> Programowanie Funkcyjne w R

Damian Skrzypiec
damian.skrzypiec@pwc.com
PwC (FRM)
27 marca 2019

Plan

- Kilka słów o mnie
- Cel prezentacji
- Czym jest programowanie „funkcyjne”?
- Krótki rys historyczny
- Czy R i tak nie jest językiem funkcyjnym?
- Funkcje czyste
- Funkcje jako podstawowe „obiekty”
- Monoidy
- Częściowe zastosowanie

Kilka słów o mnie

- Absolwent MIM UW
- Od ponad 4 lat w Financial Risk Managment w PwC
- Miłośnik R od ponad 7 lat
- Interesuje się projektowaniem języków programowania
- Na co dzień używam: R, SQL, go, C#
- W domu bawię się: C++, F#

Kilka słów o mnie

Zespół FRM w PwC zajmuje się:

- Modelowaniem, programowaniem, walidacją i wdrożeniem modeli ryzyka kredytowego w bankach
- Budowaniem narzędzi IT (wewnętrznych i zewnętrznych) związanych głównie z ryzykiem kredytowym oraz rynkiem finansowym.

- Rekrutujemy ☺

Cel prezentacji

Cel prezentacji

- Przedstawienie podstawowych (wybranych) idei programowania funkcyjnego

Cel prezentacji

- Przedstawienie podstawowych (wybranych) idei programowania funkcyjnego
- Przykłady zastosowania tych idei w programowaniu w R

Czym jest programowanie „funkcyjne”?

Czym jest programowanie „funkcyjne”?

- Jest to paradygmat programowania, gdzie funkcje są podstawową jednostką języka. Ponadto funkcje są zbliżone do funkcji matematycznych.

Czym jest programowanie „funkcyjne”?

Główne cechy FP, które poruszymy:

- Funkcje czyste i niezmienniczość
- Funkcje jako podstawowe obiekty
- Monoidy
- Częściowe zastosowanie

Krótki rys historyczny

- 1930s – Rachunek lambda – Alonzo Church
- 1945 – Początek teorii kategorii w „General Theory of Natural Equivalences” – Eilenberg & MacLane
- 1958 – LISP
- 1990 – Haskell
- 2004 – Scala
- 2005 – F#

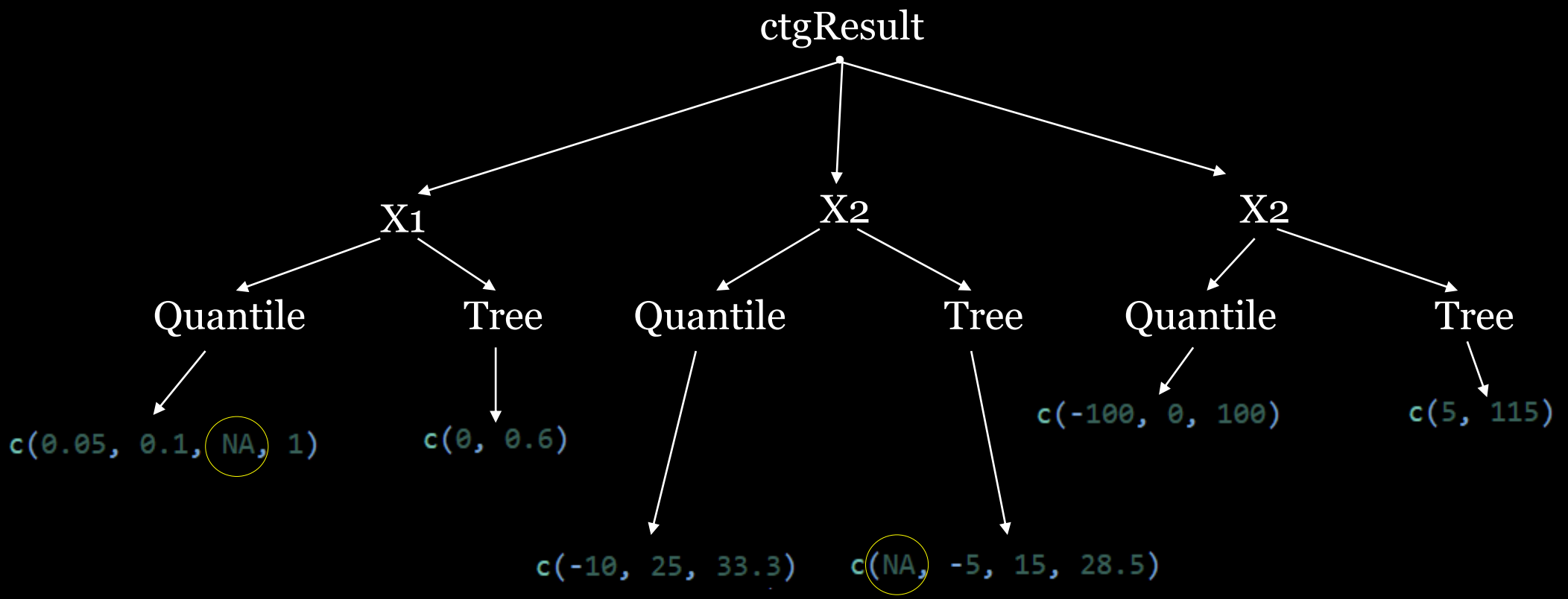
Czy R i tak nie jest funkcyjnym językiem?

```
loadQuery <- function(connString, query) { ... }
saveToSQL <- function(connString, dataFrame, dbObjectName) { ... }


connString <- "driver={SQL Server};server=server;database=project1"

loadQuery(connString, "SELECT TOP 10 * FROM dbo.X")
loadQuery(connString, "SELECT TOP 10 * FROM dbo.Y WHERE A > 10")
saveToSQL(connString, iris, "dbo.Iris")
```

