icecream

make debugging a little sweeter

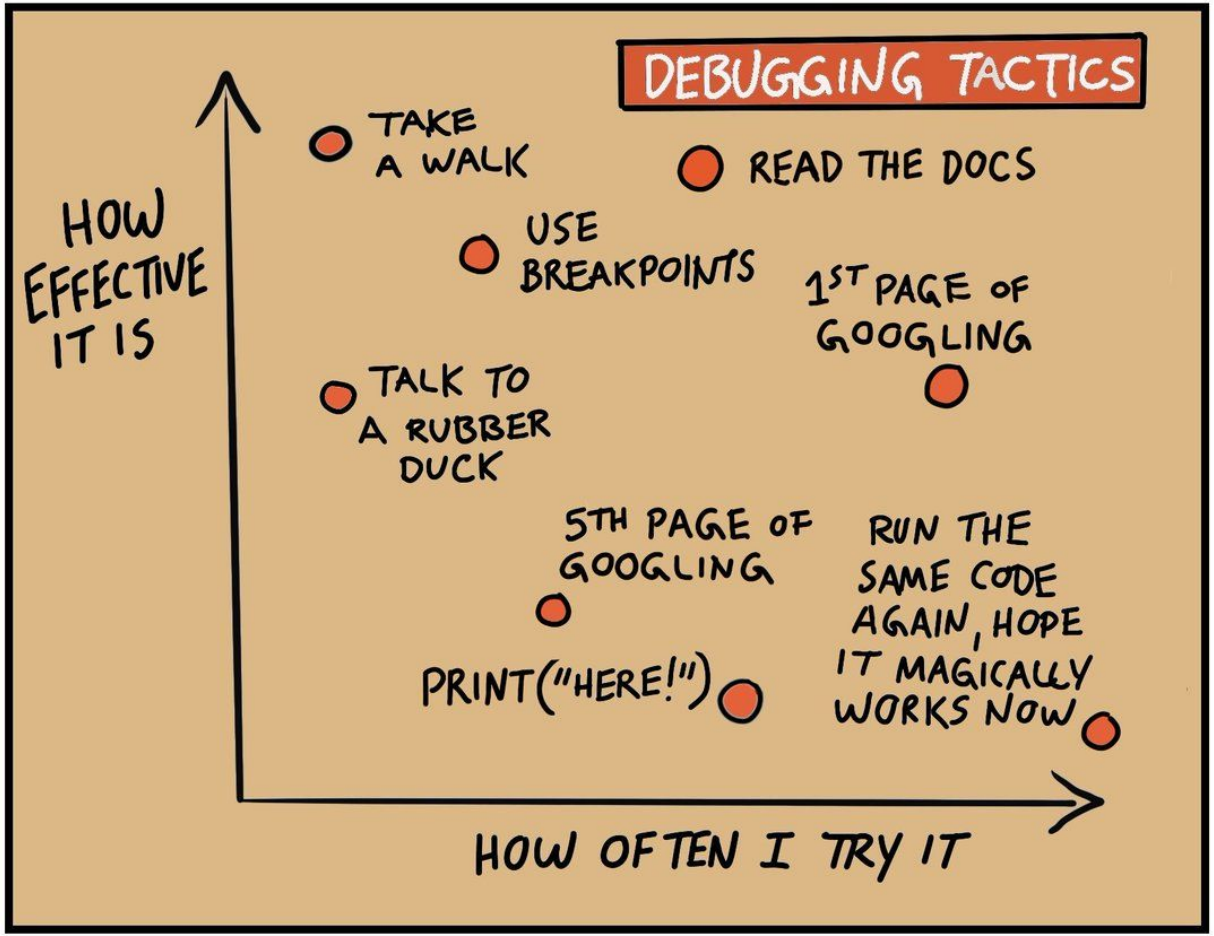
Dominik Rafacz, biogenies seminar, 21.07.21

overview

- background
- demonstration
- duel of the masters
- implementation notes
- future plans

background – motivation

FaaS and Furious by Forrest Brazeal A CLOUD GURU



background – timeline

2018 / 02 - first releases on PyPI

2021 / 03 - idea of creating port

2021 / 04 - development

2021 / 05 - release on CRAN

demo

let’s see how it works in practice!


print()
ic()

