An exploration of NSE and tidyeval

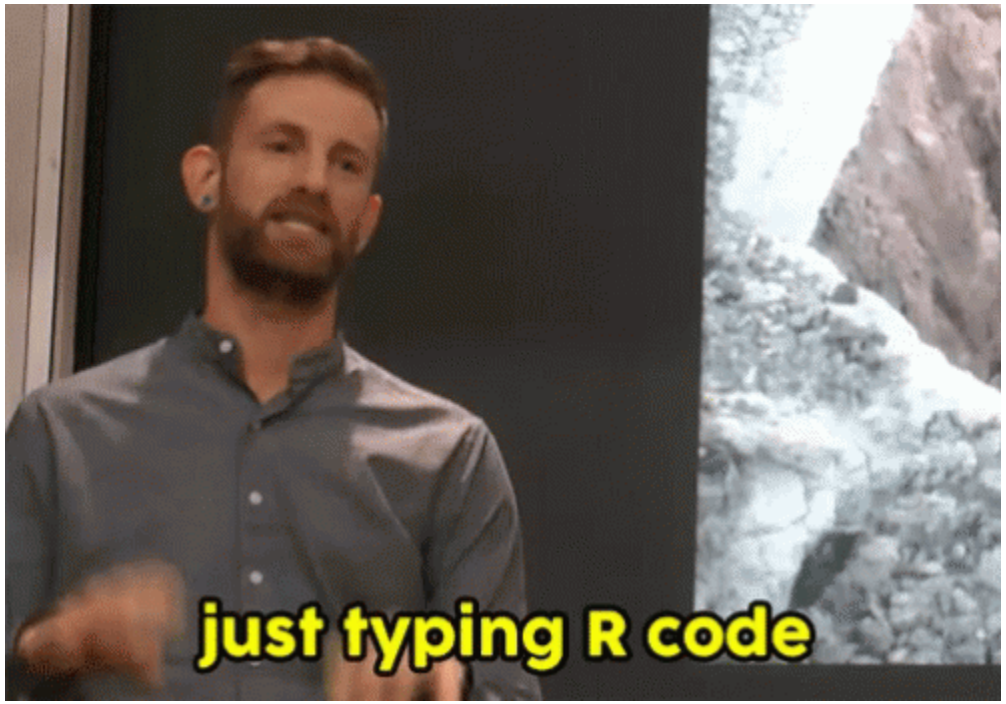
Edwin Thoen

2018-09-01

funda

whoami

whoami



whoami

Data scientist @ funda

Applied Statistics

thatssorandom.com

@edwin_thoen

CRAN: padr, GGally, recipes

What is this talk about?

Lets show some hands

Who:

Lets show some hands

Who:

- is very experienced in using NSE?

Lets show some hands

Who:

- is very experienced in using NSE?
- applies it now and then?

Lets show some hands

Who:

- is very experienced in using NSE?
- applies it now and then?
- only knows it at face value?

Lets show some hands

Who:

- is very experienced in using NSE?
- applies it now and then?
- only knows it at face value?
- is not really sure what it is?

Why do we love R so much for data-analysis?

Why do we love R so much for data-analysis?

We don't have to use R when using R!

```
mtcars$cyl_drat <- mtcars$cyl + mtcars$drat
```

Why do we love R so much for data-analysis?

We don't have to use R when using R!

```
mtcars$cyl_drat <- mtcars$cyl + mtcars$drat
```

Instead of this, we can do

```
library(dplyr)
mtcars <- mtcars %>% mutate(cyl_drat = cyl + drat)
```

or

```
mtcars_dt <- data.table::as.data.table(mtcars)
mtcars_dt[, cyl_drat := cyl + drat]
```

We all use NSE from the first day

When you started using R, did you mix up?

```
install.packages("padr")
```

and

```
library(padr)
```

Domain Specific Languages

Apparantly, things that ought not to work, are working.

```
subset(mtcars, cyl == 6)

ggplot2::ggplot(mtcars, aes(mpg, drat)) +
  geom_point()

data.table::as.data.table(mtcars)[ ,mean(mpg), by = cyl]
```

Why should you care about how NSE works?

Why should you care about how NSE works?

- Because you are a geek and can't stand not knowing :)


Het Fietscafé

Why should you care about how NSE works?

- Because you are a geek and can't stand not knowing :)
- Because most R users slip into tool design sooner or later.

So what is Standard?

