Technologie Informacyjne

R – funkcyjny język programowania

Opracował: Maciej Penar

Spis treści

[1. Wprowadzenie 3](#_Toc86433781)

[2. Podstawy 3](#_Toc86433782)

[Co to znaczy, że język jest funkcyjny 3](#_Toc86433783)

[Hello World w R 3](#_Toc86433784)

[Operatory 3](#_Toc86433785)

[Typy danych i literały 4](#_Toc86433786)

[Zmienne 4](#_Toc86433787)

[Konkatenacja wektorów 6](#_Toc86433788)

[**Pomoc** 6](#_Toc86433789)

[Szybkie rysowanie wykresów 6](#_Toc86433790)

[Troche bardziej skomplikowane rzeczy 7](#_Toc86433791)

[Pętle 7](#_Toc86433792)

[Pętla funkcyjna 7](#_Toc86433793)

[indeksowanie 7](#_Toc86433794)

[Data Frames 7](#_Toc86433795)

[Funkcje 8](#_Toc86433796)

[Inne informacje 8](#_Toc86433797)

[Kilka funkcji 8](#_Toc86433798)

[Zarządzanie pakietami 9](#_Toc86433799)

[Przydatne Pakiety 9](#_Toc86433800)

[Zadania 9](#_Toc86433801)

# 1. Wprowadzenie

R jest językiem funkcyjnym/skryptowym dedykowanym dla matematyków. Posiada naleciałości z języków proceduralnych, ale powinniśmy się ich wystrzegać (przez wzgląd na wydajność / czytelność kodu).

Częściowo wyparty przez Pythona.

IMO najlepiej korzystać z dystrybucji Microsoft R (kiedyś Revolution R): [link](https://mran.microsoft.com/open)

Sama dystrybucja pozwala na wykonywanie kodu napisanego w R – żeby wygodnie było wykonywać development kodu warto zainstalować środowisko RStudio: [link](https://www.rstudio.com/)

R jest darmowy – kuzynami tego języka/środowiska jest: Matlab, Octave

# 2. Podstawy

### Co to znaczy, że język jest funkcyjny

Funkcyjność języka oznacza, że skrypt/kod opisuje efekt końcowy – a nie sposób jego osiągnięcia. W językach funkcyjnych charakterystyczne jest to, że mamy szeroki zestaw „narzędzi” (najczęściej funkcji) **dedykowanych rozwiązywaniu konkretnych problemów.**

Problem w tym, że często opis efektu końcowego przypomina algorytm.

W językach funkcyjnych nie **powinny** występować pętle – które są naleciałością z języków proceduralnych.

### Hello World w R

|  |
| --- |
| print("Hello World") |

### Operatory

* + -
* / \* \*\*
* %/% (dzielnie całkowitoliczbowe)
* %% (modulo)
* %\*%
* <- lub = (przypisanie)
* == (porównanie - równoważność)
* != (porównanie - różnica)
* ! (negacja)
* > < >= <=

|  |
| --- |
| 2 + 2  2 \* 3  2 \*\* 3 |

### Typy danych i literały

* TRUE / FALSE – literały boolowskie
* NULL
* Liczby złożone (complex)
* Wszystkie liczby jako zmiennoprzecinkowe (**separowane kropką**)
* Teksty wpisujemy używając znaków: 'tekst'
* … istnieją też odwzorowania (nazwane niepoprawnie listami)… ale nie będziemy się nimi zajmować
* Do sprawdzania typu zmiennych można korzystać z:
  + Albo funkcje is.XXXX np. is.matrix is.vector
  + Albo typeof – co sprawdza typ pierwszego elementu w kolekcji, nie samą kolekcję
* **Wszystkie liczby jako wektory… lub macierze – implikuje to, że wszystkie operacje na ogół są ewaluowane per każdy element w wektorze… przez co nie trzeba pętli**

