1500 3000 2200  350 1000  600  300 1500 2200  900  600 2000
[29] 1750  500 4400  600 5200  850 5000
> # exclude column 1, but retain the other columns (1st 3 rows)
> hills[1:3,-1]
             climb   time
Greenmantle    650 16.083
Carnethy      2500 48.350
Craig Dunain   900 33.650
> # extract rows 4 and 7
> hills[c(4,7),]
             dist climb    time
Ben Rha       7.5   800  45.600
Bens of Jura 16.0  7500 204.617
> # extract the rows that are specified by the object x
> x <- c(4,7,nrow(hills))
> hills[x,]
             dist climb    time
Ben Rha       7.5   800  45.600
Bens of Jura 16.0  7500 204.617
Moffat Chase 20.0  5000 159.833
```

# Extracting elements logically

```
> fish <- c("cod", "haddock", "dogfish", "pollock")
> length <- c(34, 23, 75, 18)
> age <- c(6, 3, 17, 2)
> fish.data <- data.frame(fish = fish, length = length, age = age)
> fish.data$age  # a vector
[1]  6  3 17  2
> fish.data$age > 5  # a logical vector
[1]  TRUE FALSE  TRUE FALSE
> fish.data[fish.data$age > 5, ]  #rows where condition is TRUE
     fish length age
1     cod     34   6
3 dogfish     75  17
> # combining conditions
> fish.data[fish.data$age > 5 & fish.data$fish == "dogfish", ]
     fish length age
3 dogfish     75  17
> fish.data[fish.data$length < 25 | fish.data$length >= 50, ]
     fish length age
2 haddock     23   3
3 dogfish     75  17
4 pollock     18   2
```

# Tips and Tricks

**Comments**
Use comments to document the purpose of your code. Anything on a line after a # is ignored by R. RStudio uses a different color to help readability.

**SAVE** your scripts(!), not workspaces. Use meaningful variable names. Adopt a coding style and use consistently.
(e.g. `https://google.github.io/styleguide/Rguide.xml`)

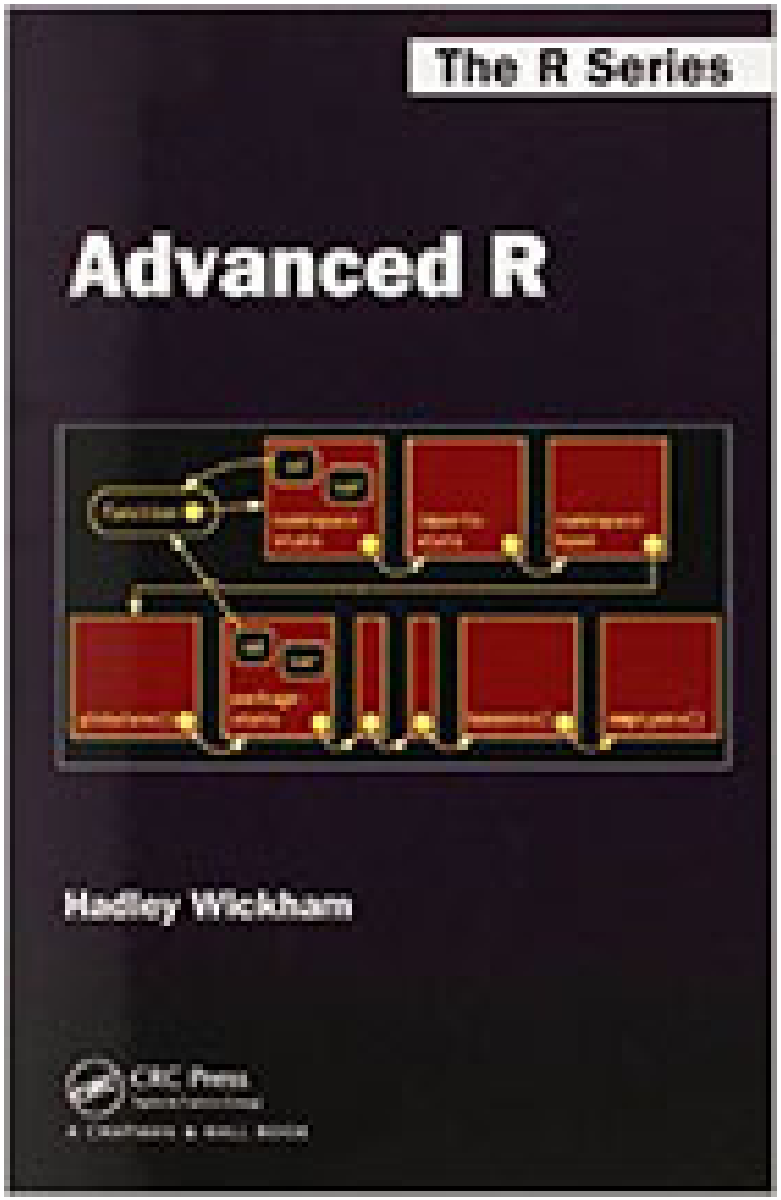
The `str()` function can be incredibly helpful when querying objects.

```
> str(hills)
'data.frame':   35 obs. of  3 variables:
 $ dist : num  2.5 6 6 7.5 8 8 16 6 5 6 ...
 $ climb: int  650 2500 900 800 3070 2866 7500 800 800 650 ...
 $ time : num  16.1 48.4 33.6 45.6 62.3 ...
```

Make use of `help()` documentation.

There are almost always multiple ways of getting the same result.
We'll mostly use low level functions to help you understand how R works.
Some advanced functions are cleaner and do things more quickly.

# Additional reading, Advanced R (Wickham) Chapters 2-3.



# Lab exercise 4/4 (data frames using `hills`)

- Display the first 5 rows of the `hills` dataframe.
- Find the fastest time.
- Display the hill races (and distance, climbs, and times) with the 3 fastest times.
- Extract and display the record time for Cairngorm.
- Find how many hill races have a climb greater than the mean.
- Display the names of the hill races that have a climb greater than the mean.
- Display the names and times of the races that are at least 10 miles long and have a climb greater than 4000 feet.
- Find the positions (indices) of hills that either have a climb greater than 5000 feet or have a record time less than 20 minutes.
- Find the standard deviation of the record times for all races except for the highest climb, the Bens of Jura.
- Display the range (minimum and maximum) of the average speed for the races.
- Find the race that had the fastest average speed.
- **bonus** Find the mean of the record times for races whose names start with letters `A` through `K`.

# Next time...

1/27: Introduction to R and R Studio, working with data
**2/03: Lists, Intro to plotting, manipulating data**
2/10: Probability, linear modeling, PCA
2/17: Programming practices, conditional statements
2/24: Creating functions, debugging
3/02: Permutation analysis
3/09: Advanced plotting