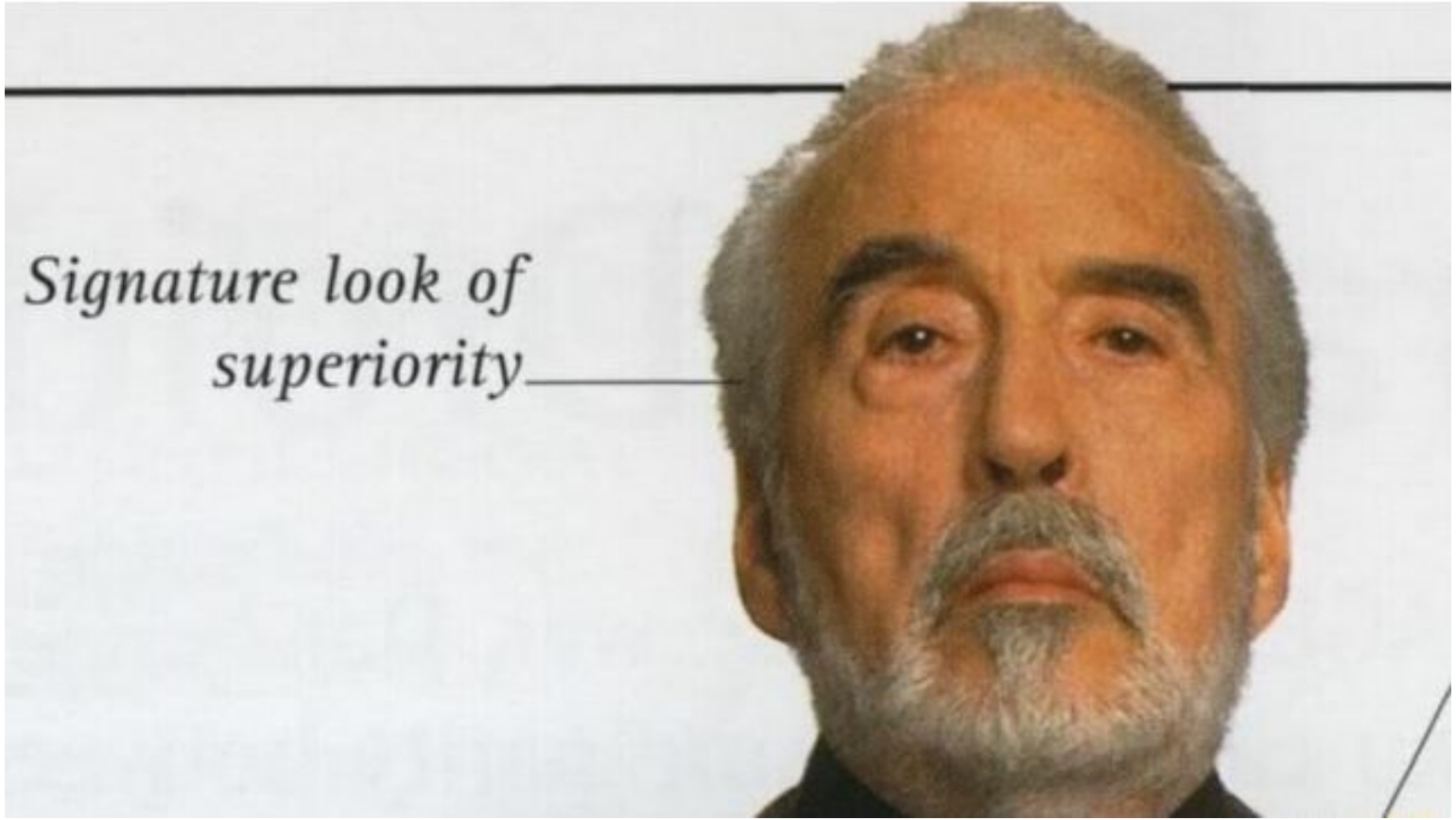
duel of the masters

- typing ‘ic’ is faster than typing ‘print’
- ‘ic’ shows debugging location
- ‘ic’ shows both the expression and the value
- ‘ic’ returns the value *(not an advantage in case of R)*
- ‘ic’ may be customized via options – e.g. peeking function and message, possibly more in the future
- ‘ic’ can be left in code without feeling ashamed

