Tutorial 01- R Programming basics

Adapted from here (http://www.andrew.cmu.edu/user/achoulde/)

Basics:

- Everything we'll do comes down to applying **functions** to **data**
- **Data**: things like 7, "seven", 7.000, the matrix $\begin{bmatrix} 7 & 7 & 7 \\ 7 & 7 & 7 \end{bmatrix}$
- **Functions**: things like log, $+$ (two arguments), $<$ (two), mod (two), `mean` (one)

> A function is a machine which turns input objects (**arguments**) into an output object (**return value**), possibly with **side effects**, according to a definite rule

Data building blocks

You'll encounter different kinds of data types

- **Booleans** Direct binary values: `TRUE` or `FALSE` in R
- **Integers**: whole numbers (positive, negative or zero)
- **Characters** fixed-length blocks of bits, with special coding; **strings** = sequences of characters
- **Floating point numbers**: a fraction (with a finite number of bits) times an exponent, like 1.87×10^6
- **Missing or ill-defined values**: `NA`, `NaN`, etc.

Operators (functions)

You can use R as a very, very fancy calculator

Command	Description
`+,-,*,\`	add, subtract, multiply, divide
`^`	raise to the power of
`%%`	remainder after division (ex: `8 %% 3 = 2`)
`( )`	change the order of operations
`log(), exp()`	logarithms and exponents (ex: `log(10) = 2.302`)
`sqrt()`	square root
`round()`	round to the nearest whole number (ex: `round(2.3) = 2`)

Command	Description
`floor(), ceiling()`	round down or round up
`abs()`	absolute value

```
7 + 5 # Addition
```

```
## [1] 12
```

```
7 - 5 # Subtraction
```

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

```
7 * 5 # Multiplication
```

```
## [1] 35
```

```
7 ^ 5 # Exponentiation
```

```
## [1] 16807
```

```
7 / 5 # Division
```

```
## [1] 1.4
```

```
7 %% 5 # Modulus
```

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

```
7 %/% 5 # Integer division
```

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

Operators cont'd.

Comparisons are also binary operators; they take two objects, like numbers, and give a Boolean

```
7 > 5
```

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

```
7 < 5
```

```
## [1] FALSE
```

```
7 >= 7
```

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

```
7 <= 5
```

```
## [1] FALSE
```

```
r   7 == 5
```

```
## [1] FALSE
```

```
r   7 != 5
```

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

Boolean operators

Basically “and” and “or”:

```
(5 > 7) & (6*7 == 42)
```

```
## [1] FALSE
```

```
(5 > 7) | (6*7 == 42)
```

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

(will see special doubled forms, `&&` and `||`, later)

More types

- `typeof()` function returns the type
- `is.`*foo*`()` functions return Booleans for whether the argument is of type *foo*
- `as.`*foo*`()` (tries to) "cast" its argument to type *foo* — to translate it sensibly into a *foo*-type value

Special case: `as.factor()` will be important later for telling R when numbers are actually encodings and not numeric values. (E.g., 1 = High school grad; 2 = College grad; 3 = Postgrad) ###

```
r   typeof(7)

## [1] "double"

r   is.numeric(7)

## [1] TRUE

r   is.na(7)

## [1] FALSE ###

r   is.character(7)

## [1] FALSE

r   is.character("7")

## [1] TRUE

r   is.character("seven")

## [1] TRUE

r   is.na("seven")

## [1] FALSE
```

Variables

We can give names to data objects; these give us **variables**

A few variables are built in:

```
pi
```

```
## [1] 3.141593
```

Variables can be arguments to functions or operators, just like constants:

```
pi*10
```

```
## [1] 31.41593
```

```
cos(pi)
```

```
## [1] -1
```

Assignment operator

Most variables are created with the **assignment operator**, `<-` or `=`

```
time.factor <- 12
time.factor
```

```
## [1] 12
```

```
time.in.years = 2.5
time.in.years * time.factor
```

```
## [1] 30
```

The assignment operator also changes values:

```
time.in.months <- time.in.years * time.factor
time.in.months
```

```
## [1] 30
```

```
time.in.months <- 45
time.in.months
```

```
## [1] 45
```

- Using names and variables makes code: easier to design, easier to debug, less prone to bugs, easier to improve, and easier for others to read
- Avoid “magic constants”; use named variables
- Use descriptive variable names
- Good: `num.students <- 35`
- Bad: `ns <- 35`

The workspace

What names have you defined values for?

```
ls()
```

```
## [1] "time.factor"    "time.in.months" "time.in.years"
```

