OO programming
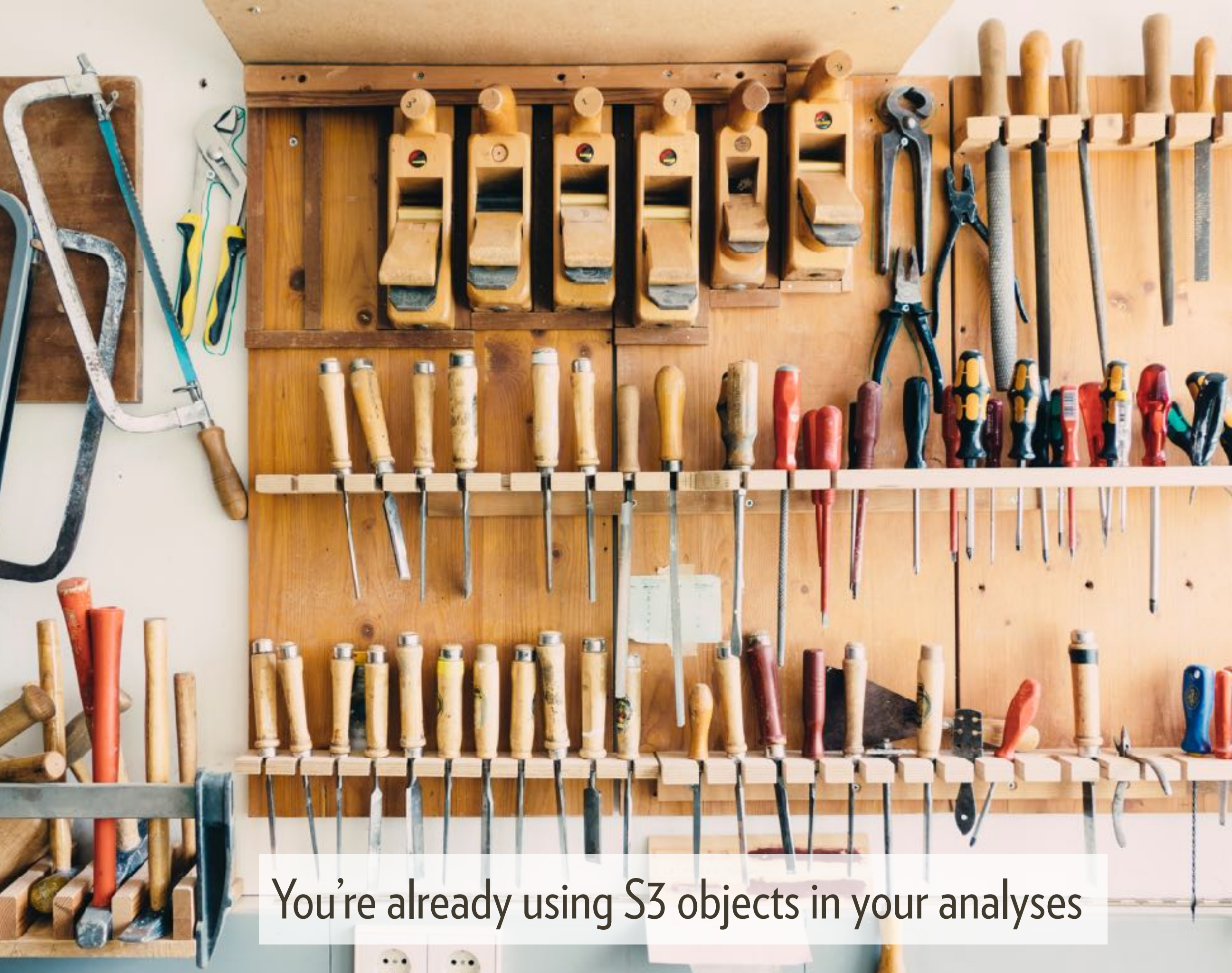
May 2018

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

Motivation

Why should you care about S3?