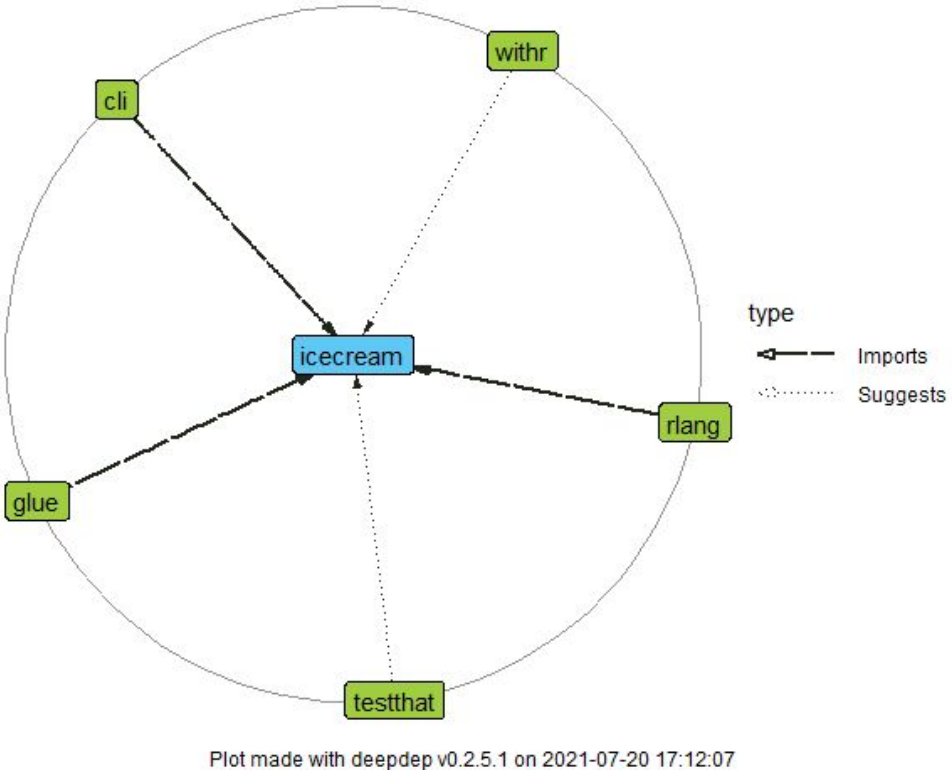
- ‘print’ is faster

fanciness

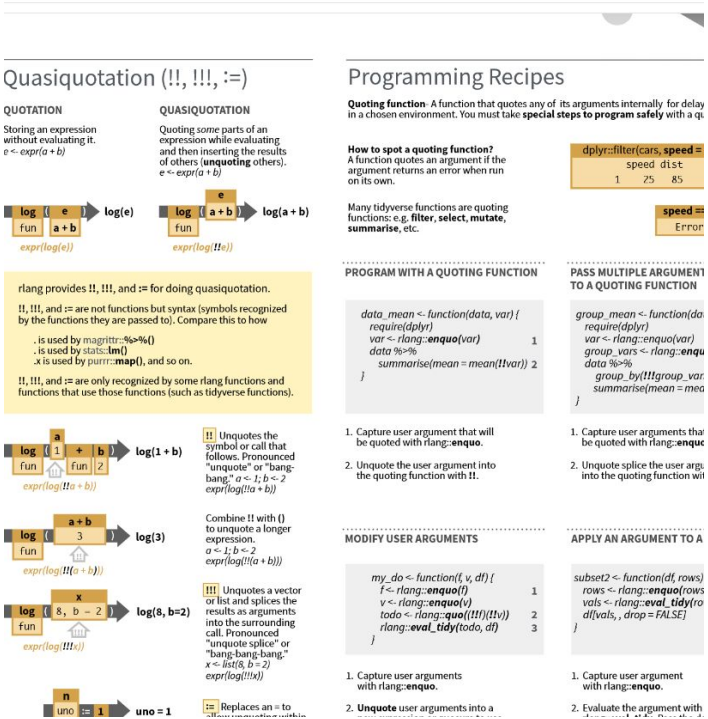
How I look when I start using icecream instead of print:



implementation – dependencies



implementation – rlang and traceback



Quasiquotation (!!, !!!, :=)

QUOTATION

Storing an expression without evaluating it.
e <- expr(a + b)

QUASIQUOTATION

Quoting *some* parts of an expression while evaluating and then inserting the results of others (**unquoting** others).
e <- expr(a + b)

log(e)

expr(log(e))

log(a + b)

expr(log(!!e))

rlang provides !!, !!!, and := for doing quasiquotation.

