MDCS R — ćwiczenia 1

17 styczeń 2014

1 Zrzut rzeczy które były na beamerze

```
1:10
```

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

```
1:100
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

```
7
```

```
## [1] 7
```

```
1.45
```

```
## [1] 1.45
```

```
2 + 2
```

```
## [1] 4
```

```
1:3 + 2:4
```

```
## [1] 3 5 7
```

```
1:6 + 1:2
```

```
## [1] 2 4 4 6 6 8
```

```
1:6 + 1
```

```
## [1] 2 3 4 5 6 7
```

```
1:6 + 1:5
```

```
## Warning: długość dłuszego obiektu nie jest wielokrotnością długości
krótszego obiektu
```

```
## [1]  2  4  6  8 10  7
```

```
a <- 1:6
b <- 1:7
a
```

```
## [1] 1 2 3 4 5 6
```

```
b
```

```
## [1] 1 2 3 4 5 6 7
```

```
a * 2
```

```
## [1]  2  4  6  8 10 12
```

```
c(a, 7)
```

```
## [1] 1 2 3 4 5 6 7
```

```
c(1:3, 7)
```

```
## [1] 1 2 3 7
```

```
c(9, c(1:3, 7))
```

```
## [1] 9 1 2 3 7
```

```
seq(2, 10)
```

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

```
seq(2, 10, 0.3)
```

```
##  [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0 5.3 5.6 5.9 6.2 6.5 6.8
## [18] 7.1 7.4 7.7 8.0 8.3 8.6 8.9 9.2 9.5 9.8
```

```
seq(2, 10, length = 20)
```

```
##  [1]  2.000  2.421  2.842  3.263  3.684  4.105  4.526  4.947  5.368  5.789
## [11]  6.211  6.632  7.053  7.474  7.895  8.316  8.737  9.158  9.579 10.000
```

```
seq(2, 10, by = 0.3)
```

```
##  [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0 5.3 5.6 5.9 6.2 6.5 6.8
## [18] 7.1 7.4 7.7 8.0 8.3 8.6 8.9 9.2 9.5 9.8
```

```
2 * 2
```

```
## [1] 4
```

```
2/2
```

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

```
2 - 2
```

```
## [1] 0
```

```
2^2
```

```
## [1] 4
```

```
2^2
```

```
## [1] 4
```

```
7%%2
```

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

```
7%/%2
```

```
## [1] 3
```

```
sum(1:10)
```

```
## [1] 55
```

```
prod(1:10)
```

```
## [1] 3628800
```

```
mean(1:10)
```

```
## [1] 5.5
```

```
length(1:10)
```

```
## [1] 10
```

```
# Uzyskaj sumę kwadratów liczb od 1 do 100
(1:100)^2
```

```
##   [1]     1     4     9    16    25    36    49    64    81   100   121
##  [12]   144   169   196   225   256   289   324   361   400   441   484
##  [23]   529   576   625   676   729   784   841   900   961  1024  1089
##  [34]  1156  1225  1296  1369  1444  1521  1600  1681  1764  1849  1936
##  [45]  2025  2116  2209  2304  2401  2500  2601  2704  2809  2916  3025
##  [56]  3136  3249  3364  3481  3600  3721  3844  3969  4096  4225  4356
##  [67]  4489  4624  4761  4900  5041  5184  5329  5476  5625  5776  5929
##  [78]  6084  6241  6400  6561  6724  6889  7056  7225  7396  7569  7744
##  [89]  7921  8100  8281  8464  8649  8836  9025  9216  9409  9604  9801
## [100] 10000
```

```
sum((1:100)^2)
```

```
## [1] 338350
```

```
# Uzyskaj wektor liczb od 1 do 10 w którym parzyste liczby zostały
# podzielone przez 2
(1:10)
```

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

```
(1:10)/(1:2)
```

```
##  [1] 1 1 3 2 5 3 7 4 9 5
```

```
(1:12)/c(1, 1, 3)
```

```
##  [1]  1  2  1  4  5  2  7  8  3 10 11  4
```

```
(1:12)^c(2, 1, 1)
```

```
##  [1]   1   2   3  16   5   6  49   8   9 100  11  12
```

```
sin(pi)
```

```
## [1] 1.225e-16
```

```
pi
```

```
## [1] 3.142
```

```
sin(pi * 2)
```

```
## [1] -2.449e-16
```

```
cos(pi * 2)
```

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

```r
log(1:100)
```

```
##   [1] 0.0000 0.6931 1.0986 1.3863 1.6094 1.7918 1.9459 2.0794 2.1972 2.3026
##  [11] 2.3979 2.4849 2.5649 2.6391 2.7081 2.7726 2.8332 2.8904 2.9444 2.9957
##  [21] 3.0445 3.0910 3.1355 3.1781 3.2189 3.2581 3.2958 3.3322 3.3673 3.4012
##  [31] 3.4340 3.4657 3.4965 3.5264 3.5553 3.5835 3.6109 3.6376 3.6636 3.6889
##  [41] 3.7136 3.7377 3.7612 3.7842 3.8067 3.8286 3.8501 3.8712 3.8918 3.9120
##  [51] 3.9318 3.9512 3.9703 3.9890 4.0073 4.0254 4.0431 4.0604 4.0775 4.0943
##  [61] 4.1109 4.1271 4.1431 4.1589 4.1744 4.1897 4.2047 4.2195 4.2341 4.2485
##  [71] 4.2627 4.2767 4.2905 4.3041 4.3175 4.3307 4.3438 4.3567 4.3694 4.3820
##  [81] 4.3944 4.4067 4.4188 4.4308 4.4427 4.4543 4.4659 4.4773 4.4886 4.4998
##  [91] 4.5109 4.5218 4.5326 4.5433 4.5539 4.5643 4.5747 4.5850 4.5951 4.6052
```

