Footmarks

[^basic-app-1]: The extra `()` on the outside are important.

`shinyApp()` only creates an app when printed, and `()` forces the printing of the last result in the file being source, which is otherwise returned invisibly.

[^basic-app-2]: Shiny strives to support all modern browsers, and you can see the set currently supported at <https://www.rstudio.com/about/platform-support/>.

Note that Internet Explorer versions prior to IE11 are not compatible when running Shiny directly from your R session.

However, Shiny apps deployed on Shiny Server or ShinyApps.io can work with IE10 (earlier versions of IE are no longer supported).

[^basic-ui-1]: All `passwordInput()` does is hide what the user is typing, so that someone looking over their shoulder can't read it.

It's up to you to make sure that any passwords are not accidentally exposed, so we don't recommend using passwords unless you have had some training in secure programming.

[^basic-ui-2]: Note that the name of that argument is different for inputs (`inputId`) and outputs (`outputId`).

I don't use the name of the first argument because it's so important and I expect you to remember what it does without an additional hint.

[^basic-reactivity-1]: Each connection to a Shiny app starts a new session whether it's connections from different people, or with multiple tabs from the same person.

[^basic-reactivity-2]: The primary exception is where there's some work that can be shared across multiple users.

For example, all users might be looking at the same large csv file, so you might as well load it once and share it between users.

We'll come back to that idea in Section \@ref(schedule-data-munging).

[^basic-reactivity-3]: If you're running the live app, notice that you have to type fairly slowly for the output to update one letter at a time.

That's because Shiny uses a technique called \*\*debouncing\*\*, which means that it waits for a few ms before sending an update.

That considerably reduces the amount of work that Shiny needs to do, without appreciably reducing the response time of the app.

[^basic-reactivity-4]: <https://xkcd.com/149/>

[^basic-reactivity-5]: If you've ever struggled to get a ggplot2 legend to look exactly the way you want, you've encountered this problem!

[^basic-reactivity-6]: Yes, Shiny doesn't update the output if you can't see it in your browser!

Shiny is so lazy that it doesn't do the work unless you can actually see the results.

[^basic-reactivity-7]: The technical term for this ordering is a "topological sort".

[^basic-reactivity-8]: If you haven't heard of a frequency polygon before, it's just a histogram that's drawn with a line, instead of bars, which makes it easier to compare multiple data sets on the same plot.

[^basic-reactivity-9]: If you're familiar with memoisation, this is a similar idea.

[^basic-reactivity-10]: The New York Times used this technique particularly effectively in their article discussing how to interpret the jobs report: <https://www.nytimes.com/2014/05/02/upshot/how-not-to-be-misled-by-the-jobs-report.html>

[^action-workflow-1]: Snippets are text macros that you can use to insert common code fragments.

See <https://support.rstudio.com/hc/en-us/articles/204463668-Code-Snippets> for more details.

If you enjoy using snippets, make sure to check the collection of Shiny specific snippets put together by ThinkR: <https://github.com/ThinkR-open/shinysnippets>.

[^action-workflow-2]: A project is a self-contained directory that is isolated from the other projects that you're working on.

If you use RStudio, but don't currently use projects, I highly recommend reading about the [project oriented lifestyle](<https://whattheyforgot.org/project-oriented-workflow.html>).

[^action-workflow-3]: I'm using `subset()` so that my app doesn't require any other packages.

In a bigger app, I'd probably prefer `dplyr::filter()` just because I'm a little more familiar with its behaviour.

[^action-workflow-4]: Regardless of how you normally load packages, I strongly recommend using multiple `library()` calls.

This eliminates a source of potential confusion for people who might not be familiar with the tool that you're using.

[^action-workflow-5]: For example, I had no idea that `is.POSIXt()` was part of the lubridate package!

[^action-layout-1]: Fonts are a little trickier than colours because you have to make sure the app viewer has the font, not just you.

Make sure to read the `bs\_theme()` docs for all the details.

[^action-layout-2]: The magic that connects inputs and outputs to R happens elsewhere (via javascript) but that's well beyond the scope of this book.

[^action-layout-3]: Introduced in R 4.0.0.

[^action-graphics-1]: When I wrote this chapter, Shiny didn't support touch events, which means that plot interactivity won't work on mobile devices.

