8_Conditions

2022-11-02

8. Conditions

8.1 Introduction

8.1.1 Prerequisites

```
library(rlang)
```

8.2 Signalling conditions

```
stop("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
```

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

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

8.2.1 Errors

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

f()
```

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

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

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

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

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

8.2.2 Warnings

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

```
fw()
```

```
## 1

## Warning in fw(): W1

## 2

## Warning in fw(): W2

## 3

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

```
options(warn = 0)
# Setting warn to 0 still doesn't fix immediate printing
fw()
```

```
## 1

## Warning in fw(): W1

## 2

## Warning in fw(): W2

## 3

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

```
formals(1)
```

```
## Warning in formals(fun): argument is not a function
```

```
## NULL
```

```
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
```

```
lag(1:3, k = 1.5)
```

```
## Warning in lag.default(1:3, k = 1.5): 'k' is not an integer
```

```
## [1] 1 2 3
## attr(,"tsp")
## [1] -1  1  1
```

```
as.numeric(c("18", "30", "50+", "345,678"))
```

```
## Warning: NAs introduced by coercion
```

```
## [1] 18 30 NA NA
```

8.2.3 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
```

```
cat("Hi!\n")
```

```
## Hi!
```

```
message("Hi!")
```

```
## Hi!
```

8.2.4 Exercises

1. Write a wrapper around file.remove() that throws an error if the file to be deleted does not exist.

```
better.file.remove <- function(file){
  if(file.exists(file)){
    file.remove(file)
  } else{
    stop("File does not exist")
  }
}

better.file.remove("test.txt")
```

```
## Error in better.file.remove("test.txt"): File does not exist
```

```
file.create("test.txt")
```

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

```
better.file.remove("test.txt")
```

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

2. What does the appendLF argument to message() do? How is it related to cat()?

```
message("No Breaks for me", appendLF = F)
```

```
## No Breaks for me
```

```
message("Now a break", appendLF = T)
```

```
## Now a break
```

```
message("Woo", appendLF = T)
```

```
## Woo
```

```
cat("Hi")
```

```
## Hi
```

```
cat("Hello")
```

```
## Hello
```

Basically you can decide to extend the message you already have by setting appendLF to False. `cat()` by default is similar to `message(appendLF = F)`

8.3 Ignoring conditions

```
f1 <- function(x) {
  log(x)
  10
}
f1("x")
```

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

```
f2 <- function(x) {
  try(log(x))
  10
}
f2("a")
```

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

## [1] 10
```

```
default <- NULL
try(default <- read.csv("possibly-bad-input.csv"), silent = TRUE)
```

```
## Warning in file(file, "rt"): cannot open file 'possibly-bad-input.csv': No such
## file or directory
```

```
default
```

```
## NULL
```

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

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

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

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

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

```
## You can still see me

## [1] 3
```

8.4 Handling conditions

```
tryCatch(
  error = function(cnd) {
    # code to run when error is thrown
  },
   code_to_run_while_handlers_are_active
)
```

```
## NULL
```

```
withCallingHandlers(
  warning = function(cnd) {
    # code to run when warning is signaled
  },
  message = function(cnd) {
    # code to run when message is signaled
  },
    code_to_run_while_handlers_are_active
)
```

```
## Error in withCallingHandlers(warning = function(cnd) {: object 'code_to_run_while_handlers_are_activ
```

8.4.1 Condition objects

```
cnd <- catch_cnd(stop("An error"))
str(cnd)
```

```
## List of 2
##  $ message: chr "An error"
##  $ call   : language force(expr)
##  - attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

```
conditionMessage(cnd)
```

```
## [1] "An error"
```

```
conditionCall(cnd)
```

```
## force(expr)
```

8.4.2 Exiting handlers

```
f3 <- function(x) {
  tryCatch(
    error = function(cnd) NA,
    log(x)
  )
}

f3("x")
```

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

```
f3(10)
```

```
## [1] 2.302585
```

```
tryCatch(
  error = function(cnd) 10,
  1 + 1
)
```

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

```
tryCatch(
  error = function(cnd) 10,
  {
    message("Hi!")
    1 + 1
  }
)
```

```
## Hi!
```

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

```
tryCatch(
  message = function(cnd) "There",
  {
    message("Here")
    stop("This code is never run!")
  }
)
```

```
## [1] "There"
```

```
tryCatch(
  error = function(cnd) {
    paste0("--", conditionMessage(cnd), "--")
  },
  stop("This is an error")
)
```

```
## [1] "--This is an error--"
```

```
path <- tempfile()
tryCatch(
  {
    writeLines("Hi!", path)
    # ...
  },
  finally = {
    # always run
    unlink(path)
  }
)
```

