

ETC4500/ETC5450

Advanced R programming

Week 6: Literate programming with
Quarto



Outline

- 1 Literate programming
- 2 roxygen2
- 3 Rmarkdown
- 4 Quarto

Outline

1 Literate programming

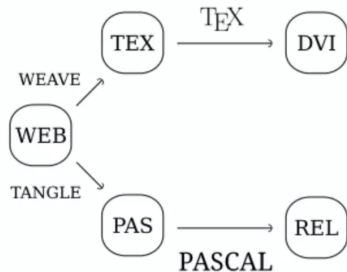
2 roxygen2

3 Rmarkdown

4 Quarto

Literate programming

- Due to Donald Knuth (Stanford), 1984
- A script or document that contains an explanation of the program logic in a natural language (e.g. English), interspersed with snippets of source code, which can be compiled and rerun.
- Generates two representations from a source file: formatted documentation and “tangled” code.



Literate programming

- As a programming approach, it never quite caught on.
- But it has become the standard approach for reproducible documents.

Literate programming examples

- WEB (combining Pascal and TeX)
- roxygen2 comments
 - technically documentation generation rather than literate programming
 - documentation embedded in code, rather than code embedded in documentation
- Sweave documents
- Jupyter notebooks
- Rmarkdown documents
- Quarto documents

Outline

1 Literate programming

2 roxygen2

3 Rmarkdown

4 Quarto

- roxygen2 documentation are just comments to R.
- roxygen2::roxygenize():
 - ▶ generates documentation from these comments in the form of Rd files
 - ▶ adds relevant lines to the NAMESPACE file.
- roxygen2::roxygenize() is called by devtools::document().
- Advantage: keeps documentation with the code. More readable, less chance for errors.

Outline

1 Literate programming

2 roxygen2

3 Rmarkdown

4 Quarto

Markdown syntax

Markdown: a “markup” language for formatting text.

- Headings:

 - # Heading 1

 - ## Heading 2

- **Bold:** *****bold*****.

- *Italic:* ****italic****.

- Blockquotes:

 - > blockquote.

Markdown and Rmarkdown

- Markdown (markup language):
 - ▶ Extension either `.md` or `.markdown`.
 - ▶ Used in many places on the web, in note-taking apps, etc.
- Rmarkdown (markup language):
 - ▶ an extension of markdown that allows for embedded R code chunks.
 - ▶ Extension `.Rmd`.
- Rmarkdown (package):
 - ▶ an R package that allows for the conversion of `.Rmd` files to other formats.

Rmarkdown files

- Structure:

- 1 YAML header

- 2 Markdown content

- 3 R code chunks surrounded by ``{r}`` and ```

- 4 Inline R surrounded by ``r`` and ```

- Rmarkdown documents can be compiled to HTML, PDF, Word, and other formats

- Compile with `rmarkdown::render("file.Rmd")`

Rmarkdown, knitr and pandoc

- `rmarkdown::render()`
 - ▶ Uses `knitr` to run all code chunks, and “knit” the results into a markdown file (replacing R chunks with output).
 - ▶ Uses `pandoc` to convert the markdown file to the desired output format.
 - ▶ If PDF output is desired, LaTeX then converts the tex file (from pandoc output) to pdf.



knitr functions

- `knitr::knit()`: knits a single Rmd file — runs all code chunks and replaces them with output in a markdown file.
- `knitr::purl()`: extracts all R code from an Rmd file and saves it to a new file.
- `knitr::spin()`: knits a specially formatted R script file into an Rmd file.

