Functions

STAT 133

Gaston Sanchez

Department of Statistics, UC–Berkeley

gastonsanchez.com
github.com/gastonstat/stat133
Course web: gastonsanchez.com/teaching/stat133

Functions

R comes with many functions and packages that let us perform a wide variety of tasks. But there are occassions in which we need to create our own functions.

Anatomy of a function

`function()` allows us to create a function. It has the following structure:

```
function_name <- function(arg1, arg2, ...)
{
  expressions
}
```

Anatomy of a function

- Generally we will give a name to a function
- A function takes one or more arguments (or none)
- The expressions forming the operations comprise the body of the function
- Braces surround the body of the function
- Functions return a single *value*

Function example

A function that squares its argument:

```
square <- function(x) {
  x * x
}
```

It works like any other function in R:

```
square(10)

## [1] 100
```

In this case, `square()` is also vectorized

```
square(1:5)

## [1]  1  4  9 16 25
```

Function example

Functions with a body consisting of a simple expression can be written with no braces (in one single line!):

```
square <- function(x) x * x

square(10)

## [1] 100
```

Function example

Once defined, functions can be used in other function definitions:

```
sum_square <- function(x) sum(square(x))

sum_square(1:5)

## [1] 55
```

A simple example

A function which, given the value r computes the value πr^2

```
area <- function(r) pi * r^2
```

- The formal argument of the function is `r`
- The body of the function consists of the simple expression `pi * r^2`
- The function has been assigned the name `"area"`

```
area(5)

## [1] 78.53982
```

Evaluation of Functions

Function evaluation involves:

- A set of variables associated to the arguments is temporarily created
- The variable definitions are used to evaluate the body function
- Temporary variables are removed at the end
- The computed values are returned

Evaluation Example

Evaluating the function call `area(5)` takes place as follows:

- Temporarily create a variable `r` with value `5`
- Use that value `5` to compute `pi * 5^2`
- Remove the temporary variable definition
- Return the value `78.53982`

Function example

```r
hello <- function(x) {
  paste("Hello", x)
}

hello('Gaston')

## [1] "Hello Gaston"
```

Another function example

```
add <- function(x, y) {
  x + y
}

add(2, 3)

## [1] 5
```

Function with no arguments

Functions can have no arguments

```
hi <- function() {
  print("Hi there!")
}

hi()

## [1] "Hi there!"
```

Missing arguments

If you specify an argument with no default value, you must give it a value everytime you call the function, otherwise you'll get an error:

```
sqr <- function(x) {
  x^2
}

sqr()

## Error in sqr():  argument "x" is missing, with no
default
```

Default arguments

You can give default values to function arguments:

```
hey <- function(x = "") {
  cat("Hey", x, "\nHow is it going?")
}

hey()

## Hey
## How is it going?

hey("Gaston")

## Hey Gaston
## How is it going?
```

The `return()` command

Sometimes the `return()` command is included to explicitly indicate the output of a function:

```
add <- function(x, y) {
  z <- x + y
  return(z)
}

add(2, 3)

## [1] 5
```

The `return()` command

If no `return()` is present, then R returns the last evaluated expression:

```r
# output with return()
add <- function(x, y) {
  z <- x + y
  return(z)
}

add(2, 3)

## [1] 5
```

```r
# output without return()
add <- function(x, y) {
  x + y
}

add(2, 3)

## [1] 5
```

The `return()` command

If no `return()` is present, then R returns the last evaluated expression, although it might not always print the output:

```
# nothing is printed
add <- function(x, y) {
  z <- x + y
}

add(2, 3)
```

```
# output printed
add <- function(x, y) {
  z <- x + y
  return(z)
}

add(2, 3)

## [1] 5
```

The `return()` command

The last evaluated expression has the same value in both cases:

```
# nothing is printed
add <- function(x, y) {
  z <- x + y
}

a1 <- add(2, 3)
a1

## [1] 5
```

```
# output printed
add <- function(x, y) {
  z <- x + y
  return(z)
}

a2 <- add(2, 3)
a2

## [1] 5
```

The `return()` command

If no `return()` is present, then R returns the last evaluated expression:

```r
add1 <- function(x, y) {
  x + y
}

add2 <- function(x, y) {
  z <- x + y
  z
}
```

```r
add3 <- function(x, y) {
  z <- x + y
}

add4 <- function(x, y) {
  z <- x + y
  return(z)
}
```

The `return()` command

`return()` can be useful when the output may be obtained in the middle of the function's body

```
f <- function(x, y, add = TRUE) {
  if (add) {
    return(x + y)
  } else {
    return(x - y)
  }
}
```

```
f(2, 3, add = TRUE)

## [1] 5

f(2, 3, add = FALSE)

## [1] -1
```

