Konstrukcje warunkowe

Statystyka I z R

Bartosz Maćkiewicz

Konstrukcje warunkowe

W tym notatniku zajmiemy się różnymi rodzajami konstrukcji warunkowych w R.

Podstawowe konstrukcje to: konstrukcjami:

- `if / else`
- `ifelse`
- `for`
- `while`

Jako materiał dydaktyczny posłuży nam tekst piosenki Ice Cube "It Was A Good Day". W piosence tej Cube opowiada o swoim dobrym dniu. Zacznijmy więc od początku.

Just wakin up in the mornin gotta thank God

I don't know but today seems kinda odd

No barkin from the dog, no smog

[. . .] I can't believe, today was a good day

Spróbujmy się dowiedzieć, czy dzisiaj był dla Ice Cube'a dobry dzień. Zacznijmy od szczekania psa.

`if`

Najprostszą konstrukcją warunkową jest pętla `if`. Wygląda dokładnie tak samo jak w każdym innym języku programowania - jeżeli warunek jest prawdziwy, to wykonuje okreslone polecenia.

```
barkinFromTheDog <- FALSE # wiemy, że pies dzisiaj nie szczekał

if (barkinFromTheDog == FALSE) print('It was a good day')
```

```
## [1] "It was a good day"
```

Możemy łatwo wprowadzić do naszego wyrażenia negację za pomocą operatora `!`.

```
!barkinFromTheDog
```

```
## [1] TRUE
```

```
if (!barkinFromTheDog) print('It was a good day')
```

```
## [1] "It was a good day"
```

Oraz konstruować z operatorów funktorów logicznych koniunkcji (`&`) i alternatywy (`|`) bardziej skomplikowane warunki. Nasze instrukcje moga być wielolinikowe, musimy je jednak otoczyć nawiasami klamrowymi (`{}`).

```
smog <- FALSE

if (barkinFromTheDog == FALSE & smog == FALSE) {
    print("It was")
    print('a good day')
    }
```

```
## [1] "It was"
## [1] "a good day"
```

Oczywiście jeżeli warunek jest fałszywy, to pętla nie zrobi nic.

```
smog = TRUE
if (barkinFromTheDog == FALSE & smog == FALSE) {
    print('It was a good day')
    }
```

Możemy naturalnie, jak w każdym języku programowania, określić co ma się dziać, jeżeli warunek nie jest spełniony. Służy nam do tego słowo kluczowe `else`.

```
if (barkinFromTheDog == FALSE | smog == FALSE) {
    print('It was a mediocre day')
} else {
    print('It was not a good day')
     }
```

```
## [1] "It was a mediocre day"
```

```
# To zadziała
```

Uwaga - `else` musi być w tej samej linijce, co nawiasy klamrowe okalające blok `if`

```
if (barkinFromTheDog == FALSE | smog == FALSE) {
    print('It was a mediocre day')
}
else {
    print('It was not a good day')
}
# A to nie zadziała
```

Przy łączeniu kilku warunków w jeden - tak samo jak na logice - należy pamiętać o nawiasach. Te dwie konstrukcje nie są równoważne!

```
smog = FALSE

if (!(barkinFromTheDog & smog)) {
    print('It was a good day')
} else {
    print('It was not a good day')
}
```

```
## [1] "It was a good day"
```

```
if (!barkinFromTheDog & smog) {
    print('It was a good day')
} else {
    print('It was not a good day')
     }
```

```
## [1] "It was not a good day"
```

Cube rapuje, że w zeszłym tygodniu grając w koszykówkę zdobył *triple-double* czyli dwucyfrową zdobycz w trzech kategoriach statystycznych (punkty, asysty, zbiórki, przechwyty lub bloki)

"Last week [...] around and got a triple double"

Spróbujmy skonstruować odpowiedni warunek. Wykorzystamy do tego interesującą własność R.

