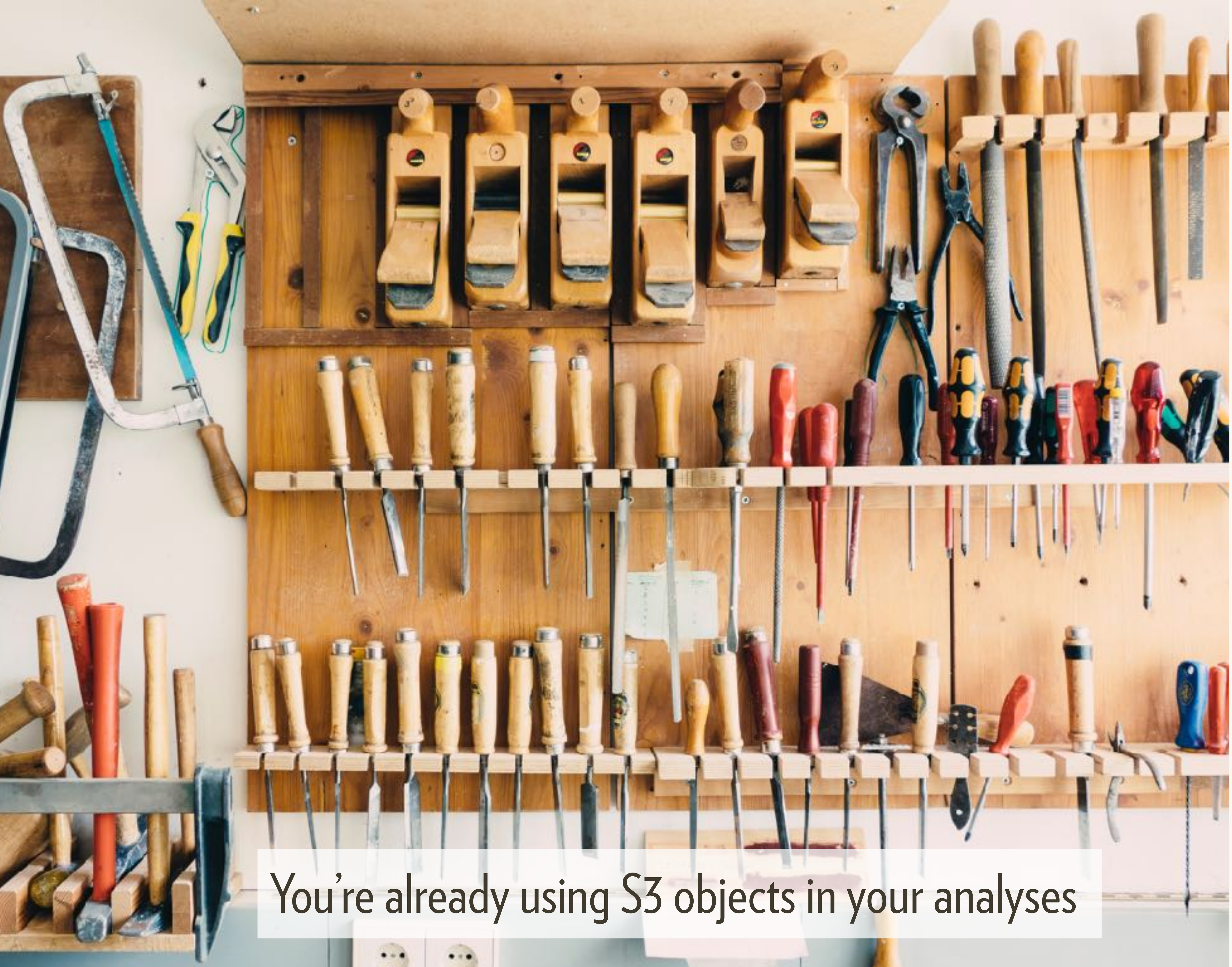
# OO programming

**_September 2017_**

Hadley Wickham
@hadleywickham
Chief Scientist, **RStudio**

# Motivation

Why should you care about S3?