What's in a NAME

By creating a variable we bind a **value** to a **name**.

```
my_val <- 123
```

Binding happens in an environment, in this case the global.

What's in a NAME

By creating a variable we bind a **value** to a **name**.

```
my_val <- 123
```

Binding happens in an environment, in this case the global.

Just call my name, I'll give you the value:

```
my_val
```

```
## [1] 123
```

This is evaluating the variable name.

Lexical scoping

R starts looking for the value of name in the environment the name is called in.

```
x <- "a variable in the global"
a_func <- function() {
  x <- "a variable in the local"
  x
}
a_func()
```

```
## [1] "a variable in the local"
```

Lexical scoping

When it can't find it locally, move up to the parent environment (where the current env was created).

```
z <- "a variable in the global"
another_func <- function() {
  z
}
another_func()
```

```
## [1] "a variable in the global"
```

Lexical scoping

Error is thrown when the variable can't be found.

```
nobody_loves_me <- function() {
  y
}
nobody_loves_me()
```

```
## Error in nobody_loves_me(): object 'y' not found
```

So this is standard evaluation in R.

Wait for it

Postpone judgement, store variable name in a **name object**.

```
quote(wait_for_it)
```

```
## wait_for_it
```

```
quote(wait_for_it) %>% class()
```

```
## [1] "name"
```

Wait for it

Postpone judgement, store variable name in a **name object**.

```
quote(wait_for_it)
```

```
## wait_for_it
```

```
quote(wait_for_it) %>% class()
```

```
## [1] "name"
```

This is the act of **quoting**, saving a variable name to be evaluated later.

(**name** is also called **symbol**)

Wait for it

Quoted variable names not evaluated. It doesn't matter if they don't exist.

```
quoted_var <- quote(wait_for_it)
quoted_var
```

```
## wait_for_it
```

```
quoted_var %>% class()
```

```
## [1] "name"
```

Wait for it

Look for the value only when we ask to evaluate it.

```
eval(quoted_var)
```

```
## Error in eval(quoted_var): object 'wait_for_it' not found
```

Wait for it

```
wait_for_it <- "I finally have a value"
eval(quoted_var)
```

```
## [1] "I finally have a value"
```

What is the use?

What is the use?

We can evaluate the `name` in a different environment.

Building our own `pull`

```
dplyr::pull(mtcars, cyl)
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

Building our own `pull`

```
diy_pull <- function(x, name) {
  eval(name, envir = x)
}

diy_pull(mtcars, quote(cyl))
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

Building our own `pull`

```
diy_pull <- function(x, name) {
  eval(name, envir = x)
}

diy_pull(mtcars, quote(cyl))
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

A data frame is an environment too. Column names act as variables.


BUT IN DPLYR I NEVER HAVE TO QUOTE MYSELF!
imgflip.com

DSLs take "bares"

```
mtcars %>% select(cyl)
as.data.table(mtcars)[, cyl]
ggplot(mtcars, aes(cyl)) + geom_bar()
```

Why does R not throw an error? There is no `cyl` in the global…

Quoting inside a function

Would this work?

```
diy_pull <- function(x, bare_name) {
  name <- quote(bare_name)
  eval(name, env = x)
}
```

Quoting inside a function

Would this work?

```
diy_pull_2 <- function(x, bare_name) {
  name <- quote(bare_name)
  eval(name, env = x)
}

diy_pull(mtcars, cyl)
```

```
## Error in eval(name, envir = x): object 'cyl' not found
```

Quoting inside a function

`substitute` quotes the argument's content

```
substitute_example <- function(x) {
  substitute(x)
}
substitute_example(cyl)
```

```
## cyl
```

```
substitute_example(cyl) %>% class()
```

```
## [1] "name"
```

Quoting inside a function

```
diy_pull <- function(x, bare_name) {
  name <- substitute(bare_name)
  eval(name, env = x)
}

diy_pull(mtcars, cyl)
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

Not just names

We can quote the following things:

- name: the name of an R object
- call: calling of a function
- pairlist: something from the past you shouldn't bother about
- literal: evaluates to the value itself

"don't be another SQL"

Call

Just like a **name**, a function **call** can be delayed by quoting.

```
my_little_filter <- function(x,
                             call) {
  call_quoted <- substitute(call)
  retain_row  <- eval(call_quoted, envir = x)
  x[retain_row, ]
}

my_little_filter(mtcars, cyl == 4 & gear == 4) %>% head(2)
```

```
##    mpg cyl  disp hp drat   wt  qsec vs am gear carb cyl_drat
## 3 22.8   4 108.0 93 3.85 2.32 18.61  1  1    4    1     7.85
## 8 24.4   4 146.7 62 3.69 3.19 20.00  1  0    4    2     7.69
```

But, but … why?

