

Back at 2:45

Tidy evaluation:

Programming with `ggplot2` and `dplyr`

July 2018

Charlotte Wickham

@cvwickham

cwickham@gmail.com

cwick.co.nz

Adapted from *Tidy Tools* by Hadley Wickham



Motivation

Remove the duplication in this code

```
library(dplyr)
```

```
mtcars %>% group_by(cyl) %>% summarise(mean = mean(mpg))
```

```
mtcars %>% group_by(cyl) %>% summarise(mean = mean(wt))
```

```
mtcars %>% group_by(gear) %>% summarise(mean = mean(mpg))
```

```
mtcars %>% group_by(gear) %>% summarise(mean = mean(wt))
```

Typo: handout uses df, x1, y1 etc.
instead of real variables in mtcars.

Your turn

Identify the parts that change.

Give them names.

Make a function.

Does it work?

Remove the duplication in this code

```
mtcars %>% group_by(cyl) %>% summarise(mean = mean(mpg))  
mtcars %>% group_by(cyl) %>% summarise(mean = mean(wt))  
mtcars %>% group_by(gear) %>% summarise(mean = mean(mpg))  
mtcars %>% group_by(gear) %>% summarise(mean = mean(wt))
```

First identify the parts that change

mtcars	%>%	group_by(cyl)	%>%	summarise(mean = mean(mpg))
mtcars	%>%	group_by(cyl)	%>%	summarise(mean = mean(wt))
mtcars	%>%	group_by(gear)	%>%	summarise(mean = mean(mpg))
mtcars	%>%	group_by(gear)	%>%	summarise(mean = mean(wt))

Then give them names

df

mtcars
mtcars
mtcars
mtcars

%>% group_by(cyl)
%>% group_by(cyl)
%>% group_by(gear)
%>% group_by(gear)

group_var

%>% summarise(mean = mean(mpg))
%>% summarise(mean = mean(wt))
%>% summarise(mean = mean(mpg))
%>% summarise(mean = mean(wt))

summary_var

Now make a function

```
grouped_mean <- function(df, group_var, summary_var) {  
  df %>%  
    group_by(group_var) %>%  
    summarise(mean = mean(summary_var))  
}
```


It doesn't work 😭

```
grouped_mean <- function(df, group_var, summary_var) {  
  df %>%  
    group_by(group_var) %>%  
    summarise(mean = mean(summary_var))  
}
```

```
grouped_mean(mtcars, cyl, mpg)
```

```
#> Error: Column `group_var` is unknown
```

Vocabulary

We need some new vocabulary

**Evaluated using
usual R rules**

```
(x - min(x)) / (max(x) - min(x))
```

```
mtcars %>%
```

```
  group_by(cyl) %>%
```

```
  summarise(mean = mean(mpg))
```

Automatically **quoted** and evaluated
in a “non-standard” way

You're already familiar with this idea

```
df <- data.frame(  
  y = 1,  
  var = 2  
)
```

```
df$y
```

```
var <- "y"  
df$var
```

**Predict the
output!**

\$ automatically quotes the variable name

```
df <- data.frame(  
  y = 1,  
  var = 2  
)
```

```
df$y  
#> [1] 1
```

```
var <- "y"  
df$var  
#> [1] 2
```


If you want refer indirectly, must use `[[` instead

```
df <- data.frame(  
  y = 1,  
  var = 2  
)
```

```
var <- "y"  
df[[var]]  
#> [1] 1
```

	Quoted	Evaluated
Direct	df\$ <u>y</u>	???
Indirect	???	var <- "y" df[[var]]

	Quoted	Evaluated
Direct	<code>df\$<u>y</u></code>	<code>df[["y"]]</code>
Indirect	<code>???</code>	<code>var <- "y"</code> <code>df[[var]]</code>

	Quoted	Evaluated
Direct	<code>df\$<u>y</u></code>	<code>df[["y"]]</code>
Indirect		<code>var <- "y"</code> <code>df[[var]]</code>

Identify which arguments are auto-quoted

```
library(MASS)
```

```
mtcars2 <- subset(mtcars, cyl == 4)
```

```
with(mtcars2, sum(vs))  
sum(mtcars2$am)
```

```
rm(mtcars2)
```

Can't tell? Try running the argument alone

```
library(MASS)
```

```
#> Works
```

```
MASS
```

```
#> Error: object 'MASS' not found
```

```
# -> The 1st argument of library() is quoted
```

Can't tell? Try running the argument alone

```
subset(mtcars, cyl == 4)
```

```
#> Works
```

```
cyl == 4
```

```
#> Error: object 'cyl' not found
```

```
# -> The 2nd argument of subset() is quoted
```

You can now identify the quoted arguments

```
library(MASS)
```

**Evaluated using
usual R rules**

```
mtcars2 <- subset(mtcars, cyl == 4)
```

```
with(mtcars2, sum(vs))  
sum(mtcars2$am)
```

Automatically **quoted** and evaluated
in a “non-standard” way

```
rm(mtcars2)
```

Base R has 3 primary ways to “unquote”

Quoted/Direct	Evaluated/Indirect	
<code>df\$<u>y</u></code>	<code>x <- "y"</code> <code>df[[<u>x</u>]]</code>	Use a different function
<code>rm(<u>mtcars</u>)</code>	<code>x <- "mtcars"</code> <code>rm(list = <u>x</u>)</code>	Use a different argument
<code>library(<u>MASS</u>)</code>	<code>x <- "MASS"</code> <code>library(<u>x</u>, character.only = TRUE)</code>	Specify an additional argument

Identify which arguments are auto-quoted

```
library(tidyverse)
```



```
mtcars %>% pull(am)
```

```
by_cyl <- mtcars %>%
```

```
  group_by(cyl) %>%
```

```
  summarise(mean = mean(mpg))
```

```
ggplot(by_cyl, aes(cyl, mean)) +
```

```
  geom_point()
```

```
# OR
```

```
library(dplyr)
```

```
library(ggplot2)
```

← **Typo**


Identify which arguments are auto-quoted


```
library(tidyverse)
```

```
mtcars %>% pull(am)
```

```
by_cyl <- mtcars %>%  
  group_by(cyl) %>%  
  summarise(mean = mean(mpg))
```

```
ggplot(by_cyl, aes(cyl, mean)) +  
  geom_point()
```


	Quoted	Evaluated	Tidy
Direct	<code>df\$<u>y</u></code>	<code>df[["y"]]</code>	<code>pull(df, <u>y</u>)</code>
Indirect		<code>var <- "y"</code> <code>df[[<u>var</u>]]</code>	???

	Quoted	Evaluated	Tidy
Direct	<code>df\$<u>y</u></code>	<code>df[["y"]]</code>	<code>pull(df, <u>y</u>)</code>
Indirect		<code>var <- "y"</code> <code>df[[<u>var</u>]]</code>	<code>var <- quo(<u>y</u>)</code> <code>pull(df, !!<u>var</u>)</code>

Everywhere in the tidyverse uses !! to unquote

Pronounced
bang-bang

```
x_var <- quo(cyl)
y_var <- quo(mpg)
```

```
by_cyl <- mtcars %>%
  group_by(!!x_var) %>%
  summarise(mean = mean(!!y_varu))
```

```
ggplot(by_cyl, aes(!!x_var, mean)) +
  geom_point()
```

Rewrite to specify the column indirectly

```
var <- quo(cyl)
```

```
# Re-write to specify column indirectly
```

```
mtcars %>%
```

```
  summarise(
```

```
    avg = mean(cyl),
```

```
    sd = sd(cyl),
```

```
    n = sum(!is.na(cyl))
```

```
)
```

Rewrite to specify the column indirectly

```
var <- quo(cyl)
```

```
# Re-write to specify column indirectly
```

```
mtcars %>%
```

```
  summarise(
```

```
    avg = mean(!var),
```

```
    sd = sd(!var),
```

```
    n = sum(!is.na(!var))
```

```
)
```

```
#           avg           sd    n
```

```
# 1  6.1875 1.785922 32
```

Rewrite to specify the column indirectly

```
var <- quo(mpg)
```

Change in **one** place

```
# Re-write to specify column indirectly
```

```
mtcars %>%
```

```
  summarise(
```

```
    avg = mean(!var),
```

```
    sd = sd(!var),
```

```
    n = sum(!is.na(!var))
```

```
)
```

```
#           avg           sd    n
```

```
# 1 20.09062 6.026948 32
```

A recipe for:

Wrapping quoting functions

(More theory later...)