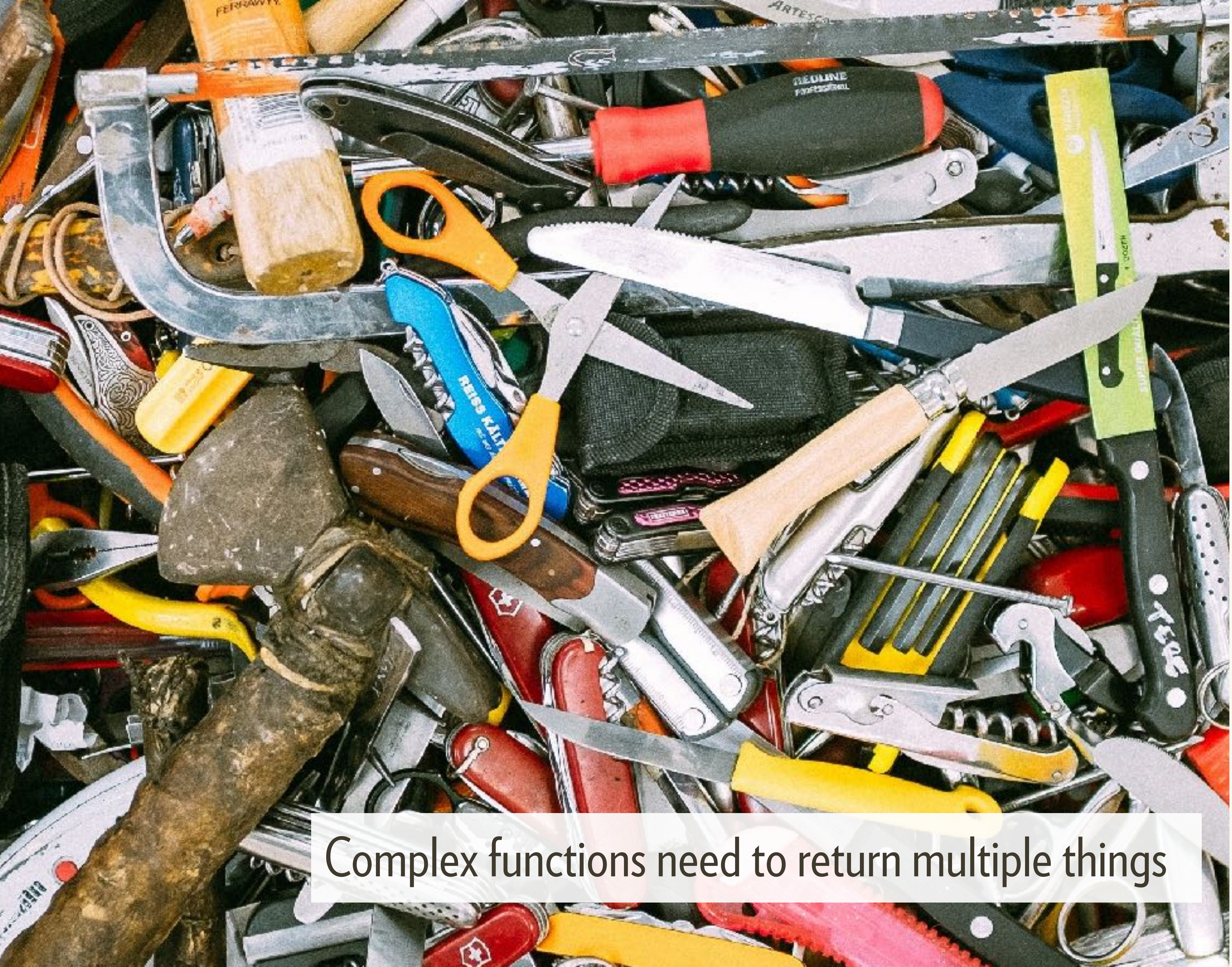
You’re already using S3 objects in your analyses

https://unsplash.com/photos/t5YUoHW6zRo

# Important S3 objects in base R

```
data.frame()

factor()
Sys.Date()
Sys.time()
table()
```



Complex functions need to return multiple things

https://unsplash.com/photos/Kw_zQBAChws

# This is obviously important for linear models

```r
mod <- lm(mpg ~ wt, data = mtcars)
str(mod)

# But also their summaries
sum <- summary(mod)
str(sum)
```

# Form follows function

https://unsplash.com/photos/gbP6hnvfYCM

# One example is linear models

```
sum
#> Call:
#> lm(formula = mpg ~ wt, data = mtcars)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -4.5432 -2.3647 -0.1252  1.4096  6.8727
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
#> wt           -5.3445     0.5591  -9.559 1.29e-10 ***
#> ---
#> Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#>
#> Residual standard error: 3.046 on 30 degrees of freedom
#> Multiple R-squared:  0.7528,  Adjusted R-squared:  0.7446
#> F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

# Another example is tibbles

Total size