Getting rid of variables:

```
rm("time.in.months")
ls()
```

```
## [1] "time.factor"   "time.in.years"
```

First data structure: vectors

- Group related data values into one object, a **data structure**
- A **vector** is a sequence of values, all of the same type
- `c()` function returns a vector containing all its arguments in order

```
students <- c("Sean", "Louisa", "Frank", "Farhad", "Li")
midterm <- c(80, 90, 93, 82, 95)
```

- Typing the variable name at the prompt causes it to display

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

Indexing

- `vec[1]` is the first element, `vec[4]` is the 4th element of `vec`

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

```
students[4]
```

```
## [1] "Farhad"
```

- `vec[-4]` is a vector containing all but the fourth element

```
students[-4]
```

```
## [1] "Sean"   "Louisa" "Frank"  "Li"
```

Vector arithmetic

Operators apply to vectors "pairwise" or "elementwise":

```
final <- c(78, 84, 95, 82, 91) # Final exam scores
midterm # Midterm exam scores
```

```
## [1] 80 90 93 82 95
```

```
midterm + final # Sum of midterm and final scores
```

```
## [1] 158 174 188 164 186
```

```
(midterm + final)/2 # Average exam score
```

```
## [1] 79 87 94 82 93
```

```
course.grades <- 0.4*midterm + 0.6*final # Final course grade
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

Pairwise comparisons

Is the final score higher than the midterm score?

```
midterm
```

```
## [1] 80 90 93 82 95
```

```
final
```

```
## [1] 78 84 95 82 91
```

```
final > midterm
```

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

Boolean operators can be applied elementwise:

```
(final < midterm) & (midterm > 80)
```

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

Functions on vectors

Command	Description
`sum(vec)`	sums up all the elements of `vec`
`mean(vec)`	mean of `vec`
`median(vec)`	median of `vec`
`min(vec), max(vec)`	the largest or smallest element of `vec`
`sd(vec), var(vec)`	the standard deviation and variance of `vec`
`length(vec)`	the number of elements in `vec`
`pmax(vec1, vec2), pmin(vec1, vec2)`	example: `pmax(quiz1, quiz2)` returns the higher of quiz 1 and quiz 2 for each student
`sort(vec)`	returns the `vec` in sorted order

Command	Description
`order(vec)`	returns the index that sorts the vector `vec`
`unique(vec)`	lists the unique elements of `vec`
`summary(vec)`	gives a five-number summary
`any(vec), all(vec)`	useful on Boolean vectors

Functions on vectors

```
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

```
mean(course.grades) # mean grade
```

```
## [1] 86.8
```

```
median(course.grades)
```

```
## [1] 86.4
```

```
sd(course.grades) # grade standard deviation
```

```
## [1] 6.625708
```

More functions on vectors

```
sort(course.grades)
```

```
## [1] 78.8 82.0 86.4 92.6 94.2
```

```
max(course.grades) # highest course grade
```

```
## [1] 94.2
```

```
min(course.grades) # lowest course grade
```

```
## [1] 78.8
```

Referencing elements of vectors

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

Vector of indices:

```
students[c(2,4)]
```

```
## [1] "Louisa" "Farhad"
```

Vector of negative indices

```
students[c(-1,-3)]
```

```
## [1] "Louisa" "Farhad" "Li"
```

More referencing

`which()` returns the `TRUE` indexes of a Boolean vector:

```
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

```
a.threshold <- 90 # A grade = 90% or higher
course.grades >= a.threshold # vector of booleans
```

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

```
a.students <- which(course.grades >= a.threshold) # Applying which()
a.students
```

```
## [1] 3 5
```

```
students[a.students] # Names of A students
```

```
## [1] "Frank" "Li"
```

Named components

You can give names to elements or components of vectors

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

```
names(course.grades) <- students # Assign names to the grades
names(course.grades)
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

```
course.grades[c("Sean", "Frank","Li")] # Get final grades for 3 students
```

```
##  Sean Frank    Li
##  78.8  94.2  92.6
```

Note the labels in what R prints; these are not actually part of the value

Useful RStudio tips

Keystroke	Description
<tab>	autocompletes commands and filenames, and lists arguments for functions. Highly useful!

Keystroke	Description
<up>	cycle through previous commands in the console prompt
<ctrl-up>	lists history of previous commands matching an unfinished one
<ctrl-enter>	paste current line from source window to console. Good for trying things out ideas from a source file.
<ESC>	as mentioned, abort an unfinished command and get out of the + prompt

Checkpoint:

Complete lab01 to check your understanding.