```r
log(1:100, 10)
```

```
##   [1] 0.0000 0.3010 0.4771 0.6021 0.6990 0.7782 0.8451 0.9031 0.9542 1.0000
##  [11] 1.0414 1.0792 1.1139 1.1461 1.1761 1.2041 1.2304 1.2553 1.2788 1.3010
##  [21] 1.3222 1.3424 1.3617 1.3802 1.3979 1.4150 1.4314 1.4472 1.4624 1.4771
##  [31] 1.4914 1.5051 1.5185 1.5315 1.5441 1.5563 1.5682 1.5798 1.5911 1.6021
##  [41] 1.6128 1.6232 1.6335 1.6435 1.6532 1.6628 1.6721 1.6812 1.6902 1.6990
##  [51] 1.7076 1.7160 1.7243 1.7324 1.7404 1.7482 1.7559 1.7634 1.7709 1.7782
##  [61] 1.7853 1.7924 1.7993 1.8062 1.8129 1.8195 1.8261 1.8325 1.8388 1.8451
##  [71] 1.8513 1.8573 1.8633 1.8692 1.8751 1.8808 1.8865 1.8921 1.8976 1.9031
##  [81] 1.9085 1.9138 1.9191 1.9243 1.9294 1.9345 1.9395 1.9445 1.9494 1.9542
##  [91] 1.9590 1.9638 1.9685 1.9731 1.9777 1.9823 1.9868 1.9912 1.9956 2.0000
```

```r
exp(1)
```

```
## [1] 2.718
```

```r
exp(2)
```

```
## [1] 7.389
```

```r
sample(1:10)
```

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

```r
sample(1:10)
```

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

```r
sample(1:10)
```

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

```r
sample(1:10, 3)
```

```
## [1] 10  4  8

sample(1:10, 30)

## Error: nie można wziąć próby większej niż rozmiar populacji gdy
'replace = FALSE'

sample(1:10, 30, replace = TRUE)

##  [1]  2  1  7  3  8  7  2  3  4  4 10  7  1  5  9  4  9  4  4  2  4  4  8
## [24]  2 10 10  8  4  4  3

sort(sample(1:10, 30, replace = TRUE))

##  [1]  1  2  2  2  2  3  3  3  4  4  4  5  5  5  5  5  6  6  6  6  7  7  7
## [24]  8  8  8  9  9 10 10

sort(sample(1:10, 30, replace = TRUE), decreasing = TRUE)

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

rev(1:3)

## [1] 3 2 1

unique(c(1, 1, 2))

## [1] 1 2

unique(c(1, 1, 2, 1))

## [1] 1 2

unique(c(1, 1, 2, 1, 3, 1, 1, 5))

## [1] 1 2 3 5

unique(c(1, 1, 2, 1, 3, 1, 1, 5, 3))

## [1] 1 2 3 5

cumsum(1:10)

##  [1]  1  3  6 10 15 21 28 36 45 55

diff(1:10)

## [1] 1 1 1 1 1 1 1 1 1

cumsum(diff(1:10))
```

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

```
diff(c(1, 3, 5, 7, 11, 13))
```

```
## [1] 2 2 2 4 2
```

```
runif(10)
```

```
##  [1] 0.77172 0.16979 0.82125 0.38072 0.01709 0.90042 0.75165 0.21273
##  [9] 0.84739 0.49258
```

```
runif(10, -1, 1)
```

```
##  [1]  0.350850 -0.522140 -0.452468  0.198276  0.970107  0.103309 -0.557731
##  [8] -0.666721  0.009373  0.230826
```

```
runif(10, -1, 1) > 0
```

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

```
b <- runif(10, -1, 1) > 0
b
```

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

```
b * 2
```

```
##  [1] 0 0 2 2 2 0 0 2 2 2
```

```
sum(b)
```

```
## [1] 6
```

```
b <- runif(10, -1, 1) >= 0
c(2, 3) == c(1, 3)
```

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

```
!(c(2, 3) == c(1, 3))
```

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

```
(c(2, 3) == c(1, 3))
```

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

```
(c(2, 3) == c(1, 3)) & c(TRUE, TRUE)
```

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

```
(c(2, 3) == c(1, 3)) | c(TRUE, TRUE)
```

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

```
j <- c(52, 23, 20, 5, 3, 4, 3, 10, 20, 7, 11)
# Policzyć liczbę Eddingtona dla j
sort(j, decreasing = TRUE)
```

```
##  [1] 52 23 20 20 11 10  7  5  4  3  3
```

2 Co powinno się wiedzieć/umieć

- Jak włączyć i wyłączyć R.
- O co chodzi z wektoryzacją i recycling'iem.
- Umieć tworzyć wektory liczbowe za pomocą `:`, `c()` i `seq()` z `length=` i z `by=`.
- Umieć wykonywać podstawowe operacje matematyczne operatorami `+ - * / ^ %% %/%` i funkcjami `sin()`, `cos()`, `tan()`, `log()` i `exp()`.
- Umieć tworzyć zmienne operatorami `->` i `<-` i ich używać.
- Znać funkcje redukujące wektor do jednoelementowego: `length()`, `mean()`, `sum()`.
- Znać podstawowe funkcje zmieniające kolejność elementów: `rev()`, `sort()` i `sample()`.
- Znać `cumsum()` i `diff()`.
- Znać `runif()`.
- Umieć tworzyć wektory logiczne operatorami porównan `> >= <= < ==` i operować na nich operatorami `! & |`.

  Wiedzieć że wektor logiczny podsunięty funkcji lub operatorowi który chce liczb zmienia się w wektor zer w miejsce `FALSE` i jedynek w miejsce `TRUE`.