Stwórzmy najpierw pięć zmiennych, w których będziemy przechowywać liczbę punktów, asyst, zbiórek, bloków i przechwytów.

```
points <- 12
assists <- 14
rebounds <- 7
blocks <- 10
steals <- 3
```

Następnie stwórzmy wektor, w którym znajdą się nasze statystyki

```
stats <- c(points, assists, rebounds, blocks, steals)
stats
```

```
## [1] 12 14  7 10  3
```

Sprawdźmy, ile ze statystyk nich było większe lub równe 10.

```
stats >= 10
```

```
## [1]  TRUE  TRUE FALSE  TRUE FALSE
```

```
greather_than_10 <- stats[stats >= 10]
greather_than_10
```

```
## [1] 12 14 10
```

Stwórzmy następnie nową zmienną, która przyjmie wartość `TRUE` tylko wtedy, gdy Ice Cube zdobył *triple-double*.

```
triple_double <- length(greather_than_10) >= 3
triple_double
```

```
## [1] TRUE
```

Następnie możemy włączyć naszą nową zmienną do warunku w konstrukcji warunkowej `if`.

```
if (!(barkinFromTheDog & smog) & triple_double) {
    print('It was a good day')
} else {
    print('It was not a good day')
     }
```

```
## [1] "It was a good day"
```

ifelse

Konstrukcja `ifelse` nie jest prawdziwą pętlą, ale jest do pętli podobna i można jej użyć do bardzo wielu zadań. Funkcja ta przyjmuje trzy argumenty i zwraca wektor. Ogólna konstrukcja wygląda tak:

```
ifelse(test,yes,no)
```

lub bardziej opisowo

```
ifelse(warunek, wartosc_kiedy_spelniony, wartosc_kiedy_niespelniony)
```

- `warunek` - wektor typu `logical`

- `wartosc_kiedy_spelniony` - wartość w wynikowym wektorze dla pozycji, dla których w `warunek` znajduje się `TRUE`
- `wartosc_kiedy_niespelniony` - wartość w wynikowym wektorze dla pozycji, dla których w `warunek` znajduje się `FALSE`

Jest ona o tyle przydatna, że jeżeli przekażemy jej na wejściu wektor, to warunek będzie sprawdzany dla każdego elementu i funkcja również zwróci wektor. Przećwiczmy wykorzystanie tej funkcji na przykładzie znanego nam zbioru danych `mtcars`.

```
data(mtcars) # załadujmy zbiór mtcars
head(mtcars) # pierwsze 6 wierszy
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Dzięki funkcji `ifelse` łatwo możemy pogrupować samochody na szybkie i wolne. Przyjmijmy arbitralną granicę, że 150 koni mechanicznych oznacza, że samochód jest szybki.

```
head(mtcars$hp)
```

```
## [1] 110 110  93 110 175 105
```

```
ifelse(mtcars$hp > 150, 'szybki', 'wolny')
```

```
##  [1] "wolny"  "wolny"  "wolny"  "wolny"  "szybki" "wolny"  "szybki" "wolny"
##  [9] "wolny"  "wolny"  "wolny"  "szybki" "szybki" "szybki" "szybki" "szybki"
## [17] "szybki" "wolny"  "wolny"  "wolny"  "wolny"  "wolny"  "wolny"  "szybki"
## [25] "szybki" "wolny"  "wolny"  "wolny"  "szybki" "szybki" "szybki" "wolny"
```

Możemy teraz do naszej ramki danych dołączyć dodatkową kolumnę z informacją, czy dany samochód jest szybki.

```
mtcars['predkosc'] = ifelse(mtcars$hp > 150, 'szybki', 'wolny')
```

```
head(mtcars, 7)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb predkosc
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    wolny
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    wolny
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1    wolny
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1    wolny
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2   szybki
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    wolny
## Duster 360        14.3   8  360 245 3.21 3.570 15.84  0  0    3    4   szybki
```

A co jeżeli mamy więcej niż dwie kategorie? Załóżmy, że chcemy podzielić samochody pod względem ekonomiczności na 3 kategorie - tanie (do 15 w kolumnie `mpg`), średnie (między 16 a 25 w kolumnie `mprg`) i drogie (więcej niż 25 w kolumnie `mpg`) ze względu na to ile palą benzyny. Do takich zadań są specjalnie pakiety, ale mozemy zrobić to również za pomocą kilku `ifelse`.

Dzielimy najpierw samochody na średnie i tanie.

