Writing functions in R

a **statsTeachR** resource

Learning goals

At the end of this lecture you should be able to...

- Understand the elements that make up a function in R.
- Write a simple function in R.

What is a function?

A function is a pre-defined algorithm

- It takes arguments as inputs.
- It returns a defined output.

```r
my_function <- function(arg1, arg2) {
    ## this is the body of the function
    ...
    return(something)
}
```

What does this function calculate?

```
my_fun <- function(x) {
    ## x is a numeric vector
    y <- sum(x)/length(x)
    return(y)
}
```

What does this function calculate?

```r
my_fun <- function(x) {
    ## x is a numeric vector
    y <- sum(x)/length(x)
    return(y)
}
a <- rnorm(100)
my_fun(a)

## [1] 0.1300216
```

You can control what people put in

```r
b <- c("fun", "with", "functions")
my_fun(b)

## Error in sum(x):  invalid 'type' (character) of argument
```

```r
my_fun <- function(x) {
    if(class(x)!="numeric")
        stop("x must be numeric")
    y <- sum(x)/length(x)
    return(y)
}
my_fun(b)

## Error in my_fun(b):  x must be numeric

my_fun(a)

## [1] 0.1300216
```

You can communicate with the user

Often these are couched in if-statements, i.e. “if some unusual condition is met, here is something you should know”.

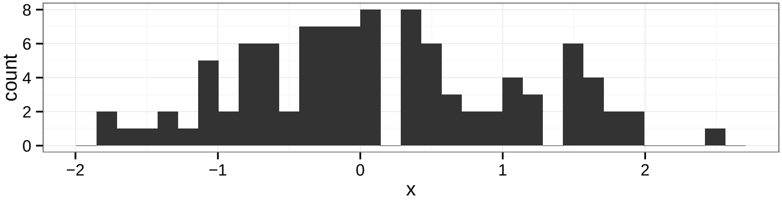
```
error_msg_fun <- function(x) {
    message("Lift off!")
    warning("Houston, we have a minor glitch. No biggie.")
    stop("Houston, we have a major problem. ABORT!")
}
error_msg_fun(a)

## Lift off!
## Warning in error_msg_fun(a):  Houston, we have a minor glitch.
No biggie.
## Error in error_msg_fun(a):  Houston, we have a major problem.
ABORT!
```

Other function features...

- You can add '...' to the argument list for your function, to enable the user to pass unspecified arguments to function calls within the function.
- `require()` ensures that the necessary packages are loaded. You should only use `require()` when defining functions, otherwise, use `library()`.

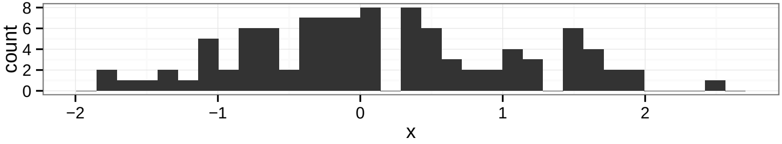
Other function features...

```
my_fun <- function(x, ...) {
    require(ggplot2)
    if(class(x)!="numeric")
        stop("x must be numeric")
    p <- qplot(x, ...)
    print(p)
    y <- sum(x)/length(x)
    return(y)
}
my_fun(a)

## [1] 0.1300216
```

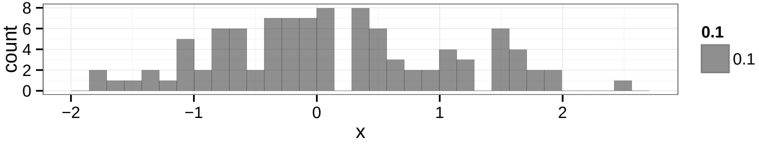


Passing an argument

```
my_fun(a)

## [1] 0.1300216
```



```
my_fun(a, alpha=0.1)

## [1] 0.1300216
```



Lexical scoping

Scoping is largely beyond the scope of this course, but a few important things:

- Scoping rules determine how “free variables” are assigned values.
- Within functions, the safest/simplest thing is to make sure that everything is defined explicitly within the function.
- R uses “Lexical scoping” which means it looks up undefined variables in the environment where your function was defined!

More detail can be found in Hadley’s book.