8.4.3 Calling handlers

```
tryCatch(
  message = function(cnd) cat("Caught a message!\n"),
  {
    message("Someone there?")
    message("Why, yes!")
  }
)
```

```
## Caught a message!
```

```
withCallingHandlers(
  message = function(cnd) cat("Caught a message!\n"),
  {
    message("Someone there?")
    message("Why, yes!")
  }
)
```

```
## Caught a message!

## Someone there?

## Caught a message!

## Why, yes!
```

```
withCallingHandlers(
  message = function(cnd) message("Second message"),
  message("First message")
)
```

```
## Second message

## First message
```

```
# Bubbles all the way up to default handler which generates the message
withCallingHandlers(
  message = function(cnd) cat("Level 2\n"),
  withCallingHandlers(
    message = function(cnd) cat("Level 1\n"),
    message("Hello")
  )
)
```

```
## Level 1
## Level 2

## Hello
```

```
# Muffles the default handler which prints the messages
withCallingHandlers(
  message = function(cnd) {
    cat("Level 2\n")
    cnd_muffle(cnd)
  },
  withCallingHandlers(
    message = function(cnd) cat("Level 1\n"),
    message("Hello")
  )
)
```

```
## Level 1
## Level 2
```

```
# Muffles level 2 handler and the default handler
withCallingHandlers(
  message = function(cnd) cat("Level 2\n"),
  withCallingHandlers(
    message = function(cnd) {
      cat("Level 1\n")
      cnd_muffle(cnd)
    },
    message("Hello")
  )
)
```

```
## Level 1
```

cnd doesn't pass to tlevel 2 and is muffled by level 1

8.4.4 Call stacks

```
f <- function() g()
g <- function() h()
h <- function() message("!")
```

```
withCallingHandlers(f(), message = function(cnd) {
  lobstr::cst()
  cnd_muffle(cnd)
})
```

```
##      x
##   1. +-base::withCallingHandlers(...)
##   2. +-global f()
##   3. | \-global g()
##   4. |   \-global h()
##   5. |     \-base::message("!")
##   6. |       +-base::withRestarts(...)
##   7. |       | \-base (local) withOneRestart(expr, restarts[[1L]])
##   8. |       |   \-base (local) doWithOneRestart(return(expr), restart)
##   9. |       \-base::signalCondition(cond)
##  10. \-global `<fn>`(`<smplMssg>`)
##  11.   \-lobstr::cst()
```

```
tryCatch(f(), message = function(cnd) lobstr::cst())
```

```
##     x
##  1. \-base::tryCatch(f(), message = function(cnd) lobstr::cst())
##  2.   \-base (local) tryCatchList(expr, classes, parentenv, handlers)
##  3.     \-base (local) tryCatchOne(expr, names, parentenv, handlers[[1L]])
##  4.       \-value[[3L]](cond)
##  5.         \-lobstr::cst()
```

8.4.5 Exercises

1. What extra information does the condition generated by `abort()` contain compared to the condition generated by `stop()` i.e. what's the difference between these two objects? Read the help for `?abort` to learn more.

```
catch_cnd(stop("An error"))
```

```
## <simpleError in force(expr): An error>
```

```
catch_cnd(abort("An error"))
```

```
## <error/rlang_error>
## Error:
## ! An error
## ---
## Backtrace:
```

abort shows you the backtrace of what triggered the error

2. Predict the results of evaluating the following code

```
show_condition <- function(code) {
  tryCatch(
    error = function(cnd) "error",
    warning = function(cnd) "warning",
    message = function(cnd) "message",
    {
      code
      NULL
    }
  )
}

show_condition(stop("!"))
```

```
## [1] "error"
```

```
show_condition(10)
```

```
## NULL
```

```
show_condition(warning("?!"))
```

```
## [1] "warning"
```

```
show_condition({
  10
  message("?")
  warning("?!")
})
```

```
## [1] "message"
```

“error”, “NULL”, “warning”, “message”

3. Explain the results of running this code:

```
withCallingHandlers(
  message = function(cnd) message("b"),
  withCallingHandlers(
    message = function(cnd) message("a"),
    message("c")
  )
)
```

```
## b
```

```
## a

## b

## c
```

`message("c")` triggers the first handler which calls `message(a)` which triggers the second handler. The second handler then messages "b", then the message from the first handler comes through with "a" which retriggers the second handler which sends "b" again. Finally the originally message of "c" is printed