```
mtcars['ekonomicznosc'] <- ifelse(mtcars$mpg > 15, 'średni', 'tani')
mtcars$ekonomicznosc
```

```
##  [1] "średni" "średni" "średni" "średni" "średni" "średni" "tani"   "średni"
##  [9] "średni" "średni" "średni" "średni" "średni" "średni" "tani"   "tani"
## [17] "tani"   "średni" "średni" "średni" "średni" "średni" "średni" "tani"
## [25] "średni" "średni" "średni" "średni" "średni" "średni" "tani"   "średni"
```

Następnie tym, które przekraczają próg "drogości" przypisujemy 'drogi', a resztę wartości zostawiamy.

```
mtcars['ekonomicznosc'] = ifelse(mtcars$mpg > 25, 'drogi', mtcars$ekonomicznosc)
mtcars$ekonomicznosc
```

```
##  [1] "średni" "średni" "średni" "średni" "średni" "średni" "tani"   "średni"
##  [9] "średni" "średni" "średni" "średni" "średni" "średni" "tani"   "tani"
## [17] "tani"   "drogi"  "drogi"  "drogi"  "średni" "średni" "średni" "tani"
## [25] "średni" "drogi"  "drogi"  "drogi"  "średni" "średni" "tani"   "średni"
```

```
tail(mtcars)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb predkosc
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2    wolny
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2    wolny
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4   szybki
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6   szybki
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8   szybki
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2    wolny
##                ekonomicznosc
## Porsche 914-2          drogi
## Lotus Europa           drogi
## Ford Pantera L        średni
## Ferrari Dino          średni
## Maserati Bora           tani
## Volvo 142E            średni
```

Inny sposób to użycie funkcji `cut`. Za za jej pomocą możemy stworzyć `factor` dzieląć wektor liczbowy na równe części.

```
cut(mtcars$mpg, 3)
```

```
##  [1] (18.2,26.1] (18.2,26.1] (18.2,26.1] (18.2,26.1] (18.2,26.1] (10.4,18.2]
##  [7] (10.4,18.2] (18.2,26.1] (18.2,26.1] (18.2,26.1] (10.4,18.2] (10.4,18.2]
## [13] (10.4,18.2] (10.4,18.2] (10.4,18.2] (10.4,18.2] (10.4,18.2] (26.1,33.9]
## [19] (26.1,33.9] (26.1,33.9] (18.2,26.1] (10.4,18.2] (10.4,18.2] (10.4,18.2]
## [25] (18.2,26.1] (26.1,33.9] (18.2,26.1] (26.1,33.9] (10.4,18.2] (18.2,26.1]
## [31] (10.4,18.2] (18.2,26.1]
## Levels: (10.4,18.2] (18.2,26.1] (26.1,33.9]
```

```
cut(mtcars$mpg, 3, labels = c('tani', 'sredni', 'drogi'))
```

```
##  [1] sredni sredni sredni sredni sredni tani   tani   sredni sredni sredni
## [11] tani   tani   tani   tani   tani   tani   tani   drogi  drogi  drogi
## [21] sredni tani   tani   tani   sredni drogi  sredni drogi  tani   sredni
## [31] tani   sredni
## Levels: tani sredni drogi
```

for

Pętla `for` wygląda dość standardowo. Jej konstrukcja wygląda następująco:

```
for (wartosc in wektor) polecenie`
```

Wewnątrz pętli `for` możemy odwoływać się do aktualnej wartości za pomocą jej nazwy (`wartosc`).

Wyświetlenie wszystkich liczb od 1 do 10 i od 20 do 30 możemy więc wykonać w ten sposób:

```
for (i in 1:10) print(i)
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
for (i in 20:30) print(i)
```

```
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
```

Staramy się nie robić tak, jak w przykładzie niżej. Nie dlatego, że to nie działa, ale dlatego, że uznawane jest to za kiepski styl w R. Jeżeli mamy wektor, po którym możemy iterować, to nie powinniśmy wprowadzać dodatkowych zmiennych w naszym kodzie. W wielu językach programowania konstrukcje, takie jak ta poniżej, są w porzadku, w R jednak nie powinniśmy ich używać.

```
n <- 20
for (i in 1:11){
    print(n)
    n <- n + 1
}
```

```
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
```

Wektory po których iteruje pętla `for` nie muszą być liczbami. w R możliwe jest iterowanie po dowolnym wektorze.

