R Camp Cheat Sheet: The Basics
R Camp 2018
Prof. Jane Sumner

Symbol	**What It is**	**Used For**	**Examples**
`<-`	Assignment arrow	Naming an object	`X <- 2.4` tells R that any time you use x you mean 2.4
`#`	Pound sign, or hashtag	“Commenting out”. (Telling R not to execute some code.) Good for making notes to yourself.	`x <- 55 # a number` `x` `[1] 55`
`==`	Equals	Asking “R, are these two things equal?”	`x <- 3` `x == 5` `[1] FALSE`
`&`	And	Grouping together two statements to determine if they are BOTH true	`x <- 3` `x==5 & x==3` `[1] FALSE`
`|`	Or	Grouping together two statements to determine if EITHER of them are true	`x <- 3` `x==5 | x==3` `[1] TRUE`
`c`	Concatenate function	Putting individual numbers or words into a vector	`X <- c(3,2,1,5,5)` `Y <- c(“Monday”, “Tuesday”, “Wednesday”)`
`=`	Argument assignment	Passing arguments into a function (in the example: `x`, `y`, `conf.level` are all arguments)	`t.test(x=c(2,3,4,2),y=c(0,1,4,5),conf.level=0.9)`
`( )`	Parentheses	Containing arguments you’re passing to a function	`cor(x=c(2,3,4.2),y=c(0,1,4))` `[1] 0.9739854`
`[ ]`	Square brackets	‘Plucking’ a value or values from a vector, matrix, or data frame	`my.data[3,2]` will pull the value in the third row, second column from the dataframe or matrix named my.data `my.data[3,]` will pull all the values in the third column of my.data `my.data[,2]` will pull all the

			values in the second row of my.data

`[[ ]]`	Double square brackets	‘Plucking’ an object from a list	`my.list <-`
			`list(humans=c("Jane","Ed","Lars"),cats=c("Maggie","Mikhail"))`
			`my.list[[1]]`
			`[1] "Jane" "Ed"   "Lars" # returns the elements of the first object`
			`my.list[1]`
			`$humans`
			`[1] "Jane" "Ed"   "Lars"`
			`# returns the first object`

`{ }`	Curly brackets
	(aka ‘curly bois’)
		Grouping lines of code together rather than executing them line-by-line (mostly used for functions and loops)	`{`
			`X <- 2.4`
			`Y <- 3*X+2`
			`X+Y`
			`}`
			None of these lines will be executed (completed) until the final curly boi ‘closes’ the code segment

`$`	Dollar sign	Pulling an object out of a data frame or list *by name*.	`my.data <-`
			`data.frame(x=c(1,2,3),y=c(0,9,8))`
			`my.data$x`
			`[1] 1 2 3`
			`my.data[,1]`
			`[1] 1 2 3`
			(in small data frames, these are equivalent, in large data sets, you probably don’t want to have to figure out that ‘democracy’ is column 154, mydata$democracy is easier)

Name	**What is is**	**Example**

Function	It takes some input(s), does something to the input(s), and spits out a desired product. There are tons built in, but you can also make your own, which is more efficient than doing the same thing over and over by hand.	`plot(x,y)`
		`c(5,2,2,2)`

Vector	A one-dimensional storage object. Usually formed by c()	`x <- c(9,0,2,1,0)`
		`x`
		`[1] 9 0 2 1 0`

Matrix	A two-dimensional (rows and columns) storage object where all the data are of the same type
	Note: if you form a matrix and your data *aren't* all the same type, it'll force them to be. Usually this means your numbers become characters/strings (words)
		`my.data <-`
		`matrix(c(1,2,3,4,5,6,7,8,9,0),nrow`
		`=5,ncol=2,byrow=T)`
		`my.data`
		`     [,1] [,2]`
		`[1,]    1    2`
		`[2,]    3    4`
		`[3,]    5    6`
		`[4,]    7    8`
		`[5,]    9    0`

Dataframe (or Data Frame)	A two-dimensional (rows and columns) storage object where the data *do not* need to all be the same type. Very similar to an Excel spreadsheet.	`my.data <-`
		`data.frame(x=c(13,14,15,16,17),y=c`
		`("Monday","Tuesday","Wednesday","T`
		`hursday","Friday"))`
		`my.data`
		`  date       day`
		`1   13    Monday`
		`2   14   Tuesday`
		`3   15 Wednesday`
		`4   16  Thursday`
		`5   17    Friday`

List	A collection of different objects OR storage objects (!!!) (meaning: you can have a list of data frames, or matrices, or vectors). The most flexible way to store things.	`my.list <-`
		`list(my.name="Jane",family.names=c`
		`("Ed","Lars"),some.numbers=c(2:10)`
		`)`
		`my.list`
		`$my.name`
		`[1] "Jane"`
		
		`$family.names`
		`[1] "Ed"   "Lars"`
		
		`$some.numbers`
		`[1]  2  3  4  5  6  7  8  9 10`
		
		`my.list$my.name`
		`[1] "Jane"`

Character	A data type that's not a number, basically. Sometimes called a 'string' or 'character string'.	`x <- "Monday"`
		`typeof(x)`
		`[1] "character"`

Factor	A type of data that looks like a string but has a specified ordering. Factors sometimes get in the way, but when they are useful, they're magical. Example: ordered logit/probit models.	`x <- sample(c(1:10),5)`
		`table(x)`
		`x`
		`2 3 4 5 9`
		`1 1 1 1 1`

	See also example to the right, where we retain knowledge that we *could* have had a 5 or a 6 when x is a factor (below), unlike when it's a number (above).	`x <- sample(as.factor(c(1:10)),5)`
		`table(x)`
		`x`
		` 1  2  3  4  5  6  7  8  9 10`
		` 1  1  1  1  0  0  1  0  0  0`

FAQs:

(1) When do I use quotation marks?!

If you're referencing a variable -- something that has inherent value to R, that R knows is just standing in for something else -- you do not use quotation marks. Otherwise, if it's not a number, use quotation marks.

Example:

```
my.data <- read.csv("the-data-set.csv")
```

Here you use quotation marks because "the-data-set.csv" means nothing at all to R. It's just the name of a file in a folder on your computer.

```
head(my.data)
```

Here you do not use quotation marks, because R knows that my.data is an object that references a data set. `head`, by the way, displays the first six rows of a data set, unless you tell it another number -- e.g., `head(my.data,10)`

(2) What's an "unexpected" parenthesis/bracket/comma/etc.?!

You gave R too many of something. It wasn't "expecting" to see an extra parenthesis, bracket, comma, etc., and now it wants to politely ask you what you really meant.

Example:

```
x <- c(3,3,3))
Error: unexpected ')' in "x <- c(3,3,3))"
my.data[3,]]
Error: unexpected ']' in "my.data[3,]]"
```