4. Read the source code for `catch_cnd()` and explain how it works.

```
catch_cnd
```

```
## function (expr, classes = "condition")
## {
##     stopifnot(is_character(classes))
##     handlers <- rep_named(classes, list(identity))
##     eval_bare(rlang::expr(tryCatch(!!!handlers, {
##         force(expr)
##         return(NULL)
##     })))
## }
## <bytecode: 0x000002753b31fad0>
## <environment: namespace:rlang>
```

First checks to see if classes is provided as a character object. Creates a list object called handlers which holds the different types of conditions to be caught. Then evaluates the expression using `tryCatch`

5. How could you rewrite `show_condition()` to use a single handler?

```
show_condition
```

```
## function(code) {
##   tryCatch(
##     error = function(cnd) "error",
##     warning = function(cnd) "warning",
##     message = function(cnd) "message",
##     {
##       code
##       NULL
##     }
##   )
## }
## <bytecode: 0x00000275384793f0>
```

```
show_condition2 <- function(code) {
  tryCatch(
    condition = function(cnd) {
      if(is_error(cnd))   return("error")
      if(is_warning(cnd)) return("warning")
```

```
      if(is_message(cnd)) return("message")
    },
    {
      code
      NULL
    }
  )
}

show_condition2(stop("!"))
```

```
## [1] "error"
```

```
show_condition2(10)
```

```
## NULL
```

```
show_condition2(warning("?!"))
```

```
## [1] "warning"
```

```
show_condition2({
  10
  message("?")
  warning("?!")
})
```

```
## [1] "message"
```

8.5 Custom conditions

```
abort(
  "error_not_found",
  message = "Path `blah.csv` not found",
  path = "blah.csv"
)
```

```
## Error:
## ! Path ‘blah.csv‘ not found
```

8.5.1 Motivation

```
log(letters)
```

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

```
log(1:10, base = letters)
```

```
## Error in log(1:10, base = letters): non-numeric argument to mathematical function
```

```
my_log <- function(x, base = exp(1)) {
  if (!is.numeric(x)) {
    abort(paste0(
      "`x` must be a numeric vector; not ", typeof(x), "."
    ))
  }
  if (!is.numeric(base)) {
    abort(paste0(
      "`base` must be a numeric vector; not ", typeof(base), "."
    ))
  }

  base::log(x, base = base)
}
```

```
my_log(letters)
```

```
## Error in `my_log()`:
## ! `x` must be a numeric vector; not character.
```

```
my_log(1:10, base = letters)
```

```
## Error in `my_log()`:
## ! `base` must be a numeric vector; not character.
```

8.5.2 Signalling

```
abort_bad_argument <- function(arg, must, not = NULL) {
  msg <- glue::glue("`{arg}` must {must}")
  if (!is.null(not)) {
    not <- typeof(not)
    msg <- glue::glue("{msg}; not {not}.")
  }

  abort("error_bad_argument",
    message = msg,
    arg = arg,
    must = must,
    not = not
  )
}
```

```
stop_custom <- function(.subclass, message, call = NULL, ...) {
  err <- structure(
    list(
```

```
      message = message,
      call = call,
      ...
    ),
    class = c(.subclass, "error", "condition")
  )
  stop(err)
}

err <- catch_cnd(
  stop_custom("error_new", "This is a custom error", x = 10)
)
class(err)
```

```
## [1] "error_new" "error"     "condition"
```

```
err$x
```

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

```
my_log <- function(x, base = exp(1)) {
  if (!is.numeric(x)) {
    abort_bad_argument("x", must = "be numeric", not = x)
  }
  if (!is.numeric(base)) {
    abort_bad_argument("base", must = "be numeric", not = base)
  }

  base::log(x, base = base)
}
```

```
my_log(letters)
```

```
## Error in `abort_bad_argument()`:
## ! `x` must be numeric; not character.
```

```
my_log(1:10, base = letters)
```

```
## Error in `abort_bad_argument()`:
## ! `base` must be numeric; not character.
```

8.5.3 Handling

```
library(testthat)
```

```
##
## Attaching package: 'testthat'
```

```
## The following objects are masked from 'package:rlang':
##
##     is_false, is_null, is_true
```

```
err <- catch_cnd(my_log("a"))
expect_s3_class(err, "error_bad_argument")
expect_equal(err$arg, "x")
expect_equal(err$not, "character")
```

```
tryCatch(
  error_bad_argument = function(cnd) "bad_argument",
  error = function(cnd) "other error",
  my_log("a")
)
```

```
## [1] "bad_argument"
```

```
tryCatch(
  error = function(cnd) "other error",
  error_bad_argument = function(cnd) "bad_argument",
  my_log("a")
)
```

```
## [1] "other error"
```

