Handling Conditions in R

01-23-2020

Conditions

There are three conditions that you can signal in code: errors, warnings, and messages.

- Errors are the most severe; they indicate that there is no way for a function to continue and execution must stop.
- Warnings fall somewhat in between errors and message, and typically indicate that something has gone wrong but the function has been able to at least partially recover.
- Messages are the mildest; they are way of informing users that some action has been performed on their behalf.

Actually, there is one more condition called SIGINT which interrupts execution by pressing `Escape`, `Ctrl + Break`, or `Ctrl + C` (depending on the platform).

```
stop("This is what an error looks like")
```

```
## Error in eval(expr, envir, enclos): This is what an error looks like
```

```
#> Error in eval(expr, envir, enclos): This is what an error looks like

warning("This is what a warning looks like")
```

```
## Warning: This is what a warning looks like
```

```
#> Warning: This is what a warning looks like

message("This is what a message looks like")
```

```
## This is what a message looks like
```

```
#> This is what a message looks like
```

Errors

```
f <- function() g()
g <- function() h()
h <- function() stop("This is an error!")

f()
```

```
## Error in h(): This is an error!
```

Without the call information.

```r
h <- function() stop("This is an error!", call. = FALSE)
f()
```

```
## Error: This is an error!
```

Warnings

```r
fw <- function() {
  cat("1\n")
  warning("W1")
  cat("2\n")
  warning("W2")
  cat("3\n")
  warning("W3")
}
```

By default, warnings are cached and printed only when control returns to the top level.

```r
fw()
```

```
## 1

## Warning in fw(): W1

## 2

## Warning in fw(): W2

## 3

## Warning in fw(): W3
```

The behavior could be controled by the `warn` option.

- The default behaviour with `options(warn = 0)`
- To make warnings appear immediately, set `options(warn = 1)`
- To turn warnings into errors, set `options(warn = 2)`

```r
log(-1)
```

```
## Warning in log(-1): NaNs produced

## [1] NaN
```

```r
file.remove("this-file-doesn't-exist")
```

```
## Warning in file.remove("this-file-doesn't-exist"): cannot remove file 'this-
## file-doesn't-exist', reason 'No such file or directory'

## [1] FALSE
```

Messages

```
fm <- function() {
  cat("1\n")
  message("M1")
  cat("2\n")
  message("M2")
  cat("3\n")
  message("M3")
}

fm()
```

```
## 1

## M1

## 2

## M2

## 3

## M3
```

When start a long running progress, it may be a better idea to use `progress_bar` that `message`.

```
for (i in 1:5) {
  message("tick")
  Sys.sleep(1)
}
```

```
library(progress)
pb <- progress_bar$new(total = 5)
for (i in 1:5) {
  pb$tick()
  Sys.sleep(1)
}
```

It's important to compare `message()` with `cat()`. At the first glance, they may look similar.

```
cat("hello\n")
```

```
## hello
```

```
message("hello")
```

```
## hello
```

However, they are piped to different channels. For instance, `capture.output` cannot be used to capture `message`.

```r
capture.output(cat("hello\n"))
```

```
## [1] "hello"
```

```r
capture.output(message("hello"))
```

```
## hello
```

```
## character(0)
```

Ignoring conditions

The simplest way of handling conditions in R is to simply ignore them:

- Ignore errors with try().
- Ignore warnings with suppressWarnings().
- Ignore messages with suppressMessages().

```r
f2 <- function(x) {
  try(log(x))
  10
}
# the error message will be displayed but execution will continue
f2("a")
```

```
## Error in log(x) : non-numeric argument to mathematical function
```

```
## [1] 10
```

```r
# `try` doesn't catch `warnings`.
f2(-1)
```

```
## Warning in log(x): NaNs produced
```

```
## [1] 10
```

```r
suppressWarnings({
  warning("Uhoh!")
  warning("Another warning")
  1
})
```

```
## [1] 1
```

```r
suppressMessages({
  message("Hello there")
  2
})
```

```
## [1] 2
```

```r
suppressWarnings({
  message("You can still see me")
  3
})
```

```
## You can still see me
```

```
## [1] 3
```

Handling conditions

- Use `tryCatch` to handle errors, warnings and messges
- Use `withCallingHandlers` to handle warnings and messages

```r
tryCatch(
  error = function(cnd) {
    # code to run when error is thrown
    message("we got an error")
    5
  },
  log("a")
)
```

```
## we got an error
```

```
## [1] 5
```

```r
tryCatch(
  warning = function(cnd) {
    # code to run when message is signalled
    message("we got a warning")
    5
  },
  log(-1)
)
```

```
## we got a warning
```

```
## [1] 5
```

```r
withCallingHandlers(
  warning = function(cnd) {
    # code to run when message is signalled
    message("we got a warning")
    5  # is not used
  },
  log(-1)
)
```

```
## we got a warning
```

```
## Warning in log(-1): NaNs produced
```

```
## [1] NaN
```

```
tryCatch(
  message = function(cnd) {
    message("hi")
    5
  },
  message("hello")
)
```

```
## hi
```

```
## [1] 5
```

```
withCallingHandlers(
  message = function(cnd) {
    message("hi")
    5  # is not used
  },
  message("hello")
)
```

```
## hi
```

```
## hello
```

The condition object `cnd`

In the above examples, there is an object `cnd` in the handler. You might wonder what it is.

```
(cnd <- tryCatch(
  error = function(cnd) {
    # return cnd
    cnd
  },
  log("a")
))
```

```
## <simpleError in log("a"): non-numeric argument to mathematical function>
```

```
str(cnd)
```

```
## List of 2
##  $ message: chr "non-numeric argument to mathematical function"
##  $ call   : language log("a")
##  - attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

```r
# the error message
conditionMessage(cnd)
```

```
## [1] "non-numeric argument to mathematical function"
```

```r
# if you want to re-evaluate the call
eval(conditionCall(cnd))
```

```
## Error in log("a"): non-numeric argument to mathematical function
```

Finally, the `finally` handler

`finally` expression is always executed.

```r
path <- tempfile()
tryCatch(
  error = function(cnd) {
    message("we got a error: ", conditionMessage(cnd))
  },
  finally = {
    file.remove(path)
  },
  cat(a, file = path)
)
```

```
## we got a error: object 'a' not found
```

Reference

Advanced R Chapter 8 https://adv-r.hadley.nz/debugging.html