New: Identify quoted vs. evaluated arguments

```
mtcars %>% group_by(cyl) %>% summarise(mean = mean(mpg))  
mtcars %>% group_by(cyl) %>% summarise(mean = mean(wt))  
mtcars %>% group_by(gear) %>% summarise(mean = mean(mpg))  
mtcars %>% group_by(gear) %>% summarise(mean = mean(wt))
```

Typo: handout uses df, x1, y1 etc.
instead of real variables in mtcars.

New: Identify quoted vs. evaluated arguments

```
mtcars %>% group_by(cyl) %>% summarise(mean = mean(mpg))  
mtcars %>% group_by(cyl) %>% summarise(mean = mean(wt))  
mtcars %>% group_by(gear) %>% summarise(mean = mean(mpg))  
mtcars %>% group_by(gear) %>% summarise(mean = mean(wt))
```

Then identify the parts that could change

mtcars	%>% group_by(<u>cyl</u>)	%>% summarise(mean = <u>mean(mpg)</u>)
mtcars	%>% group_by(<u>cyl</u>)	%>% summarise(mean = <u>mean(wt)</u>)
mtcars	%>% group_by(<u>gear</u>)	%>% summarise(mean = <u>mean(mpg)</u>)
mtcars	%>% group_by(<u>gear</u>)	%>% summarise(mean = <u>mean(wt)</u>)

These become the function arguments

df

mtcars
mtcars
mtcars
mtcars

group_var

%>% group_by(cyl)
%>% group_by(cyl)
%>% group_by(gear)
%>% group_by(gear)

summary_var

%>% summarise(mean = mean(mpg))
%>% summarise(mean = mean(wt))
%>% summarise(mean = mean(mpg))
%>% summarise(mean = mean(wt))

Next write the function template & identify quoted arguments

```
grouped_mean <- function(df, group_var, summary_var) {  
  
  df %>%  
    group_by(group_var) %>%  
    summarise(mean = mean(summary_var))  
}
```

New: Wrap every quoted argument in enquo()

```
grouped_mean <- function(df, group_var, summary_var) {  
  group_var <- enquo(group_var)  
  summary_var <- enquo(summary_var)  
  
  df %>%  
    group_by(group_var) %>%  
    summarise(mean = mean(summary_var))  
}
```

New: And then unquote with !!

```
grouped_mean <- function(df, group_var, summary_var) {  
  group_var <- enquo(group_var)  
  summary_var <- enquo(summary_var)  
  
  df %>%  
    group_by(!!group_var) %>%  
    summarise(mean = mean(!!summary_var))  
}
```

What happens when you call grouped_mean()?

```
grouped_mean(mtcars, cyl, mpg)
```

```
grouped_mean <- function(df, group_var, summary_var) {  
  group_var <- enquo(group_var)  
  summary_var <- enquo(summary_var)  
  
  df %>%  
    group_by(!!group_var) %>%  
    summarise(mean = mean(!!summary_var))  
}
```

What happens when you call grouped_mean()?

```
grouped_mean(mtcars, cyl, mpg)
```

```
grouped_mean <- function(df, group_var, summary_var) {  
  group_var <- quo(cyl)  
  summary_var <- quo(mpg)  
  
  df %>%  
    group_by(!!group_var) %>%  
    summarise(mean = mean(!!summary_var))  
}
```


What happens when you call grouped_mean()?

```
grouped_mean(mtcars, cyl, mpg)
```

```
grouped_mean <- function(df, group_var, summary_var) {  
  group_var <- quo(cyl)  
  summary_var <- quo(mpg)  
  
  df %>%  
    group_by(cyl) %>%  
    summarise(mean = mean(mpg))  
}
```

Reduce the duplication here

```
mtcars %>% summarise(avg = mean(cyl), sd = sd(cyl), n = sum(!is.na(cyl)))  
mtcars %>% summarise(avg = mean(mpg), sd = sd(mpg), n = sum(!is.na(mpg)))  
mtcars %>% summarise(avg = mean(displacement), sd = sd(displacement), n = sum(!is.na(displacement)))
```

Reduce the duplication here

```
mtcars %>% summarise(avg = mean(cyl), sd = sd(cyl), n = sum(!is.na(cyl)))  
mtcars %>% summarise(avg = mean(mpg), sd = sd(mpg), n = sum(!is.na(mpg)))  
mtcars %>% summarise(avg = mean(displacement), sd = sd(displacement), n = sum(!is.na(displacement)))
```

New: Identify quoted vs. evaluated arguments

```
mtcars %>% summarise(avg = mean(cyl), sd = sd(cyl), n = sum(!is.na(cyl)))  
mtcars %>% summarise(avg = mean(mpg), sd = sd(mpg), n = sum(!is.na(mpg)))  
mtcars %>% summarise(avg = mean(displ), sd = sd(displ), n = sum(!is.na(displ)))
```

Then identify the parts that could change

```
mtcars %>% summarise(avg = mean(cyl), sd = sd(cyl), n = sum(!is.na(cyl)))  
mtcars %>% summarise(avg = mean(mpg), sd = sd(mpg), n = sum(!is.na(mpg)))  
mtcars %>% summarise(avg = mean(displ), sd = sd(displ), n = sum(!is.na(displ)))
```

These become the function arguments

mtcars
mtcars
mtcars

```
%>% summarise(avg = mean(cyl), sd = sd(cyl), n = sum(!is.na(cyl)))  
%>% summarise(avg = mean(mpg), sd = sd(mpg), n = sum(!is.na(mpg)))  
%>% summarise(avg = mean(displ), sd = sd(displ), n = sum(!is.na(displ)))
```

df

var

var

var

Next write the function template & identify quoted arguments

```
summarise_column <- function(df, var){
```

```
  df %>%
```

```
    summarise(avg = mean(var),
```

```
              sd = sd(var),
```

```
              n = sum(!is.na(var)))
```

```
}
```

New: Wrap every quoted argument in enquo()

```
summarise_column <- function(df, var){  
  
  var <- enquo(var)  
  
  df %>%  
    summarise(avg = mean(var),  
              sd = sd(var),  
              n = sum(!is.na(var)))  
}
```


New: And then unquote with !!

```
summarise_column <- function(df, var){  
  
  var <- enquo(var)  
  
  df %>%  
    summarise(avg = mean(!!var),  
              sd = sd(!!var),  
              n = sum(!is.na(!!var)))  
}
```

Try it out!

```
summarise_column <- function(df, var){  
  
  var <- enquos(var)  
  
  df %>%  
    summarise(avg = mean(!!var),  
              sd = sd(!!var),  
              n = sum(!is.na(!!var)))  
}
```

```
summarise_column(mtcars, cyl)  
#           avg           sd    n  
# 1  6.1875  1.785922  32
```

Plays nicely with other tools

```
# Fits in a dplyr pipeline
```

```
mtcars %>%
```

```
  group_by(cyl) %>%
```

```
  summarise_column(mpg)
```

```
# Use purrr to iterate
```

```
my_cols <- list(quo(cyl), quo(mpg), quo(displacement))
```

```
map(my_cols, summarise_column, df = mtcars)
```

Plays nicely with other tools

```
# Fits in a dplyr pipeline
```

```
mtcars %>%
```

```
  group_by(cyl) %>%
```

```
  summarise_column(mpg)
```

```
# Use purrr to iterate
```

```
my_cols <- quos(cyl, mpg, disp)
```

```
map_dfr(my_cols, summarise_column, df = mtcars)
```

Plural version for a list

Is it worth it?