8.5.4 Exercises

1. Inside a package, it's occasionally useful to check that a package is installed before using it. Write a function that checks if a package is installed (`with requireNamespace("pkg", quietly = FALSE)`) and if not, throws a custom condition that includes the package name in the metadata.

```
abort_not_installed <- function(pkg, must){
  msg <- glue::glue("`{pkg}` must {must}")

  abort(
    "error_not_installed",
    message = msg,
    pkg = pkg,
    must = must
    )
}

got_it <- function(pkg) {
  if (!requireNamespace(pkg, quietly = FALSE)) {
    abort_not_installed(pkg = pkg, must = "be previously installed to use")
  }
  TRUE
}

got_it("ggplot2")
```

```
## Loading required namespace: ggplot2
```

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

```
got_it("ggplot3")
```

```
## Loading required namespace: ggplot3

## Error in `abort_not_installed()`:
## ! `ggplot3` must be previously installed to use
```

2. Inside a package you often need to stop with an error when something is not right. Other packages that depend on your package might be tempted to check these errors in their unit tests. How could you help these packages to avoid relying on the error message which is part of the user interface rather than the API and might change without notice?

Make some custom metadata in a custom condition which the other packages can check for. Make an error class which is unique to your package/error

8.6 Applications

8.6.1 Failure value

```
fail_with <- function(expr, value = NULL) {
  tryCatch(
    error = function(cnd) value,
    expr
  )
}

fail_with(log(10), NA_real_)
```

```
## [1] 2.302585
```

```
fail_with(log("x"), NA_real_)
```

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

```
try2 <- function(expr, silent = FALSE) {
  tryCatch(
    error = function(cnd) {
      msg <- conditionMessage(cnd)
      if (!silent) {
        message("Error: ", msg)
      }
      structure(msg, class = "try-error")
    },
    expr
  )
}

try2(1)
```

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

```
try2(stop("Hi"))
```

```
## Error: Hi

## [1] "Hi"
## attr(,"class")
## [1] "try-error"
```

```
try2(stop("Hi"), silent = TRUE)
```

```
## [1] "Hi"
## attr(,"class")
## [1] "try-error"
```

8.6.2 Success and failure values

```
foo <- function(expr) {
  tryCatch(
    error = function(cnd) error_val,
    {
      expr
      success_val
    }
  )
}
```

```
does_error <- function(expr) {
  tryCatch(
    error = function(cnd) TRUE,
    {
      expr
      FALSE
    }
  )
}
```

```
catch_cnd <- function(expr) {
  tryCatch(
    condition = function(cnd) cnd,
    {
      expr
      NULL
    }
  )
}
```

```r
safety <- function(expr) {
  tryCatch(
    error = function(cnd) {
      list(result = NULL, error = cnd)
    },
    list(result = expr, error = NULL)
  )
}

str(safety(1 + 10))
```

```
## List of 2
##  $ result: num 11
##  $ error : NULL
```

```r
str(safety(stop("Error!")))
```

```
## List of 2
##  $ result: NULL
##  $ error :List of 2
##   ..$ message: chr "Error!"
##   ..$ call   : language doTryCatch(return(expr), name, parentenv, handler)
##   ..- attr(*, "class")= chr [1:3] "simpleError" "error" "condition"
```

8.6.3 Resignal

```r
warning2error <- function(expr) {
  withCallingHandlers(
    warning = function(cnd) abort(conditionMessage(cnd)),
    expr
  )
}
```

```r
warning2error({
  x <- 2 ^ 4
  warn("Hello")
})
```

```
## Error:
## ! Hello
```