Lazy, lazy R

Lazy, lazy R

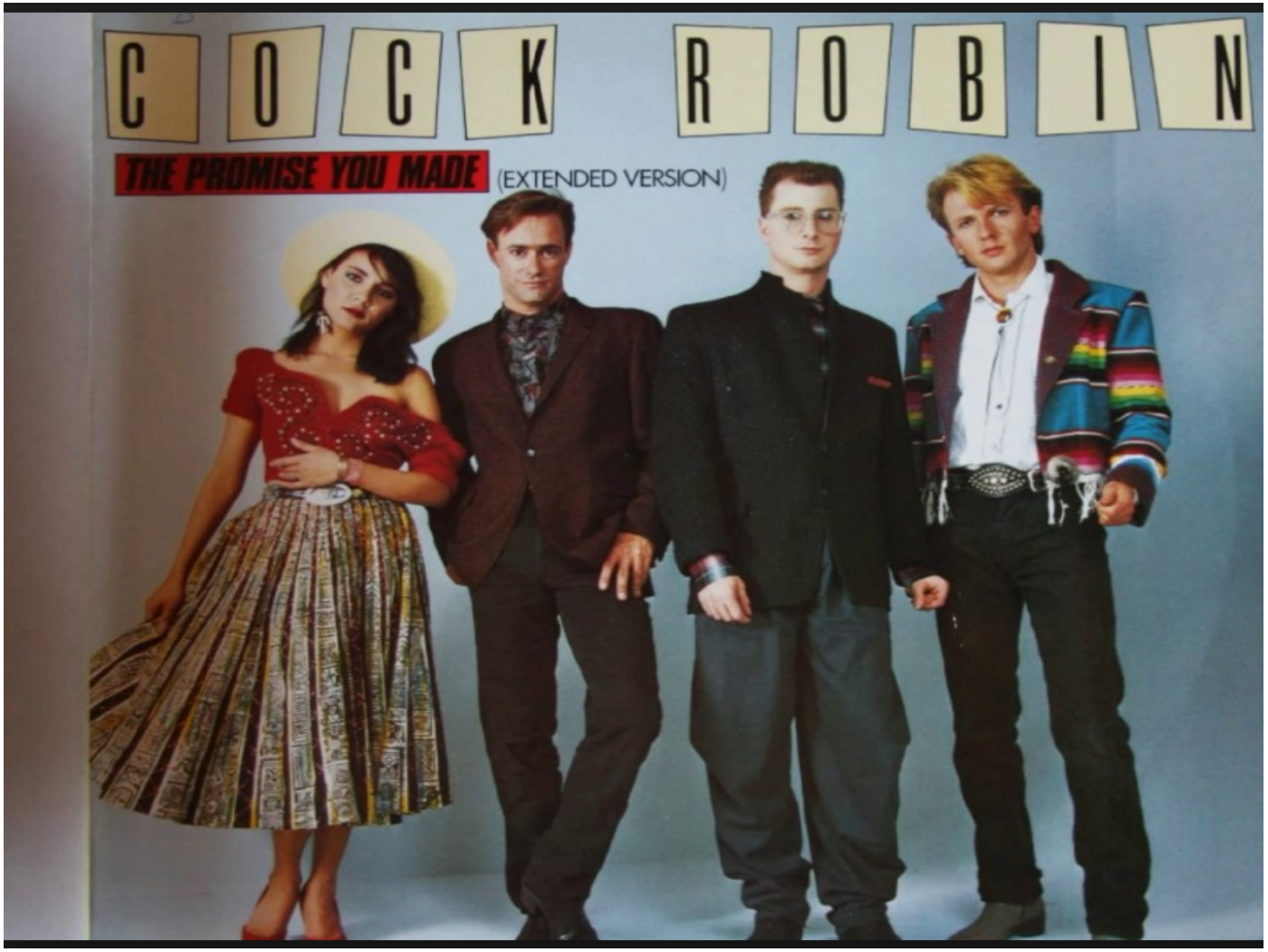
```
koala <- function(x, y) {
  x + 42
}

koala(3)
```

```
## [1] 45
```

