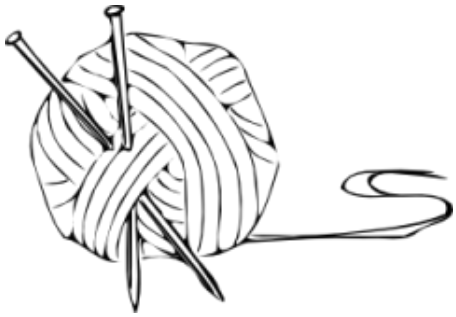
Control output



Manipulate six types of chunk output and inline output

As introduced in the main manual, **knitr** uses the **evaluate** package to evaluate R code chunks, and there are six types of output: source code, normal text output, messages, warnings, errors and plots. Below we give a summary on the fine control over them.

1. source code: use chunk option `echo`, e.g. `echo=FALSE` hides the R code

2. normal text output: use option `results` (`markup` marks up the results; `asis` return texts as-is; `hide` hides the results)

3. messages: option `message` (`FALSE` hides messages in the output)

4. warnings: option `warning` (`FALSE` hides warnings in the output)

5. errors: option `error` (`FALSE` will make R stop if an error occurs; `TRUE` will show the error messages in the output)

6. plots: option `fig.keep` (`none` discards all plots; `all` for all plots including low-level plots; `high` for high-level plots)

These options are orthogonal to each other, so you are free to turn on/off one type of output without affecting other pieces.

Note all options in **knitr** can take values from R expressions, which brings the feature of conditional evaluation introduced in the main manual. In short, `eval=dothis` means the real value of `eval` is taken from a variable named `dothis` in the global environment; by

manipulating this variable, we can turn on/off the evaluation of a batch of chunks.

Advanced usage of the `echo` option

Besides `TRUE/FALSE`, the chunk option `echo` can also take a numeric vector to selectively echo source code in the output. This vector indexes the complete R expresions in the code chunk, e.g. `echo=1` means only include the first expression in the output. Below is a complete example:

```
<<hide-par, echo=3:4>>=
## 'ugly' code that I do not want to show
par(mar = c(4, 4, 0.1, 0.1), cex.lab = 0.95, cex.axis =
0.9,
    mgp = c(2, 0.7, 0), tcl = -0.3)
plot(mtcars[, 1:2])
plot(mtcars[, 4:5])
@
```

The `par()` expression is not essential to this code chunk, and it is even distracting to the readers, so we may want to hide it in the output. In this particular case, we do not want the first expression (comments) either; `echo=3:4` means the third and fourth expressions are to be included in the output. Note the index of an expression is not necessarily the same as its line number. Alternatively, we can use `echo=-(1:2)` to remove the first two expressions.

It may be confusing to the reader if you select a discrete subset of the source code; often times we should use `a:b` or `-(a:b)` to select a (relatively) complete subset. However, nobody can stop you from doing so:

```
% select 3rd and 5th expressions
<<hide-par, echo=c(3, 5)>>=
## 'ugly' code that I do not want to show
par(mar = c(4, 4, 0.1, 0.1), cex.lab = 0.95, cex.axis =
0.9,
    mgp = c(2, 0.7, 0), tcl = -0.3)
plot(mtcars[, 1:2])
par(mar = c(4, 4, 1, 0.5)) # reset margins
plot(mtcars[, 4:5])
@
```

Inline output

Besides chunk output, there is another type of output:

inline R code output (e.g. `\Sexpr{t.test(x)$p.value}`). Numeric output is treated specially: the numbers will be written in scientific notations if they are too big or small. The threshold between scientific notation and fixed notation is the R option `scipen` (see `?options` for details). Basically if a number is bigger than `10^4` or smaller than `10^{-4}` (applies to the absolute values of negative numbers too), it will be denoted in scientific notation. Depending on the output format (LaTeX or HTML), **knitr** will use the appropriate code like `$3.14 \times 10^5$` or `3.14 &times; 10<sup>5</sup>`.

Another R option `digits` controls how many digits a number should be rounded with. If you do not like the default `options(scipen = 0, digits = 4)`, you can change them in the first chunk like:

```
<<setup, echo=FALSE, cache=FALSE>>=
## numbers >= 10^5 will be denoted in scientific
notation,
## and rounded to 2 digits
options(scipen = 1, digits = 2)
@
```

A Markdown example:

```
Inline code looks like this `r 1+1`
```

The above would print `2`, inline with the rest of the sentence.

An Rnw example (LaTeX):

```
Inline code looks like this \Sexpr{1+1}
```

An R HTML example:

```
<p>Inline code looks like this <!--rinline 1+1 --></p>
```

For R HTML documents, the character results are written in `<code></code>` by default. If you would like to drop the `<code></code>` tags in the output, simply wrap your R code with `I()`. For example,

```
<p>Inline code looks like this <!--rinline I(1+1) -->
</p>
```

See this SO post for another example.

Long lines of text output

Normally R respects the option `width` (set via `options(width = ??)`) when printing text output, e.g. `rnorm(100)`. The default value for `width` is set to `75` and it is likely you want a smaller value if you use LaTeX. In some cases, you may see the output is still too width even if you have set this option to be a small value, and that is usually because R does not respect this option. I cannot do much about this, although there can be some hacks on **knitr**'s side. See #421 for one example.

One comment on messages

Many people like to use `cat()` to write messages in R, and this is a very bad practice, because it is inconvenient to turn off such messages. When we really mean *messages*, we should use `message()`. These regular messages can be conveniently suppressed by `suppressMessages()` or captured by **knitr**. Some package authors did not seem to pay attention to this issue; when we load a package or use functions in a package, we may see fake messages that cannot be supppressed. If a startup message is really necessary, it should be put in `packageStartupMessage()`. When you are unable to turn off some messages by `message=FALSE` in **knitr**, it is probably time to request the package authors to make some changes.

Similarly, we should use `warning()` when we really mean a warning.

So watch out your `cats`!

Misc

If you use `echo=FALSE, results='hide'` on a chunk, you may see extra empty lines in the output; if you do not want these empty lines, see #231 for a possible solution.

If you have any questions, please consider asking them on *StackOverflow*, where you may get more attention and fast answers.