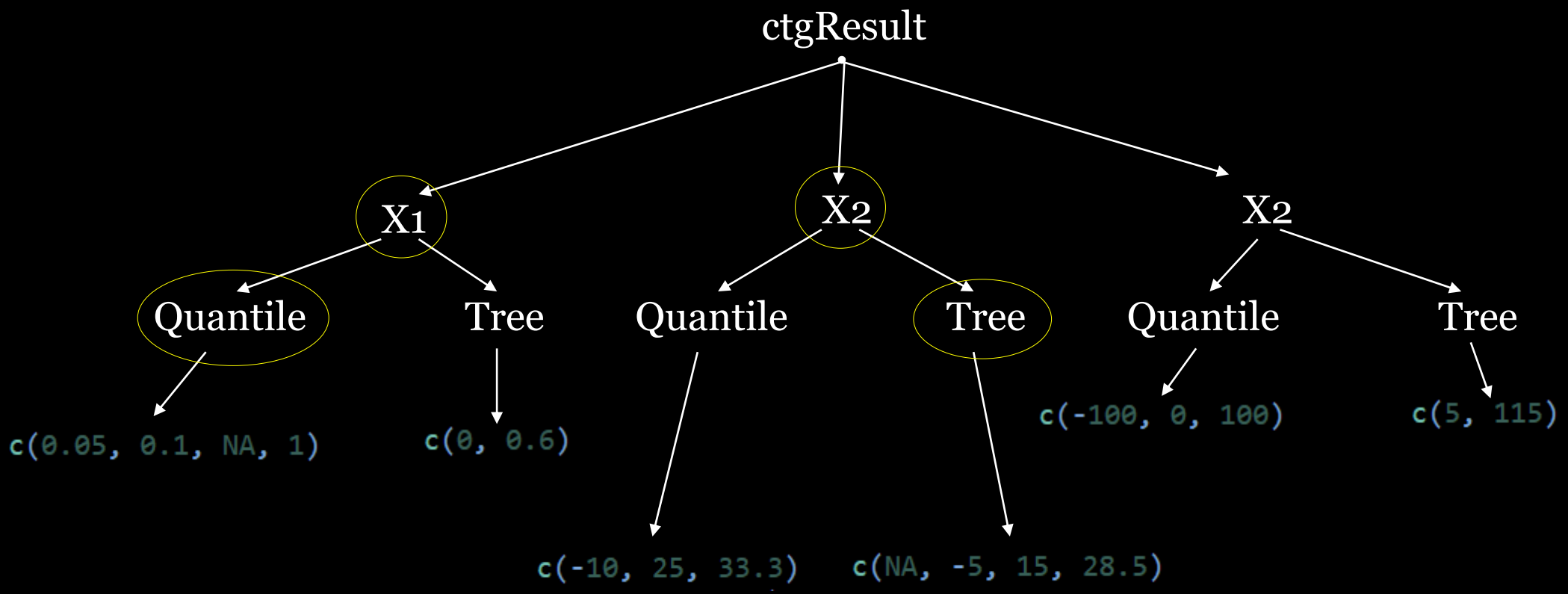
Czy R i tak nie jest funkcyjnym językiem?

```
# OO approach
createDBHandler <- function(connectionString) {
    dbHandler <- new.env()
    class(dbHandler) <- "DBHandler"

    dbHandler$LoadQuery <- function(query) { ... }
    dbHandler$Save <- function(dataFrame, dbObjectName) { ... }

    return(dbHandler)
}

connString <- "driver={SQL Server};server=server;database=project1"

projectDB <- createDBHandler(connString)
projectDB$LoadQuery("SELECT TOP 10 * FROM dbo.X")
projectDB$LoadQuery("SELECT TOP 11 * FROM dbo.Y WHERE A > 10")
projectDB$Save(iris, "dbo.Iris")
```