8.6.4 Record

```r
catch_cnds <- function(expr) {
  conds <- list()
  add_cond <- function(cnd) {
    conds <<- append(conds, list(cnd))
    cnd_muffle(cnd)
```

```
  }

  withCallingHandlers(
    message = add_cond,
    warning = add_cond,
    expr
  )

  conds
}

catch_cnds({
  inform("a")
  warn("b")
  inform("c")
})
```

```
## [[1]]
## <message/rlang_message>
## Message:
## a
##
## [[2]]
## <warning/rlang_warning>
## Warning:
## b
##
## [[3]]
## <message/rlang_message>
## Message:
## c
```

```
catch_cnds <- function(expr) {
  conds <- list()
  add_cond <- function(cnd) {
    conds <<- append(conds, list(cnd))
    cnd_muffle(cnd)
  }

  tryCatch(
    error = function(cnd) {
      conds <<- append(conds, list(cnd))
    },
    withCallingHandlers(
      message = add_cond,
      warning = add_cond,
      expr
    )
  )

  conds
}
```

```
catch_cnds({
  inform("a")
  warn("b")
  abort("C")
})
```

```
## [[1]]
## <message/rlang_message>
## Message:
## a
##
## [[2]]
## <warning/rlang_warning>
## Warning:
## b
##
## [[3]]
## <error/rlang_error>
## Error:
## ! C
## ---
## Backtrace:
```

8.6.5 No default behaviour

```
log <- function(message, level = c("info", "error", "fatal")) {
  level <- match.arg(level)
  signal(message, "log", level = level)
}
```

```
log("This code was run")
```

```
record_log <- function(expr, path = stdout()) {
  withCallingHandlers(
    log = function(cnd) {
      cat(
        "[", cnd$level, "] ", cnd$message, "\n", sep = "",
        file = path, append = TRUE
      )
    },
    expr
  )
}

record_log(log("Hello"))
```

```
## [info] Hello
```

```
ignore_log_levels <- function(expr, levels) {
  withCallingHandlers(
    log = function(cnd) {
      if (cnd$level %in% levels) {
        cnd_muffle(cnd)
      }
    },
    expr
  )
}

record_log(ignore_log_levels(log("Hello"), "info"))
```

```
withRestarts(signalCondition(cond), muffle = function() NULL)
```

If you create a condition object by hand, and signal it with signalCondition(), cnd_muffle() will not work. Instead you need to call it with a muffle restart defined, like this:

```
## Error in signalCondition(cond): object 'cond' not found
```

8.6.6 Exercises

1. Create `suppressConditions()` that works like `suppressMessages()` and `suppressWarnings()` but suppresses everything. Think carefully about how you should handle errors.

```
suppressConditions <- function(expr){
  tryCatch(
    error = function(cnd) invisible(cnd),
    interrupt = function(cnd) invisible(cnd),
    warning = function(cnd) invisible(cnd),
    message = function(cnd) invisible(cnd),
    expr
  )
}

error_obj <- suppressConditions({
  message("message")
  warning("warning")
  abort("error")
})

error_obj
```

```
## <simpleMessage in message("message"): message
## >
```

```
error_obj2 <- suppressConditions({log10("a")})
error_obj2
```

```
## <simpleError in log10("a"): non-numeric argument to mathematical function>
```

```
error_obj3 <- suppressConditions({log10(10)})
error_obj3
```

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

2. Compare the following two implementations of `message2error()`. What is the main advantage of `withCallingHandlers()` in this scenario? (Hint: look carefully at the traceback.)

```
message2error <- function(code) {
  withCallingHandlers(code, message = function(e) stop(e))
}
message2error(message("aaa"))
```

```
## aaa
```

```
message2error <- function(code) {
  tryCatch(code, message = function(e) stop(e))
}
message2error(message("aaa"))
```

```
## aaa
```

It doesn't throw an error and prints the message in the first one. Easier to work with the first than the second.

3. How would you modify the `catch_cnds()` definition if you wanted to recreate the original intermingling of warnings and messages?

```
catch_cnds <- function(expr) {
  conds <- list()
  add_cond <- function(cnd) {
    conds <<- append(conds, list(cnd))
    cnd_muffle(cnd)
  }

  tryCatch(
    error = function(cnd) {
      conds <<- append(conds, list(cnd))
    },
    withCallingHandlers(
      message = add_cond,
      warning = add_cond,
      expr
    )
  )

  conds
}
```

4. Why is catching interrupts dangerous? Run this code to find out.

```
bottles_of_beer <- function(i = 99) {
  message(
    "There are ", i, " bottles of beer on the wall, ",
    i, " bottles of beer."
  )
  while(i > 0) {
    tryCatch(
      Sys.sleep(1),
      interrupt = function(err) {
        i <<- i - 1
        if (i > 0) {
          message(
            "Take one down, pass it around, ", i,
            " bottle", if (i > 1) "s", " of beer on the wall."
          )
        }
      }
    )
  }
  message(
    "No more bottles of beer on the wall, ",
    "no more bottles of beer."
  )
}
```

Death loops where you can't exit until i is equal to 0