It saves a lot of typing

```
filter(diamonds, x > 0 & y > 0 & z > 0)
```

vs

```
diamonds[  
  diamonds$x > 0 &  
  diamonds$y > 0 &  
  diamonds$z > 0,  
]
```

It saves a lot of typing

```
filter(diamonds, x > 0 & y > 0 & z > 0)
```

vs

```
diamonds[  
  diamonds[["x"]] > 0 &  
  diamonds[["y"]] > 0 &  
  diamonds[["z"]] > 0,  
]
```

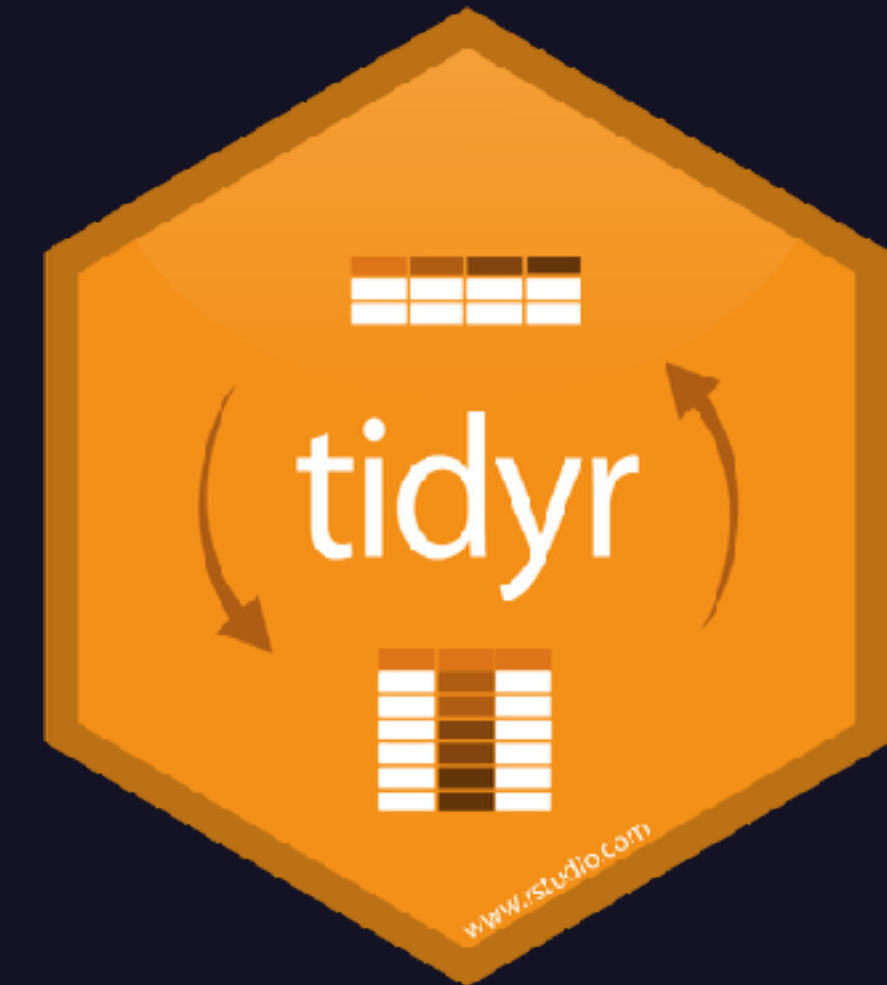
And makes it possible to translate to other languages

```
con <- DBI::dbConnect(RSQLite::SQLite(), filename = ":memory:")  
mtcars_db <- copy_to(con, mtcars)
```

```
mtcars_db %>%  
  filter(cyl > 2) %>%  
  select(mpg:hp) %>%  
  head(10) %>%  
  show_query()
```

```
#> SELECT `mpg`, `cyl`, `disp`, `hp`  
#> FROM `mtcars`  
#> WHERE (`cyl` > 2.0)  
#> LIMIT 10
```