```
# A tibble: 53,940 x 10
   carat cut         color clarity depth table price     x     y     z
   <dbl> <ord>       <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 0.230 Ideal       E     SI2      61.5  55.0   326  3.95  3.98  2.43
 2 0.210 Premium     E     SI1      59.8  61.0   326  3.8
 3 0.230 Good        E     VS1      56.9  65.0   327  4.0
 4 0.290 Premium     I     VS2      62.4  58.0   334  4.20  4.23  2.63
 5 0.310 Good        J     SI2      63.3  58.0   335  4.34  4.35  2.75
 6 0.240 "Very Good" J     VVS2     62.8  57.0   336  3.94  3.96  2.48
 7 0.240 "Very Good" I     VVS1     62.3  57.0   336  3.95  3.98  2.47
 8 0.260 "Very Good" H     SI1      61.9  55.0   337  4.07  4.11  2.53
 9 0.220 Fair        E     VS2      65.1  61.0   337  3.87  3.78  2.49
10 0.230 "Very Good" H     VS1      59.4  61.0   338  4.00  4.05  2.39
# ... with 53,930 more rows
```

Variable type

Only shows first 10 rows

Intermediate objects can allow you to partition API complexity

https://unsplash.com/photos/gbP6hnvfYCM

# e.g. string manipulation in base R vs stringr

```r
str_replace(x, fixed("y"), "")
str_replace(x, regex(".", ignore_case = TRUE), "")

# vs.

gsub("y", "", x, fixed = TRUE)
gsub("y", "", x, fixed = FALSE, ignore.case = TRUE)
```

# S3 makes packages extensible

## New methods

Lets you extend other packages

## New generics

Write packages in way that others can easily extend.

# You could use a nested if statement

```
mean <- function(x, ...) {
  if (is.Date(x)) {
    ...
  } else if (is.difftime(x)) {
    ...
  } else if (is.POSIXct(x)) {

  } else if (is.POSIXlt(x)) {
    ...
  } else {
    ...
  }
}
```

# But a generic function lets anyone extend

```
mean <- function(x, ...) {
  UseMethod("mean")
}

mean.Date <- function(x, ...) ...
mean.difftime <- function(x, ...) ...
mean.POSIXct <- function(x, ...) ...
mean.POSIXlt <- function(x, ...) ...

mean.default <- function(x, ...) ...
```

# Vector classes

https://adv-r.hadley.nz/S3.html

# Your turn

What is an attribute? What types of objects can have attributes?

How do you *get* the value of an attribute? How do you *set* the value an attribute?

What's *the* most important attribute?

# Attributes add arbitrary metadata to any object

```
x <- 1:6
attr(x, "max") <- 5
attr(x, "max")
attributes(x)

# structure returns a modified object
structure(1:10, min = 1, max = 10)

# Most important attribute is names()
```

# Your turn

Every S3 class is built on a base type (e.g. a vector). The two most important S3 classes are factor and data frame.

What are factors built on top of?
What attributes do they use?

What are data frames built on top of?
What attributes do they use?

```r
f <- factor(c("a", "b", "c"))
typeof(f)     # Built on top of integer
attributes(f) # Use levels and class attributes

d <- data.frame(f)
typeof(d)     # Built on top of list
attributes(d) # names, row.names and class
```

# “Scalar” classes

Principle:

Provide consistent structure and print method for complex return values

Change working directory/project to:

[safely]

# Challenge: how can improve the output of safely?

```
library(purrr)
safe_log <- safely(log)
```

```
safe_log("a")
#> result
#> NULL
#>
#> $error
#> <simpleError in log(...):
#>  non-numeric argument to
#>  mathematical function>
```

```
safe_log(10)
#> $result
#> [1] 2.302585
#>
#> $error
#> NULL
```

1. Figure out name
2. Define properties of the class
3. Write the constructor
4. Write methods

# Your turn

What are the constraints on the results of safely?

# Now, write the constructor

```
new_safely <- function(result = NULL, error = NULL) {
  if (!xor(is.null(result), is.null(error))) {
    stop(
      "One of `result` and `error` must be NULL",
      call. = FALSE
    )
  }

  structure(
    list(
      result = result,
      error = error
    ),
    class = "safely"
  )
}
```

Most S3 classes will have this form

# Then use the constructor