Rmarkdown packages

- rmarkdown (to html, pdf, docx, odt, rtf, md, etc.)
- bookdown (to html, pdf, epub)
- blogdown (to html) – uses hugo rather than pandoc
- xaringan (to html) – uses remark.js rather than pandoc
- beamer (to pdf)
- rticles (to pdf)
- tufte (to html, pdf)
- vitae (to pdf)
- distill (to html)
- flexdashboard (to html)

Outline

1 Literate programming

2 roxygen2

3 Rmarkdown

4 Quarto

- Generalization of Rmarkdown (not dependent on R)
- Supports R, Python, Javascript and Julia chunks by using either knitr, jupyter or ObservableJS engines.
- More consistent yaml header and chunk options.
- Many more output formats, and many more options for customizing format.
- Heavier reliance on pandoc Lua filters
- Uses pandoc templates for extensions



Choose your engine

Specify the engine in the yaml header:

```
---  
engine: knitr  
---
```

```
---  
engine: jupyter  
jupyter: python3  
---
```

Default: If any `{r}` blocks found, use `knitr` engine; otherwise use `jupyter` (with kernel determined by first block).

Code chunks

Chunk options use the hash-pipe #|

```
```{r}  
#| label: fig-chunklabel
#| fig-caption: My figure
#| fig-width: 6
#| fig-height: 4
mtcars |>
 ggplot(aes(x = mpg, y = wt)) +
 geom_point()
```
```

Reference the figure using @fig-chunklabel.

Chunk options

- Quarto consistently uses hyphenated options (`fig-width` rather than `fig.width`)
- The Rmarkdown `knitr` options are recognized for backwards compatibility.
- Options that are R expressions need to be prefaced by `!expr`

```
```{r}  
#| fig-cap: !expr paste("My figure", 1+1)
```
```

Execute options

- execute option in yaml header can be used instead of a setup chunk:

```
execute:  
  cache: true  
  echo: false  
  warning: false
```

Some chunk options

- `label`: name of chunk. Useful for cross-references
- `eval`: whether to evaluate the code chunk
- `echo`: whether to display the code chunk
- `output`: whether to show chunk output
- `results`: 'asis' includes the output without markup
- `message`: whether to display messages
- `warning`: whether to display warnings
- `error`: `true`: continue even if code returns an error.
- `fig-cap`: caption for the figure
- `fig-width`, `fig-height`: width and height of the figure
- `cache`: whether to cache the code chunk
- `dependson`: cache dependencies

Debugging

- The Quarto document is compiled in a different environment from your R console.
- If you get an error, try running all chunks (Ctrl+Alt+R).
- If you can't reproduce the error, check the working directory (add `getwd()` in a chunk).
- Try setting `error: true` on problem chunk to help you diagnose what happens. (But change it back!)
- Look at the intermediate files (`.md` or `.tex`) to see what is happening.

Caching

```
```{r}  
#| cache: true
```
```

- When evaluating code chunks, knitr will save the results of chunks with caching to files to be reloaded in subsequent runs.
- Caching is useful when a chunk takes a long time to run.
- It will re-run if the code in the chunk changes in any way (even comments or spacing).
- Beware of inherited objects from earlier chunks. Without explicit dependencies, a chunk will not re-run if inherited objects change.
- Beware of dependence on external files.

Caching

```
```{r}
#| label: chunk1
#| cache = TRUE
x <- 1
```

```{r}
#| label: chunk2
#| cache: true
#| dependson: "chunk1"
y <- x*3
```
```

Cache will be rebuilt if:

- Chunk options change except `include`
- Any change in the code, even a space or comment
- An explicit dependency changes

Do not cache if:

- setting R options like `options('width')`
- setting knitr options like `opts_chunk$set()`
- loading packages via `library()` if those packages are used by uncached chunks

Caching with random numbers

```
```{r}  
#| label: setup
#| include: false
knitr::opts_chunk$set(cache.extra = knitr::rand_seed)
```
```

- `rand_seed` is an unevaluated expression.
- Each chunk will check if `.Random.seed` has been changed since the last run.
- If it has, the chunk will be re-run.

Some caching options

- `cache-comments` If `false`, changing comments does not invalidate the cache.
- `cache-rebuild` Force rebuild of cache.
- `dependson` A character vector of labels of chunks that this chunk depends on.
- `autodep` If `true`, the dependencies are automatically determined. (May not be reliable.)