Tidy evaluation = principled NSE

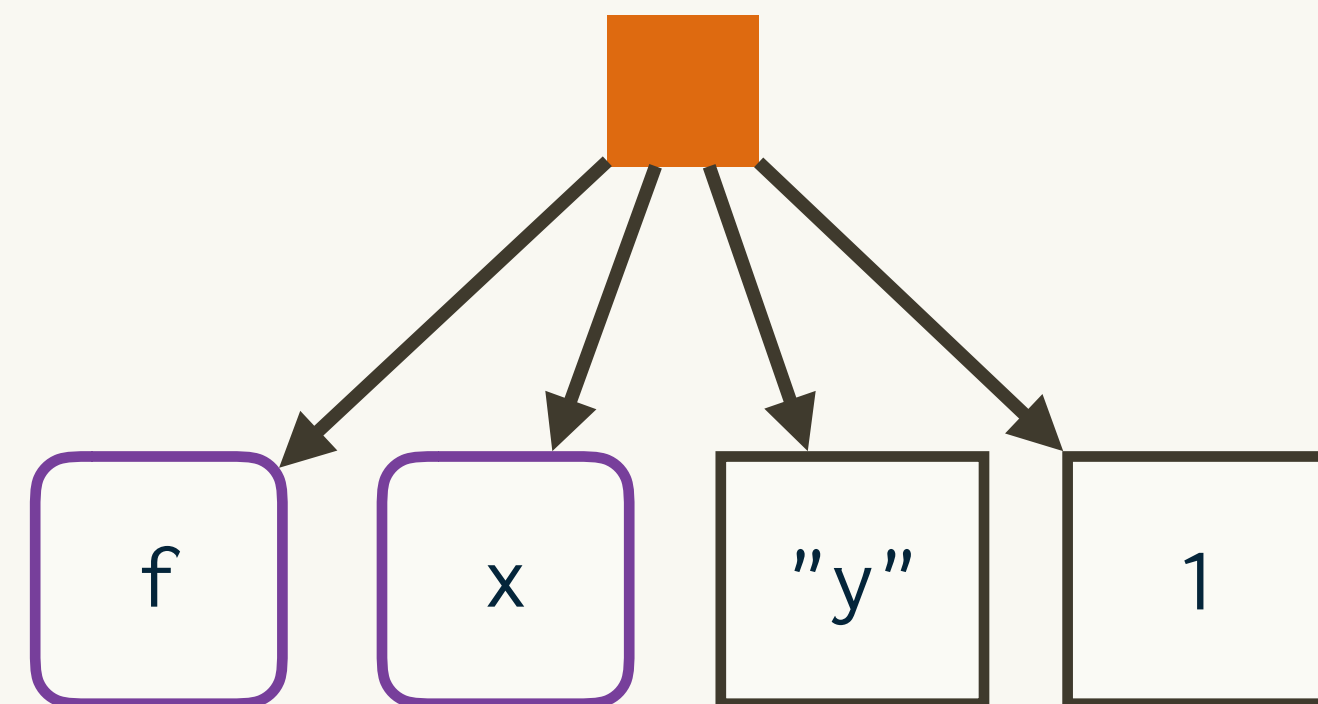


Now for some ~~game~~ theory

1. R code is a tree
2. Unquoting builds trees
3. Environments map
names to values

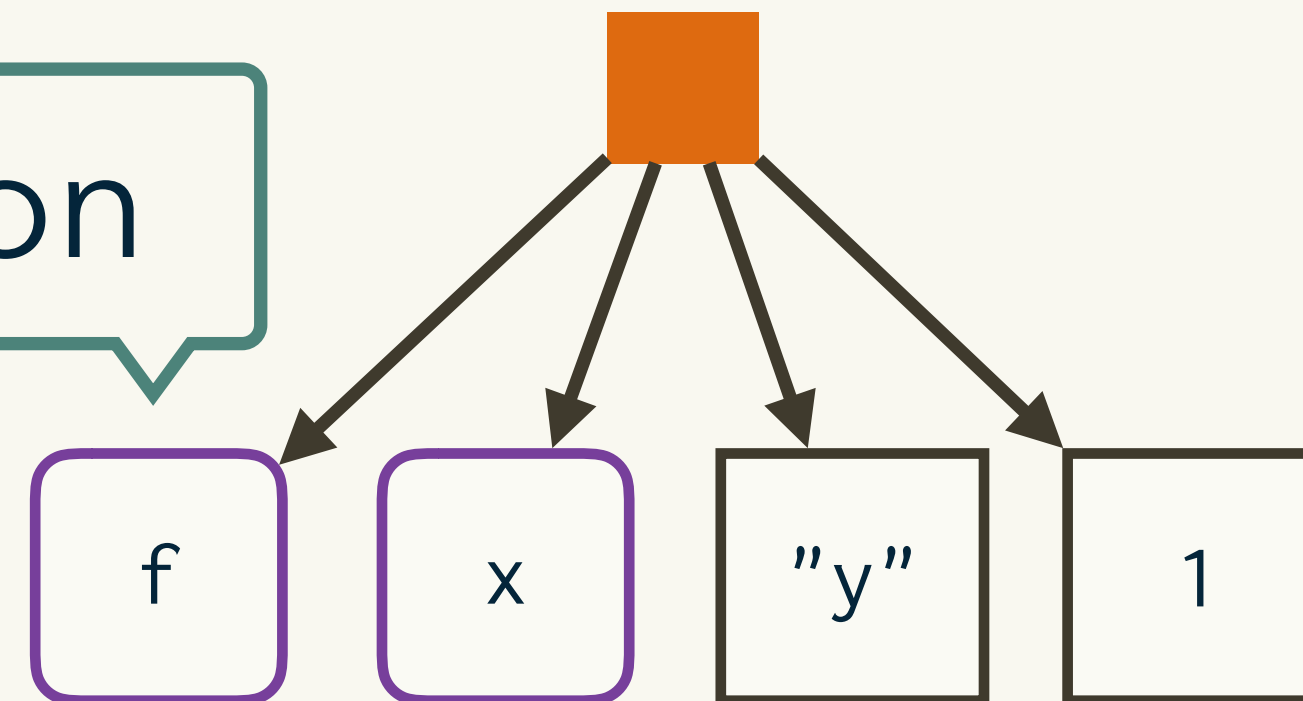
R code is a tree

$f(x, "y", 1)$



A function call

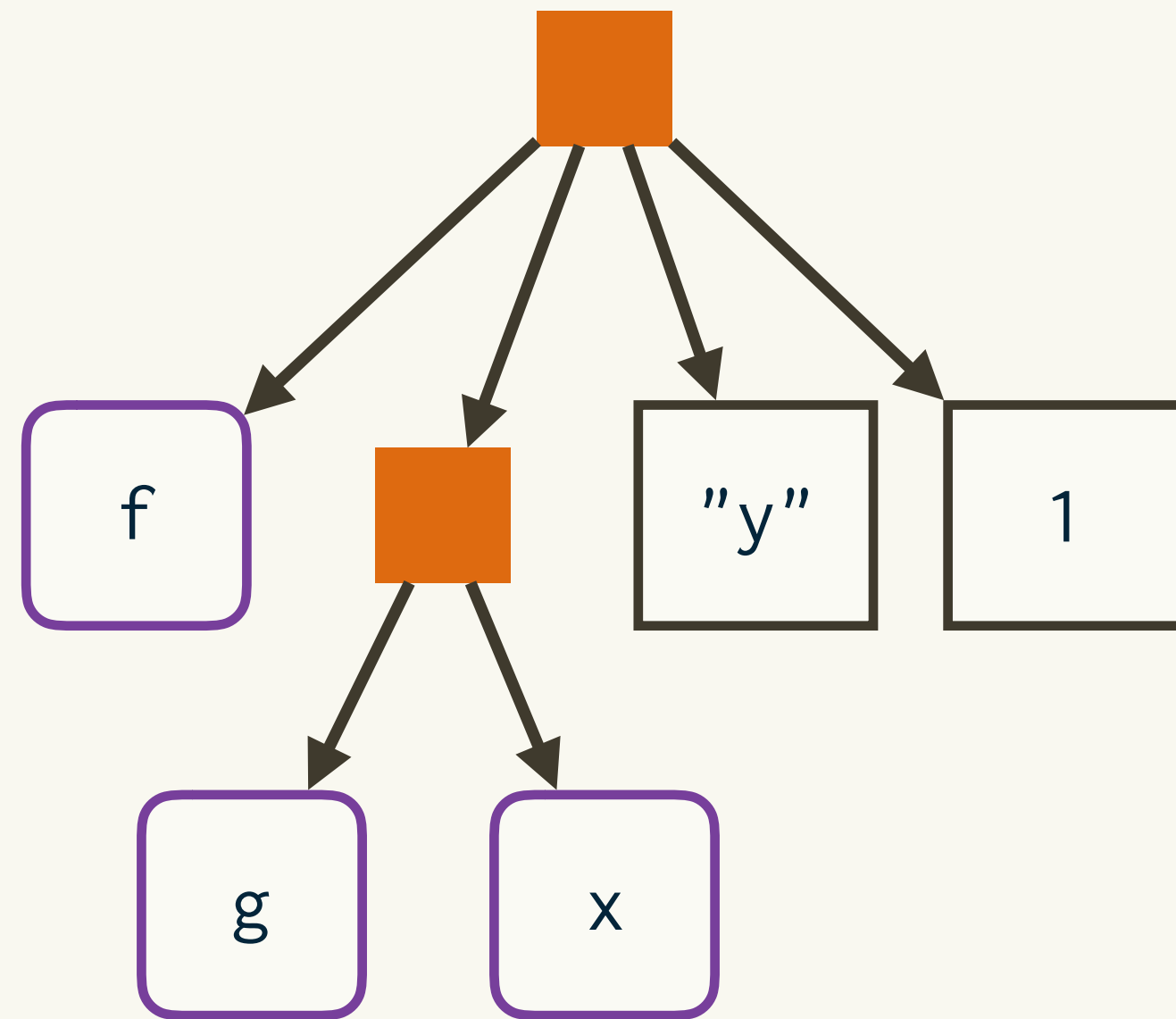
First child = function



Other children = arguments

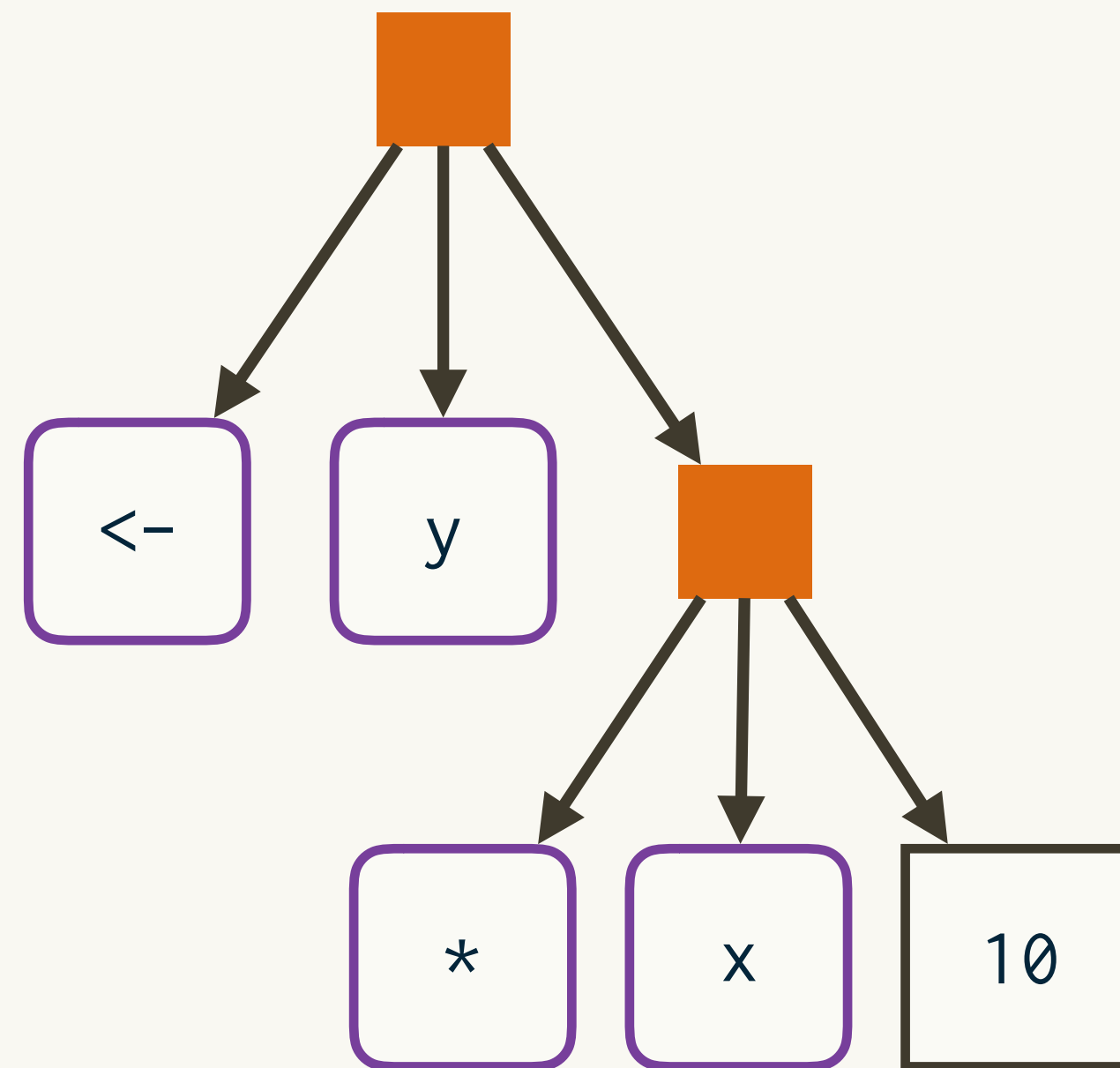
More complex calls have multiple levels

`f(g(x), "y", 1)`



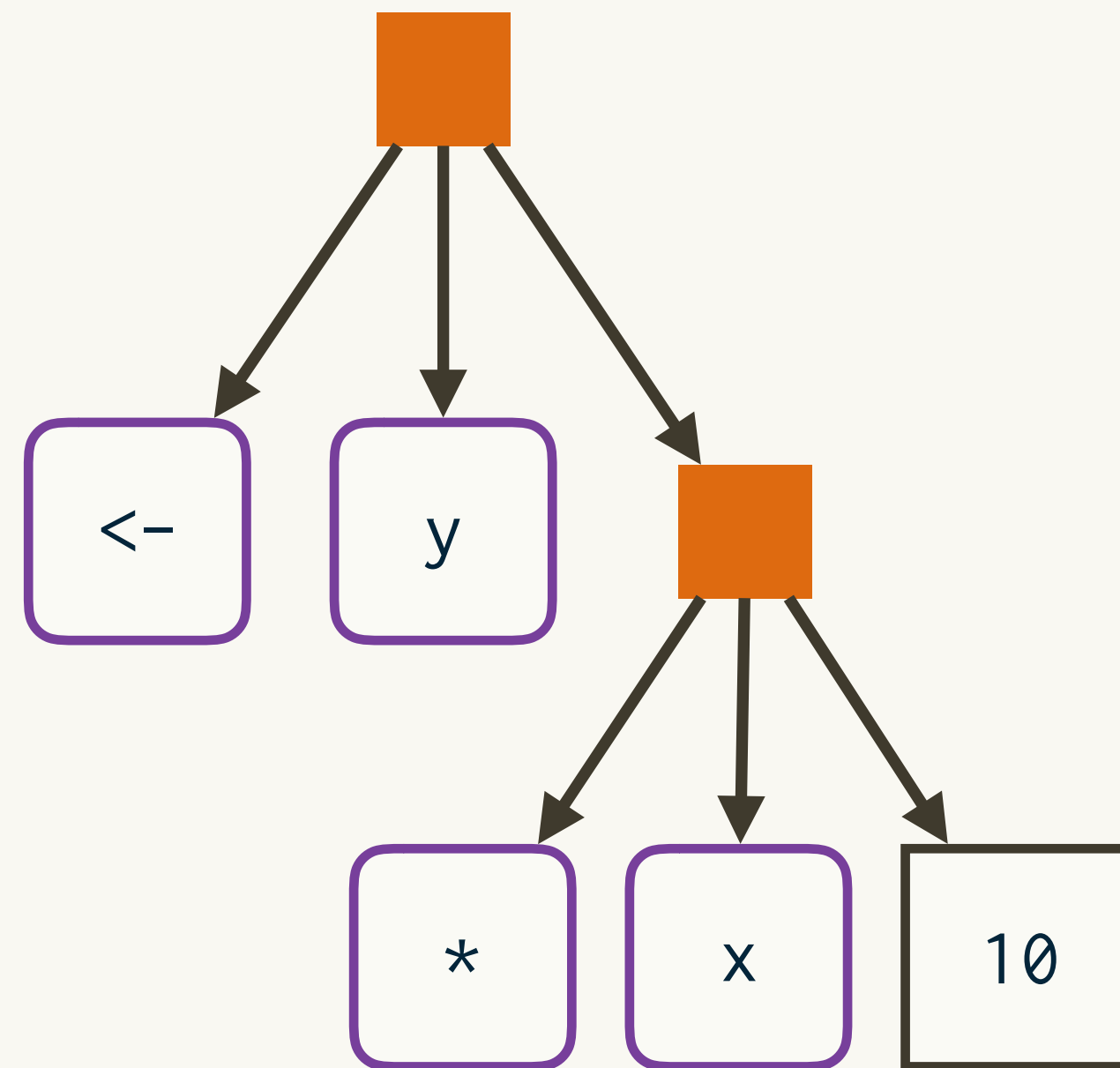
Every expression has a tree

`y <- x * 10`



Because every expression can be rewritten

```
'<-' (y, '*'(x, 10))
```



You can see this yourself with `lobstr::ast()`