You're already using S3 objects in your analyses
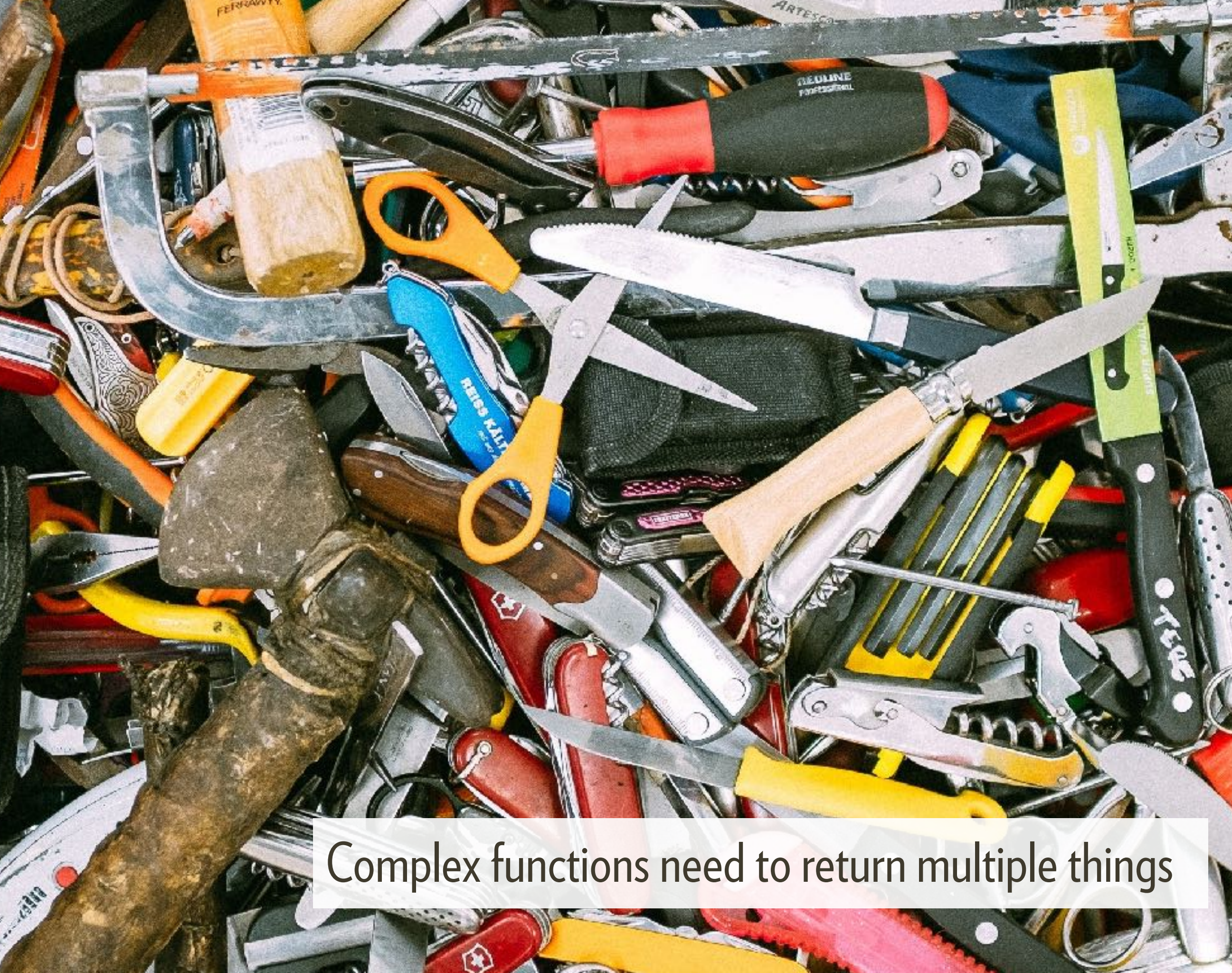
Important S3 objects in base R

```
data.frame()

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

Complex functions need to return multiple things

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

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

```
# 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
```

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

```r
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

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()
```

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?

```
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

What are the invariants of the results of safely?

```
safe_log <- purrr::safely(log)
# what do we know to be always true
# about the result of safe_log?
```

Invariants

Return a list
- two components: result and error
- result should always come first
- one is always NULL

Now, write the constructor

```r
new_safely <- function(result = NULL, error = NULL) {
  if (!(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

Definition of safely

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

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

Then use the constructor

```r
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() expect_s3_class() expect_s4_class()	Check that inherits from a given base type, S3 class, or S4 class.
expect_true() expect_false()	Catch all expectations for anything not otherwise covered

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

There is a bug in your version (think about what happens if the function returns NULL)!

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 **generics**, not classes

	Date	POSIXct	integer
print			
mean			
sum			

	Date	POSIXct	integer
print			
mean			
sum			

```r
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.
```

```
f <- function() NULL
safe_f <- safely(f)
safe_f()

# What has gone wrong?
# And how can you fix it?
```

My print method

```r
print.safely <- function(x, ...) {
  if (!is.null(x$error)) {
    cat_line("<safely: error>")
    cat_line("Error: ", x$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

```r
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)
}
```

New generic

Change working directory/project to:

[bizarro]

Goal: create a bizarro function

```
bizarro("abc")
#> [1] "cba"

bizarro(1)
#> [1] -1

bizarro(c(TRUE, FALSE))
#> [1] FALSE TRUE
```

We could eliminate using if + else

```r
str_reverse <- function(x) {
  purrr::map_chr(stringr::str_split(x, ""),
    ~ stringr::str_flatten(rev(.x))
  )
}

bizarro <- function(x) {
  if (is.character(x)) {
    str_reverse(x)
  } else if (is.numeric(x)) {
    -x
  } else if (is.logical(x)) {
    !x
  } else {
    stop(
      "Don't know how to make bizzaro <", class(x)[[1]], ">",
      call. = FALSE)
  }
}
```

But instead we'll create a new S3 generic

```
bizarro <- function(x) {
  UseMethod("bizarro")
}

bizarro.character <- function(x) {
  str_reverse(x)
}


bizarro("abc")
#> [1] cba
```

Magically passes all arguments to correct method

method = generic.class

Implement numeric and logical methods
Implement a data frame method that bizarros the column names, as well as each column.
(i.e. get tests parsing)

```r
bizarro.integer <- function(x) {
  -x
}


bizarro.logical <- function(x) {
  !x
}

bizarro.data.frame <- function(x) {
  names(x) <- bizarro(names(x))
  x[] <- purrr::map(x, bizarro)
  x
}
```

What happens when a method isn't available?

```r
bizarro(factor(letters))
#> Error in UseMethod("bizarro") :
#>  no applicable method for 'bizarro'
#>  applied to an object of class "factor"

# How can we do better?
# We need to provide a catch-all default method
```

```
bizarro.default <- function(x) {
  stop(
    "Don't know how to make bizzaro <",
    class(x)[[1]], ">",
    call. = FALSE
  )
}

bizarro(factor(letters))
#> Error: Don't know how to make
#> bizzaro <factor>
```

Your turn

What should bizzaro(factor("abc")) return?
Decide, encode your decisions in tests, then implement
bizarro.factor().

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