Build automatic dependencies among chunks

```
...  
execute:  
  cache: true  
  autodep: true  
...
```

Child documents

```
```{r}  
#| child: file1.qmd, file2.qmd
```
```

Child documents

```
```{r}  
#| child: file1.qmd, file2.qmd
```
```

Conditional inclusion

```
```{r}  
#| child: !expr if(condition) 'file1.qmd' else 'file2.qmd'
```
```

Child documents

```
```{r}  
#| child: file1.qmd, file2.qmd
```
```

Conditional inclusion

```
```{r}  
#| child: !expr if(condition) 'file1.qmd' else 'file2.qmd'
```
```

R Script files

```
```{r}  
#| file: "Rscript1.R"
```
```

- Better than `source("Rscript1.R")` because output of script included and dependencies tracked.

Other language engines

```
```{python}  
print("Hello Python!")
```
```

```
```{stata}  
sysuse auto
summarize
```
```

- Python and Stata need to be installed with executables on PATH

Other language engines

```
names(knitr::knit_engines$get())
```

```
[1] "awk"      "bash"      "coffee"    "gawk"      "groovy"
[6] "haskell"  "lein"      "mysql"      "node"      "octave"
[11] "perl"     "php"       "psql"       "Rscript"   "ruby"
[16] "sas"      "scala"     "sed"        "sh"        "stata"
[21] "zsh"      "asis"      "asy"        "block"     "block2"
[26] "bslib"    "c"         "cat"        "cc"        "comment"
[31] "css"      "ditaa"     "dot"        "embed"     "eviews"
[36] "exec"     "fortran"   "fortran95"  "go"        "highlight"
[41] "js"       "julia"     "python"     "R"         "Rcpp"
[46] "sass"     "scss"     "sql"        "stan"      "targets"
[51] "tikz"     "verbatim"  "ojs"        "mermaid"   "glue"
[56] "glue_sql" "gluesql"
```

Extensions and templates

- Quarto extensions modify and extend functionality.
- They are stored locally, in the `_extensions` folder alongside the qmd document.
- See <https://quarto.org/docs/extensions/> for a list.
- Templates are extensions used to define new output formats.
- Journal templates at <https://quarto.org/docs/extensions/listing-journals.html>
- Monash templates at <https://github.com/quarto-monash>

quarto on the command line

- `quarto render` to render a quarto or Rmarkdown document.
- `quarto preview` to preview a quarto or Rmarkdown document.
- `quarto add <gh-org>/<gh-repo>` to add an extension from a github repository.
- `quarto update <gh-org>/<gh-repo>` to update an extension
- `quarto remove <gh-org>/<gh-repo>` to remove an extension
- `quarto list extensions installed`
- `quarto use template <gh-org>/<gh-repo>` to use existing repo as starter template.

Add a custom format

From the CLI: `quarto add quarto-monash/memo`

Add a custom format

From the CLI: `quarto add quarto-monash/memo`

New folder/files added

```
├── _extensions
│   ├── quarto-monash
│   │   └── memo
│   │       └── ...
```

Add a custom format

From the CLI: `quarto add quarto-monash/memo`

New folder/files added

```
├── _extensions
│   ├── quarto-monash
│   │   └── memo
│   │       └── ...
```

Update YAML

```
---
title: "My new file using the Monash memo format"
format: memo-pdf
---
```

Activity 1

- Set up a new project.
- Create a quarto document using an html format.
- Add a code chunk to generate a figure with a caption.
- Reference the figure in the text using `@fig-chunklabel`.
- Add the monash memo extension and generate a pdf output.

Activity 2

- Create a thesis using the `quarto-monash/thesis` template, and add your name and details to the front page.

Activity 3

- Work on your Assignment 2.
- Ask for help if you need it.