Intro to R

Functions

Introduction to R for Public Health Researchers

Writing your own functions

So far we’ve seen many functions, like `c()`, `class()`, `filter()`, `dim()` …

Why create your own functions?

- Cut down on repetitive code (easier to fix things!)
- Organize code into manageable chunks
- Avoid running code unintentionally
- Use names that make sense to you

Writing your own functions

Here we will write a function that returns the second element of a vector:

```
return2a = function(x) x[2]
```

When you run the line of code above, you make it ready to use (no output yet!). Let’s test it!

```
return2a(x = c(1,4,5,76))
```

```
[1] 4
```

Writing your own functions

Adding the curly brackets - `{ }` - allows you to use functions spanning multiple lines:

```r
return2b = function(x) {
  x[2]
}
return2b(x = c(1,4,5,76))
```

```
[1] 4
```

Writing your own functions

If we want something specific for the function's output, we use `return()`:

```
return2c = function(x) {
  output = x[2]
  return(output)
}
return2c(x = c(1,4,5,76))
```

```
[1] 4
```

Writing your own functions

Review: The syntax for a function is:

```
functionName = function(inputs) {
 <function body>
return(value)
}
```

Writing your own functions

Functions can take multiple inputs. Maybe you want users to select which element to extract:

```
return_n = function(x, n) x[n]
return_n(x = c(1,4,5,76), n = 3)
```

```
[1] 5
```

Writing your own functions

Functions can have “default” arguments. This lets us use the function without using an argument later.

```r
return_n2 = function(x = c(1,2,3), n = 2) x[n]
return_n2()
```

```
[1] 2
```

Writing a simple function

Let’s write a function, `sqdif`, that:

1. takes two numbers `x` and `y` with default values of 2 and 3.
2. takes the difference
3. squares this difference
4. then returns the final value

Writing a simple function

```
sqdif <- function(x=2,y=3) (x-y)^2

sqdif()
```

```
[1] 1
```

```
sqdif(x=10,y=5)
```

```
[1] 25
```

```
sqdif(10,5)
```

```
[1] 25
```

Writing your own functions

Try to write a function called `top()` that takes a `tibble`, and returns the first `n` rows and columns, with the default value of `n=5`.

Writing your own functions

Try to write a function called `top()` that takes a `tibble`, and returns the first `n` rows and columns

```
top = function(df, n=5) df[1:n, 1:n]
bike = jhur::read_bike()
```

```
top(bike) # Note that we are using the default value for n
```

```
# A tibble: 5 x 5
  subType name            block                    type          numLanes
  <chr>   <chr>           <chr>                    <chr>            <dbl>
1 <NA>    <NA>            <NA>                     BIKE BOULEVARD        1
2 <NA>    <NA>            <NA>                     SIDEPATH              1
3 <NA>    <NA>            <NA>                     SIGNED ROUTE          1
4 <NA>    HUNTINGDON PATH <NA>                     SIDEPATH              1
5 STCLN   EDMONDSON AVE   5300 BLK EDMONDSON AVE   BIKE LANE             1
```

Custom functions in `apply`

You can also “apply” functions easily with `sapply()`.

These functions take the form:

```
sapply(???, some_function) # No parentheses on the function
```

Custom functions in `apply`

```
sapply(bike, class)
```

```
      subType          name         block          type      numLanes 
  "character"   "character"   "character"   "character"     "numeric" 
      project         route        length dateInstalled 
  "character"   "character"     "numeric"     "numeric" 
```

```
sapply(bike$length, log)
```

```
  [1] 6.077041 6.932130 8.229330     -Inf 5.198085 4.997062 5.901772 5.569763
  [9] 6.544916 3.764470 4.216844 4.330822 3.897970 4.248787 6.639784 5.136674
 [17] 7.452495 4.308122 4.555367 4.790242 5.009860 5.522817 5.549253 5.523975
 [25] 5.528365 5.527894 4.868185 4.965629 5.842121 5.520994 5.373899 5.281683
 [33] 5.309748 5.404436 4.434243 5.463881 5.509687 5.504178 5.729384 5.208887
 [41] 5.225900 5.896902 5.426478 5.581274 5.849967 5.806604 5.752419 5.815334
 [49] 7.179772 3.913531 4.769848 6.499783 7.201789 3.045399 2.818975 3.585692
 [57] 4.079736 4.591600 5.805138 4.808667 4.891159 4.410427 4.624717 5.358758
 [65] 4.445556 4.573303 4.639545 4.738702 4.758811 4.866848 4.646820 5.023409
 [73] 5.554352 5.649389 5.212269 5.239913 5.197247 5.225199 5.440054 5.583290
 [81] 4.227466 5.147534 4.348386 4.851520 5.438399 4.519382 4.527825 4.667593
 [89] 4.704452 4.561311 4.601683 4.811872 4.831482 4.859074 4.942993 5.548694
 [97] 5.596376 5.268160 5.750434 5.412915 5.793715 5.564473 4.673357 4.837849
[105] 5.253504 5.292955 6.118034 5.854753 5.455541 5.724412 5.967190 5.961206
[113] 6.467741 5.861670 3.524487 3.707389 5.166544 5.739166 5.754792 4.687172
[121] 5.320737 5.544181 5.906076 5.149766 5.179001 5.781746 5.598589 5.365700
[129] 5.443331 3.838176 4.284184 5.346309 4.314462 5.271321 3.247318 3.447757
[137] 3.462880 6.135929 5.146842 5.494053 6.040775 5.491808 5.500108 4.871648
[145] 4.883670 4.842304 4.954753 5.188784 5.192162 5.223832 5.239730 6.486323
[153] 4.528981 5.235028 4.287634 4.352214 4.358823 4.376095 4.435957 4.478382
```

Website

Website