```
> lobstr::ast(if(x > 5) y + 1)
```

■ — `if`

├─ ■ — `>`

| └─ x

| └─ 5

└─ ■ — `+`

└─ y

└─ 1

Your turn

```
library(lobstr)
```

```
# Compare to my hand drawn diagrams
```

```
ast(f(x, "y", 1))
```

```
ast(y <- x * 10)
```

```
# What does this tree tell you?
```

```
ast(function(x, y) {
```

```
  if (x > y) {
```

```
    x
```

```
  } else {
```

```
    y
```

```
  }
```

```
})
```

What isn't in the AST?

```
ast(1          + 2)
ast({
  1
  # comment
  2
})
```

Unquoting builds trees

expr() captures your expression

```
library(rlang)
```

```
expr(y + 1)
```

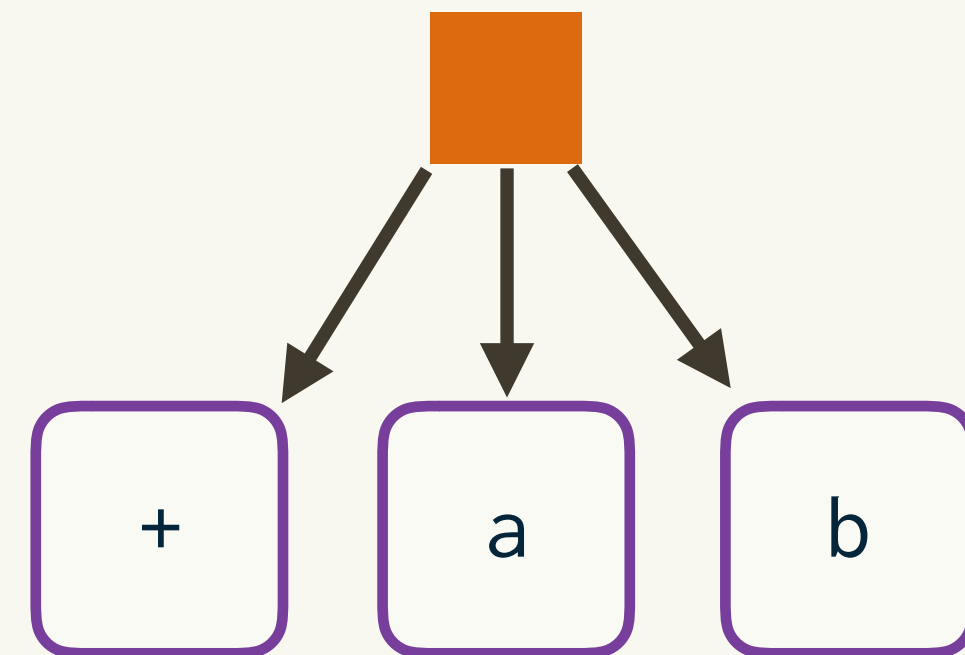
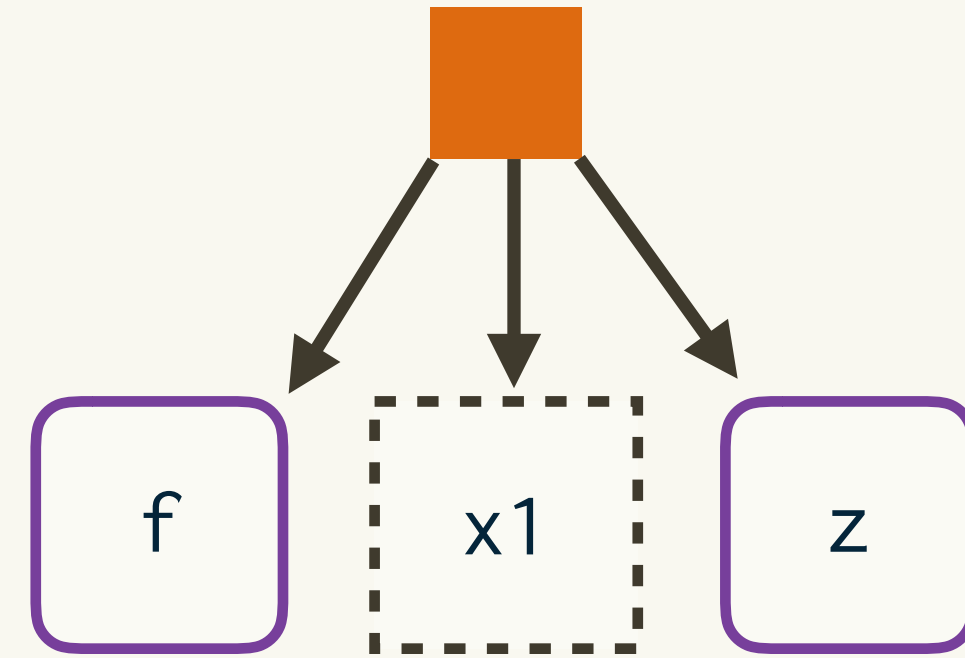
```
#> y + 1
```