Exercise

Write a function that checks if a number is positive (output `TRUE`) or negative (output `FALSE`)

Exercise

Write a function that checks if a number is positive (output `TRUE`) or negative (output `FALSE`)

```
is_positive <- function(x) {
  if (x > 0) TRUE else FALSE
}

is_positive(2)

## [1] TRUE

is_positive(-1)

## [1] FALSE
```

Exercise

What happens in these cases?

```
is_positive <- function(x) {
  if (x > 0) TRUE else FALSE
}

is_positive(0)
is_positive(NA)
is_positive(TRUE)
is_positive("positive")
is_positive(1:5)
```

Writing Functions

Writing Functions

Writing Functions

- Choose meaningful names of functions
- Preferably a verb
- Choose meaningful names of arguments
- Think about the users (who will use the function)
- Think about extreme cases
- If a function is too long, maybe you need to split it

Names of functions

Avoid this:

```
f <- function(x, y) {
  x + y
}
```

This is better

```
add <- function(x, y) {
  x + y
}
```

Describing functions

Also add a short description of what the arguments should be like. In this case, the description is outside the function

```
# function for adding two numbers
# x: number
# y: number
add <- function(x, y) {
  x + y
}
```

Describing functions

In this case, the description is inside the function

```
add <- function(x, y) {
  # function for adding two numbers
  # x: number
  # y: number
  x + y
}
```

Exercise

R has a function `summary()` that when applied on a numeric vector provides something like this:

```
summary(1:10)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

Create a `describe()` function that takes a numeric vector and returns: minimum, maximum, mean, and standard deviation

Exercise

First attempt

```
describe <- function(x) {
  x_min <- min(x)
  x_max <- max(x)
  x_mean <- mean(x)
  x_sd <- sd(x)
  return(c(x_min, x_max, x_mean, x_sd))
}

describe(1:10)

## [1]  1.00000 10.00000  5.50000  3.02765
```

Exercise

Second attempt (adding names)

```
describe <- function(x) {
  x_min <- min(x)
  x_max <- max(x)
  x_mean <- mean(x)
  x_sd <- sd(x)
  values <- c(x_min, x_max, x_mean, x_sd)
  names(values) <- c("min", "max", "mean", "sd")
  return(values)
}

describe(1:10)
```

```
##       min       max      mean        sd
##   1.00000 10.00000   5.50000   3.02765
```

Exercise

Third attempt (using a list as output)

```r
describe <- function(x) {
  list(
    min = min(x),
    max = max(x),
    mean = mean(x),
    sd = sd(x)
  )
}
```

```r
describe(1:10)
```

```
## $min
## [1] 1
##
## $max
## [1] 10
##
## $mean
## [1] 5.5
##
## $sd
## [1] 3.02765
```

Handling Function Arguments

Exercise

Assertions

```
describe <- function(x) {
  if (!is.numeric(x)) {
    stop("x is not numeric")
  }
  # output
  c(mean = mean(x), sd = sd(x))
}
```

Exercise

Passing arguments

```
describe <- function(x, na.rm = FALSE) {
  if (!is.numeric(x)) {
    stop("x is not numeric")
  }
  # output
  c(mean = mean(x, na.rm = na.rm),
    sd = sd(x, na.rm = na.rm))
}

describe(c(1:5, NA), na.rm = TRUE)

##     mean       sd
## 3.000000 1.581139
```

Exercise

Probability Density of the Normal Distribution:

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Write a function that takes a value x (with parameters μ and σ) which computes the probability density distribution of the normal distribution

Exercise

Normal Distribution:

```
normal_dist <- function(x, mu = 0, sigma = 1) {
  constant <- 1 / (sigma * sqrt(2*pi))
  constant * exp(-((x - mu)^2) / (2 * sigma^2))
}

normal_dist(2)

## [1] 0.05399097
```

Argument Matching

- Arguments can be named or not
- Named arguments are used in preference to position which
- It is important to be clear about which argument corresponds to which formal parameter of the function

Argument Matching

```r
normal_dist <- function(x, mu = 0, sigma = 1) {
  constant <- 1 / (sigma * sqrt(2*pi))
  constant * exp(-((x - mu)^2) / (2 * sigma^2))
}

normal_dist(2)
normal_dist(2, sigma = 3, mu = 1)
normal_dist(mu = 1, sigma = 3, 2)
normal_dist(mu = 1, 2, sigma = 3)
```

Argument Matching

R is "smart" enough in doing pattern matching with arguments' names (not recommended though)

```
normal_dist(2)

## [1] 0.05399097

normal_dist(2, m = 0, s = 1)

## [1] 0.05399097

normal_dist(2, sig = 1, m = 0)

## [1] 0.05399097
```