!!, !!!, and := are not functions but syntax (symbols recognized by the functions they are passed to). Compare this to how

. is used by magrittr::**%>%()**
. is used by stats::**lm()**
.x is used by purrr::**map()**, and so on.

!!, !!!, and := are only recognized by some rlang functions and functions that use those functions (such as tidyverse functions).

log(1 + b)

expr(log(!!a + b))

!! Unquotes the symbol or call that follows. Pronounced "unquote" or "bang-bang." *a <- 1; b <- 2*
expr(log(!!a + b))

log(3)

expr(log(!!(a + b)))

Combine !! with () to unquote a longer expression.
a <- 1; b <- 2
expr(log(!!(a + b)))

log(8, b=2)

expr(log(!!!x))

!!! Unquotes a vector or list and splices the results as arguments into the surrounding call. Pronounced "unquote splice" or "bang-bang-bang."
x <- list(8, b = 2)
expr(log(!!!x))

uno = 1

:= Replaces an = to

Programming Recipes

Quoting function: A function that quotes any of its arguments internally for delay
in a chosen environment. You must take **special steps to program safely** with a qu

How to spot a quoting function?
A function quotes an argument if the argument returns an error when run on its own.

Many tidyverse functions are quoting functions: e.g. **filter**, **select**, **mutate**, **summarise**, etc.

dplyr::filter(cars, **speed =**

```
  speed dist
1    25   85
```

speed ==

```
Error
```

PROGRAM WITH A QUOTING FUNCTION

```
data_mean <- function(data, var) {
  require(dplyr)
  var <- rlang::enquo(var)                1
  data %>%
    summarise(mean = mean(!!var))         2
}
```

1. Capture user argument that will be quoted with rlang::**enquo**.
2. Unquote the user argument into the quoting function with !!.

PASS MULTIPLE ARGUMENT TO A QUOTING FUNCTION

```
group_mean <- function(da
  require(dplyr)
  var <- rlang::enquo(var)
  group_vars <- rlang::enqu
  data %>%
    group_by(!!!group_var
    summarise(mean = mea
}
```

1. Capture user arguments tha be quoted with rlang::enquo
2. Unquote splice the user arg into the quoting function wit

MODIFY USER ARGUMENTS

```
my_do <- function(f, v, df) {
    f <- rlang::enquo(f)                  1
    v <- rlang::enquo(v)
    todo <- rlang::quo((!!f)(!!v))        2
    rlang::eval_tidy(todo, df)            3
}
```

1. Capture user arguments with rlang::**enquo**.
2. **Unquote** user arguments into a

APPLY AN ARGUMENT TO A

```
subset2 <- function(df, rows)
  rows <- rlang::enquo(rows
  vals <- rlang::eval_tidy(ro
  df[vals, , drop = FALSE]
}
```

1. Capture user argument with rlang::**enquo**.
2. Evaluate the argument with

```
> rlang::last_trace()
   ■
 1. └─chifishr::chi_fisher_p(treatment, "sex", "treatment")
 2.   └─purrr:::chisq_wrapper(tbl, var, treatment)
 3.     ├─purrr:::capture_output(.f(...))
 4.     │ └─base::withCallingHandlers(code, warning = wHandler, message = mHandler)
 5.     └─chifishr:::.f(...)
 6.       └─tbl %>% dplyr::pull(var) %>% as.factor()
 7.         ├─base::withVisible(eval(quote(`_fseq`(`_lhs`)), env, env))
 8.         └─base::eval(quote(`_fseq`(`_lhs`)), env, env)
 9.           └─base::eval(quote(`_fseq`(`_lhs`)), env, env)
10.             └─chifishr:::`_fseq`(`_lhs`)
11.               └─magrittr::freduce(value, `_function_list`)
12.                 └─function_list[[i]](value)
13.                   ├─dplyr::pull(., var)
14.                   └─dplyr:::pull.data.frame(., var)
15.                     └─tidyselect::vars_pull(names(.data), !!enquo(var))
16.                       └─tidyselect:::match_var(var, vars)
```

future plans

what almost works (to fix and release):

- ic_autopeek & switching peeking function

what does not work as intended (to upgrade):

- finding context in test files, knitted documents and some other cases

what does not work at all (to implement):

- returning output as string
- outputting timestamp
- multiple inputs
- some convenience options from original package

credits

- the original idea: @gruns
- co-developement and maintenance: @lewinfox
- support, suggestions, tests: @ErdaradunGaztea

the end