Seeking Help

- To be able to read R help files for functions and special operators.
- To be able to use CRAN task views to identify packages to solve a problem.
- To be able to seek help from your peers.

- How can I get help in R?

Reading Help Files

R, and every package, provide help files for functions. The general syntax to search for help on any function, "function_name", from a specific function that is in a package loaded into your namespace (your interactive R session) is:

```
?function_name
help(function_name)
```

For example take a look at the help file for `write.table()`, we will be using a similar function in an upcoming episode.

```
?write.table()
```

This will load up a help page in RStudio (or as plain text in R itself).

Each help page is broken down into sections:

- Description: An extended description of what the function does.
- Usage: The arguments of the function and their default values (which can be changed).
- Arguments: An explanation of the data each argument is expecting.
- Details: Any important details to be aware of.
- Value: The data the function returns.
- See Also: Any related functions you might find useful.
- Examples: Some examples for how to use the function.

Different functions might have different sections, but these are the main ones you should be aware of.

Notice how related functions might call for the same help file:

```
?write.table()
?write.csv()
```

This is because these functions have very similar applicability and often share the same arguments as inputs to the function, so package authors often choose to document them together in a single help file.

Tip: Running Examples

From within the function help page, you can highlight code in the Examples and hit `Ctrl`+`Return` to run it in RStudio console. This gives you a quick way to get a feel for how a function works.

Tip: Reading Help Files

One of the most daunting aspects of R is the large number of functions available. It would be prohibitive, if not impossible to remember the correct usage for every function you use. Luckily, using the help files means you don't have to remember that!

Special Operators

To seek help on special operators, use quotes or backticks:

```
?"<-"
?`<-`
```

Getting Help with Packages

Many packages come with "vignettes": tutorials and extended example documentation. Without any arguments, `vignette()` will list all vignettes for all installed packages; `vignette(package="package-name")` will list all available vignettes for `package-name`, and `vignette("vignette-name")` will open the specified vignette.

If a package doesn't have any vignettes, you can usually find help by typing `help("package-name")`.

RStudio also has a set of excellent cheatsheets for many packages.

When You Remember Part of the Function Name

If you're not sure what package a function is in or how it's specifically spelled, you can do a fuzzy search:

```
??function_name
```

A fuzzy search is when you search for an approximate string match. For example, you may remember that the function to set your working directory includes "set" in its name. You can do a fuzzy search to help you identify the function:

```
??set
```

When You Have No Idea Where to Begin

If you don't know what function or package you need to use CRAN Task Views is a specially maintained list of packages grouped into fields. This can be a good starting point.

When Your Code Doesn't Work: Seeking Help from Your Peers

If you're having trouble using a function, 9 times out of 10, the answers you seek have already been answered on Stack Overflow. You can search using the `[r]` tag. Please make sure to see their page on how to ask a good question.

If you can't find the answer, there are a few useful functions to help you ask your peers:

```
?dput
```

Will dump the data you're working with into a format that can be copied and pasted by others into their own R session.

```
sessionInfo()
```

```
## R version 4.3.0 (2023-04-21)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK versio
n 3.11.0
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] digest_0.6.33   R6_2.5.1        fastmap_1.1.1   xfun_0.42
##  [5] cachem_1.0.8    knitr_1.43      htmltools_0.5.7 rmarkdown_2.23
##  [9] lifecycle_1.0.3 cli_3.6.1       sass_0.4.7      jquerylib_0.1.4
## [13] compiler_4.3.0  rstudioapi_0.14 tools_4.3.0     evaluate_0.21
## [17] bslib_0.6.1     yaml_2.3.7      rlang_1.1.1     jsonlite_1.8.7
```

Will print out your current version of R, as well as any packages you have loaded. This can be useful for others to help reproduce and debug your issue.

Challenge 1

Look at the help page for the `c` function. What kind of vector do you expect will be created if you evaluate the following:

```
c(1, 2, 3)
c('d', 'e', 'f')
c(1, 2, 'f')
```

Solution to Challenge 1

The `c()` function creates a vector, in which all elements are of the same type. In the first case, the elements are numeric, in the second, they are characters, and in the third they are also characters: the numeric values are "coerced" to be characters.

Challenge 2

Look at the help for the `paste` function. You will need to use it later. What's the difference between the `sep` and `collapse` arguments?

Solution to Challenge 2

To look at the help for the `paste()` function, use:

```
help("paste")
?paste
```

The difference between `sep` and `collapse` is a little tricky. The `paste` function accepts any number of arguments, each of which can be a vector of any length. The `sep` argument specifies the string used between concatenated terms — by default, a space. The result is a vector as long as the longest argument supplied to `paste`. In contrast, `collapse` specifies that after concatenation the elements are *collapsed* together using the given separator, the result being a single string.

It is important to call the arguments explicitly by typing out the argument name e.g `sep = ","` so the function understands to use the "," as a separator and not a term to concatenate. e.g.

```
paste(c("a","b"), "c")
```

```
## [1] "a c" "b c"
```

```
paste(c("a","b"), "c", ",")
```

```
## [1] "a c ," "b c ,"
```

```
paste(c("a","b"), "c", sep = ",")
```

```
## [1] "a,c" "b,c"
```

```
paste(c("a","b"), "c", collapse = "|")
```

```
## [1] "a c|b c"
```

```
paste(c("a","b"), "c", sep = ",", collapse = "|")
```

```
## [1] "a,c|b,c"
```

(For more information, scroll to the bottom of the `?paste` help page and look at the examples, or try `example('paste')`.)

Challenge 3

Use help to find a function (and its associated parameters) that you could use to load data from a tabular file in which columns are delimited with "\t" (tab) and the decimal point is a "." (period). This check for decimal separator is important, especially if you are working with international colleagues, because different countries have different conventions for the decimal point (i.e. comma vs period). Hint: use `??"read table"` to look up functions related to reading in tabular data.

Solution to Challenge 3

The standard R function for reading tab-delimited files with a period decimal separator is read.delim(). You can also do this with `read.table(file, sep="\t")` (the period is the *default* decimal separator for `read.table()`), although you may have to change the `comment.char` argument as well if your data file contains hash (#) characters.

Other Resources

- Quick R
- RStudio cheat sheets
- Cookbook for R

- Use `help()` to get online help in R.