Introduction to R Coding

Election Data Science

Peter Licari

2020-09-08

Let's get coding in R

Let's get acquainted with R Studio.

- Console
- R scripts
- Plot window

Data types in R

1. Character (`"Alabama"`)
2. Numeric (`1.3`, `2.0`)
3. Integer (`13`, `13L`)
4. Logical (`TRUE` and `FALSE`)
5. Imaginary (`i`)
6. Raw

Don't really use 5 and 6 much, but they're important to know.

```r
typeof("Alabama")
```

```
## [1] "character"
```

```r
typeof(1.3)
```

```
## [1] "double"
```

```r
typeof(13L)
```

```
## [1] "integer"
```

```r
typeof(TRUE)
```

```
## [1] "logical"
```

There are also missing values:

- `NA` `Inf` `NaN` `NULL`

You can combine multiple elements together with `c()`

```r
c(1,2,3,4,5)
```

```
## [1] 1 2 3 4 5
```

```r
c("Alabama","Alaska","Arizona","Arkansas")
```

```
## [1] "Alabama"  "Alaska"   "Arizona"  "Arkansas"
```

```r
c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

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

Mathematical, Relational, and Logical Operations

- **Mathematical:** `+ ^ - / * %% %/%`
- **Relational Operations:** `< > >= <= == != ()`
- **Logical:** `! & && | ||`

```
5*4
```

```
## [1] 20
```

```
42000/1609
```

```
## [1] 26.10317
```

```
5 <= 8
```

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

```
TRUE | FALSE
```

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

You can also assign objects to other objects (including the same object)

```r
x <- 1
x
```

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

```r
y <- x
y
```

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

```r
y <- y + 3
y
```

```
## [1] 4
```

And do operations on longer objects:

```r
y <- c(1,2,3,4,5,6)
y * 5
```

```
## [1]  5 10 15 20 25 30
```

```r
y < 3
```

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

```r
typeof(y)
```

```
## [1] "double"
```

```r
sum(y)
```

```
## [1] 21
```

You can nest functions and store outputs

```r
y <- c(1,2,3,4,5,6)
y1 <- typeof(mean(y))
y
```

```
## [1] 1 2 3 4 5 6
```

```r
y1
```

```
## [1] "double"
```

It's similar to when you'd have $h(g(f(x)))$ back in algebra. Things are evaluated inside out. (Mostly).

You can create your own functions

```r
f <- function(x){
  x^2-1
}

f(2)
```

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

```r
values <- c(1,2,35,3,5,323,44,NA,445)

f(values)
```

```
## [1]      0      3   1224      8     24 104328   1935     NA 198024
```

Functions can take in multiple values.

```r
f <- function(x,y){
  print(x ^ y)

}

f(13,2)
```

```
## [1] 169
```

You can loop through things.

```r
for(i in c("apple","banana","orange","cucumber")){
  x <- paste(i,"is my favorite fruit", sep = " ")
  print(x)
  }
```

```
## [1] "apple is my favorite fruit"
## [1] "banana is my favorite fruit"
## [1] "orange is my favorite fruit"
## [1] "cucumber is my favorite fruit"
```

And use logical relationships to better control the flow of your steps.

```r
for(i in c("apple","banana","orange","cucumber")){
  if(i=="apple"){
  x <- paste(i,"is my favorite fruit", sep = " ")
  print(x)
  }
  else if(i == "banana"){
    x <- paste(i,"is pretty good on a sunday", sep = " ")
    print(x)
  }  else {
    x <- paste(i, "is as terrible as pineapple on pizza.", sep = " ")
    print(x)
  }
}
```

```
## [1] "apple is my favorite fruit"
## [1] "banana is pretty good on a sunday"
## [1] "orange is as terrible as pineapple on pizza."
## [1] "cucumber is as terrible as pineapple on pizza."
```

The limits of R.

- You can do a *lot* with so-called "Base R." But there's a lot you either can't do or, frankly, don't want to do because it's too onerous.
- Because R is open source, thousands of people have contributed "packages" that you can download to add additional functionality.
- You can install many of these packages with the `install.packages` function.

```r
install.packages("nnet")
```

More on packages

- Once you've installed the package, you don't have to do it again (unless you want updates.)
- But *installing* it does not mean it's immediately available like the Base R functions (e.g., `sum`, `mean`, etc.)
    - When you run R (and RStudio), you're operating within a particular *environment.*
    - Everytime you reopen RStudio, you're operating in a *different* environment.
- Two options for package management.
    - Use `::` to reference functions within specific packages (`nnet::multinom`)
    - Use the `library` function at the beginning of your script (generally, but not always, preferred)

```r
library(nnet)
```

The goal when writing code isn't to write code that works and that is only decipherable by machines. It's to write functional code that's readable by people. Either collaborators or your future self.

Style guidelines and tips.

- Make your object names understandable and readable.
  - If you have a row of voter IDs, don't call it `var9930`, call it `Voter_ID`
  - Be consistent with your naming conventions: snake_case, camelCase, dot.case, etc.
- Use `#` to write comments to yourself and your collaborators in your code.

```r
# The machine won't read this line, but you'll be able to.

# This function takes its input and squares it.
f<-function(x){x^2}
f(2)
```

```
## [1] 4
```

Style guidelinees (continued)

- Limit your lines to ~80-100 characters. -You can end lines with `,` and/or `+` depending on the function.

- Don't be afraid of white space and `()`

```
x<-7*max(c(1,2,3,4,5))+1
# Is way less readable

x <- (7 * max( c(1,2,3,4,5) )) + 1
# Than this.
```

- Check out this style guide for other good practices

Quick look over at RStudio.