Unquoting allows you to build your own trees

```
x1 <- expr(a + b)
expr(f(!!x1, z))
#> f(a + b, z)
```

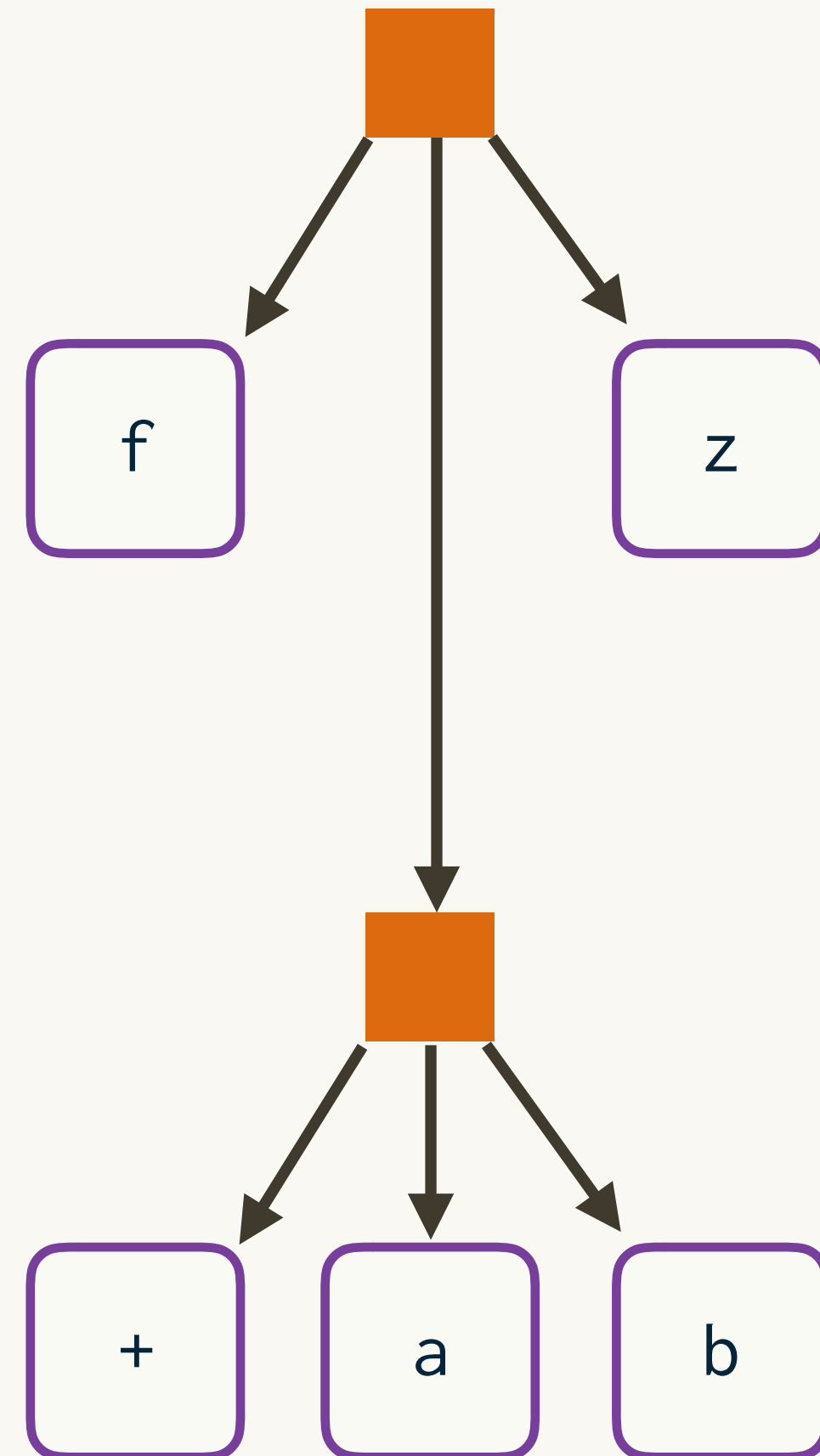
```
# !! is called the unquoting operator
# And is pronounced bang-bang
```

`expr(f(!!x1, z))`

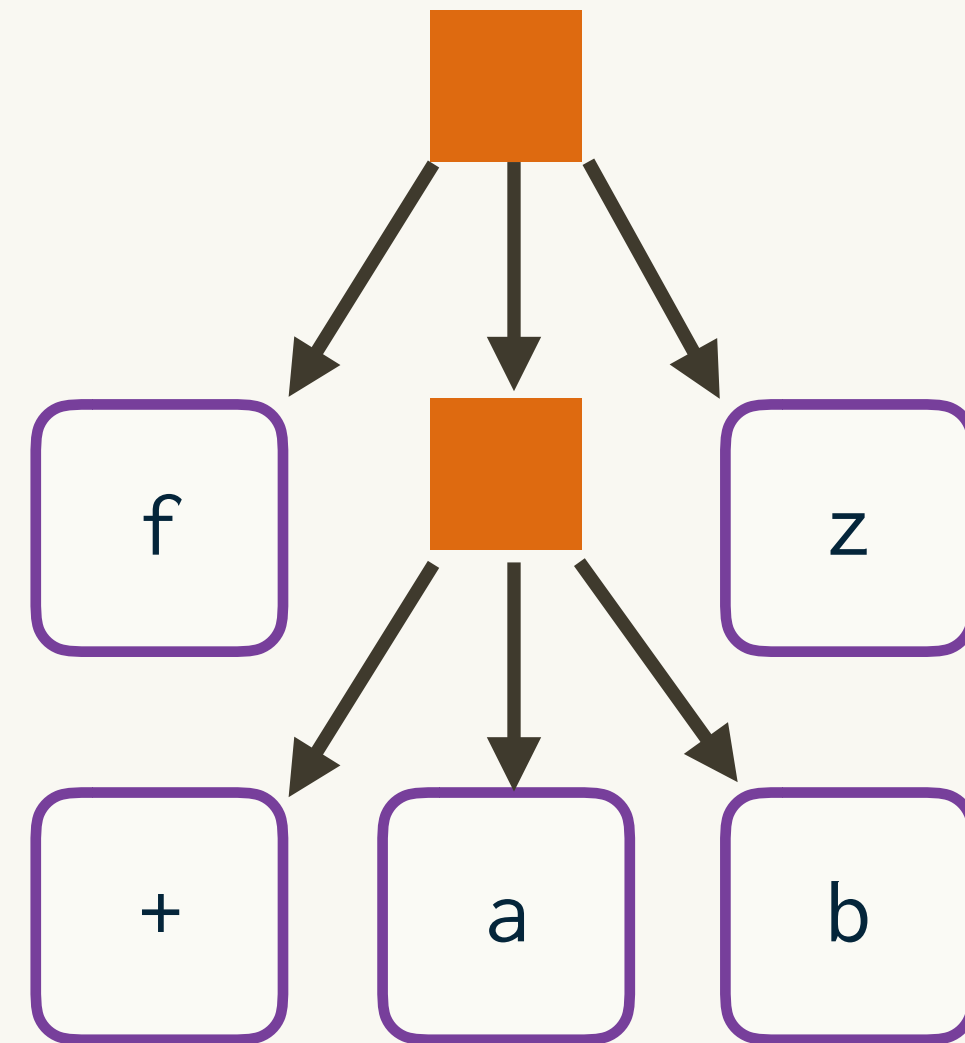


`x1 <- expr(a + b)`

`expr(f(!!x1, z))`



expr(f(!!x1, z))



Predict the contents of these expressions

```
ex1 <- expr(x + y)
```

```
ex2 <- expr (!!ex1 + z)
```

```
ex3 <- expr(1 / !!ex1)
```

Predict the contents of these expressions

```
ex1 <- expr(x + y)
```

```
# x + y
```

```
ex2 <- expr (!!ex1 + z)
```

```
ex3 <- expr(1 / !!ex1)
```

