Defining functions

A function definition looks like

```r
median <- function(x, na.rm = FALSE)
{
  ... lots of code...
  ## a return value
  sort(x, partial = half)[half]
}
```

This function has two arguments, `x` and `na.rm`. The second argument has a default value of `FALSE`, so it is optional. The first argument is required.

The last line of the function computes a return value, which is not assigned to anything.
Suppose the function is called as

\[
\text{median}(height \times width)
\]

and \text{height} and \text{width} are \(c(1, 2, 3)\) and 10 respectively.

- The function argument \text{x} is now the value of \(height \times width\), \(c(10, 20, 30)\). It still has this value even if there are new variables \text{height} and \text{width} defined inside the function.

- The argument \text{na.rm} has its default value \text{FALSE}, since its value wasn’t specified.

Example

We could complete the \text{median} function as

```r
median <- function (x, na.rm = FALSE){
  if (na.rm) {remove missing values}
    x <- x[!is.na(x)]
  else if (any(is.na(x)))
    return(NA)
  n <- length(x)
  half <- (n + 1)/2
  if (n%%2 == 1) { ## odd n
    sort(x)[half]
  } else{ ## even n
    sum(sort(x)[c(half, half+ 1)])/2
  }
}
```

It sorts \text{x} and then returns the middle observation or the average of the middle two.
There may be variables with the same name (eg x, n) inside and outside a function. Rules for working out which one to use are called scoping rules. R’s are:

- First look for a variable inside the function.
- Then look for a variable inside the function that the function was defined in (if any), and so on up.
- Finally look in the global environment, the variables visible at the command line.

This is different from S, where the second step doesn’t happen, but the difference only matters in some specialized cases. The third step is usually accidental for looking up variables (it’s important for looking up functions).

**Scope example**

The value of this function is itself a function

```r
power <- function(lambda){
  function(x) {x^lambda}
}
```

```r
square <- power(2)
cube <- power(3)

> square(1:2)
[1] 1 4

> cube(1:2)
[1] 1 8
```

Inside the `square` function, what is `lambda`? It isn’t a local variable, so R looks at the function where `square` was defined. Here `lambda` exists. Its value was 2.
Unevaluated arguments

Earlier we said that in

\[ \text{median}(\text{height} \times \text{width}) \]

the function argument \( x \) just stored the value of \( \text{height} \times \text{width} \). This isn’t quite true. Until you look at \( x \) it stores the whole expression \( \text{height} \times \text{width} \). Graphics commands use this to get plot labels, since the \text{substitute} function lets you copy the expression without looking at it.

```r
> label <- function(x) {list(value=x, actual=substitute(x))}
> label(1+1)
$\text{value}
[1] 2
$\text{actual}
1 + 1
```

This \text{lazy evaluation} is also used in handling model formulas.