```
rymowanka <- c('Entliczek', 'Pętliczek', 'Czerwony', 'Stoliczek')
for (slowo in rymowanka){ # tak robimy w R
    print(slowo) # możemy odwołać się do wartości za pomocą `slowo`
}
```

```
## [1] "Entliczek"
## [1] "Pętliczek"
## [1] "Czerwony"
## [1] "Stoliczek"
```

Dlatego też staramy się nie robić tak jak tutaj:

```
for(i in 1:length(rymowanka)) print(rymowanka[i]) # tak nie robimy w R PROSZĘ!
```

```
## [1] "Entliczek"
## [1] "Pętliczek"
## [1] "Czerwony"
## [1] "Stoliczek"
```

Czasami będziemy jednak do tego zmuszeni. Wyobraźmy sobie, że chcemy za pomocą R stworzyć krótkie podsumowanie naszej dotychczasowej pracy nad samochodami. Nauczymy się później jak zrobić to bardziej idiomatycznie dla R, teraz zróbmy to jednak za pomocą pętli `for`. Idea jest taka, żeby dla każdego samochodu wydrukować na ekranie zdanie dotyczace jego ekonomiczności. Skorzystamy w tym celu z funkcji `paste`.

```
for (wiersz in 1:nrow(mtcars)){
    zdanie <- paste( # funkcja umożliwiająca konkatenację zmiennych typu `character`
      row.names(mtcars)[wiersz], # row.names służy do wydobywania z ramki danych nazw wierszy
      'to samochód',
      mtcars[wiersz, 'ekonomicznosc'], # wybieramy dla każdego samochodu wartość z kolumny `ekonomiczno
      'w eksploatacji.'
      )
    print(zdanie)
}
```

```
## [1] "Mazda RX4 to samochód średni w eksploatacji."
## [1] "Mazda RX4 Wag to samochód średni w eksploatacji."
## [1] "Datsun 710 to samochód średni w eksploatacji."
## [1] "Hornet 4 Drive to samochód średni w eksploatacji."
## [1] "Hornet Sportabout to samochód średni w eksploatacji."
## [1] "Valiant to samochód średni w eksploatacji."
## [1] "Duster 360 to samochód tani w eksploatacji."
## [1] "Merc 240D to samochód średni w eksploatacji."
## [1] "Merc 230 to samochód średni w eksploatacji."
## [1] "Merc 280 to samochód średni w eksploatacji."
## [1] "Merc 280C to samochód średni w eksploatacji."
## [1] "Merc 450SE to samochód średni w eksploatacji."
## [1] "Merc 450SL to samochód średni w eksploatacji."
## [1] "Merc 450SLC to samochód średni w eksploatacji."
## [1] "Cadillac Fleetwood to samochód tani w eksploatacji."
## [1] "Lincoln Continental to samochód tani w eksploatacji."
## [1] "Chrysler Imperial to samochód tani w eksploatacji."
## [1] "Fiat 128 to samochód drogi w eksploatacji."
## [1] "Honda Civic to samochód drogi w eksploatacji."
## [1] "Toyota Corolla to samochód drogi w eksploatacji."
## [1] "Toyota Corona to samochód średni w eksploatacji."
## [1] "Dodge Challenger to samochód średni w eksploatacji."
```

```
## [1] "AMC Javelin to samochód średni w eksploatacji."
## [1] "Camaro Z28 to samochód tani w eksploatacji."
## [1] "Pontiac Firebird to samochód średni w eksploatacji."
## [1] "Fiat X1-9 to samochód drogi w eksploatacji."
## [1] "Porsche 914-2 to samochód drogi w eksploatacji."
## [1] "Lotus Europa to samochód drogi w eksploatacji."
## [1] "Ford Pantera L to samochód średni w eksploatacji."
## [1] "Ferrari Dino to samochód średni w eksploatacji."
## [1] "Maserati Bora to samochód tani w eksploatacji."
## [1] "Volvo 142E to samochód średni w eksploatacji."
```

Jeżeli już *koniecznie* musimy z jakiegoś powodu iterować po każdej wartości z macierzy, możemy to zrobić za pomocą zagnieżdżonych dwóch pętli `for`. Spróbujmy stworzyć macierz i dodać 5 do każdej niepodzielnej przez trzy liczby oraz odjąć 5 od każdej podzielnej przez 3.