### Zmienne

Zmienne rezydują w tzw. Środowisku – wszystkie zmienne w ramach danego środowisko „widzą się” wzajemnie. Domyślnie zmienne i funkcje alokowane są w Globalnym Środowisku (Global Environment).

|  |
| --- |
| nazwana\_liczba = 5  nazwany\_tekst = 'kot'  # Potęgowanie zmiennej  nazwana\_liczba \*\* 2 |

Przez to, że najważniejszymi obywatelami w R są wektory, to istnieje skrót do szybkiego generowania ciągów liczb całkowitych:

|  |
| --- |
| # Generowanie ciągów  1:10  nazwany\_ciag = -100:100  inny\_ciag = 10:1  # Jeśli potrzebujemy większej kontroli nad ciągiem to mamy do dyspozycji funkcję **seq:**  nazwany\_ciag\_2 = seq(from = - 50, to = 50, by = 0.5) |

Jeśli wyrażenie w R nie jest przepisaniem to wynik jest drukowany na konsoli np.:

|  |
| --- |
| nazwany\_ciag\_2  # lub to: seq(from = - 50, to = 50, by = 0.5) |

By potwierdzić, że domyślnie wszystko jest ciągiem liczbowym można użyć funkcji is.vector

|  |
| --- |
| czy\_to\_vector = 1:4  is.vector(czy\_to\_vector)  is.matrix(czy\_to\_vector) |

Obiekty (można myśleć o zmiennych) w R mają (lub nie mają) właściwości – jeśli właściwości przyjmują odpowiednie wartości – obiekty zmieniają swój typ. Najbardziej istotnym przykładem są macierze – które mają właściwość **dim()**

|  |
| --- |
| czy\_to\_vector = 1:4  dim(czy\_to\_vector)  # Skopiujmy macierz  czy\_to\_macierz = czy\_to\_vector  **# Uwaga na linijkę poniżej – taki zapis występuje tylko w R**  **dim(czy\_to\_macierz) = c(2,2)**  is.vector(czy\_to\_macierz)  is.matrix(czy\_to\_macierz)  czy\_to\_macierz |

Zwróćmy uwagę, że operacje są ewaluowane per każda komórka:

|  |
| --- |
| czy\_to\_vector \* 100  czy\_to\_vector + 100  czy\_to\_macierz + 100  czy\_to\_macierz \* 100 |

**Operator mnożenia macierzowego to %\*%**

### Konkatenacja wektorów

**By łączyć (konkatenować) wektory używamy funkcji c()**

|  |
| --- |
| **c(1,100,5,2)**  **c(1,2,c(3,4))** |

### **Pomoc**

**Pamiętajmy – R jest językiem funkcyjnym – nie jest możliwe zapamiętać jak działają wszystkie funkcje w tym języku. Z gro funkcji korzysta się doraźnie. W celu uzyskania pomocy należy wpisać:**

* **?[nazwa funkcji] – wyszukuje dokumentację funkcji o podanej nazwie**
* **??[tekst] – przeszukuje dokumentację w poszukiwaniu danego tekstu**

|  |
| --- |
| ??`regular expression`  ?plot.default |

### Szybkie rysowanie wykresów

Domyślnie R ma bibliotekę do szybkiego rysowania wykresów. Służy do tego funkcja **plot.**

|  |
| --- |
| # Rysowanie funkcji kwadratowej:  plot(-10:10, (-10:10) \*\* 2)  plot(-10:10, (-10:10) \*\* 2, 'l')  plot(-10:10, (-10:10) \*\* 2, 'l', xlim = c(-20,20)) |

# Troche bardziej skomplikowane rzeczy

## Pętle

Pomimo, że powinniśmy się wystrzegać pętli, to czasem nie da się ich obejść (nieprawda).

|  |
| --- |
| for(i in 1:100){  print(i)  } |

## Pętla funkcyjna

Raczej pętle robimy tak:

|  |
| --- |
| # Załózmy że mamy jakiś ciąg wejściowy:  ciag = c("ala", "ma", "kota")  # Konwersja na macierz:  dim(ciag) = c(length(ciag), 1)  # Aplikacja funkcji nchar do każdego elementu:  apply(ciag, 1, nchar) |

## indeksowanie

Wektory indeksowane są od 1 (! Nietypowe jak na język programowania).