Predict the contents of these expressions

```
ex1 <- expr(x + y)
```

```
# x + y
```

```
ex2 <- expr (!!ex1 + z)
```

```
# x + y + z
```

```
ex3 <- expr(1 / !!ex1)
```

Predict the contents of these expressions

```
ex1 <- expr(x + y)
```

```
# x + y
```

```
ex2 <- expr(!ex1 + z)
```

```
# x + y + z
```

```
ex3 <- expr(1 / !ex1)
```

```
# 1 / (x + y)
```

```
# Not 1 / x + y
```

Recreate these expressions

```
# Using:
```

```
var <- expr(cyl)
```

```
df <- expr(mtcars)
```

```
f <- expr(foo)
```

```
# Recreate these expressions:
```

```
# mean(log(cyl))
```

```
# lm(cyl ~ 1, data = mtcars)
```

```
# foo(x = cyl)
```

Recreate these expressions

```
# Using:
```

```
var <- expr(cyl)
```

```
df <- expr(mtcars)
```

```
f <- expr(foo)
```

```
# Recreate these expressions:
```

```
expr(mean(log(!!var)))
```

```
# mean(log(cyl))
```

```
expr(lm(!!var ~ 1, data = !!df))
```

```
# lm(cyl ~ 1, data = mtcars)
```

```
expr((!!f)(x = !!var))
```

```
# foo(x = cyl)
```

enexpr() lets you capture user expressions

```
# expr() quotes your expression
```

```
f1 <- function(z) expr(z)
```

```
f1(a + b)
```

```
#> z
```

```
# enexpr() quotes user's expression
```

```
f2 <- function(z) enexpr(z)
```

```
f2(x + y)
```

```
#> x + y
```


Environments map
names to values

Capturing just expression isn't enough

```
add_y <- function(df, var) {  
  n <- 10  
  var <- enexpr(var)  
  mutate(df, y = !!var)  
}
```

```
df <- tibble(x = 1)  
n <- 100  
add_y(df, x + n)
```

```
#>           x      y  
#> 1  1.00    11
```

Capturing just expression isn't enough

```
add_y <- function(df, var) {  
  n <- 10  
  var <- enexpr(var)  
  mutate(df, y = !!var)  
}
```

```
df <- tibble(x = 1)  
n <- 100  
add_y(df, x + n)
```

```
#>           x      y  
#> 1  1.00    11
```

Capturing just expression isn't enough

```
add_y <- function(df, var) {  
  n <- 10  
  var <- enexpr(var)  
  mutate(df, y = !!var)  
}
```

```
df <- tibble(x = 1)  
n <- 100  
add_y(df, x + n)
```

```
#>           x      y  
#> 1  1.00    11
```

Capturing just expression isn't enough

```
add_y <- function(df, var) {  
  n <- 10  
  var <- expr(x + n)  
  mutate(df, y = !!var)  
}
```

```
df <- tibble(x = 1)  
n <- 100  
add_y(df, x + n)
```

```
#>           x      y  
#> 1  1.00    11
```

Capturing just expression isn't enough

```
add_y <- function(df, var) {  
  n <- 10  
  var <- expr(x + n)  
  mutate(df, y = x + n)  
}
```

```
df <- tibble(x = 1)  
n <- 100  
add_y(df, x + n)
```

```
#>           x      y  
#> 1  1.00    11
```

quo() captures expression and environment

```
# quo() quotes your expression
```

```
f1 <- function(z) quo(z)
```

```
f1(a + b)
```

```
#> <quosure>
```

```
#>  expr: ^z
```

```
#>  env:  0x10d3b9308
```

```
# enquo() quotes user's expression
```

```
f2 <- function(z) enquo(z)
```

```
f2(x + y)
```

```
#> <quosure>
```

```
#>  expr: ^x + y
```

```
#>  env:  0x10d3b9309
```

	Function author	Function user
Expression	<code>expr(x)</code>	<code>enenxpr(x)</code>
Expression + environment	<code>quo(x)</code>	<code>enquo(x)</code>

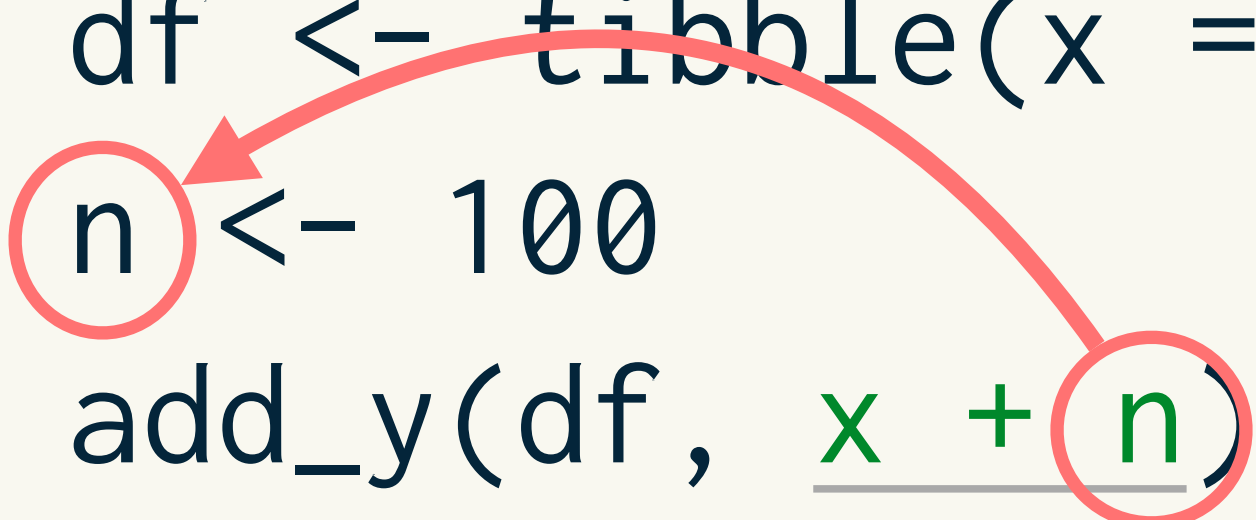
Think enrich


```
add_y <- function(df, var) {  
  n <- 10  
  var <- enquos(var)  
  mutate(df, y = !!var)  
}
```

```
df <- tibble(x = 1)  
n <- 100  
add_y(df, x + n)
```

```
#>           x      y  
#> 1    1.00   101
```

```
add_y <- function(df, var) {  
  n <- 10  
  var <- enquos(var)  
  mutate(df, y = !!var)  
}
```

```
df <- tibble(x = 1)  
  
n <- 100  
add_y(df, x + n)  
#>           x      y  
#> 1  1.00  101
```

Key pattern is to quote and unquote

```
df <- data.frame(x = 1:5, y = 5:1, z = 0:4) ← Typo
```

```
filter(df, abs(x) > 1e-3)
```

```
filter(df, abs(y) > 1e-3)
```

```
filter(df, abs(z) > 1e-3)
```

```
my_filter <- function(df, var) {
```

```
  var <- enquos(var)
```

```
  filter(df, abs(!!var) > 1e-3)
```

```
}
```

```
my_filter(df, x)
```

Quote

Unquote

Case Study

Switch to project
case_study

Your Turn

Open 03-report.R

Adapt summarise_weekly() to remove dependence on exact column names.

E.g. should be able to specify as arguments the columns for the: date, variable to be summarised, and a grouping variable.