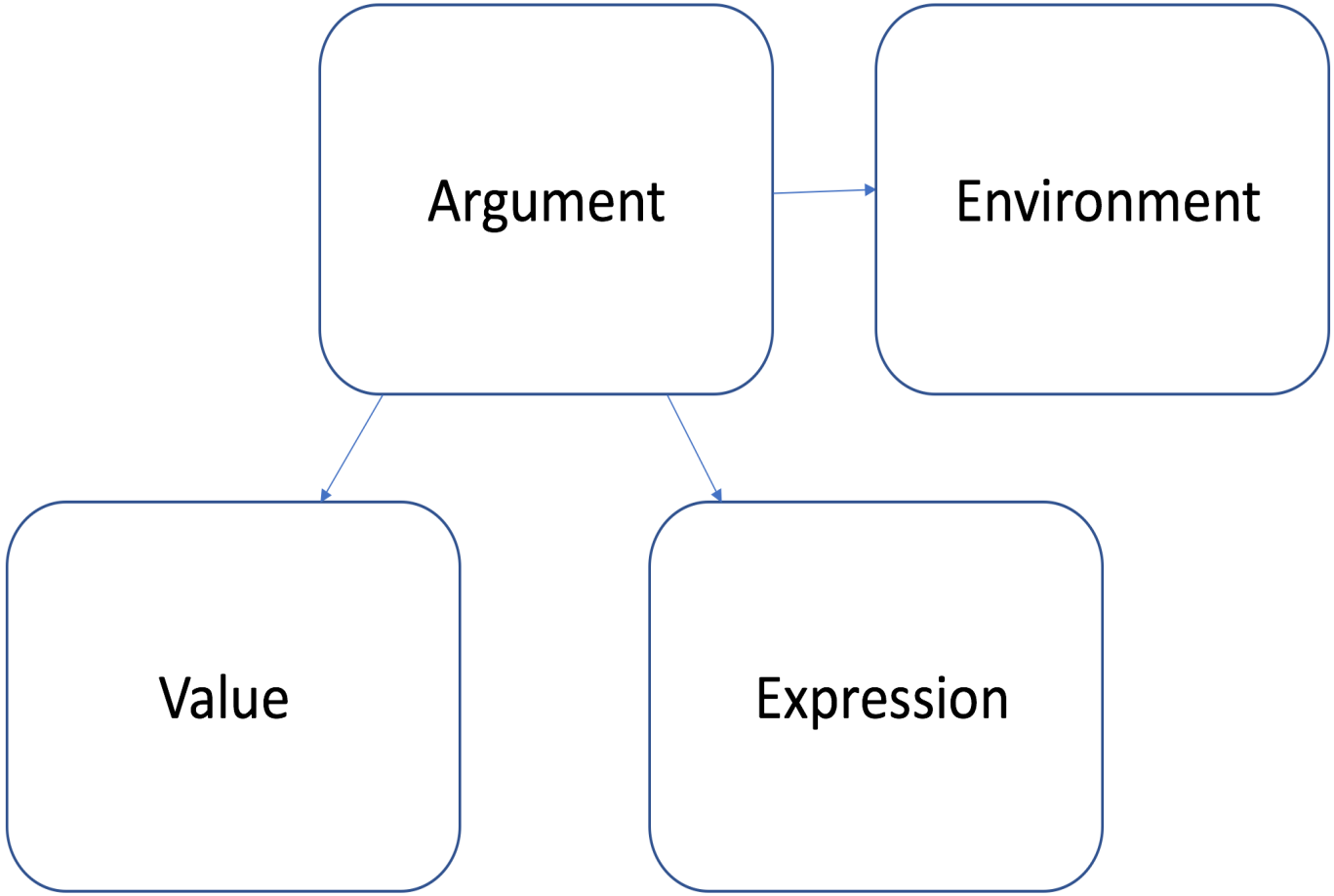
Industrious Python

```
def koala(x, y):
  return(x + 42)
koala(3)
```

```
## TypeError: koala() takes exactly 2 arguments (1 given)
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
```


COCK ROBIN
THE PROMISE YOU MADE
(EXTENDED VERSION)

That's a promise



That's a promise

The value slot is empty at promise creation.

Only when the argument's expression is evaluated in the function, we start looking for it.