Pobranie elementów można wykonać za pomocą składni []

|  |
| --- |
| # Utworzenie ciągu 5…10..15…  z = (1:10) \* 5  # Pobranie 3 i 1 elementu  z[c(3,1)] |

Jeśli w [] podamy wektor boolowski, to zwrócone zostaną elementy które mają odpowiadającą pozycję TRUE

|  |
| --- |
| z[(z %% 25)==0]  z[z > 25] |

## Data Frames

To kolekcja ustrukturalizowanych wektorów. Każdy ma taką samą długość. Powołanie następuje zazwyczaj przez użycie funkcji read.XXX np. read.csv…

… ale można też użyć funkcji data.frame()

|  |
| --- |
| df = data.frame(imie = c("Maciej", "Adam", "Janusz"), wiek = c(18,19,20), wzrost = c(170,167,185))  # Odwołanie się do wektorów przez składnie z $  df$wiek |

**R posiada wbudowany dataframe o nazwie iris (i też jest cars).**

## Funkcje

Powtarzalne fragmenty kodu możemy opakować w funkcję. Funkcja w R posiada nazwę oraz argumenty:

|  |
| --- |
| pow <- function(x,n){  x \*\* n  } |

albo

|  |
| --- |
| pow <- function(x,n){  return (x \*\* n)  } |

W funkcjach możemy też stosować instrukcje if-then-else… ale znowu – nie powinniśmy.

# Inne informacje

### Kilka funkcji

* sample(x, size, replace = FALSE, prob = NULL)
  + x jest wektorem próbek – może być to wektor ciągów znaków
* Rozkład równomierny
  + runif(n, min = 0, max = 1) – pobranie wartości
  + punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE) - dystrybuanta
* Rozkład normalny:
  + rnorm(n, mean = 0, sd = 1)
  + dnorm(x, mean = 0, sd = 1, log = FALSE)
* plot - rysowanie wykresu
* nchar – długość tekstu
* length – długość wektora np. length(5:10)
* names – nazwy kolumn (data.frame)
* dim – wymiary wektora
* seq – tworzenie sekwencji
* mean / sd / var / sum – funkcje agregujące
* seq\_along – tworzenie sekwencji nad wektorem
* ifelse – operator trójargumentowy
* t.test – test t-studenta
* chisq.test – test chi-kwadrat

### Zarządzanie pakietami

* install.packages(“nazwa”) – instalacja pakietu z zewnętrznego repozytorium
* library(nazwa) – ładowanie paketu do pamięci

### Przydatne Pakiety

* ggplot2 – wykresy od Google-a
* caret – klasyfikacja / SI
* dplyr – lepsza obsługa data.frames

# Zadania

1. Narysować wykres prezentujący działanie funkcji **ifelse()** np. użyć tej funkcji do implementacji funkcji znaku (signum)
2. Narysować wykres dystrybuanty rozkładu normalnego o średniej 10 i odchyleniu standardowym 10 (funkcje dnorm i plot)
3. Narysować wykres pierwszych kilku wartości funkcji Fibonacciego
4. Wyznaczyć prawdopodobieństwo problemu Monty-Halla dla N bramek [link](https://pl.wikipedia.org/wiki/Paradoks_Monty%E2%80%99ego_Halla)
   1. Nie myśleć o oficjalnym wzorze
   2. Zaproponować funkcję która decyduje dla pojedynczej gry czy zwyciężono czy nie
   3. Zasymulować odpowiednio dużą liczbie gier