Your Turn

```
summarise_weekly <- function(data){  
  data %>%  
    mutate(week = lubridate::week(date)) %>%  
    group_by(type, week) %>%  
    summarise(  
      date = first(date),  
      n = sum(!is.na(n_sales)),  
      mean = mean(n_sales, na.rm = TRUE))  
}
```

Your Turn

```
summarise_weekly <- function(data, date, var, group){  
  date <- enquo(date)  
  var <- enquo(var)  
  group <- enquo(group)  
  
  data %>%  
    mutate(week = lubridate::week(!date)) %>%  
    group_by(!group, week) %>%  
    summarise(  
      date = first(!date),  
      n = sum(!is.na(!var)),  
      mean = mean(!var, na.rm = TRUE))  
}  
all_states_weekly <- map(all_states, summarise_weekly,  
  date = date, var = n_sales, group = type)
```


How could `plot_weekly()` be improved?

```
plot_weekly <- function(data, title){  
  data %>%  
    ggplot(aes(date, mean, color = type)) +  
    geom_point(size = 3) +  
    geom_line(alpha = 0.5) +  
    labs(title = title,  
         x = "Week starting",  
         y = "Average number of sales per day") +  
    theme_bw() +  
    scale_color_brewer(type = "qual")  
}
```

How could `plot_weekly()` be improved?

```
plot_ts <- function(data, x, y, group, title = ""){
  x <- enquo(x)
  y <- enquo(y)
  group <- enquo(group)

  data %>%
    ggplot(aes(!!x, !!y, color = !!group)) +
    geom_point(size = 3) +
    geom_line(alpha = 0.5) +
    labs(title = title,
         x = "Week starting",
         y = "Average number of sales per day") +
    theme_bw() +
    scale_color_brewer(type = "qual")
}

all_states_plots <- map2(all_states_weekly, states_long_names, plot_ts,
  x = date, y = mean, group = type)
```

Hopefully, the end result is code that is easier to

04-report.R

```
library(tidyverse)
library(fs)
```

```
source("functions.R")
```

```
# Get file names and paths to data
files <- dir("data") %>% path_ext_remove()
file_paths <- path("data", files, ext = "csv")
states <- str_sub(files, 1, 2)
states_long_names <- c(
  "OR" = "Oregon",
  "BC" = "British Columbia",
  "WA" = "Washington")[states]
```

```
# Check data isn't too old -----
ages <- check_not_outdated(file_paths)
```

```
# Import data -----
all_states <- map(file_paths, read_csv)
```

```
# Summarise by week -----
all_states_weekly <- map(all_states, summarise_weekly,
  date = date, var = n_sales, group = type)
```

```
# Plot weekly summary -----
all_states_plots <- map2(all_states_weekly, states_long_names, plot_ts,
  x = date, y = mean, group = type)
```

1. understand,
2. maintain, and
3. extend

Once you embrace this workflow use tibbles + dplyr

sales

```
# # A tibble: 3 x 9
#   files          file_paths      states states_long_names ages
#   <fs::path>    <fs::path>      <chr>   <chr>                <time>
# 1 BC_2018-07-14 data/BC_2018-07-14.csv BC       British Columbia    6
# 2 OR_2018-07-15 data/OR_2018-07-15.csv OR        Oregon              5
# 3 WA_2018-07-18 data/WA_2018-07-18.csv WA        Washington          2
# # ... with 4 more variables: data <list>, weekly_n_sales <list>,
# #   n_sales_weekly_plot <list>, image_paths <fs::path>
```

See 05-report.R

Challenge

Extend the analysis to create the weekly plots for all three variables: `n_sales`, `total_dollar_amount`, `n_existing_customer`.

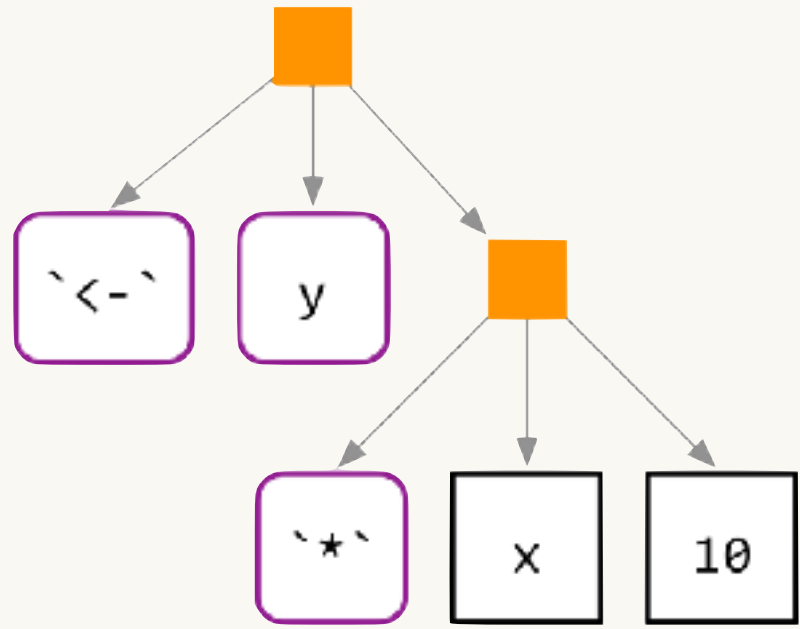
Learning more

Theory

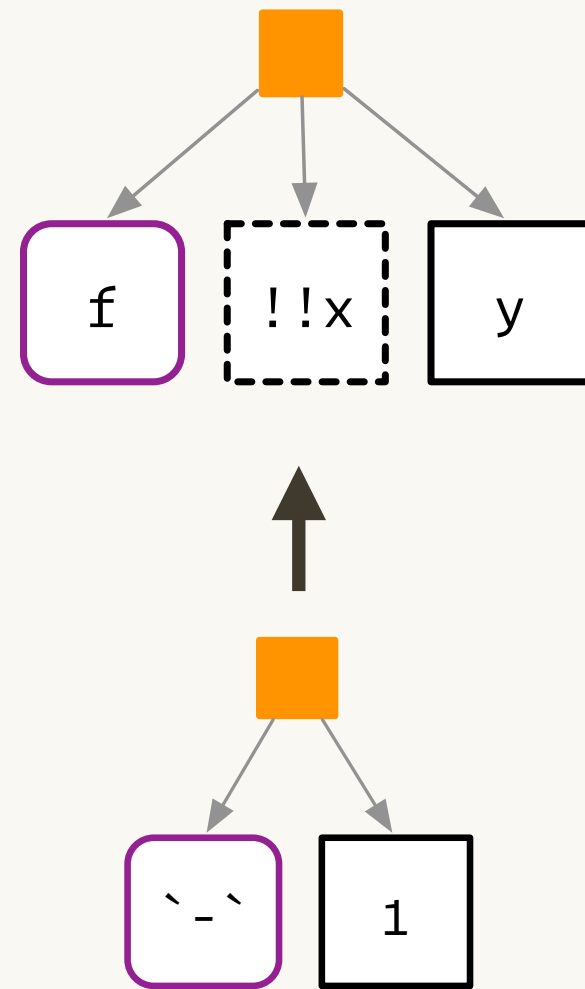
<https://youtu.be/nERXS3ssntw>

Practice

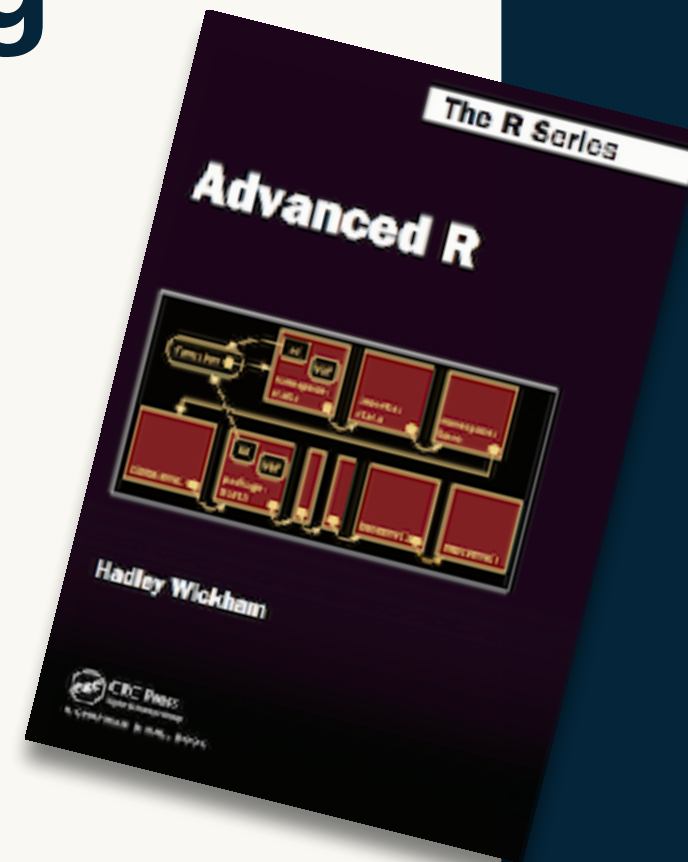
Code is a tree



enquo()



Build trees with unquoting



<https://adv-r.hadley.nz/expressions.html>

<https://adv-r.hadley.nz/quasiquotation.html>

<https://adv-r.hadley.nz/evaluation.html>



Adapted from *Tidy Tools* by Hadley Wickham

This work is licensed as
Creative Commons
Attribution-ShareAlike 4.0
International

To view a copy of this license, visit
[https://creativecommons.org/licenses/by-sa/
4.0/](https://creativecommons.org/licenses/by-sa/4.0/)