```
macierz = matrix(1:25, 5,5)
macierz
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

```
# Podzielność przez 3 sprawdzamy operatorem modulo czyli %
for (kolumna in 1:ncol(macierz)){ # pierwsza pętla for iteruje po kolumnach
    for (wiersz in 1:nrow(macierz)){ # druga pętla for po wierszach
        if (macierz[wiersz,kolumna] %% 3 == 0){
            macierz[wiersz,kolumna] <- macierz[wiersz,kolumna] - 5} else {
            macierz[wiersz,kolumna] <- macierz[wiersz,kolumna] + 5}
}}
macierz
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    6    1   16   21   16
## [2,]    7   12    7   22   27
## [3,]   -2   13   18   13   28
## [4,]    9    4   19   24   19
## [5,]   10   15   10   25   30
```

Bardziej obeznany z R zrobiłby to oczywiście inaczej. Niezłym pomysłem jest przekonwertowanie macierzy do postaci wektora, zaaplikowanie funkcji `ifelse` i stworzenie z wektora nowej macierzy.

```
macierz <- matrix(1:25, 5,5)
m <- as.numeric(macierz)
matrix(ifelse(m %% 3 == 0,  # warunek
              m - 5,  # jeśli warunek spełniony
              m + 5) # jeśli warunek niespełniony
       ,5,5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    6    1   16   21   16
## [2,]    7   12    7   22   27
## [3,]   -2   13   18   13   28
## [4,]    9    4   19   24   19
## [5,]   10   15   10   25   30
```

while

Istotą pętli `while` jest powtarzanie pewnego bloku kodu, dopóki `warunek` jest prawdziwy.

```
while (warunek) polecenie
```

Spróbujmy za pomocą pętli `while` stworzyć kod do sprawdzania, czy dana liczba jest liczbą pierwszą implementując kiepski algorytm, który próbuje podzielić ją przez kolejne liczby naturalne większe od 2.

```
n <- 421 # liczba do sprawdzenia
pierwsza <- FALSE # czy jest to liczba pierwsza?
d <- 2 # pierwszy potencjalny dzielnik jaki sprawdzamy

while(!pierwsza){
    if (n %% d == 0){
      print(d)
      pierwsza = TRUE
      }
    d <- d+1
}
```

```
## [1] 421
```

```
pierwsza
```

```
## [1] TRUE
```

Zamiast while możemy użyć pętli `for` ze kluczowym słowem `break`. `break` użyte w kodzie sprawia, że R "ucieka" z nadrzędnej pętli. W naszym wypadku po znalezieniu dzielnika R opuszcza nadrzędną pętle `for`.

```
n <- 35
for(i in 2:n){
    if(n %% i == 0){
        print(paste('Dzielnik to', as.character(i)))
        break # jeżeli warunek się spełni, to nie wykona się już żadna iteracja `for`
    } else{
        print(paste(as.character(i), 'nie jest dzielnikiem.'))
    }
}
```

```
## [1] "2 nie jest dzielnikiem."
## [1] "3 nie jest dzielnikiem."
## [1] "4 nie jest dzielnikiem."
## [1] "Dzielnik to 5"
```

Tutaj mamy trochę lepszy algorytm, który wykorzystuje fakt że:

- jeżeli liczba n nie jest liczbą pierwszą, to pierwiastek każdego jej dzielnika jest od niej mniejszy
- po sprawdzeniu podzielności przez 2 nie musimy sprawdziać, czy dana liczb dzieli się przez liczby parzyste
- po sprawdzeniu podzielności przez 3 nie musimy sprawdzać, czy dzieli się przez wielokrotności 3

```
n <- 421
pierwsza <- FALSE

while(!pierwsza){
    if (n <= 3){
      print('To jest liczba pierwsza')
      pierwsza = TRUE
```

```
      }
    if (n %% 2 == 0 | n %% 3 == 0){
      print('Dzielnik to 2 lub 3')
      pierwsza = FALSE
      }
    i = 5
    while (sqrt(i) < n){
        if((n %% i) == 0 | (n %% (i+2)) == 0){
            p <- paste('Dzielnik to', as.character(i), 'lub', as.character(i+2))
            print(p)
            pierwsza = TRUE}
        i <- i + 6
    }
}
```

```
## [1] "Dzielnik to 419 lub 421"
```