```
safely <-  function(.f) {
  stopifnot(is.function(.f))

  function(...) {
    tryCatch({
      new_safely(result = .f(...))
    }, error = function(e) {
      new_safely(error = e)
    })
  }
}
```

# How could we test the constructor?

| Abbreviation | Test |
|---|---|
| expect_null() | Checks if a literal NULL |
| expect_type()<br>expect_s3_class()<br>expect_s4_class() | Check that inherits from a given base type, S3 class, or S4 class. |
| expect_true()<br>expect_false() | Catch all expectations for anything not otherwise covered |

# Your turn

Write tests to ensure that our new safely() function returns the correct type of output regardless of whether or not an error occurs.

# Now we can improve the output with a print method

```
safe_log(10)
#> <safely: ok>
#> [1] 2.302585
```

I think it's good practice to include type in <>

```
safe_log("a")
#> <safely: error>
#> Error: non-numeric argument to
#>  mathematical function
```

# S3 methods all have the same basic structure

generic.class

Same arguments as generic

```
print.safely <- function(x, ...) {



}
```

Methods belong to **functions**, not classes

# Your turn: fill in the blanks

```
print.safely <- function(x, ...) {



}

# Useful helper found in utils.R
cat_line <- function(...) {
  cat(..., "\n", sep = "")
}
# See https://github.com/r-lib/cli for
# many more helpers.
```

# My print method

```r
print.safely <- function(x, ...) {
  if (is.null(x$result)) {
    cat_line("<safely: error>")
    cat_line("Error: ", x$error$message)
  } else {
    cat_line("<safely: ok>")
    print(x$result)
  }

  invisible(x)
}
```

Called primarily for side-effects

# How do we test printing code?

```
expect_output(
  new_safely(result = 1:10),
  "<safely: ok>"
)
```

# How do we test printing code?

```
expect_output(
  print(new_safely(result = 1:10)),
  "<safely: ok>"
)
```

# But this is tedious & error prone

```
# It's hard to define precisely what the output
# should be (but we know it when we see it)

# So instead we can use a regression test.
# This is a weaker guarantee than a unit test:
# we'll just get alerted when it changes
expect_known_output(
  new_safely(result = 1:10),
  test_path("safely-ok.txt"),
  print = TRUE
)
```

# A little colour can be transformative

```
print.safely <- function(x, ...) {
  if (is.null(x$result)) {
    cat_line("<safely: ", crayon::bold(crayon::red("error")), ">")
    cat_line(crayon::red("Error: "), x$error$message)
  } else {
    cat_line("<safely: ", crayon::green("ok"), ">")
    print(x$result)
  }

  invisible(x)
}
```

# Parameter objects

https://refactoring.guru/introduce-parameter-object

Principle:

Extract repeated groups of arguments into an object

https://www.refactoring.com/catalog/introduceParameterObject.html

# Your turn: brainstorm problems

```
strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)

grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE, fixed = FALSE, useBytes = FALSE, invert = FALSE)

grepl(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

regexpr(pattern, text, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

gregexpr(pattern, text, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

regexec(pattern, text, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)
```

# The arguments of grepl() interact in complex ways

| | `perl = FALSE` | `perl = TRUE` |
|---|---|---|
| `fixed = FALSE` | POSIX 1003.2 | perl |
| `fixed = TRUE` | exact matching | exact matching (with warning) |

And ignore.case does not apply to exact matching

# stringr takes a different approach

```
str_replace(x, fixed("y"))
str_replace(x, regex("."))
str_replace(x, boundary("word"))

# Allows different modes to have different
# arguments and hides details until you need
# to know about them
```

```r
fixed <- function(pattern, ignore_case = FALSE) {
  structure(
    pattern,
    ignore_case = ignore_case,
    class = c("fixed", "pattern", "character")
  )
}
```

```
type <- function(x) UseMethod("type")

type.boundary <- function(x) "bound"

type.regex <- function(x) "regex"

type.coll <- function(x) "coll"

type.fixed <- function(x) "fixed"

type.character <- function(x) {
  if (identical(x, "")) "empty" else "regex"
}
```

```
# In str_replace
str_detect <- function(string, pattern) {
  switch(type(pattern),
    empty = ,
    bound = str_count(string, pattern) > 0,
    fixed = stri_detect_fixed(...),
    coll  = stri_detect_coll(...),
    regex = stri_detect_regex(...)
  )
}
```

# Learning more

# Advanced R (2nd ed) has four chapters

**S3**: https://adv-r.hadley.nz/s3.html

**S4**: https://adv-r.hadley.nz/s4.html

**R6**: https://adv-r.hadley.nz/r6.html

**Trade-offs**: https://adv-r.hadley.nz/oo-tradeoffs.html