There are lots of R packages. We’ll limit ourselves to those packages which ship with R, and which load on startup. Which ones are they?

**What packages load on starting R?**  
Start a new R session and type search(). Here’s the result on my machine:

search()  
[1] ".GlobalEnv" "tools:rstudio" "package:stats" "package:graphics" "package:grDevices"  
"package:utils" "package:datasets" "package:methods" "Autoloads" "package:base"

We’re interested in the packages with *priority = base*. Next question:

**How can I see and filter for package priority?**  
You don’t need dplyr for this, but it helps.

library(tidyverse)

installed.packages() %>%

as.tibble() %>%

filter(Priority == "base") %>%

select(Package, Priority)

# A tibble: 14 x 2

Package Priority

1 base base

2 compiler base

3 datasets base

4 graphics base

5 grDevices base

6 grid base

7 methods base

8 parallel base

9 splines base

10 stats base

11 stats4 base

12 tcltk base

13 tools base

14 utils base

Comparing to the output from search(), we want to look at: *stats, graphics, grDevices, utils, datasets, methods* and *base*.

**How can I see all the objects in a package?**  
Like this, for the *base* package. For other packages, just change *base* to the package name of interest.

ls("package:base")

However, not every object in a package is a function. Next question:

**How do I know if an object is a function?**  
The simplest way is to use is.function().

is.function(ls)

[1] TRUE

What if the function name is stored as a character variable, “ls”? Then we can use get():

is.function(get("ls"))

[1] TRUE

But wait: what if two functions from different packages have the same name and we have loaded both of those packages? Then we specify the package too, using the *pos* argument.

is.function(get("Position", pos = "package:base"))

[1] TRUE

is.function(get("Position", pos = "package:ggplot2"))

[1] FALSE

So far, so good. Now, to the arguments.

**How do I see the arguments to a function?**  
Now things start to get interesting. In R, function arguments are called *formals*. There is a function of the same name, formals(), to show the arguments for a function. You can also use formalArgs() which returns a vector with just the argument names:

formalArgs(ls)

[1] "name" "pos" "envir" "all.names" "pattern" "sorted"

But that won’t work for every function. Let’s try abs():

formalArgs(abs)

NULL

The issue here is that abs() is a *primitive* function, and primitives don’t have formals. Our next two questions:

**How do I know if an object is a primitive?**  
Hopefully you guessed that one:

is.primitive(abs)

[1] TRUE

**How do I see the arguments to a primitive?**  
You can use args(), and you can pass the output of args() to formals() or formalArgs():

args(abs)

function (x)

NULL

formalArgs(args(abs))

[1] "x"

However, there are a few objects which are primitive functions for which this doesn’t work. Let’s not worry about those.

is.primitive(`:`)

[1] TRUE

formalArgs(args(`:`))

NULL

Warning message:

In formals(fun) : argument is not a function

**So what was the original question again?**  
Let’s put all that together. We want to find the base packages which load on startup, list their objects, identify which are functions or primitive functions, list their arguments and count them up.

We’ll create a tibble by pasting the arguments for each function into a comma-separated string, then pulling the string apart using unnest\_tokens() from the *tidytext* package.

library(tidytext)

library(tidyverse)

pkgs <- installed.packages() %>%

as.tibble() %>%

filter(Priority == "base",

Package %in% c("stats", "graphics", "grDevices", "utils",

"datasets", "methods", "base")) %>%

select(Package) %>%

rowwise() %>%

mutate(fnames = paste(ls(paste0("package:", Package)), collapse = ",")) %>%

unnest\_tokens(fname, fnames, token = stringr::str\_split,

pattern = ",", to\_lower = FALSE) %>%

filter(is.function(get(fname, pos = paste0("package:", Package)))) %>%

mutate(is\_primitive = ifelse(is.primitive(get(fname, pos = paste0("package:", Package))),

1,

0),

num\_args = ifelse(is.primitive(get(fname, pos = paste0("package:", Package))),

length(formalArgs(args(fname))),

length(formalArgs(fname)))) %>%

ungroup()

That throws out a few warnings where, as noted, args() doesn’t work for some primitives.

And the winner is –

pkgs %>%

top\_n(10) %>%

arrange(desc(num\_args))

Selecting by num\_args

# A tibble: 10 x 4

Package fname is\_primitive num\_args

1 graphics legend 0 39

2 graphics stars 0 33

3 graphics barplot.default 0 30

4 stats termplot 0 28

5 utils read.table 0 25

6 stats heatmap 0 24

7 base scan 0 22

8 graphics filled.contour 0 21

9 graphics hist.default 0 21

10 stats interaction.plot 0 21

– the function legend() from the *graphics* package, with 39 arguments. From the base package itself, scan(), with 22 arguments.

Just to wrap up, some histograms of argument number by package, suggesting that the base *graphics* functions tend to be the more verbose.

pkgs %>%

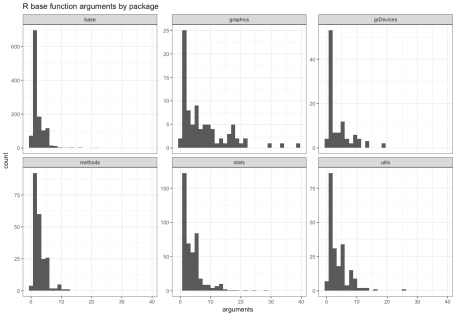
ggplot(aes(num\_args)) +

geom\_histogram() +

facet\_wrap(~Package, scales = "free\_y") +

theme\_bw() +

labs(x = "arguments", title = "R base function arguments by package")

[](https://nsaunders.files.wordpress.com/2018/06/rfunctions.png)