# Intro to R

October 5, 2021

## Demo: Introduction to R

### R syntax

There are some important basic syntax things to learn in R:

- **Comments** R will ignore everything on a line that follows a `#`:

```
# this is a comment
1 + 2 + 3 # this is also a comment
```

```
## [1] 6
```

- **Assignment** We can assign values to variables using `<-`:

```
# set x to 98.6
x <- 98.6
```

*Note that x appears in the environment panel!*

We can also ask R what it is, by typing `x`:

```
x
```

```
## [1] 98.6
```

It is always good practice to give your variables meaningful names

```
bodyTemp <- 98.6

# side note: you can also assign variables with `x=98.6` but it is better practice to use `<-`
# there is a shortcut for this symbol: Mac (Option + -), Windows (Alt + -)
```

- **Arithmetic** R has usual arithmetic operations (`+ - * / ^`)

```
# convert body temp from F to C
(5/9)*(bodyTemp-32)
```

```
## [1] 37
```

```
# we can also assign this to another variable
bodyTempC <- (5/9)*(bodyTemp-32)
bodyTempC
```

```
## [1] 37
```

It can also do things like square roots, logs, etc.

```
# square root of 4
sqrt(4)
```

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

```
# the natural log of 1
log(1)
```

```
## [1] 0
```

```
# e^2.5
exp(2.5)
```

```
## [1] 12.18249
```

- **Comparison** R can compare things:

```
# == means 'is equal too?'
3 + 2 == 5 # should be true
```

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

```
2 + 2 == 5 # should be false
```

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

```
# > and >= are 'greater than' and 'greater than or equal to'
5 > 3
```

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

```
# and < and <= are 'less than' and 'less than or equal to'
3 < 3 + 2
```

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

**Getting help!** R has a help function, called `help()`, which will pull up the manual page for the function. You give as an argument the thing you want help with:

```
help(sqrt)

# you can also use ?
?sqrt
```

The help panel in R studio also has a search panel. Try pulling up the help page for the log function.

- The **usage** line tells you the syntax of the function. Sometimes multiple functions are documented on the same help page. Let's focus on the first on on the log page, the one with the usage line `log(x, base = exp(1))`
- This tells us that we can feed the `log()` function two arguments: `x` and `base`. It's also telling us that the base argument has a default value, namely the natural base `e = e1`, or `exp(1)`. This means that if we don't specify the base, it will assume we want a natural log.
- Further down in the help page you can read about the arguments and the return values for the function
- Arguments can be passed by name or passed by position

So, for example:

```
log(x=4) # natural log of 4
```

```
## [1] 1.386294
```

```
log(4) # also natural log of 4
```

```
## [1] 1.386294
```

```
log(x=4,base=2) # log base 2 of 4
```

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

```
log(4,2) # same -- it assumes that x is first and base is second, just like the help says
```

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

```
log(base=2,x=4) # notice!  order doesn't matter if you tell it which argument is which!
```

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

## Data structures in R

Data in R can exist in one of several **types** or **storage modes**, including:

- **numeric** - holds numeric values
- **character** - holds string values (i.e. "x" or "hello world")
- **logical** - holds Boolean TRUE or FALSE values (can also be shown as 1 or 0, respectively)
- **factor** and **ordered** - hold *categorical* data
- **NA** - denotes a missing value

You can find out what type of value something is with the `is.` functions:

```r
x <- 5

is.logical(x)
```

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

```r
is.character(x)
```

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

```r
is.na(x)
```

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

```r
is.numeric(x)
```

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

```r
# mode will tell you the storage mode of a value
mode(x)
```

```
## [1] "numeric"
```

```r
# str (structure) displays the internal structure of an object
# (also helpful for more complex objects like dataframes or lists)
str(x)
```

```
##  num 5
```

Changing from one type to another is called coercion. You can coerce values with the "as." functions:

```r
y <- as.character(x)

y # quotes will indicate that it's a character string
```

```
## [1] "5"
```

```r
# can't add a string to a character!
# uncomment to try to run this line
# y + 2

# let's try coercing it back:
as.numeric(y) + 2
```

```
## [1] 7
```

Note that not all strings are coercible!

```
 # now y is the LETTER 'x'
y<-"x"

# note that it's the LETTER, not the variable x!
y == x
```

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

```
# can't coerce the letter x to a number
# as.numeric(y)
```

And finally, a few examples of boolean logic:

```
# & means 'logical and'
# something cannot be true and false, so returns false
TRUE & FALSE
```

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

```
# | means 'logical or'
# everything is either true or false...
TRUE | FALSE
```

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

```
# remember, TRUE also = 1 and FALSE = 0, we can add TRUE and FALSE
TRUE + TRUE
```

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

These **atomic variables** can be organized into **data structures**:

- **vector** - a vector of values all having the same mode
- **matrix** - a 2-dimensional array of values having the same mode
- **list** - like a vector, but the elements can be of different types (they can even be vectors or lists themselves)
- **data.frame** - a specialized structure for tables of values (you can think of this as a list of vectors all with the same length, or as a special sort of matrix where the columns can be different types)

## Vectors in R

As mentioned above, vectors are a sequence of elements all having the same type, such as (1, 2, 3, 4) or ("hello","world"). NA's, which are a special type for missing data, may also be mixed in.

### Creating vectors

We can **concatenate** vectors with `c()`:

```
# c() concatenates its arguments into a vector, eg:
x <- c(1,2,4,8)
x
```

```
## [1] 1 2 4 8
```

```
# note that x is both numeric (the storage mode) AND a vector!
is.numeric(x)
```

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

```
is.vector(x)
```

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

```
# note also that a single value is still a vector -- a vector of length 1:
is.vector(5.5)
```

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

```
# "a:b" can be used as a shorthand for "from a to b in steps of 1"
y <- 10:20
y
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```
# you can concatenate two vectors, eg:
c(x,y)
```

```
##  [1]  1  2  4  8 10 11 12 13 14 15 16 17 18 19 20
```

```
# R will COERCE values if you have mixed types in a vector:
x <- c(1,2,3,"hello","world")
x # note that the first three elements are CHARACTERS, not numeric!
```

```
## [1] "1"     "2"     "3"     "hello" "world"
```

### Operating on vectors

Most (but not all) functions in R operate elementwise, meaning they apply to all elements of a vector. For example,

```
y  # just to remind ourselves
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```
1+y
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20 21
```

```
5*y
```

```
##  [1]  50  55  60  65  70  75  80  85  90  95 100
```

```
y^2
```

```
##  [1] 100 121 144 169 196 225 256 289 324 361 400
```

```
sqrt(y)
```

```
##  [1] 3.162278 3.316625 3.464102 3.605551 3.741657 3.872983 4.000000 4.123106
##  [9] 4.242641 4.358899 4.472136
```

Some functions operate on the complete vector, eg:

```
# find the sum of vector y
sum(y)
```

```
## [1] 165
```

```
# find the mean of y
mean(y)
```

```
## [1] 15
```