Zacznijmy od przykładu



Zacznijmy od przykładu



Zacznijmy od przykładu

```r
# Non-functional approach
filterNASplitPoints <- function(ctgResult) {
    empty <- list()
    for (xName in names(ctgResult)) {
        for (method in names(ctgResult[[xName]])) {
            singleRes <- ctgResult[[xName]][[method]]
            if (containNA(singleRes)) {
                if (is.null(empty[[xName]])) {
                    empty[[xName]] <- list()
                    empty[[xName]][[method]] <- singleRes
                } else {
                    empty[[xName]][[method]] <- singleRes
                }
            }
        }
    }
    return(empty)
}

containNA <- function(singleCtg) {
    singleCtg %>% is.na %>% which %>% any
}
```

Zacznijmy od przykładu

```r
# Functional approach
filterNASplitPoints(ctgResult) {
    xWithNaMethod <- lapply(ctgResult, methodsWithNA) %>% filterEmptyVectors
    xFiltered <- ctgResult[names(xWithNaMethod)]

    sapply(names(xFiltered), function(name) {
        naMethodNames <- xWithNaMethod[[name]]
        xFiltered[[name]][naMethodNames]
    }, simplify = FALSE)
}

filterEmptyVectors <- function(list) {
    list[sapply(list, function(x) length(x) > 0)]
}

methodsWithNA <- function(xResult) {
    sapply(xResult, containNA) %>% which %>% names
}
```

Funkcje czyste (pure functions)

Zdefiniowanie i używanie funkcji czystych jest próbą zbliżenia funkcji informatycznych do funkcji matematycznych

Funkcje czyste (pure functions)

Funkcje czyste charakteryzują się

- Robią dokładnie jedną rzecz
- Nie posiadają efektów ubocznych
- Wynik funkcji zależy tylko od jej argumentów
- Dla tego samego zestawu argumentów zwraca zawszę ten sam wynik

Funkcje czyste (pure functions)

Dlaczego funkcje czyste są spoko?

Funkcje czyste (pure functions)

Dlaczego funkcje czyste są spoko?

- Łatwe w testowaniu
- Zapewniają izolacje
- Łatwe w zrównoleglaniu
- Są czytelne
- Łatwe do wykorzystania w innym projekcie

Funkcje czyste (pure functions)

Dlaczego funkcje czyste są spoko?

- Łatwe w testowaniu
- Zapewniają izolacje
- Łatwe w zrównoleglaniu
- Są czytelne
- Łatwe do wykorzystania w innym projekcie

Stąd też większość (dobrych) R-owych pakietów składa się z funkcji czystych.