That's a promise

The value slot is empty at promise creation.

Only when the argument's expression is evaluated in the function, we start looking for it.

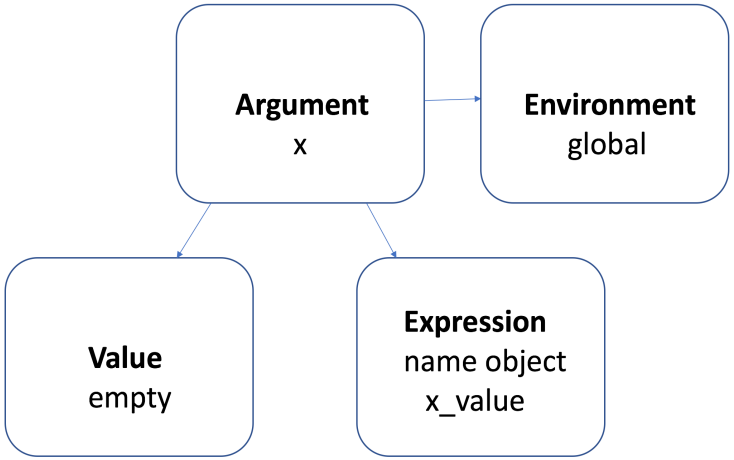
Remember `koala`?

```
koala <- function(x, y) {
  x + 42
}
```

Promising koala

When we call `koala` we create the following promise

```
x_value <- 42
koala(x = x_value)
```



Accessing the expression slot

That's how `subsitute` works!

Accesses the expression in the promise without evaluating it.

```
subs_func <- function(val) {
  vals_expr <- substitute(val)
  deparse(vals_expr)
}
subs_func(anything_goes)
```

```
## [1] "anything_goes"
```

Note that `deparse` coerces the expression to a character. Its inverse is `parse`.

Tidyeval

The `tidyverse` NSE dialect.

```
mtcars %>% select(cyl)
```

We now know that `cyl` gets somehow quoted by `select` and evaluated within the data frame.

But what if we want to wrap tidyverse code in a custom function?

Tidyeval - custom function

This won't work

```
my_tv_func <- function(x, grouping_var) {
  x %>%
    group_by(grouping_var) %>%
    summarise(max_drat = max(drat))
}
my_tv_func(mtcars, cyl)
```

Why?

Tidyeval - custom function

In order to get it to work:

- quote the variable upfront
- unquote again before the argument is swallowed by the tidyverse function
- then tidyverse function can go back and quote it again

Tidyeval - custom function

In order to get it to work:

- quote the variable upfront
- unquote again before the argument is swallowed by the tidyverse function
- then tidyverse function can go back and quote it again

```
my_tv_func <- function(x, grouping_var) {
  x %>%
    group_by(!!grouping_var) %>%
    summarise(max_drat = max(drat))
}
my_tv_func(mtcars, quo(cyl))
```

Tidyeval - custom function

Just like using `substitute` you can quote the arguments value with `enquo`.

```
my_grouping_func <- function(x, grouping_var) {
  grouping_var_q <- enquo(grouping_var)
  x %>%
    group_by(!!grouping_var_q) %>%
    summarise(max_drat = max(drat))
}
my_grouping_func(mtcars, cyl)
```

```
## # A tibble: 3 x 2
##     cyl max_drat
##   <dbl>    <dbl>
## 1     4     4.93
## 2     6     3.92
## 3     8     4.22
```

Summary

Summary

```
my_filter <- function(x, bare_call) {
  call <- substitute(bare_call)
  x[eval(call, envir = x), ]
}

my_filter(mtcars, cyl == 4) %>% head(1)
```

```
##    mpg cyl disp hp drat   wt  qsec vs am gear carb cyl_drat
## 3 22.8   4  108 93 3.85 2.32 18.61  1  1    4    1     7.85
```

- `cyl == 4` on itself is invalid, there is no `cyl` in the global.
- But, R refrains from judgement, stores it in a promise.
- `substitute` gets the expression, which is the quoted call.
- This expression is evaluated within the environment of `x`.
- Here it is completely valid, because there is a `cyl` column.

Things not covered (extensively)

- quasiquotation
- quosures
- environments

Thank You!

edwinthoen@gmail.com

@edwin_thoen

github.com/EdwinTh/satRday

edwinth.github.io/blog/nse

edwinth.github.io/blog/dplyr-recipes