Hopefully it will support these events by the time you read this.

[^action-graphics-2]: Note that it's not called `nearestPoints()`; it won't return any thing if you don't click near an existing data point.

[^action-graphics-3]: As a general rule, adding explanatory text suggests that your interface is too complex, so is best avoided, where possible.

This is the key idea behind "affordances", the idea that an object should suggest naturally how to interact with it as introduced by Don Norman in the \*"Design of Everyday Things"\*.

[^action-graphics-4]: Unfortunately there's no easy way to keep them exactly the same because it's currently not possible to find out the size of the fixed elements around the borders of the plot.

[^action-feedback-1]: "Condition" is a technical term that includes errors, warnings, and messages.

If you're interested, you can learn more of the details of R's condition system in <https://adv-r.hadley.nz/conditions.html>.

[^action-feedback-2]: More precisely, `req()` proceeds only if its inputs are \*\*truthy\*\*, i.e. any value apart from `FALSE`, `NULL` , `""`, or a handful of other special cases described in `?isTruthy`.

[^action-feedback-3]: If reading csv files is a bottleneck in your application should consider using `data.table::fread()` and `vroom::vroom()` instead; they can be orders of magnitude faster than `read.csv()`.

[^action-feedback-4]: If your code doesn't involve a for loop or a apply/map function, it's going to be very difficult to make a progress bar.

[^action-transfer-1]: MIME type is short for "\*\*m\*\*ulti-purpose \*\*i\*\*nternet \*\*m\*\*ail \*\*e\*\*xtensions type".

As you might guess from the name, it was originally designed for email systems, but now it's used widely across many internet tools.

A MIME type looks like `type/subtype`.

Some common examples are `text/csv`, `text/html`, `image/png`, `application/pdf`, `application/vnd.ms-excel` (excel file).

[^action-dynamic-1]: I introduced `observeEvent()` in Section \@ref(observers) and will discuss in more detail in Section \@ref(observers-details).

[^action-dynamic-2]: The first argument, `session`, exists for backward compatibility but is very rarely needed.

[^action-dynamic-3]: To be more precise, any attempt to read a frozen input will result in `req(FALSE).`

[^action-dynamic-4]: This is generally only a concern when you are changing the `value`, but be some other parameters can change the value indirectly.

For example, if you modify the `choices` for `selectInput()`, or `min` and `max` for `sliderInput()`, the current `value` will be modified if it's no longer in the allowed set of values.

[^action-tidy-1]: `dplyr::filter()` is inspired by `base::subset()`.

`subset()` uses data-masking, but not through tidy evaluation, so unfortunately the techniques discussed in this chapter don't apply to it.

[^action-tidy-2]: In Shiny apps, the most common form of indirection is having the name of data-variable stored in a reactive value.

There's another form of indirection that happens when you're writing functions which is solved using `{{ x }}`, called embracing.

You can learn more about that in [\*Programming with dplyr\*](<http://dplyr.tidyverse.org/dev/articles/programming.html>).

[^action-tidy-3]: You might wonder if the same problem applies to variables called `.data` and `.env`.

In the unlikely event of having columns with those names you'll need to refer to them with explicitly `.data$.data` and `.data$.env`.

[^action-tidy-4]: In older versions of tidyselect and dplyr, you'll need to use `one\_of()`.

It has the same semantics as `any\_of()`, but a less informative name.

[^reactivity-motivation-1]: R uses "lexical scoping" for looking up the values associated with variable names.

You can learn more about it in <https://adv-r.hadley.nz/functions.html#lexical-scoping>.

[^reactivity-motivation-2]: `<<-` is called the super-assignment operator, and here it modifies `temp\_f` in the global environment, rather than creating a new `temp\_f` variable inside the function as `<-` would.

You can learn more about `<<-` in <https://adv-r.hadley.nz/environments.html#super-assignment-->.

[^reactivity-motivation-3]: If you happen to have ever used R's active bindings, you might notice that the syntax is very similar.

This is not a coincidence.

[^reactivity-motivation-4]: You can tell it doesn't re-compute because "Converting" is not printed.

[^reactivity-graph-1]: Anywhere you see output, you can also think observer.

The primary difference is that certain outputs that aren't visible will never be computed.

We'll discuss the details in Section \@ref(observers-details).