Funkcje czyste (pure functions)

```r
# Pure function
naPerc <- function(x) {
    x %>% is.na %>% which %>% length / x %>% length
}


# Nonpure function
naPercToRds <- function(x, fileName) {
    saveRDS(naPerc(x), fileName)
}
```

Funkcje jako podstawowe obiekty

```
# Calculates [f(y, x1), f(y, x2), ..., f(y, xn)]

calcYXs <- function(dataFrame, yColName, xColNames, f, ...) {
    y <- dataFrame[[yColName]]
    sapply(dataFrame[, xColNames], function(x) {f(y, x, ...)})
}
```

Funkcje jako podstawowe obiekty

```
> calcYX(iris, "Sepal.Length", names(iris)[2:4], cor, method = "pearson")
 Sepal.Width Petal.Length  Petal.Width
  -0.1175698    0.8717538    0.8179411
```

Monoid

Monoid

Monoid jest półgrupą, której działanie ma element neutralny. Formalnie, monoid to algebra $(S, e, *)$, sygnatury $(0, 2)$, gdzie S jest niepustym zbiorem, natomiast

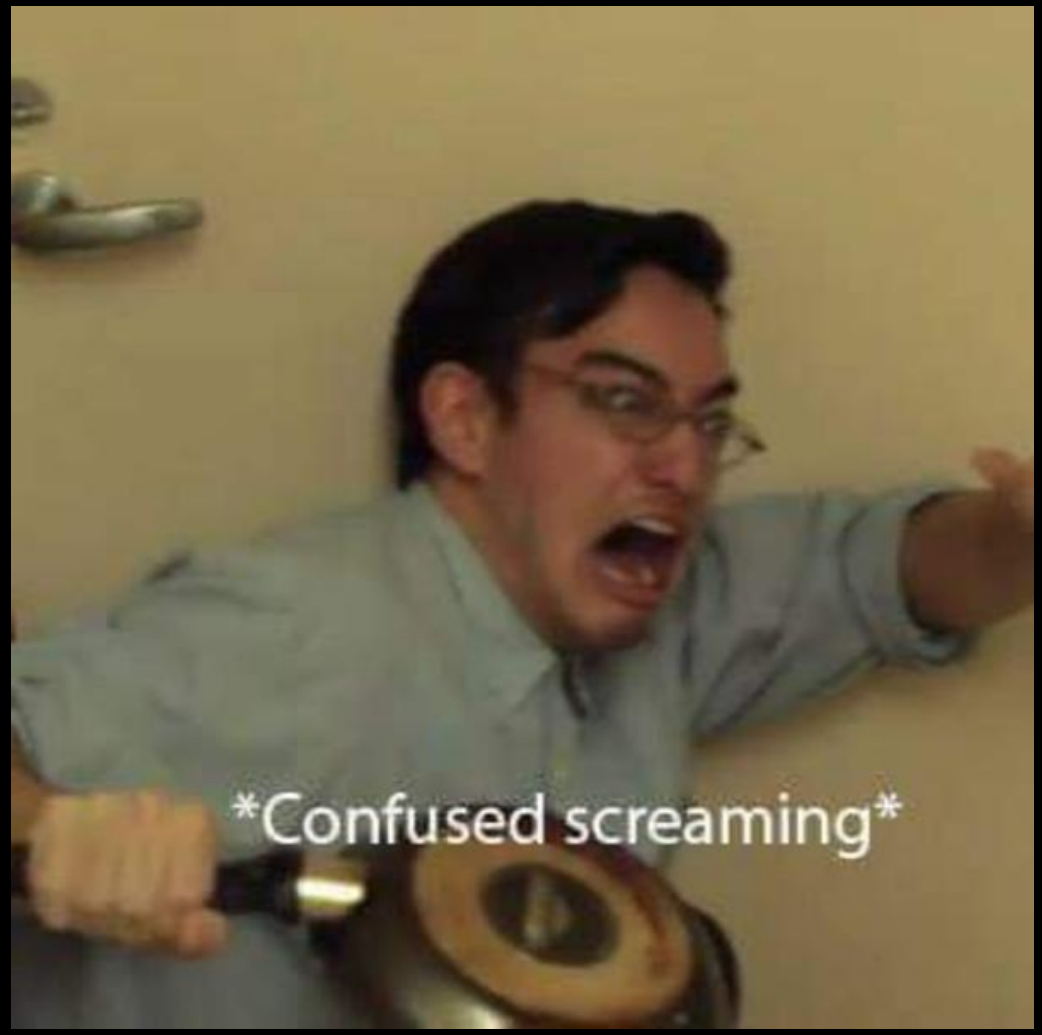
$$* : S \times S \longrightarrow S$$

jest działaniem dwuargumentowym, spełniającym warunki:

1. $\forall_{a \in S} \quad e * a = a * e = a$
2. $\forall_{a,b,c \in S} \quad (a * b) * c = a * (b * c)$

https://pl.wikipedia.org/wiki/Monoid

Monoid



Monoid

$1 + 0 = 0 + 1 = 1$

$1 + (2 + 3) = (1 + 2) + 3$

Monoid

$1 + 0 = 0 + 1 = 1$

$1 + (2 + 3) = (1 + 2) + 3$

$1 * 5 = 5 * 1 = 5$

$1 * (2 * 3) = (1 * 2) * 3$

Monoid

$1 + 0 = 0 + 1 = 1$

$1 + (2 + 3) = (1 + 2) + 3$

$1 * 5 = 5 * 1 = 5$

$1 * (2 * 3) = (1 * 2) * 3$

Dlaczego jest to warte uwagi?

Monoid

$(1 + 2 + 3 + 4 + 5 + 6 + 7 + \ldots + 100) = 5050$

Monoid

$$(1 + 2 + 3 + 4 + 5 + 6 + 7 + \ldots + 100) = 5050$$

Ok, ale jednak musimy policzyć sumę aż do 101.

Monoid

$(1 + 2 + 3 + 4 + 5 + 6 + 7 + \ldots + 100) = 5050$

To już policzyliśmy!

Ok, ale jednak musimy policzyć sumę aż do 101.

$(1 + 2 + \ldots + 101) = (1 + 2 + \ldots + 100) + 101 = 5050 + 101 = 5151$

łączność

Monoid

purrr::reduce(x, f, init) = f(f(f(f(f(… f(f(init, x[1]), x[2]), x[3], …, x[n])

Monoid

```r
joinPath <- function(prev, act) {
    paste0(prev, "/", act)
}

parts <- c("some", "path", "to", "file.R")


purrr::reduce(.x = parts, .f = joinPath, .init = ".")
```

Monoid

```
[1] "./some/path/to/file.R"
```

Częściowe zastosowanie (partial application)

Częściowe zastosowanie (partial application)

```r
addPar <- function(doc, value, style) {
    officer::body_add_par(doc, value = value, style = style)
}

emptyLine <- function(doc) {
    addPar(doc, "", "Normal")
}

tableRef <- function(doc, refString) {
    addPar(doc, refString, "tablereference")
}

plotRef <- function(doc, refString) {
    addPar(doc, refString, "figurereference")
}

normal <- function(doc, txt) {
    addPar(doc, txt, "Normal")
}
```

Częściowe zastosowanie (partial application)

```
officer::read_docx() %>%
    officer::body_add_par(., "", style = "Normal") %>%
    officer::body_add_par(., "", style = "Normal") %>%
    officer::body_add_par(., "Some text",
                          style = "Normal") %>%
    officer::body_add_gg(., plotHistogram()) %>%
    officer::body_add_par(., "Some histogram",
                          style = "figurereference") %>%
    officer::body_add_par(., "New chapter",
                          style = "headin 1") %>%
    officer::body_add_par(., "Text in new chapter",
                          style = "Normal")
```

Częściowe zastosowanie (partial application)

```
officer::read_docx() %>%
    emptyLine %>%
    emptyLine %>%
    normal(., "Some text") %>%
    addPlot(., plotHistogram()) %>%
    plotRef(., "Some histogram") %>%
    heading(., "New chapter", 1) %>%
    normal(., "Text in new chapter")
```

Dziękuję za uwagę!

https://github.com/DSkrzypiec/WDI2019FP