[^reactivity-graph-2]: If you have observers whose side effects must happen in a certain order, you're generally better off re-designing your system.

Failing that, you can control the relative order of observers with the `priority` argument to `observe()`.

[^reactivity-foundations-1]: See more details at <https://adv-r.hadley.nz/names-values.html#copy-on-modify>

[^reactivity-foundations-2]: By default, you'll see the whole error message.

You can show a generic error message by turning error sanitising on.

See <https://shiny.rstudio.com/articles/sanitize-errors.html> for details.

[^reactivity-foundations-3]: Technically, a custom condition.

See <https://adv-r.hadley.nz/conditions.html#custom-conditions> for more details on the underlying theory.

[^reactivity-foundations-4]: In rare cases, you may prefer to process even outputs that are hidden.

You can use the `outputOptions()` function's `suspendWhenHidden` to opt out of the automatic suspension feature on an output-by-output basis.

[^reactivity-foundations-5]: Assuming that it's used by some output or observer; otherwise it will stay in its initial invalidated state forever.

[^reactivity-escaping-1]: As a debugging aid, the reactlog package can capture and draw these connection when you modify reactive values from an observer, but this information is not used by Shiny.

[^reactivity-escaping-2]: This is rather similar to a notification, as seen in Section \@ref(notifications).

[^scaling-functions-1]: If you're not, and you'd like to learn the basics, you might try reading the [Functions chapter](https://r4ds.had.co.nz/functions.html) of \*R for Data Science\*.

[^scaling-modules-1]: Unlike an app, both module UI and server are functions.

[^scaling-modules-2]: The tidyverse style guide, <https://style.tidyverse.org/functions.html#return>, recommends reserving `return()` only for cases where you are returning early.

[^scaling-modules-3]: Not every page will have both buttons (more on that shortly) so I mark them as optional by supplying a default value of `NULL`.

[^scaling-packages-1]: I'd expect most other ways of deploying Shiny apps would also work since `app.R` is the most common way of structuring apps.

[^scaling-packages-2]: The distinction between Imports and Suggests is not generally important for app packages.

If you do want to make a distinction, the most useful is to use Imports for packages that need to be present on the deployment machine (in order for the app to work) and Suggests for packages that need to be present on the development machine (in order to develop the app).

[^scaling-testing-1]: If you don't use RStudio, you'll need to give `use\_test()` the name of the file, like `usethis::use\_test("load")`.

[^scaling-testing-2]: Like `usethis::use\_test()` this only works if you're using RStudio.

[^scaling-testing-3]: Snapshot tests require the third edition of testthat.

New package will automatically use the testthat 3e, but you'll need to manually update older packages.

See [testthat 3e](https://testthat.r-lib.org/articles/third-edition.html) for more details.

[^scaling-security-1]: If you can't assume that, you have bigger problems!

That said, some companies do have a "zero-trust" model, so you should double check with your IT team.

[^scaling-security-2]: If your app does work these types of data, it's imperative that you partner with a software engineer with security expertise.

[^scaling-security-3]: The only exception is if they don't involve user-supplied data in any way.

[^scaling-security-4]: [\<https://xkcd.com/327/\>](<https://xkcd.com/327/){.uri}>

[^scaling-performance-1]: Thanks to Sean Lopp for this analogy from his rstudio::conf(2018) talk [Scaling Shiny to 10,000 users](https://rstudio.com/resources/rstudioconf-2018/scaling-shiny/){.uri}.

I highly recommend watching it if you have any doubt that Shiny apps can handle thousands of users.

[^scaling-performance-2]: Again, this depends on exactly how your app is deployed, but typically you can dynamically control the number of processes based on the number of users.

See <https://shiny.rstudio.com/articles/scaling-and-tuning-ssp-rsc.html> for advice on RStudio's deployment offerings.

[^scaling-performance-3]: Or vertical scaling

[^scaling-performance-4]: Or horizontal scaling

[^scaling-performance-5]: The easiest way to do this with RStudio is to open another RStudio instance.

Alternatively, open a terminal and type `R`.

[^scaling-performance-6]: This function was introduced in Shiny 1.6.0, generalising the older `renderCachedPlot()` which only worked for plots.

[^scaling-performance-7]: See <https://ursalabs.org/blog/2020-feather-v2/> for some benchmarks.